BSDCon Europe 2002

2nd European BSD Conference

lovember 15-17, 2002

lercure Hotel, Amsterdam, The Netherlands

Conference Proceedings

http://2002.eurobsdcon.org

Sidsel Jensen BSD-DK
purple.dhemster.dk..

Proceedings

BSDCon Europe 2002

http://2002.eurobsdcon.org

November 15–17, 2002
Amsterdam, The Netherlands

Organisers

Program Committee

Walter Belgers (chair) <walter@belgers.com>
Paul Kranenburg <pk@cs.few.eur.nl>
Frank van der Linden <fvdl@wasabisystems.com>
Wim Vandeputte <wvdputte@kd85.com>

Board members

Chair: Guido van Rooij *(Madison Gurkha)*
Secretary: Walter Belgers *(Madison Gurkha)*
Treasurer: Robert Kochheim *(SNOW)*

Volunteer

Jos Jansen *(SNOW)*

Conference Organiser (PCO)

ICONIQ — *The Professional Conference Organisers*
Mariëlle Klatten & Sabina Beeke <info@iconiq.nl>

Special thanks to the sponsors:
GANDI, **NLUUG**, **Stichting NLnet**, **TUNIX Internet Security & Training** and **USENIX**

Contents

Welcome to the European BSD Conference 2002

It is my pleasure to present to you the proceedings of the European BSD Conference, an international conference aimed at those who work with, develop for, or just plain *like* any of the BSD-derived operating systems.

These conference proceedings contain the printed version of the papers that have been accepted by the program committee. We received more submissions than we have time slots, so we had to disappoint a few people. Don't give up and submit your abstract to other conferences as well! I thank all speakers whose papers are included in these proceedings. I know how much work it is to get it finished in time.

The program committee members were (deliberately) chosen to represent more than just one BSD variant. The program committee members are all long-time BSD users with a background in NetBSD, OpenBSD and FreeBSD. This fact shows in the program (which features all three and also MacOS X). I thank all the program committee members for their work. We certainly have been able to create a very interesting program, with top speakers.

On the first day there are two tracks with four in-depth tutorials about various topics: device drivers, network forensics, firewalls and a modular disk I/O subsystem.

During the weekend, there will be plenty of activities. I am very pleased that Michael J. Karels is willing to give the keynote lecture. He will be talking about the past, present and future of BSD. After that, you will be able to make your selection from 18 different lectures. Participating yourself is also possible by hosting a Birds of a Feather (BOF) session.

We also have a vendor exhibition. The book vendors at the exhibition will be giving interesting discounts. There's also an internet access room. But it's not all computers: come to the social event to talk with others, have a beer and enjoy the entertainment. And... the conference is in Amsterdam so there will be something to see or do around the clock.

The European BSD Conference will be a great opportunity to learn, and also to meet other users and developers, not only from your favourite BSD flavour but from all BSDs. This event gives you a chance to learn from the developments, mistakes and new ideas of the other BSDs.

Thanks goes to the conference organisation, and especially Mariëlle and Sabina from ICONIQ who did a truly excellent job. Thanks also goes to Gandi, TUNIX, USENIX, Stichting NLnet, the NLUUG and all sponsors who have made this event possible.

And thanks to *you*, for coming to Amsterdam. I'll see you around!

Walter Belgers, Program Chair
Madison Gurkha, Eindhoven

Lectures

Virtual Private Networks using FreeBSD - a case study

Eilko Bos
Le Reseau Netwerksystemen B.V.
<eilko@reseau.nl>

People want to, for more than one reason, work at home, sharing information from the corporate network. Companies want to link their networks together over the internet. But nobody wants their private traffic to be readable on the internet. Various solutions have been made, and one of them is IPSec. IPSec stands for 'IP Security'. It provides authentication and encryption of networktraffic. But there's more to secure your data. It makes no sense to encrypt your data and have your endpoint compromised. Thus a firewall is needed as well. On both ends. And some more issues to cover.

This little piece of paper shows us how FreeBSD can help us with this. Although there are some differences between the various BSD's, it will reasonly apply to Open- and NetBSD as well.

Starting at the Gasunie I went to Philips CP (Now Atos Origin). Spend several years working for Origin and went back to Groningen (the north of the Netherlands). Since 2 years I am working for Le Reseau, an information security company. Le Reseau is an independant company, which does pentesting, gives security-related workshops, advises companies about to-be-implemented environments and so on. Most of my time I do pentesting and workshops.

Virtual Private Networks using *BSD
- a case study -

Index

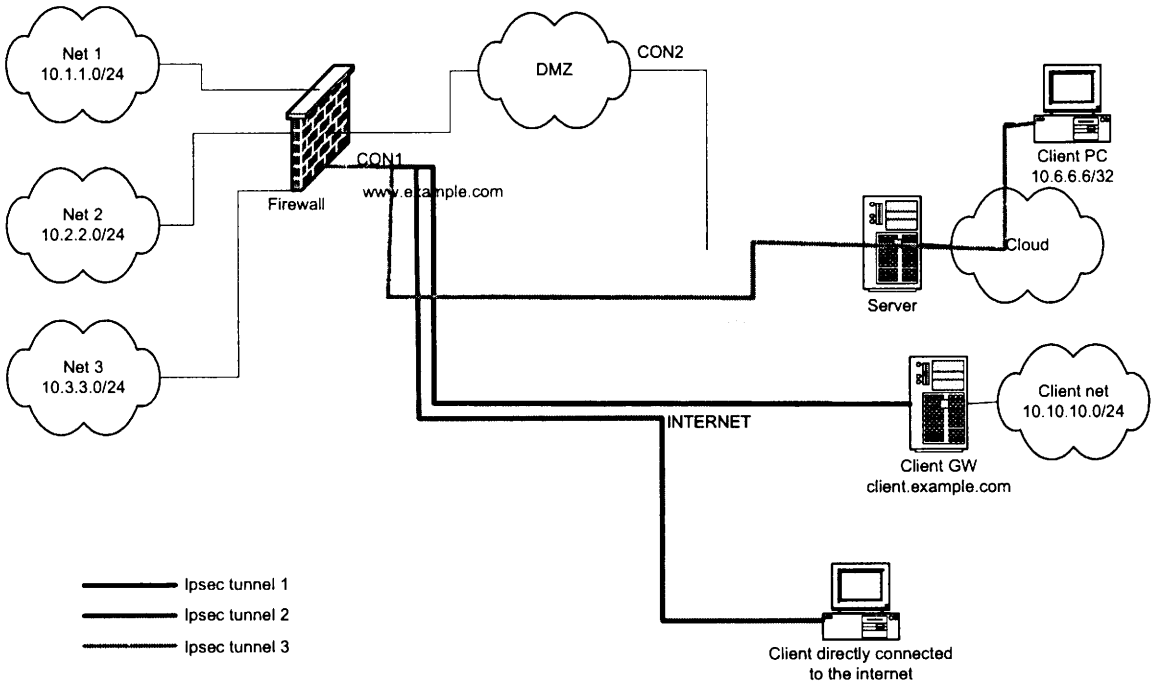
Introduction

People want to, for more than one reason, work at home, using information from the corporate network. Companies want to link their networks together over the Internet, but nobody wants his or her network traffic to be readable on the Internet. Various solutions have been developed, and one of them is IPsec.

This paper covers the use of IPsec on a FreeBSD client and an OpenBSD server, both using isakmpd in an example environment. This environment is pretty generic, so what is used here can be easily translated to other organizations.

Scenario

In the case described in this paper we have the following scenario:



The client must be able to connect from everywhere on the Internet independent from IP-addresses of the client. It might be connected via cable, which changes IP-address once in a while, or on a network behind such a connection. It can also be a gateway with a network behind it, let's say 10.10.10.0/24.

Since it does not have a fixed IP-address (as seen from the Internet) we call this a roaming client. In this paper it will be a FreeBSD machine running isakmpd and ipfilter.
The remote location can be configured as the roaming client.

The firewall has multiple interfaces. The client will enter the network on CON1. It is running OpenBSD with isakmpd and packet filter.

IPsec / ISAKMP

Introduction

IPsec is an open set of protocols, implemented in several operating systems by several companies. It does encryption, decryption and verification. It is implemented in the IP layer.

There are to protocols in IPsec:

- Authentication Header (AH) guarantees integrity of IP packet and protects it from intermediate alteration or impersonation. This is done by attaching a cryptographic checksum. This is protocol 51.
- Encapsulated Security Payload (ESP) protects the payload of an IP-packet by encrypting it with a secret key. This is protocol 51.

(Note that we are talking about protocols here, TCP is protocol 6, UDP is protocol 17)

The roaming client must be able to travel through a NATting firewall. Since NAT requires header rewriting it makes the AH-protocol unusable in this situation. So only ESP is used.

There are 2 modes to use IPsec in:

- Transport mode is used for peer-to-peer communication between nodes.
- Tunnel mode is used to let security gateways communicate with each other. Behind one or both gateways there can be a network using the tunnel-mode to communicate with the other end.

We use tunnel mode since the client has to reach a network behind the firewall.

Setting up an IPsec connection between two peers takes to phases:

- In the first phase, a secure channel is created to communicate between the peers. In this stage things like passphrase or certificates are exchanged.
- In the second phase, security associations (SA's) are negotiated by services such as IPsec. In this phase the details of the IPsec-link will be negotiated, such as encapsulation mode, lifetime and algorithms to use. Also the identities like host and network are specified.

Once these two steps are successfully finished, an encrypted tunnel exists. Later in this document, after showing how configuration and certificates are made, there is output of packets written by isakmpd (-l <file> option) and decoded by tethereal.

Security Policies and associations

Policies
During negotiating a security association (SA) and a security policy (SP) are made. The SP is ment to check wether an inbound or outbout IP packet applies to a SA.
Packets are identified by:

- Security Parameter Index (SPI)
- Source / Destination address
- ESP or AH protocol

For example, we have the following SP on the client from IPsec tunnel 3:
(192.0.34.72 is the firewall, 212.203.6.138 is the client, and 10.1.1.0/24 is the network behind the firewall.)

```
10.1.1.0/24[any] 212.203.6.138[any] any
       in ipsec
       esp/tunnel/192.0.34.72-212.203.6.138/use
       spid=26 seq=1 pid=27305
       refcnt=1
212.203.6.138[any] 10.1.1.0/24[any] any
       out ipsec
       esp/tunnel/212.203.6.138-192.0.34.72/require
       spid=25 seq=0 pid=27305
       refcnt=1
```

If a packet enters the client with destination 212.203.6.138, and it originates (encrypted) from 192.0.34.72 it will be decrypted according to what is associated in the SA (see below). If in the decrypted packet 10.1.1.0/24 is the source, the packet will pass, otherwise it will be dropped.

If a packet wants to leave the client, it is checked against the SP. If it has 10.1.1.0/24 as destination, from 212.203.6.138, it must be encrypted and sent to 192.0.34.72.

This information can be retrieved using the command 'setkey -DP' on FreeBSD.
On the OpenBSD server this information is shown with the command 'netstat':

```
$ netstat -rn -f encap
Routing tables

Encap:
Source             Port  Destination        Port  Proto
SA(Address/Proto/Type/Direction)
212.203.6.138/32    0     10.1.1.0/24         0     0     192.0.34.72/50/use/in
10.1.1.0/24         0     212.203.6.138/32    0     0
192.0.34.72/50/require/out
```

Packets that apply to one of those routing rules are treated accordingly.

Associations

If a packet passes the SP it is brought to the SA.
How the packets must be encrypted or decrypted is stored in the Security Associations Database (SAD). The content of this database can, on FreeBSD, be retreived using the command '*setkey -D*'. The content looks like this:

```
212.203.6.138 192.0.34.72
        esp mode=any spi=903662016(0x35dcc9c0) reqid=0(0x00000000)
        E: 3des-cbc  946b0af0 c3c001f7 87e87f4e 25eb9c73 e9a04dbe c1b8cfe1
        A: hmac-md5  080a57ad 6a82512c 0e40e506 f00a0578
        seq=0x00000000 replay=0 flags=0x00000000 state=mature
        created: Oct 11 11:55:40 2002    current: Oct 11 11:57:50 2002
        diff: 130(s)  hard: 600(s)  soft: 540(s)
        last:                       hard: 0(s)    soft: 0(s)
        current: 0(bytes)    hard: 0(bytes)       soft: 0(bytes)
        allocated: 0 hard: 0       soft: 0
        sadb_seq=1 pid=27304 refcnt=1
192.0.34.72 212.203.6.138
        esp mode=any spi=914016611(0x367ac963) reqid=0(0x00000000)
        E: 3des-cbc  89e40f1d 663e82e9 9130eb12 c48482d1 3dfa4c2b c05e1d40
        A: hmac-md5  ba89e8a3 fc523573 228df1d4 c46069b2
        seq=0x00000000 replay=0 flags=0x00000000 state=mature
        created: Oct 11 11:55:40 2002    current: Oct 11 11:57:50 2002
        diff: 130(s)  hard: 600(s)  soft: 540(s)
        last:                       hard: 0(s)    soft: 0(s)
        current: 0(bytes)    hard: 0(bytes)       soft: 0(bytes)
        allocated: 0 hard: 0       soft: 0
        sadb_seq=0 pid=27304 refcnt=1
```

Configuration

Kernel configuration

We will now discuss how the server and the clients are configured. As example we take the IPsec connection #1, a gateway with a network 10.10.10.0/24 behind it, connecting to the firewall/VPN gateway to reach network 1.

First we need to make the kernel of the machines IPsec enabled.
On OpenBSD this is done by adding:

```
Option              IPSEC         # IPsec
pseudo-device       enc     1     # IPSEC needs the encapsulation interface
```

to the kernel configuration and make and install a kernel with this new configuration.

FreeBSD uses the lines:

```
Options         IPSEC          #IP security
Options         IPSEC_ESP      #IP security (crypto; define w/ IPSEC)
Options         IPSEC_DEBUG    #debug for IP security
```

Also for FreeBSD a new kernel must be built and installed.

Now the machines have an IPsec enabled kernel, the next step will be setting up IPsec. Before doing so, decisions have to be made on the type of authentication to use. Either passphrases or X509 certificates can be used. Since passphrases will be to weak for this environment, X509 certificates are used. This implies the need of a Certificate Authority (CA). You can go out on the Internet and go to a CA that can deliver you the needed certificates. But if you trust yourself and your network, you can setup your own.

Setting up a Certificate Authority

In our environment we have setup a dedicated machine that can only be accesses by certain people using ssh-keys. All other access to the machine is denied. This is the machine where certificates are created and stored. They must be in a safe place (we allow physical access to the CA-server, though...).

To be able to create a CA openssl is needed. Be sure to use one of the latest versions.

As root, you need to do the following:

```
# mkdir -p /etc/ssl/private
# openssl genrsa -out /etc/ssl/private/ca.key 1024
# openssl req -new -key /etc/ssl/private/ca.key \
        -out /etc/ssl/private/ca.csr
```

During the third step some questions needs to be answered. The answers are printed bold.

```
Country Name (2 letter code) [AU]:NL
State or Province Name (full name) [Some-State]:Groningen
Locality Name (eg, city) []:Groningen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:EuroBSDcon
Organizational Unit Name (eg, section) []:EuroBSDcon CA dept.
Common Name (eg, YOUR name) []:BOFH
Email Address []:bofh@eurobsdcon.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:Tru5t'nmyCA
An optional company name []:
```

Now we have a CA key (ca.key) and a certificate signing request (ca.csr) file. The latter one is used to create a certificate that will be signed with the key:

```
# openssl x509 -req -days 365 -in /etc/ssl/private/ca.csr \
        -signkey /etc/ssl/private/ca.key \
        -extfile /etc/ssl/x509v3.cnf -extensions x509v3_CA \
        -out /etc/ssl/ca.crt
```

Note that the file "/etc/ssl/x509v3.cnf" is not available in a default FreeBSD install. However it can be fetched from OpenBSD and used without problems.

Now we have a CA certificate, ca.crt. Together with the CA key this certificate is used to create client certificates. Setting up a CA is done.

Creating client certificates

Creating certificates is done on the same machine as the CA because the keys of the CA are needed. For each peer, certificates are needed. Since the solution needs to be IP-address independent, we choose to use FQDN certificates.

To store the certificates for more than one peer, a directory structure is set up like it would be used in isakmpd:

```
/usr/local/cert/FQDN1/ca
                |     |--/certs
                |     |--/private
                |
                |-/FQDN2/ca
                |     |--/certs
                |     |--/private
                |
```

where 'FQDN{1|2}' represents the FQDN of the peer, e.g. www.example.com. For each new peer a new set of directories is created.

After creating this directory structure a certificate signing request for the server is created:

```
# openssl genrsa -out
/usr/local/cert/www.example.com/private/www.example.com.key 1024
# openssl req -new -key
/usr/local/cert/www.example.com/private/www.example.com.key \
              -out /usr/local/cert/www.example.com/private/www.example.com.csr
```

Now the certificate itself must be created for the server:

```
# cd /usr/local/cert/www.example.com/private
# setenv CERTFQDN www.example.com
# openssl x509 -req -days 365 -in www.example.com.csr \
        -CA /etc/ssl/ca.crt -CAkey /etc/ssl/private/ca.key \
        -CAcreateserial \
        -extfile /etc/ssl/x509v3.cnf -extensions x509v3_FQDN \
        -out www.example.com.crt
```

Now the certificate is copied to the 'cert' directory and the CA cert is included:

```
# cp www.example.com.crt ../certs/
# cp /etc/ssl/ca.crt /usr/local/cert/www.example.com/ca
```

Now there is a complete set of certificates that can be transferred to the server.
For the client the same steps must be performed, but now with the FQDN the client will use. This can be a non-existing FQDN like 'client.example.com' since the mechanism of checking certificates only checks an abstract of the certificate and it is not checked against the real FQDN. This enables the use for roaming clients.

Distribution and use of certificates

At this stage the isakmpd configuration files are entering the scene.
On the server side, a generic setup is needed which will allow all clients to connect and authenticate, without having the configuration file updated each time a client is added. The same CA signs certificates from the client and server. When peers authenticate each other they will create an abstract of information from the received certificate and compare that with an abstract of what is in the policy file. Since the abstract of the CA certificate will be the same, it is possible to add the ca.crt to the policy file and distribute this to all the peers (including the server). In this case the policy file needs to be edited (depending on some settings).

We can create a generic policy file that will look like this:

```
# cat isakmpd.policy
Comment: This policy accepts ESP SAs from a remote that uses the right
password
        $OpenBSD: policy,v 1.6 2001/06/20 16:36:19 angelos Exp $
        $EOM: policy,v 1.6 2000/10/09 22:08:30 angelos Exp $
Authorizer: "POLICY"
licensees: "x509-base64:\
        MIIC1TCCAj6gAwIBAgIBADANBgkqhkiG9w0BAQQFADCBnTELMAkGA1UEBhMCTkwx\
        EjAQBgNVBAgTCUdyb25pbmdlbjESMBAGA1UEBxMJR3JvbmluZ2VuMRUwEwYDVQQK\
        EwxCcmFzYXBlbiBvcmcxFzAVBgNVBAsTDkhlYWRxdWFydGVyIENBMRIwEAYDVQQD\
        Ew1FaWxrbyBCb3MxIjAgBgkqhkiG9w0BCQEWE3RhZmthbUBicmFzYXBlbi5vcmcw\
        HhcNMDIU                                                  EBhMCTkwx\
        EjAQBgNV   THIS IS THE ca.crt from the ./ca directory    wEwYDVQQK\
        EwxCcmFz                                                 wEAYDVQQD\
        Ew1FaWxrbyBCb3MxIjAgBgkqhkiG9w0BCQEWE3RhZmthbUBicmFzYXBlbi5vcmcw\
        gZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJjcx3Hg66XEDz9kDX+6qE5lgX6e\
        70kbrmV1CQXyYV2wc3p9+bUjqqkm5AVYj3bUfyXOrngSSnPNQBO0Jqv8iRWcoGvy\
        qBeqYvxvmyoysTt5FAregstvg34whgbh375u65yh45egbdfvbtzNHq1nl7z+DRF9\
        UN88scqffYm/AJrPAgMBAAGjIzAhMBIGA1UdEwEB/wQIMAYBAf8CAQEwCwYDVR0P\
        BAQDAgKEMA0GCSqGSIb3DQEBBAUAA4GBAFdYPJEz5keXA7/6/JE4W5pFWSjnsPj3\
        wC/qR+ki/dQ0tAgt15f9W6zI6BUs8xUP4Va4EppbdDxIhyX+aiMJ3ETMlMu2h/rZ\
        MaBAByPZzNmMwP1b5J8rEP4rk9VwHpbQFA0RFJdCDo2ekIrcVM2bT89cz5IS91GT\
        NWLa0vTsNm9b"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "aes" &&
            esp_auth_alg == "hmac-sha" -> "true";
```

As extra check the Conditions can be added with the FQDN of the client:

```
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "aes" &&
            esp_auth_alg == "hmac-sha" &&
            (remote_id == "client.example.com"  ||
             remote_id == "another.example.com") -> "true";
```

This policy can be copied to the /usr/local/cert/FQDN directory for each client, so it can be transferred together with the certificates.

ISAKMPD configuration files

How a policy file is created is described in the previous part.
The only thing missing is a configuration file. This one differs from peer to peer, but a template can be generated. Let's walk though the configuration file of the server first. this file is different from the rest. Since it must be possible to use roaming users no IP-addresses of the client must appear in the configuration file. Furthermore, to improve readability, all unneeded lines are removed.

It looks like this, lines starting with a '#' and italic font are comments of the author.

```
# $ Id: $
# Paths in this file are paths as they would be on
# the final destination, in this case the server.
# General section
[General]
Policy-File=
/usr/local/etc/isakmpd/policy
Retransmits=            5
Exchange-max-time=      120
Listen-on=              192.0.34.72

# Incoming phase 1 negotiations are multiplexed on
# the source IP address
[Phase 1]
Default=                ISAKMP-peer-GNU

# These connections are walked over after config
# file parsing and told to the application layer
# so that it will inform us when traffic wants to
# pass over them. This means we can do on-demand
# keying.
[Phase 2]
Connections=            IPsec-OBSD-GNU

# The peers
[ISAKMP-peer-GNU]
Phase=                  1
Transport=              udp
Local-address=          192.0.34.72
# We don't want the peer in the configfile.
# Instead we point to our identity and leave the
# rest to certificates and the policy file
ID=                     work-ID
Configuration=          Default-main-mode

[work-ID]
ID-type=                FQDN
Name=                   www.example.com

# The different connections
[IPsec-OBSD-GNU]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-GNU
Configuration=          Default-quick-mode
Local-ID=               Net-OBSD
Remote-ID=              Net-GNU

# Our Networks
[Net-GNU]
# This is the remote network. We leave it generic
# it will be negotiated, and the configuration of
# the peer will be used.
ID-type=                IPV4_ADDR_SUBNET
Network=                0.0.0.0
Netmask=                0.0.0.0

[Net-OBSD]
ID-type=                IPV4_ADDR_SUBNET
Network=                10.1.1.0
Netmask=                255.255.255.0

# Certificates stored in PEM format
[X509-certificates]
CA-directory=           /usr/local/etc/isakmpd/ca
Cert-directory=
/usr/local/etc/isakmpd/certs/
Private-key=      \
/usr/local/etc/isakmpd/private/www.example.com.ke

# Phase 1 descriptions
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=             3DES-SHA,3DES-MD5

# Main mode transforms
#########################
# 3DES
# Note "AUTHENTICATION_METHOD" it is not
#"PRE_SHARED"
# as it would by default
[3DES-SHA]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         SHA
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

[3DES-MD5]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

< snip rest of default config file >
```

This file can be stored in the directory /usr/local/certs/www.example.com/.
Now the files for the server are all in place. They can be transferred to the server in /usr/local/etc/isakmpd(/{ca|certs|private}/). Change the mode of the files to 600 so others cannot read them. If this is not done, isakmpd refuses to read the configuration file (*too open permissions*) and will not work.

What is left is the configuration of the client. The configuration file looks pretty much the same but in this file the peer (server) is explicitly named:

```
# cat /usr/local/certs/isakmpd.conf.template

# General section
# The IP-address of the client is
# @@MY_IP_ADDRESS@@ so it can be
substituted.
# The FQDN is @@MY_FQDN@@
[General]
Retransmits=            5
Exchange-max-time=      120
Listen-on=
@@MY_IP_ADDRESS@@
# Listen-on=            212.203.6.138

# Incoming phase 1 negotiations are
multiplexed on the source IP address
[Phase 1]
193.78.174.81=          ISAKMP-peer-GNU
Default=                ISAKMP-peer-GNU

# These connections are walked over
after config
# file parsing and told to the
application layer
# so that it will inform us when traffic
wants
# to pass over them. This means we can
do on-
# demand keying.
[Phase 2]
Connections=            IPsec-OBSD-GNU

# The peers
[ISAKMP-peer-GNU]
# over here we explicitely declare the
server's
# address.
Phase=                  1
Transport=              udp
Local-address=
@@MY_IP_ADDRESS@@
Address=                192.0.34.72
ID=                     my-ID
Configuration=          Default-main-
mode

[my-ID]
ID-type=                FQDN
Name=                   @@MY_FQDN@@
# Name=
client.example.com

# The different connections
[IPsec-OBSD-GNU]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-GNU
Configuration=          Default-quick-
mode
Local-ID=               Net-OBSD
Remote-ID=              Net-GNU

# Our Networks
[Net-GNU]
# this is the remote network to be reached.
ID-type=                IPV4_ADDR_SUBNET
Network=                10.1.1.0
Netmask=                255.255.255.0

[Net-OBSD]
# this is our network
ID-type=                IPV4_ADDR_SUBNET
Network=                10.10.10.0
Netmask=                255.255.255.0

# Certificates stored in PEM format
[X509-certificates]
CA-directory=    /etc/isakmpd/ca/
Cert-directory=  /etc/isakmpd/certs/
Private-key= \
/etc/isakmpd/private/@@MY_FQDN@@.key

# Phase 1 descriptions
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=             3DES-SHA,3DES-MD5

# Main mode transforms
#########################
# 3DES

[3DES-SHA]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         SHA
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

[3DES-MD5]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

< snip rest of default configfile >
```

This file can be used for every new client, only the @@MY_IP_ADDRESS@@ and @@MY_FQDN@@ must be substituted. Copy this template to /usr/local/certs/FQDN(client)/ and do the substitution.

Now the configuration of the client is done as well. The directory /usr/local/certs/FQDN(client) can be transferred to the client machine in /usr/local/etc/isakmpd/.

Setting up the connection

Now on both sides isakmpd can be started. For debugging purposes both can write everything to STDOUT (-D A=99), which is very noisy. It is better to redirect it to a file. Also both can be started with the '-l <file>' option, causing isakmpd to write negotiation packets to a pcap-file.

On both sides isakmpd is (for debugging) started as:

```
# cd /usr/local/etc/isakmpd
# isakmpd -c isakmpd.conf -l /tmp/ike.pcap -D A=99 -d > /tmp/ike.debug 2>&1 &
```

The file /tmp/ike.pcap can now be read into tethereal (which is installed with the net/ethereal port):

```
# tethereal -V -r /tmp/ike.pcap > /tmp/ike_readable.txt
```

The result is a file that is human readable. The negotiation took 9 packets in total:

```
22:20:02.015224 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident: [|sa]
22:20:02.160553 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident: [|sa]
22:20:02.242995 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident: [|ke]
22:20:02.516224 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident: [|ke]
22:20:02.676648 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident[E]:
       [encrypted id]
22:20:03.478109 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident[E]:
       [encrypted id]
22:20:03.590456 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 2/others I
       oakley-quick[E]: [encrypted hash]
22:20:03.963916 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 2/others R
       oakley-quick[E]: [encrypted hash]
22:20:03.986819 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 2/others I
       oakley-quick[E]: [encrypted hash]
```

Below is the (snipped) output of each packet from the servers side, the connection is iniated by the server:

```
Frame 1 (144 on wire, 144 captured)
Null/Loopback
    Family: IP (0x00000002)
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
    Version: 4
    Protocol: UDP (0x11)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Initiator cookie, Responder cookie
    Next payload: Security Association (1)
    Version: 1.0
    Exchange type: Identity Protection (Main Mode) (2)
    Flags
        .... ...0 = No encryption
        .... ..0. = No commit
        .... .0.. = No authentication
    Message ID: 0x00000000
    Length: 112
    Security Association payload
        Next payload: NONE (0)
        Length: 84
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 72
```

```
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI size: 0
            Number of transforms: 2
            Transform payload
                Next payload: Transform (3)
                Length: 32
                Transform number: 0
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): SHA (2)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)
            Transform payload
                Next payload: NONE (0)
                Length: 32
                Transform number: 1
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): MD5 (1)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)

Frame 2 (112 on wire, 112 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
    Version: 4
    Protocol: UDP (0x11)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Security Association (1)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 80
    Security Association payload
        Next payload: NONE (0)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI size: 0
            Number of transforms: 1
            Transform payload
                Next payload: NONE (0)
                Length: 32
                Transform number: 0
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): SHA (2)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)
```

```
Frame 3 (212 on wire, 212 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Key Exchange (4)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 180
    Key Exchange payload
        Next payload: Nonce (10)
        Length: 132
        Key Exchange Data
    Nonce payload
        Next payload: NONE (0)
        Length: 20
        Nonce Data

Frame 4 (212 on wire, 212 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Key Exchange (4)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 180
    Key Exchange payload
        Next payload: Nonce (10)
        Length: 132
        Key Exchange Data
    Nonce payload
        Next payload: NONE (0)
        Length: 20
        Nonce Data

Frame 5 (999 on wire, 999 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Identification (5)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 967
    Identification payload
        Next payload: Certificate (6)
        Length: 32
        ID type: FQDN (2)
        Protocol ID: Unused
        Port: Unused
        Identification data: www.example.com
    Certificate payload
        Next payload: Signature (9)
        Length: 747
        Certificate encoding: 4 - X.509 Certificate - Signature
        Certificate Data
    Signature payload
        Next payload: Notification (11)
        Length: 132
        Signature Data
    Notification payload
        Next payload: NONE (0)
        Length: 28
```

```
        Domain of Interpretation: IPSEC (1)
        Protocol ID: ISAKMP (1)
        SPI size: 16
        Message type: INITIAL-CONTACT (24578)
        Security Parameter Index

Frame 6 (980 on wire, 980 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Identification (5)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 948
    Identification payload
        Next payload: Certificate (6)
        Length: 24
        ID type: FQDN (2)
        Protocol ID: Unused
        Port: Unused
        Identification data: client.example.com
    Certificate payload
        Next payload: Signature (9)
        Length: 732
        Certificate encoding: 4 - X.509 Certificate - Signature
        Certificate Data
    Signature payload
        Next payload: Notification (11)
        Length: 132
        Signature Data
    Notification payload
        Next payload: NONE (0)
        Length: 28
        Domain of Interpretation: IPSEC (1)
        Protocol ID: ISAKMP (1)
        SPI size: 16
        Message type: INITIAL-CONTACT (24578)
        Security Parameter Index
    Extra data: 00000000

Frame 7 (320 on wire, 320 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 288
    Hash payload
        Next payload: Security Association (1)
        Length: 24
        Hash Data
    Security Association payload
        Next payload: Nonce (10)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: IPSEC_ESP (3)
            SPI size: 4
```

```
            Number of transforms: 1
            SPI: 4FF93BE9
            Transform payload
                Next payload: NONE (0)
                Length: 28
                Transform number: 1
                Transform ID: 3DES (3)
                SA-Life-Type (1): Seconds (1)
                SA-Life-Duration (2): Duration-Value (600)
                Encapsulation-Mode (4): Tunnel (1)
                Authentication-Algorithm (5): HMAC-MD5 (1)
                Group-Description (3): Group-Value (2)
    Nonce payload
        Next payload: Key Exchange (4)
        Length: 20
        Nonce Data
    Key Exchange payload
        Next payload: Identification (5)
        Length: 132
        Key Exchange Data
    Identification payload
        Next payload: Identification (5)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 10.1.1.0/255.255.255.0
    Identification payload
        Next payload: NONE (0)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 212.203.6.138/255.255.255.255

Frame 8 (324 on wire, 324 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 292
    Hash payload
        Next payload: Security Association (1)
        Length: 24
        Hash Data
    Security Association payload
        Next payload: Nonce (10)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: IPSEC_ESP (3)
            SPI size: 4
            Number of transforms: 1
            SPI: 5838A401
            Transform payload
                Next payload: NONE (0)
                Length: 28
                Transform number: 1
                Transform ID: 3DES (3)
```

```
                SA-Life-Type (1): Seconds (1)
                SA-Life-Duration (2): Duration-Value (600)
                Encapsulation-Mode (4): Tunnel (1)
                Authentication-Algorithm (5): HMAC-MD5 (1)
                Group-Description (3): Group-Value (2)
    Nonce payload
        Next payload: Key Exchange (4)
        Length: 20
        Nonce Data
    Key Exchange payload
        Next payload: Identification (5)
        Length: 132
        Key Exchange Data
    Identification payload
        Next payload: Identification (5)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 10.1.1.0/255.255.255.0
    Identification payload
        Next payload: NONE (0)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 212.203.6.138/255.255.255.255
    Extra data: 00000000

Frame 9 (84 on wire, 84 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 52
    Hash payload
        Next payload: NONE (0)
        Length: 24
        Hash Data
```

The message ID of the phase 1 part of the conversation is 0x00000000. As soon as phase 2 is reached, the message ID is changed to 0x29a5c670. In this conversation it is clear to see that in phase 1 authentication is done, while negotiation about network properties is done in phase 2.

Helpful tools

This conversation resulted in a correct working Ipsec link. But in many cases of new developed IPsec links, many things can go wrong. Captured packets as shown before are nice to have for debugging but will often not suffice. Looking at the outputfile of isakmpd can help a lot. As said before, this output is very noisy, but it contains a lot of useful information.

It can happen that routing tables are not correctly setup. E.g. a VPN-client with 2 IP-addresses on the same interface might place packets destined for the tunnel on the wrong IP-address. Look at the routing table to see if something missing.

In the following chapter a brief overview of firewall settings will be given. When testing IPSec in secure environments, firewalls might cause problems with respect to negotiation or ESP-traffic. If possible, stop the firewall for testing, or log all the blocked traffic. On FreeBSD this is done with 'ipmon(8)'. On OpenBSD this is done with a tcpdump on pflog (man pflog).

Firewalling

Client site

There are some demands on the client site for firewalling. It is not allowed to access the Internet directly. If an IPsec connection is active, all Internet activity should go via the proxy server on network 1. DNS-requests are sent to the name server on network 1.

To keep control over the traffic, only a few protocols are allowed to work remotely. These protocols are ICA for Citrix and ssh.

Assuming that the network interface to the Internet is called 'fxp0' and the internal interface called 'fxp1', a firewall configuration on the client gateway running FreeBSD with ipfilter, would look like this:

```
# 10.10.10.1 = internal address of client gateway
# 10.10.10.2 = nameserver
# 10.10.10.16 = admin's PC
# No antispoofing is needed on this interface. We only allow from one IP
address.
# Allow incoming and outgoing IPsec traffic from the server
pass in quick on fxp0 proto udp from 97.0.34.72 to 212.203.6.138 port = 500 \
   keep state
pass out quick on fxp0 proto udp from 212.203.6.138 to 97.0.34.72 port = 500 \
   keep state
pass in quick on fxp0 proto esp from 97.0.34.72 to 212.203.6.138
pass out quick on fxp0 proto esp from 212.203.6.138 to 97.0.34.72
pass in quick on lo0 all
pass out quick on lo0 all
# Allow admin access from admin's PC
pass in quick on fxp1 proto tcp from 10.10.10.16 to 10.10.10.1 port = ssh \
   keep state
# allow outgoing dns queries
pass out quick on fxp1 proto udp from 10.10.10.1 port=53 to 10.10.10.2 \
   port=52 keep state
block in log quick on fxp1 from any to 10.10.10.1
# Perform some security on incoming traffic. It must come from network 1 or
# ourselves.
pass in quick on fxp1 from 10.10.10.0/24 to 10.1.1.0/24
pass in quick on fxp1 from 10.10.10.0/25 to 10.10.10.0/24
pass out quick on fxp1 from 10.1.1.0/24 to 10.10.10.0/24
pass out quick on fxp1 from 10.10.10.0/24 to 10.10.10.0/24
# block and log the rest
block in log level auth.alert quick all
block out log level auth.alert quick all
```

Note that this very simple firewall configuration does not allow e.g. ICMP traffic. This might cause problems with e.g. MTU discovery or other administrative network information. Restriction on type of traffic will be done on the server, since firewalling on the client is not that reliable. If it happens to get compromised it is easy to change the rules.

Server site

This site is a bit more complicated. On this site traffic of the client is checked, but that can only be done after it is decrypted. So it can't be done on the incoming interface. We filter on other interfaces.
The server is running OpenBSD with packet filter. It has 5 interfaces. For VPN traffic only 2 interfaces are used: the interface to the Internet where the VPN-traffic comes in and the interface to network 1. It is easy to add rules so VPN-traffic can go to the other networks as well.

```
# See pf.conf(5) for syntax and examples
# interfaces by network
lo_if         = "lo0"
ext_if        = "rl0"
net1_if       = "dc0"
net2_if       = "dc1"
net3_if       = "dc2"
dmz_if        = "dc3"
# network addresses
ifext_net     = "rl0/8"
if1_net       = "dc0/24"
if2_net       = "dc1/24"
if3_net       = "dc2/24"
ifdmz_net     = "dc3/24"
internal_net  = "10.0.0.0/8"      # internal network, including VPN networks
ifext_ip      = "192.0.34.72"     # www.example.com
# special machines for which there are distinct rules
proxy         = "10.1.1.10
nameserver    = "10.1.1.11"
citrix        = "10.1.1.12"
caserver      = "10.1.1.14"
admin         = "10.1.1.100"

#############################################################################
#  The Rules
#############################################################################
# default deny
block in log all
block out log all

# loopback interace
pass in quick on $lo_if all
pass out quick on $lo_if all

# VPN/Internet interface
# rules for $ext_if (in)
pass in quick on $ext_if proto udp from any port = isakmp to $ext_if_ip \
   port = isakmp keep state
pass in quick on $ext_if proto esp from any to $ext_if_ip keep state

# other rules for $ext_if (out)
pass out quick on $ext_if proto udp from $ext_if_ip port = isakmp to any \
   port = isakmp keep state
pass out quick on $ext_if proto esp from $ext_if_ip to any keep state
pass out quick on $ext_if proto icmp from $ext_if_ip to any icmp-type echoreq
keep state
pass out quick on $ext_if proto tcp from $proxy to any keep state
pass out quick on $ext_if proto tcp from $nameserver port = 53 to any \
   port = 53 keep state
pass out quick on $ext_if proto udp from $nameserver to any port = 53 keep
state

pass in quick on enc0 all
pass out quick on enc0 all
```

```
#####################
# network 1 interface
#####################
# ICA traffic
pass out quick on $net1_if proto tcp from $internal_net to $citrix \
   port = 1496 keep state
pass out quick on $net1_if proto udp from $internal_net to $citrix \
   port = 1604 keep state
# DNS traffic
pass out quick on $net1_if proto tcp from $internal_net port= 53 \
   to $nameserver port = 53 keep state
pass out quick on $net1_if proto udp from $internal_net to $nameserver \
   port = 53 keep state
# Proxy traffic
pass out quick on $net1_if proto tcp from $internal_net to $proxy \
   port = 3128 keep state
# ssh traffic
block out quick on $net1_if proto tcp from $internal_net to $net1_if port = 22
pass out quick on $net1_if proto tcp from $internal_net to $if1_net port = 22
keep state

# Incoming for management
pass in quick on $net1_if proto tcp from {$caserver, $admin, $proxy} to \
   $net1_if port = ssh flags S/FSRA keep state
block in log quick on $net1_if from any to $net1_if
block out log quick on $net1_if from $net1_if to any
```

This is a very simple firewall setup,but it allows certain traffic from the VPN-client to some services on network 1. Note that only some services are allowed. Blocking the rest at this stage, after the tunnel, is more reliable than blocking it at the client side.

Extra attention

The setup above applies to a client acting as a single machine behind a gateway or as a single machine directly connected to the Internet.

To avoid changing the configurations too much when the IP-address given by the ISP changes, it is possible to add an extra IP-address on the network interface:

```
# ifconfig ep0 add 10.7.7.7 netmask 255.255.255.0
# route add 10.0.0.0/8 10.7.7.7
```

An extra rule in the firewall configuration will be needed to allow traffic from 10.7.7.7 on the interface. After this has been done, the isakmpd configuration can be configured to listen on 10.7.7.7 instead of the external IP-address. When that external IP-address changes, only the firewall configuration needs to be changed and reloaded.

Running and tuning of OpenBSD network servers in a production environment

Philipp Bühler & Henning Brauer
sysfive.com GmbH / BS Web Services
<pb@sysfive.com>

Heavily loaded network servers can experience resource exhaustion. At best, resource exhaustion will slow server response, but left uncorrected, it can result in a crash of the server. In order to understand and prevent such situations, a knowledge of the internal operation of the operating system is required, especially how memory management works. This paper will provide an understanding of the memory management of OpenBSD, how to monitor the current status of the system, why crashes occur and how to prevent them.

Audience
Experienced network administrators and interested kernel hackers.

Required Skills
Good knowledge in the TCP/IP suite, basic understandig of memory management under UNIX and basic understandig of an UNIX kernel.

Philipp Bühler has over eight years UNIX experience (Solaris, Linux, FreeBSD, HP/UX, OpenBSD) and is co-founder of sysfive.com GmbH. Main working area is network services and security. If company leaves him enough time, he is working on the OpenBSD packetfilter. http://sysfive.com/

Henning Brauer is founder of BS Web Services, an Internet Service Provider, and is responsible for the server operations, planning, customer interfaces and backend development. He's an OpenBSD developer, mainly working on the packet filter and maintaining the in-tree apache. He recently developed privilege seperation for Apache. He started working with and on UNIX back in 1995 and worked with Linux, Solaris, FreeBSD, and of course OpenBSD. http://bsws.de

Running and tuning OpenBSD network servers in a production environment

Philipp Bühler
sysfive.com GmbH
pb@sysfive.com

Henning Brauer
BS Web Services
hb@bsws.de

October 8, 2002

Abstract

Heavily loaded network servers can experience resource exhastion. At best, resource exhaustion will slow server response, but left uncorrected, it can result in a crash of the server.

In order to understand and prevent such situations, a knowledge of the internal operation of the operating system is required, especially how memory management works.

This paper will provide an understanding of the memory management of OpenBSD, how to monitor the current status of the system, why crashes occur and how to prevent them.

1 Motivation

Our main motivation for this paper was the lack of comprehensive documentation about tuning network servers running under OpenBSD [Ope02], especially with regard to the memory usage of the networking code in the kernel.

Either one can get general information, or is "left alone" with the source code. This paper outlines how to deal with these issues, without reading the source code. At least one does not need to start in "nowhere-land" and dig through virtually everything.

This paper aims to give a deeper understanding on how the kernel handles connections and interacts with userland applications like the Apache webserver.

2 Resource Exhaustions

Running a publicly accessible server can always lead to unexpected problems. Typically it happens that resources get exhausted. There are numerous reasons for this, including:

Low Budget There's not enough money to buy "enough" hardware which would run an untuned OS.

Peaks Overload situations which can be expected (e. g. special use) or not (e. g. getting "slashdotted").

DoS Denial-of-Service by attackers flooding the server.

No matter what reason leads to an exhaustion, there are also different types of resources which can suffer from such a situation. We briefly show common types and countermeasures. Afterwards we go into detail about memory exhaustion.

2.1 I/O Exhaustion

It's very typical for network servers to suffer in this area. Often people just add more CPU to "help" a slowly reacting server, but this wouldn't help in such a case.

Usually one can detect such an exhaustion by using `vmstat(8)` or `systat(8)`. Detailed usage is shown in Section 5.1 There are also numerous I/O "bottlenecks" possible, but one typical indication is the CPU being mostly idle and blocked processes waiting for resources. Further distinctions can be made:

Disk

The process is waiting for blocks from (or to) the disk and cannot run on the CPU, even if the CPU is idle. This case could be resolved by moving from IDE to SCSI, and/or using RAID technology. If repetitive writes/reads are being done an increase of the filesystem-cache could also help [1]. Filesystem-cache can be configured with the kernel option `BUFCACHEPERCENT`[2].

NIC

Choosing the right network card is important for busy servers. There are lots of low-end models like the whole Realtek range. These cards are relatively dumb themselves. On the other hand, there are chipsets with more intelligence. DEC's 21143, supported by the dc(4) driver, and Intel's newer chipsets, supported by the fxp(4) driver, have been proven to work well in high-load circumstances.

Low-end cards usually generate an interrupt for every packet received, which leads to the problems we describe in the next subsection. By using better cards, like the mentioned DEC and Intel ones, packets are getting combined, thus reducing the amount of interrupts.

Another important point is the physical media interface, e. g. `sqphy(4)`. Noise and distortion is a normal part of network communications, a good PHY will do a better job of extracting the data from the noise on the wire than a poor PHY will, reducing the number of network retransmissions required.

It might be a good idea to use Gigabit cards, even when running 100 MBit/s only. They are obviously built for much higher packet rates (and this is the real problem, not bandwidth) than FastEthernet ones, thus have more own intelligence and deal better with high loads.

[1] Though this has implications on the KVM, see the appropriate section

[2] for most kernel configurations, see `options(4)` and `config(8)`.

IRQ

Every interrupt requires a context switch, from the process running when the IRQ took place, to the interrupt handler. As a number of things must be done upon entering the interrupt handler, a large quantity of interrupts can result in excess time required for context switching. One non-obvious way to reduce this load is to share interrupts between the network adapters, something permitted on the PCI bus. As many people are not even aware of the the possibility of interrupt sharing, and the benefits are not obvious, let's look at this a little closer.

With separate adapters on separate interrupt lines, when the first interrupt comes in, a context switch to the interrupt handler takes place. If another interrupt comes in from the other adapter while the first interrupt is still being handled, it will either interrupt the first handler, or be delayed until the first handler has completed, depending on priority, but regardless, two additional context switches will take place – one into the second handler, one back out.

In the case of the PCI and EISA busses, interrupts are level triggered, not edge triggered, which makes interrupt sharing possible. As long as the interrupt line is held active, a device needs servicing, even if the first device which triggered the interrupt has already been serviced. So, in this case, when the first adapter triggers the interrupt, there will be a context switch to the handler. Before the handler returns, it will see if any other devices need servicing, before doing a context switch back to the previous process.

In a busy environment, when many devices are needing service, saving these context switches can significantly improve performance by permitting the processor to spend more time processing data, rather than switching between tasks. In fact, in a very high load situation, it may be desireable to switch the adapters and drivers from an interrupt driven mode to a polling mode, though this is not supported on OpenBSD at this time.

2.2 CPU Exhaustion

Of course the CPU can be overloaded also while other resources are still fine. Besides buying more CPU power, which is not always possible, there are other ways to resolve this problem. Most common cases for this are:

CGI Excessive usage of CGI scripts, usually written in interpreter languages like PHP or Perl. Better (resource-wise) coding can help, as well as using modules like mod_perl[3] to reduce load.

RDBM Usually those CGI scrips use a database. Optimization of the connections and queries (Indexing, ..) is one way. There is also the complete offloading of the database to a different machine [4].

SSL Especially e-commerce systems or online banking sites suffer here. OpenBSD supports hardware-accelerators [5]. Typical cryptographic routines used for SSL/TLS can be offloaded to such cards in a transparent manner, thus freeing CPU time for processing requests.

3 Memory Exhaustion

Another case of overloading can be the exhaustion of memory resources. Also the speed of the allocator for memory areas has significant influence on the overall performance of the system.

3.1 Virtual Memory (VM)

VM is comprised of the physical RAM and possible swap space(s). Processes are loaded into this area and use it for their data structures. While the kernel doesn't really care about the current location of the process' memory space (or address space) it is recommended that especially the most active tasks (like the webserver application) never be swapped out or even subjected to paging.

With regard to reliability it's not critical if the amount of physical RAM is exhausted and heavy paging occurs, but performance-wise this should not happen. The paging could compete for Disk I/O with the server task, thus slowing down the general performance of the server. And, naturally, harddisks are slower than RAM by magnitudes.

It's most likely that countermeasures are taken after the server starts heavy paging, but it could happen that also the swap space, and thus the whole VM, is exhausted. If this occurs, sooner or later the machine will crash.

Even if one doesn't plan for the server starting to page out memory from RAM to swap, there should be some swap space. This prevents a direct crash, if the VM is exhausted. If swap is being used, one has to determine if this was a one-time-only peak, or if there is a general increase of usage on the paging server. In the latter case one should upgrade RAM as soon as possible.

In general it's good practice to monitor the VM usage, especially to track down when the swap space is being touched. See section 5 for details.

3.2 Kernel Virtual Memory (KVM)

Besides VM there is a reserved area solely for kernel tasks. On the common i386 architecture (IA-32) the virtual address space is 4GB. The OpenBSD/i386 kernel reserves 768MB since the 3.2 release (formerly 512MB) of this space for kernel structures, called KVM.

KVM is used for addressing the needs of managing any hardware in the system and small allocations[6] being needed by syscalls. The biggest chunks being used are the management of the VM (RAM and swap), filesystem-cache and storage of network buffers (mbuf).

Contrary to userland the kernel allocations can-

[3] This can have security implications, but this is another story.

[4] This could be unfeasible due to an already overloaded network or due to budget constraints.

[5] `crypto(4)`

[6] like pathname translations

not be paged out ("wired pages"). Actually it's possible to have pageable kernel memory, but this is rarely used (e. g. for pipe buffers) and not a concern in the current context. Thus, if the KVM is exhausted, the server will immediatly crash. Of course 768MB is the limit, but if there is less RAM available, this is the absolute limit for wired pages then. Non-interrupt-safe pages could be paged out, but this is a rare exception.

Since RAM has to be managed by kernel maps also, it's not wise to just upgrade RAM without need. More RAM leaves `less` space for other maps in KVM. Monitoring the "really" needed amount of RAM is recommended, if KVM exhaustions occur. For example, 128MB for a firewall is usually more than enough. Look at Section 7.2 for a typical hardware setup of a busy firewall.

This complete area is called `kernel_map` in the source and has several "submaps"[7]. One main reason for this is the locking of the address space. By this mapping other areas of the kernel can stay unlocked while another map is locked.
Main submaps are `kmem_map`, `pager_map`, `mb_map` and `exec_map`. The allocation is done at boot-time and is never freed, the size is either a compile-time or boot-time option to the kernel.

4 Resource Allocation

Since the exhaustion of KVM is the most critical situation one can encounter, we will now concentrate on how those memory areas are allocated.

Userland applications cannot allocate KVM needed for network routines directly. KVM is protected from userland processes completely, thus there have to be routines to pass data over this border. The userland can use a `syscall(2)` to accomplish that. For the case of networking the process would use `socket(2)` related calls, like `bind(2)`, `recv(2)`, etc.

Having this layer between userland and kernel, we will concentrate on how the kernel is allocating memory; the userland process has no direct influence on this. The indirect influence is the sending and receiving of data to or from the kernel by the userland process. For example the server handles a lot of incoming network data, which will fill up buffer space (mbufs) within the KVM. If the userland process is not handling this data fast enough, KVM could be exhausted. Of course the same is true if the process is sending data faster than the kernel can release it to the media, thus freeing KVM buffers.

4.1 mbuf

Historically, BSD uses `mbuf(9)`[8] routines to handle network related data. An mbuf is a data structure of fixed size of 256 bytes [9]. Since there is overhead for the mbuf header (m_hdr{}) itself, the payload is reduced by at least 20 bytes and up to 40 bytes[10].

Those additional 20 bytes overhead appear, if the requested data doesn't fit within two mbufs. In such a case an external buffer, called cluster, with a size of 2048 bytes[11], is allocated and referenced by the mbuf (m_ext{}).

Mbufs belonging to one payload packet are "chained" together by a pointer `mh_next`. `mh_nextpkt` points to the next chain, forming a queue of network data which can be processed by the kernel. The first member of such a chain has to be a "packet header" (mh_type M_PKTHDR).

Allocation of mbufs and clusters are obtained by macros (MGET, MCLGET, ..). Before the release of OpenBSD 3.0 those macros used `malloc(9)` to obtain memory resources.

If there were a call to MGET but no more space is left in the corresponding memory map, the kernel would panic[12].

[7] see /sys/uvm/uvm_km.c

[8] memory buffer

[9] defined by MSIZE.

[10] see `/usr/include/sys/mbuf.h` for details.

[11] defined by MCLBYTES

[12] "malloc: out of space in kmem_map"

4.2 pool

Nowadays OpenBSD uses `pool(9)` routines to allocate kernel memory. This system is designed for fast allocation (and freeing) of fixed-size structures, like mbufs.

There are several advantages in using `pool(9)` routines instead of the ones around `malloc(9)`:

- faster than malloc by caching constructed objects
- cache coloring (using offsets to more efficiently use processor cache with real-world hardware and programming techniques)
- avoids heavy fragmentation of available memory, thus wasting less of it
- provides watermarks and callbacks, giving feedback about pool usage over time
- only needs to be in kmem_map if used from interrupts
- can use different backend memory allocators per pool
- VM can reclaim free chunks before paging occurs, not more than to a limit (Maxpg) though

If userland applications are running on OpenBSD (> 3.0), `pool(9)` routines will be used automatically. But it's interesting for people who plan (or do so right now) to write own kernel routines where using `pool(9)` could gain significant performance improvements.

Additionally large chunks formerly in the kmem_map have been relocated to the kernel_map by using pools. Allocations for inodes, vnodes, .. have been removed from kmem_map, thus there is more space for mbufs, which need protection against interrupt reentrancy, if used for e. g. incoming network data from the NIC [13].

[13] kmem_map has to be protected by `splvm()`, see `spl(9)`.

5 Memory Measurement

Obviously one wants to know about memory exhaustion *before* it occurs. Additionally it can be of interest, which process or task is using memory. There are several tools provided in the base OpenBSD system for a rough monitoring of what is going on. For detailed analysis one has to be able to read and interpret the values provided by those tools, but sometimes one needs more details and can rely on 3rd party tools then.

Example outputs of the tools mentioned can be found in the Appendix.

5.1 Common tools

These are tools provided with OpenBSD, where some are rather well-known, but some are not. In any case, we have found that often the tools are used in a wrong fashion or the outputs are misinterpreted. It's quite important to understand what is printed out, even if it's a "known tool".

top

One of the most used tools is `top(1)`. It shows the current memory usage of the system. In detail one could see the following entries:

Real: `68M/117M act/tot`, where 68MB are currently used and another 49MB are allocated, but not currently used and may be subject to be freed.

Free: `3724K`, shows the amount of free physical RAM

Swap: `24M/256M used/tot`, 24MB of 256MB currently available swap space is used.

If one adds 3724kB to 117MB, the machine would have nearly 122MB RAM. This is, of course, not true. It has 128MB of RAM; the "missing" 6MB are used as filesystem-cache[14].

[14] dmesg: `using 1658 buffers containing 6791168 bytes (6632K) of memory`

Besides this rough look on the memory usage of the system, there are indicators for other resource exhaustions. In the line `CPU states:` there is an entry `x.y% interrupt`. See how to resolve high values, they slow down the performance.

Blocking disks can be detected in the `WAIT` column. For example an entry `getblk` shows that the process is waiting for data from a disk (or any other block device).

ps

Another very common tool is `ps(1)` and it's related to `top(1)`. Where `top(1)` is usually used for an overview of the system, one can use `ps(1)` for detailed picking on the exact state of a process (or process group).

Additionally it can be closer to reality and the output is more flexible, thus one can do better post-processing in scripts or similar.

Probably most interesting are the options showing how much percentage CPU and VM a process is using. One can sort by CPU ('u') or VM usage ('v') to find a hogging process quickly.

vmstat

`vmstat(8)` is the traditional "swiss army knife" for detailed looks on the systems current usage. It's perfect for a first glance on potential bottlenecks.

A vmstat-newbie will probably be baffled by the output, but with some experience it's rather easy to find out, what's happening and where potential problems are located.

The default output consists of six areas (procs, memory, page, disks, faults, cpu). Each areas has columns for related values:

procs `r b w`, shows how many processes are (r)unning, are being (b)locked or are (w)aiting. Blocked processes cannot change to running before the block is resolved, e. g. a process "hangs" in a `getblk` state and waits for disk I/O.
Waiting means that the process is ready to run, but has still not been scheduled, most likely because the CPU is overloaded with processes.

memory `avm fre`, number of pages (1024b) being allocated and on the free list. The `avm` value gives a better insight on the allocation, than the values from `top(1)`.

page `flt re at pi po fr sr`, page-in (pi) and page-out (po) are most interesting here. It indicates if, and how much, paging (or even swapping) occurs.

disks `sd0 cd0`, the columns here depend on the disk setup of course. Values are transfer per seconds on this device. If high values here correspond with blocked processes below `procs` this is a good indication that the disk subsystem could be too slow.

faults `in sys cs`, can indicate too many interrupts and context switches on the CPU. `sys` counts syscalls brought to the kernel, a rather hard value to interpret with regard to bottlenecks, but one can get an idea of how much traffic has to pass between userland and kernel for completing the task.

cpu `us sy id`, looked at separately not too informative, but in combination with other values it's one keypoint in figuring out the bottleneck. If processes are in 'w' state and 'id' is very low, a CPU exhaustion occurs. Processes being (b)locked and having high (id)le values detect I/O exhaustions. Having high (sy)stem values and (w)aiting and/or (b)locked processes indicate that the kernel is busy with itself too much; this is usually because of "bad" drivers. Compare to 'faults in' to find out if interrupts are killing the performance. If not it's still possible that the CPU is busy transfering blocks from disk devices, indicated by low disk transfers and blocked processes.

Already impressive diagnostic possibilities, but `vmstat(8)` can show even more interesting things.

Besides the options `-i` to show summaries about interrupt behaviour and `-s` to get information about the swap area, `vmstat -m` can

provide a very detailed look on the current memory usage.

Like we already have shown OpenBSD uses `pool(9)` for network data, thus we concentrate now on the last chunk `vmstat -m` is reporting. Most interesting are the lines `mbpl` and `mclpl`, which represent the memory usage for mbufs (mbpl) and clusters (mclpl).

Interesting columns are `Size`, `Pgreq`, `Pgrel`, `Npage` and `Maxpg`. One can obtain the following information from that:

Size the size of a pool item

Pgreq reports how many pages have ever been allocated by this pool.

Pgrel the pool freed those pages to the system.

Npage currently allocated/used pages by the pool.

Maxpg maximum number of pages the pool can use, even if paging would occur. More precise: the pool can grow over this limit, but the pagedaemon can reclaim free pages being over this limit, if VM is running low.

netstat

Usually `netstat(1)` is used for gathering network configurations, but it also provides information about different memory usages.

`netstat -f inet`[15] shows information about current network activity. With regard to memory consumption the columns `Recv-Q` and `Send-Q` are of major interest.

Typically one will encounter entries in Send-Q for a busy webserver with a good network connection. Clients usually have significant smaller bandwith, thus the provided data of the webserver application cannot "leave" the system. It's getting queued on the network stack, eating up mbuf clusters.

Pending requests will show up in Recv-Q, indicating that the userland cannot process the data as fast as it is coming in over the network. The latter case should be resolved, even if memory is not running low, since the system would appear sluggish to the client, which is usually not appreciated (by the admin and/or client).

In addition to `vmstat -m`, `netstat -m` can report further values about current mbuf and cluster usage. Most notably it reports how much memory is "really" used. `vmstat -m` shows how many pool items are allocated, but `netstat -m` then reports how many pool items are actually filled with data to be processed.

In fact one could calculate this in `vmstat -m` by substracting `Releases` from `Requests`, but with numbers like 10599250 and 10599245, this is not really practical. Another pitfall is that `vmstat -m` reports memory pages, where `netstat -m` reports pool items[16] used, despite its output of `mapped pages in use`.

Furthermore it splits up what type of, and how many, mbufs are used (packet headers, sockets, data, ..), and it gives a summary about how much memory is needed by the network stack, which would be rather tedious to calculate from the `vmstat -m` output.

systat

This tool provides a `top(1)` like display of information the previous tools would provide. Especially `systat vmstat` is a perfect overview about load, disk usage, interrupts, CPU and VM usage.

One can monitor the system in intervals, or collect the information over time.

5.2 Special tools

Besides the tools we have shown so far, there are additional possibilities to monitor the system. *symon* and *pftop* are in the ports collection. *KVMspy* is not even published for now, but it shows that it's possible to write own tools for specific monitorings without enormous effort[17].

[15] or -f inet6

[16] usually a factor of two.

[17] the source code is below 300 lines.

symon

For monitoring overall resource usage over time frames, *symon* [Dij02] is a perfect tool. It queries the kernel via `sysctl` about common resources. It uses `rrdtool` [Oet02] as data storage backend. There is a data collector daemon, called `symon`, which runs on every monitored machine, sending the collected data to `symux`, usually running on a central machine, which stores them on disk. Additionally there is a web-interface, `symon-web`, providing graphical representation of the collected data.

After machines have been set up with detailed analysis, this output is enough to detect high-load situations and trigger countermeasures before exhaustion occurs.

If one wants a long-term analysis of detailed data, it's relativly easy to extend this tool. Symon is pretty new and under active development by Willem Dijkstra, but already very useful.

pftop

If one wants to monitor specific areas, like `pf(4)`, *pftop* [Aca02] is a curses-based, real-time monitoring application providing that.

One can consider it as a netstat-variant, providing similar information, about the paket filter.

KVMspy

For the absolute curious one, there will be *KVMspy*. Currently it shows a bit more (offsets) information than `vmstat -m` about pools and a bit less (only current and highwater).

But, for the interested hacker, this is maybe better example code how to poll the kernel states via `kvm(3)` routines. Queries via `sysctl(3)` can be found in symon or are added to KVMspy in the future.

6 Countermeasures

And finally we come to the interesting pieces. Several ways to determine where a lack of KVM resources occurs have been shown. So, what to do if it actually happens?

There are three important kernel options defining the KVM layout with regard to networking. `NMBCLUSTERS` and `NKMEMPAGES` are compile-time options, but can be set via `config(8)` as well. `MAX_KMAPENT` can only be set at compile-time.

6.1 NMBCLUSTERS

The maximum number of clusters for network data can be defined here. Naturally, it's difficult to calculate this value in advance. Most tuning guides recommend a value of 8192 here. We usually use this value, too.

People tend to raise this value further, not knowing what implications this can have on the system. A value of 8192 potentially uses 16MB for `mb_map`: $8192 * 2048 = 16777216$ (MCLBYTES is usually 2048).

Since this is only a "pre-allocation" and not real usage in the first place, this value can be sane. On the other hand, if there are other problems with KVM, this value may be lowered.

Looking at real-life usage of busy webservers (see 7.1) the high watermark of `mclpl` is 524 (1048 clusters), thus even the default of 2048 clusters would be sufficient. This high watermark (Hiwat in `vmstat -m`) is also perfect to determine the `mclpl` size for load-balanced servers.

Imagine a Hiwat of 1000 on both machines. If one machine has to go out of service, due to a crash or simply hardware maintenance, a pool size of >4000 would ensure that the remaining machine doesn't run out of clusters. Remember that `vmstat -m` reports pages, not items, thus one has to calculate 1000*2*2 for `NMBCLUSTERS`.

Additionally it's important to track why clusters are used in larger numbers. We have shown

in 5.1/netstat that it is important to have a quick passing from the `Recv-Q` to the server application. It's a better idea to improve the application performance in this area, than increasing `NMBCLUSTERS` and let the data sit in KVM. At least a rather empty `Recv-Q` leaves more space for the `Send-Q`, which cannot be influenced directly to free clusters.

After all, it's dangerous to use high-values for this (and the following) options without very detailed knowledge about what is happening in the kernel. A "just to be safe" tuning can easily lead to an unstable machine. We have seen people using a value of 65535 for `NMBCLUSTERS`, rendering a pre-allocation of 128MB – not a good idea and usually it doesn't gain anything, except problems. Think twice about those values.

6.2 NKMEMPAGES

This option defines the total size of `kmem_map`. Since this is not exclusively used for networking data, it is a bit difficult to calculate the value for this option.

Since `kmem_map` was freed from other usage (4.2) and the introduction of `pool(9)` ensures that there is more space here for mbufs anyway, so an exhaustion of `kmem_map` is less likely than before.

Tracking of the usage is still possible, though. Looking again at
tt vmstat -m, this time at `mbpl`, one can see a correlation between `mbpl` and `mclpl`. It's common that the page value is usually half (or less) the value from `mclpl`. Yet again, one has to take care of "items vs page-size". Mbufs are way smaller then a cluster, thus 16 mbufs fit in one page of memory.

A network connection using clusters needs at least two mbufs, one for the paket header and one for the reference to the cluster. Since not every connection uses clusters it's sane to assume that a value for `NKMEMPAGES` being twice the value of `NMBCLUSTERS` is a good starting point.

Again, one should raise this value very carefully. Blindly changing these values can introduce more problems, than are solved.

Additionally, if the option is not touched, the kernel gets a sane default value for `NKMEMPAGES` at compile-time, based on RAM available in the system. If the kernel is compiled on a different machine with a different amount of RAM, this option should be used. A typical calculation value is 8162 for a machine with 128MB of RAM; this can be determined by `sysctl -n vm.nkmempages`.

6.3 MAX_KMAPENT

Definition of the number of static entries in `kmem_map`. Like `NKMEMPAGES`, the value is calculated at compile-time if unset. The default of 1000 (at least, it is based on "maxusers") is usually enough.

Raising this value is discouraged, but could be needed, if panics (`uvm_mapent_alloc: out of static map entries ..`) occur. Usually this happens if `kmem_map` is highly fragmented, for example by a lot of small allocations.

7 Real-life Examples

Putting everything together, we provide two examples of machines running OpenBSD under high load. It shows that a careful kernel configuration and hardware selection has great influence on the performance and reliability.

7.1 chat4free.de Webserver

This machine, hosted by BSWS, is running the webserver for one of Germany's biggest chat systems, chat4free.de.

The site consists of static pages and public forums. The unusual problem here is the both the overall load and the enormous peaks which happen when numbers of users are disconnected from the chat server due to external network problems or crashes of the server itself. Unlike many web applications, this server has

a huge volume of small packets, which demonstrates that loading is more an issue of users and packet counts than raw data transfer.

Originally, it was running one Apache instance for the entire application, on an 700MHz Athlon system with 1.5G RAM, running a highly modified OpenBSD 3.0. Unfortunately, this system sometimes crashed due to KVM exhaustion.

To address this problem, the system was switched to a new system, again an 700MHz Athlon with 512M RAM, running two Apache instances in chroot jails, on a fairly stock OpenBSD 3.1 system. The system has a network adapter based on a DEC/Intel 21143, with a Seeq 84220 PHY, and runs "headless" with a serial console.

One of the two Apache instances is stripped down as much as I could make it, and serves the static pages. This httpd binary is only 303k in size, compared to the almost 600k of the stock Apache. The second instance of Apache is much bigger, as it has PHP compiled in. I always use static httpds, rather than Dynamic Shared Objects (DSOs).

The kernel configuration is fairly stock. All unused hardware support and emulations are removed, option DUMMY_NOPS is enabled. NMBCLUSTERS is bumped to 8192, NKMEMPAGES to 16384. I considered raising MAX_KMAPENT from its default of 1000 to 1500 or so to be able to have even more concurrent Apache processes running, though there has been no real need yet in this application. The machine has an ordinary IDE hard disk for the system, content and logs are on a separate machine's RAID subsystem, mounted via NFS. Most static content ends up being cached, reducing network traffic.

The "lean" httpd instance is configured for up to 1000 concurrent httpd tasks, the "fat" one for up to 600. I've seen both reach their maximum limits at the same time, and the smaller machine handles this load without incident. This is due to the superior memory management in OpenBSD 3.1 and the smaller Apache configurations.

Detailed kernel configuration and dmesg(8) can be found in the Appendix.

7.2 A firewall at BSWS

One important fact about firewalling and filtering is that the bandwidth isn't the important issue, the issue is the packet rate (i.e., packets per second). Each packet needs to be handled by the network adapter, the TCP/IP stack and the filter, which each need to do roughly the same amount of work whether the packet is small or large.

The firewall that protects a number of the servers at BSWS is under rather heavy load, not really due to total bandwidth, but the large number of small packets involved. It is running on a 700MHz Duron with 128M RAM and three DEC/Intel 21143-based NICs (one is currently not in use). It boots from a small IDE hard disk, which is quite unimportant to this application.

The machine is running a highly customized version of OpenBSD. The base system is OpenBSD 3.0, but many pieces of what became OpenBSD 3.1 were imported, including 3.1's packet filter `pf(4)`. At the time this was put together, there was no other option for this application. Many of pf's newer features were needed, but it was not possible to wait for 3.1-Release, as the previous OpenBSD 2.9 firewall running IPFilter had saturated the processor at near 100% utilization at peak usage times, and delays were being noticed. The kernel configuration has had all uneeded hardware support and binary emulations removed, and the always famous NKMEMCLUSTERS=16384 and NMBCLUSTERS=8192 modifications. The number of VLAN interfaces was raised to 20 (from 2 in GENERIC).

As of October 5, the expanded ruleset has 1132 rules. The "quick" keyword is used in most places to reduce the number of rules that must be evaluated for each packet, otherwise the entire ruleset must be evaluated for each packet. The rules are ordered so that the ones I expect the most matches from are towards the top of the file. All pass rules keep state; not only is this good practice for security, but with pf, state table lookups are usually much faster than rule evaluation. No NAT takes place on

this machine, only packet filtering.

On the external interface, there is only spoofing protection taking place. Incoming packets with a source IP of the internal networks, outgoing packets with an IP which is not from one of the internal networks, and all 127.0.0.0/8 traffic is dropped. Normally, one would also drop packets with RFC1918 ("private IP space"), however in this case, it is handled externally by the BSWS core routers, because there is valid traffic with RFC1918 IPs from other internal networks crossing this firewall.

The actual filtering policies are enforced on the inside (VLAN) interfaces, which has the benefit that packets attempting to cross between VLANs encounter the same rules as packets from the outside. Every packet passing the firewall is normalized using the scrub directive. OpenBSD 3.2 will support multiple scrub methods besides the classic buffering fragment cache. One of the more interesting is the crop method, which almost completely avoids buffering fragments.

The results have been impressive. In September, 2002, the state table reached a peak size of 29,390, with an average size of 11,000. Up to 15,330 state table lookups per second were performed with average of 5600. State table inserts and removals peaked at slightly over 200 per second each. The CPU load seldom exceeds 10%. Compare this to the old IPFilter solution running on the same hardware doing much the same task, where the CPU was maxed out with only 600 rules and a peak of 15,000 packets per second. pf has permitted considerable growth in the complexity of the rule sets and traffic, and as you can see, still leaves BSWS considerable room to grow. Since this firewall went into operation in March, 2002, there hasn't been a single problem with its hardware or software.

8 Conclusions

Running OpenBSD servers under high load is pretty safe nowadays. We have shown that the introduction of `pool(9)` made operation way better with regard to memory usage and performance.

We have shown how network traffic influences the memory usage of the kernel and how the pieces are related together.

The provided knowledge about monitoring a running system and potential countermeasures against resource exhaustions should help to deal with high-load situations better.

9 Acknowledgements

A big "thank you" goes to Nick Holland, who turned our crappy english into something useful and provided a lot of input on how to explain this difficult area better.

Thanks also to Artur Grabowski for implementing `pool(9)` in the OpenBSD kernel and for further explanations about KVM.

Several proof-readers helped on finding spelling errors and inconsistencies within the paper, a special thanks here for Daniel Lucq, who also wrote KVMspy.

And, of course, thanks to the OpenBSD developer team for working on a system which provides already sane defaults for operating a high-load server, and, not to forget, a very high level of security.

References

[Aca02] Can E. Acar. Openbsd pf state viewer. http://www.eee.metu.edu.tr/~canacar/pftop/, 2002.

[Dij02] Willem Dijkstra. The small and secure active system monitor. http://www.xs4all.nl/~wpd/symon/, 2002.

[McK96] Marshall Kirk (et. al.) McKusick. *The design and implementation of the 4.4BSD operating system.* Addison-Wesley, 1996.

[Oet02] Tobi Oetiker. Round robin database. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/, 2002.

[Ope02] OpenBSD. http://www.openbsd.org/, 2002.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated, Vol. 2.* Addison-Wesley, 1994.

A top

This machine is the main server of sysfive.com GmbH, slightly tuned it is really idle.

```
load averages:  0.19,  0.12,  0.09                                    14:19:57
68 processes:  1 running, 64 idle, 3 zombie
CPU states:  0.3% user,  0.9% nice,  0.3% system,  0.0% interrupt, 98.4% idle
Memory: Real: 49M/80M act/tot  Free: 41M  Swap: 0K/256M used/tot

  PID USERNAME PRI NICE  SIZE   RES STATE WAIT     TIME    CPU COMMAND
15902 root       2    0 2308K 1832K idle  select  19:39  0.00% isakmpd
27679 pb         2    0  964K 1468K sleep select   7:00  0.00% screen-3.9.11
19945 gowry      2    0 4644K 5096K idle  select   4:30  0.00% screen-3.9.11
 3605 postfix    2    0  304K  736K sleep select   4:29  0.00% qmgr
22360 root      18    0  640K 9944K sleep pause    2:53  0.00% ntpd
11827 pb         2    0  516K 1312K sleep poll     2:18  0.00% stunnel
[..]
```

B ps

Same machine, same processes reported by `ps -axv`

```
USER       PID %CPU %MEM   VSZ   RSS TT  STAT  STARTED      TIME COMMAND
root     22360  0.0  7.6   640  9944 ??  Ss     8Aug02    2:48.24 ntpd -c /etc/ntp.conf
gowry    19945  0.0  3.9  4644  5096 ??  Ss     9Aug02    4:30.56 SCREEN (screen-3.9.11)
root     15902  0.0  1.4  2308  1832 ??  Is    31Jul02   19:39.33 isakmpd
pb       27679  0.0  1.1   964  1468 ??  Ss    13Jul02    6:59.75 SCREEN (screen-3.9.11)
pb       11827  0.0  1.0   516  1312 ??  Ss    13Jul02    2:15.55 stunnel
postfix   3605  0.0  0.6   304   736 ??  S      6Aug02    4:30.29 qmgr -l -t fifo -u
```

C vmstat

Current vmstat output of the same machine (`vmstat 1 5`)

```
 procs   memory        page                    disks     faults      cpu
 r b w    avm    fre   flt  re  pi  po  fr  sr cd0 sd0   in     sy   cs us sy id
 1 0 0  50324  41608    14   0   0   0   0   0   0   1  234   7151  160  0  0 99
 0 0 0  50324  41608    10   0   0   0   0   0   0   0  233   1602  165  0  0 100
 0 0 0  50324  41608     6   0   0   0   0   0   0   0  233   1589  165  0  1 99
```

If the machine would have disk I/O blocking problems, the output could look like this. Note the idle CPU, but blocked processes are waiting for blocks from the busy drive.

```
 procs   memory        page                    disks     faults      cpu
 r b w    avm    fre   flt  re  pi  po  fr  sr cd0 sd0   in     sy   cs us sy id
 1 2 0  50324  41608    14   0   0   0   0   0   0 271  234   7151  160  1  3 96
 0 1 0  50324  41608    10   0   0   0   0   0   0 312  233   1602  165  0  4 96
 0 1 0  50324  41608     6   0   0   0   0   0   0 150  233   1589  165  0  2 98
```

Worst-case scenario, the machine does heavy paging, thus overloading the disk subsystem. Additionally the CPU is maxed out. Processes are waiting, interrupts cause massive context-switching. The values are arbitrary.

```
 procs   memory        page                    disks     faults      cpu
 r b w    avm    fre   flt  re  pi  po  fr  sr cd0 sd0   in     sy    cs us sy id
 1 2 1    324    608   314   0  25  35   0   0   0 271  412   7151 1931 80 19  1
 1 3 2    324    608   310   0  28  42   0   0   0 312  501   1602 1876 81 19  0
 1 2 1    324    608   306   0  21  38   0   0   0 150  467   1589 1911 85 12  3
```

Now let's have a look at the pool situation of a firewall. A nice example that the pool can grow over the initial limit (Maxpg 512, Hiwat 516), but somehow KVM is low, since a lot of requests are failing (Fail 14725). The kernel should be reconfigured with NMBCLUSTERS > 1024 (`vmstat -m | grep mclpl`).

```
Name        Size Requests Fail Releases Pgreq Pgrel Npage Hiwat Minpg Maxpg Idle
mclpl       2048  1758499 14725 1757480   518     2   516   516     4   512    4
```

D netstat

All packet data is getting delivered to/from the sshd fast enough, so no queuing occurs.

```
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  172.23.1.1.22          10.172.2.32.1156       ESTABLISHED
tcp        0      0  172.23.1.1.22          172.23.1.3.39679       ESTABLISHED
tcp        0      0  172.23.1.1.22          192.168.1.5.42456      ESTABLISHED
```

Somehow either the uplink is saturated, or the remote clients are not retrieving data fast enough, thus the Send-Q is growing.

```
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0   5346  172.23.1.1.22          10.172.2.32.1156       ESTABLISHED
tcp        0      0  172.23.1.1.22          172.23.1.3.39679       ESTABLISHED
tcp        0   7159  172.23.1.1.22          192.168.1.5.42456      ESTABLISHED
```

For whatever reason, sshd is not processing data fast enough. Maybe the deciphering needs more CPU then available?

```
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp     8811      0  172.23.1.1.22          10.172.2.32.1156       ESTABLISHED
tcp     5820      0  172.23.1.1.22          172.23.1.3.39679       ESTABLISHED
tcp    11631      0  172.23.1.1.22          192.168.1.5.42456      ESTABLISHED
```

Let's have a look at the memory usage with **`netstat -m`**. The stack has to keep 85 clusters in KVM, somehow the application is processing data either too fast (Send-Q) or too slow (Recv-Q).

```
384 mbufs in use:
        100 mbufs allocated to data
        178 mbufs allocated to packet headers
        106 mbufs allocated to socket names and addresses
85/1048 mapped pages in use
3144 Kbytes allocated to network (8% in use)
0 requests for memory denied
0 requests for memory delayed
0 calls to protocol drain routines
```

E systat

Looks like the machine is doing nothing? Wrong, look at the interrupt counting for dc0 and dc2. It's the BSWS' firewall described earlier.

```
    1 users    Load  0.05  0.08  0.08                  Sat Oct  5 17:22:05 2002

         memory totals (in KB)              PAGING   SWAPPING     Interrupts
        real   virtual     free              in  out   in  out     7903 total
Active  91472     93712   10848      ops                            100 clock
All    116216    118456  270684      pages                              pccom0
                                                                    128 rtc
Proc:r  d  s  w       Csw  Trp  Sys  Int  Sof  Flt        forks    3669 dc0
        1  9            6    5   21 7936    4    2        fkppw         dc1
                                                          fksvm         pciide0
   0.0% Sys   0.0% User   0.0% Nice  90.0% Idle           pwait    4006 dc2
|    |    |    |    |    |    |    |    |    |          relck
                                                          rlkok
                                                          noram
Namei         Sys-cache    Proc-cache    No-cache       ndcpy
    Calls     hits    %    hits     %    miss   %        fltcp
        2        2  100                                  zfod
                                                          cow
Discs  wd0                                          128 fmin
seeks                                               170 ftarg
xfers                                              8446 itarg
Kbyte                                                39 wired
  sec                                                    pdfre
                                                         pdscn
```

F iostat

Medium, but constant, traffic on sd0. In fact I was generating traffic with `dd(1)`.

```
      tty             cd0              sd0              sd1              fd0              cpu
 tin tout  KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s  us ni sy in id
   0  540  0.00   0 0.00  0.50 2614 1.28  0.00   0 0.00  0.00   0 0.00   1  1  5  3 90
   0  179  0.00   0 0.00  0.50 2560 1.25  0.00   0 0.00  0.00   0 0.00   0  0  2  2 95
   0  344  0.00   0 0.00  0.50 2601 1.27  0.00   0 0.00  0.00   0 0.00   0  0  3  5 92
   0  181  0.00   0 0.00  0.50 2601 1.27  0.00   0 0.00  0.00   0 0.00   0  1  5  3 91
```

G pftop

Easy and quick overview about current traffic filtering:

```
pfTop: Up State 1-3/64, View: default, Order: none

PR   DIR SRC                  DEST                 STATE   AGE   EXP  PKTS BYTES
icmp Out 192.168.100.32:361   192.168.100.22:361    0:0      9     1     2    96
icmp Out 192.168.100.32:361   192.168.100.23:361    0:0      9     1     2    96
tcp  In  192.168.100.7:1029   192.168.100.32:443    4:4   4165 86302 25871 9251K
```

H KVMspy

The full output would be too long, thus shortened to relevant pools/maps. Somehow this machine is not really exhausted, even with the default settings.

```
_kmem_map @ 0xd0518cdc: total size = 33431552 bytes, [0xd0890000, 0xd2872000]
_kmem_map @ 0xd0518cdc: 103 entries, actual size = 2453504 bytes (7.34%)
_mb_map @ 0xd0890c00: total size = 4194304 bytes, [0xda63e000, 0xdaa3e000]
_mb_map @ 0xd0890c00: 5 entries, actual size = 118784 bytes (2.83%)
_socket_pool @ 0xd05424c8: currently has 6 pages (24576 bytes)
_socket_pool @ 0xd05424c8: high water mark of 12 pages (49152 bytes)
_nkmempages @ 0xd05029d4: 8162 (_nkmempages * PAGE_SIZE = 33431552 bytes)
_nmbclust @ 0xd04fb278: 2048 (_nmbclust * MCLBYTES = 4194304 bytes)
```

I chat4free.de Webserver

I'm using a bit more aggressive timeouts on this machine to lower the number of concurrent connections. This inlcudes a shortened KeepAliveTimeout to 10 seconds in apache's config and the following addition to `/etc/sysctl.conf`:

```
net.inet.tcp.keepinittime=10
net.inet.tcp.keepidle=30
net.inet.tcp.keepintvl=30
net.inet.tcp.rstppslimit=400
net.inet.ip.redirect=0
net.inet.ip.maxqueue=1000
kern.somaxconn=256
```

The timeouts depend heavily on your usage profile and need to be tried. The above ones work fine here, and should fit for most well connected webservers.
dmesg:

```
OpenBSD 3.1 (windu) #0: Wed Apr 17 20:10:40 CEST 2002
    root@ozzel:/usr/src/sys/arch/i386/compile/windu
cpu0: AMD Athlon Model 4 (Thunderbird) ("AuthenticAMD" 686-class) 700 MHz
cpu0: FPU,V86,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SYS,MTRR,PGE,MCA,CMOV,PAT,PSE36,MMX,FXSR
real mem  = 536457216 (523884K)
avail mem = 494899200 (483300K)
```

```
using 5689 buffers containing 26927104 bytes (26296K) of memory
mainbus0 (root)
bios0 at mainbus0: AT/286+(86) BIOS, date 04/02/02, BIOS32 rev. 0 @ 0xfb210
apm0 at bios0: Power Management spec V1.2
apm0: AC on, battery charge unknown
pcibios0 at bios0: rev. 2.1 @ 0xf0000/0xb690
pcibios0: PCI IRQ Routing Table rev. 1.0 @ 0xfdbd0/176 (9 entries)
pcibios0: PCI Exclusive IRQs: 11
pcibios0: PCI Interrupt Router at 000:07:0 ("VIA VT82C596A PCI-ISA" rev 0x00)
pcibios0: PCI bus #1 is the last bus
pci0 at mainbus0 bus 0: configuration mode 1 (no bios)
pchb0 at pci0 dev 0 function 0 "VIA VT8363 Host" rev 0x03
ppb0 at pci0 dev 1 function 0 "VIA VT8363 PCI-AGP" rev 0x00
pci1 at ppb0 bus 1
pcib0 at pci0 dev 7 function 0 "VIA VT82C686 PCI-ISA" rev 0x40
pciide0 at pci0 dev 7 function 1 "VIA VT82C571 IDE" rev 0x06: ATA100, channel 0
 \configured to compatibility, channel 1 configured to compatibility
wd0 at pciide0 channel 0 drive 0: <IC35L060AVER07-0>
wd0: 16-sector PIO, LBA, 58644MB, 16383 cyl, 16 head, 63 sec, 120103200 sectors
wd0(pciide0:0:0): using PIO mode 4, Ultra-DMA mode 5
pchb1 at pci0 dev 7 function 4 "VIA VT82C686 SMBus" rev 0x40
dc0 at pci0 dev 8 function 0 "DEC 21142/3" rev 0x41: irq 11 address 00:00:cb:53:62:c3
sqphy0 at dc0 phy 17: Seeq 84220 10/100 media interface, rev. 0
isa0 at pcib0
isadma0 at isa0
pckbc0 at isa0 port 0x60/5
pckbd0 at pckbc0 (kbd slot)
pckbc0: using irq 1 for kbd slot
wskbd0 at pckbd0: console keyboard
pcppi0 at isa0 port 0x61
sysbeep0 at pcppi0
npx0 at isa0 port 0xf0/16: using exception 16
pccom0 at isa0 port 0x3f8/8 irq 4: ns16550a, 16 byte fifo
pccom0: console
pccom1 at isa0 port 0x2f8/8 irq 3: ns16550a, 16 byte fifo
biomask 4000 netmask 4800 ttymask 4802
pctr: user-level cycle counter enabled
mtrr: Pentium Pro MTRR support
dkcsum: wd0 matched BIOS disk 80
root on wd0a
rootdev=0x0 rrootdev=0x300 rawdev=0x302
```

Kernel config:

```
machine i386 # architecture, used by config; REQUIRED
option DIAGNOSTIC # internal consistency checks
option CRYPTO # Cryptographic framework
option SYSVMSG # System V-like message queues
option SYSVSEM # System V-like semaphores
option SYSVSHM # System V-like memory sharing
option FFS # UFS
option FFS_SOFTUPDATES # Soft updates
option QUOTA # UFS quotas
option MFS # memory file system
option TCP_SACK # Selective Acknowledgements for TCP
option NFSCLIENT # Network File System client
option NFSSERVER # Network File System server
option FIFO # FIFOs; RECOMMENDED
option KERNFS # /kern
option NULLFS # loopback file system
option UMAPFS # NULLFS + uid and gid remapping
option INET # IP + ICMP + TCP + UDP
option INET6 # IPv6 (needs INET)
option PULLDOWN_TEST # use m_pulldown for IPv6 packet parsing
pseudo-device pf 1 # packet filter
pseudo-device pflog 1 # pf log if
pseudo-device loop 2 # network loopback
```

```
pseudo-device bpfilter 8 # packet filter
pseudo-device vlan  2 # IEEE 802.1Q VLAN
pseudo-device pty 64 # pseudo-terminals
pseudo-device tb 1 # tablet line discipline
pseudo-device vnd 4 # paging to files
#pseudo-device ccd 4 # concatenated disk devices
pseudo-device ksyms 1 # kernel symbols device

option BOOT_CONFIG # add support for boot -c
option I686_CPU
option USER_PCICONF # user-space PCI configuration
option DUMMY_NOPS # speed hack; recommended
option COMPAT_LINUX # binary compatibility with Linux
option COMPAT_BSDOS # binary compatibility with BSD/OS

option  NMBCLUSTERS=8192
option  NKMEMPAGES=16384

maxusers 64 # estimated number of users
config bsd swap generic

mainbus0 at root
bios0 at mainbus0
apm0 at bios0 flags 0x0000 # flags 0x0101 to force protocol version 1.1
pcibios0 at bios0 flags 0x0000 # use 0x30 for a total verbose
isa0 at mainbus0
isa0 at pcib?
pci* at mainbus0 bus ?
option PCIVERBOSE
pchb* at pci? dev ? function ? # PCI-Host bridges
ppb* at pci? dev ? function ? # PCI-PCI bridges
pci* at ppb? bus ?
pci* at pchb? bus ?
pcib* at pci? dev ? function ? # PCI-ISA bridge
npx0 at isa? port 0xf0 irq 13 # math coprocessor
isadma0 at isa?
isapnp0 at isa?
option WSDISPLAY_COMPAT_USL # VT handling
option WSDISPLAY_COMPAT_RAWKBD # can get raw scancodes
option WSDISPLAY_DEFAULTSCREENS=6
option WSDISPLAY_COMPAT_PCVT # emulate some ioctls
pckbc0 at isa? # PC keyboard controller
pckbd* at pckbc? # PC keyboard
vga* at pci? dev ? function ?
wsdisplay* at vga? console ?
wskbd* at pckbd? console ?
pcppi0 at isa?
sysbeep0 at pcppi?
pccom0 at isa? port 0x3f8 irq 4 # standard PC serial ports
pccom1 at isa? port 0x2f8 irq 3
pciide* at pci ? dev ? function ? flags 0x0000
wd* at pciide? channel ? drive ? flags 0x0000
dc* at pci? dev ? function ? # 21143, "tulip" clone ethernet
sqphy* at mii? phy ? # Seeq 8x220 PHYs
pseudo-device pctr 1
pseudo-device mtrr 1 # Memory range attributes control
pseudo-device sequencer 1
pseudo-device wsmux 2
pseudo-device crypto 1
```

Xperteyes - keeping your system under control

Pim Buurman
X|support
<pim.buurman@summix.nl>

How would life change if you had every morning some lists of points where your computers would fail the criteria set by the system administrators, their boss, the EDP-auditor and the software manufacturers. So if you had yesterday installed a powerful tool for managing your network, today you would notice that the install log (owned by root) has mode 07777. Or if a user leaves the company, a check is performed that his account is also removed from the database, and his name is removed from the cron users.

With this new tool, xperteyes, this is easily possible. In some minutes all relevant properties of many objects are read and stored. In a separate step (which may be performed on another machine) all these properties can be checked through a configuration script, which will take also a few minutes. Naturally, a graphical interface is available for viewing the properties, the checks and the results. The results can also be viewed in HTML. For the more daring system administrators it is possible to (automatically) repair some of the properties.

The tool is currently running on Mac OS X, Linux and Windows, implemented in Python with some extensions in C. The main object's properties that are collected are:

- the properties of files, directories and other objects in the file system
- /etc/passwd, /etc/group, etc.
- the windows registry and the windows user/group/machine database
- the Mac OS netinfo tree

But, if you want to check later on e.g. file magic, such extensions can be implemented in less than one hour, depending on the amount of code that can be reused.

The checks that may be performed are partly predefined. These are simple checks like checking that a single property has a predefined value, or its value belongs to some range. More complex checks, like is_hardlink or uid_valid are also available. And comparing objects collected at different times or from different systems is possible. But it is likely that these checks are not enough. If you add the file magic to your object's properties, you want to check it. Well, you can easily define new checks yourself, once you have formulated them (that's the difficult part).

Some areas where this tool can be deployed are:

- Checking, initially and daily, the configuration of one system
- Checking the effect of (un)installing some packages
- Keeping two systems "identical" (clustering)

Pim Buurman started out as a mathematician, but he was always more interested in writing software in complex environments. The simple UNIX environment gives enough challenges to keep him happy. Since May 2001 he works for X|support. He partially developed the public domain tool TimeWalker (http://www.NLnet.nl/projects/timewalker), a tool to visualize interactively huge amounts of eventdata. Xperteyes is his new tool. New functions can be defined instantly by the user.

Xperteyes - keeping your system under control

Pim Buurman
X|support
Pim.Buurman@summix.nl

Introduction

The increasing complexity of computer systems and infrastructures asks for good solutions. The installation and configuration of all components should be valid (correct syntax), consistent and adhere to the standards of the organization. Fortunately, many configuration settings can be set with a graphical tool, making the task simpler and sometimes more intuitive. Therefore, syntax errors in configuration files hardly occur any more. Installing is made easy with the emerging of install packages, so each file will have the correct owner and group and permission bits. In theory, installing new packages will not interfere with already installed packages, nor will it introduce a security risk.

But configuring a system is still prone to human error, and many inconsistencies may occur. Some errors in configurations are noticed immediately, but many others may slumber for a long time, until an irregularity happens to the system. Then an emergency will occur due to a problem that was introduced three months ago.

Another source of problems are deliberate attempts by people to change the system to their own ideas. These people can be both outsiders (hackers) and regular users. A misguided administrative action may introduce serious security problems, but even a simple typo may lead to a break in. It would be useful for system administrators to detect configuration problems as well as the occurrence of a security breach.

The system administrators are not the only users who want a report on the current configuration. Their manager will be interested to measure the general quality of the computer systems. And the EDP-auditor will have an easier job.

Xperteyes is a solution for these problems. It can check many properties of a system within a few minutes, as well as checking several systems against each other, for instance to check that a fail-over system is configured correctly. Therefore, Xperteyes can be used on a daily basis to find incorrect or insecure configuration settings. Major changes can be tested before the system goes live, important minor changes can be tested immediately. Furthermore, implicit management rules can be made explicit, making it easier to maintain them and to train novice administrators. Experienced system managers will have more time to do complicated jobs, knowing that they can check everything before going on line.

For daring system administrators, Xperteyes offers the possibility to repair the system based on the reported problems. Because this is a dangerous action, repairing needs to be governed by a user.

Each system administrator thinks his computer system has a unique combination of applications and that only he knows to address all of them. This is usually true, so Xperteyes initially handles only the most common configuration properties. But the application is easily extendible. It will take only a few hours to extend Xperteyes to read the contents of a configuration, and check these contents for specific values.

Architecture

Xperteyes is an application containing several programs. The programs can be divided into three classes: collectors, checkers and viewers. The collectors gather all (or at least all interesting) properties from an object of your computer system. The object may be as simple as the password file, or can be as complex as the file system or the domain name configuration. The information is saved in a generic format, which we will call a collection.

The checkers use several collections, together with the criteria the user wants to check, and produce a new collection, based on the original collections and enriched with the failures of the criteria. Because the user can specify both the criteria and the collections, it is possible to check if a system has not changed since yesterday, or that the users on the Mac server are the same as on the Linux clients.

The viewers are used to browse or print the collections. Browsing through the system data of remote machines is possible, but the main purpose is viewing the failures in relation to their environment.

A good performance was one of the design goals. The performance of the collectors is usually limited by the performance of the OS. E.g. the file system collector uses typically only 30% CPU time, because it is disk read bound. The performance of the checkers is CPU bound, checking typically 100,000 properties per minute. The viewers are visually bound; they are only slow when many results need to be shown.

Collectors

The collectors are intended to gather all relevant information about a system. Each collector can handle one object. This object can be a UNIX configuration file (`/etc/passwd`, `/etc/group`, ...) or a more complex object as the file system. The result is either a list of records, or a tree of records. Each record in the result has the same fields, one field per property. It is possible that a special property (e.g. the `rdev` for device files) is only set for a limited number of objects.

The collectors are basically read-only, so the system should be nearly unaffected by it. On purpose, the collectors use the permissions of the owner of the process to probe the system. If the permissions do not allow a full scan of the system, the checkers may give unexpected results. Any error during the collect process is added to the result, including permission errors.

Each collector is principally system specific, so the file system collectors of Mac OS X, generic UNIX and Windows differ. However, they all specify a tree where each node has the name, owner, group and content modification time of a file system object. The Mac file system has amongst others Mac-type, creator and alias as extra fields.

A supercollector is built from a group of collectors. Its result is a tree containing the lists and trees of its subcollectors. In contrast to the simple collectors, this tree has records that depend on the place in the tree.

The currently defined collectors are:

for generic UNIX:

- file system - all file object properties (not the data), including optional storage of checksum or content of some files.
  This collector is configurable, e.g. to skip NFS-mounted file systems and to store the contents of `/etc/profile`.
- many configuration files, mostly in `/etc` (`passwd`, `group`, `fstab`, `named`, ...)
- configuration API (`getpwent`, `getgrent`, etc.)

for Mac OS X:

- extended UNIX file system
- generic UNIX configuration files and configuration API
- netinfo, open directory
- installed packages

for Linux:

- generic UNIX
- rpm information

for Windows:

- file system
- registry
- user/group/machine database

Checking

The checkers test if the system satisfies some criteria. Unlike many other tools, the checkers in Xperteyes have no internal criteria. The user defines the criteria, and different criteria can be tested on the same collection, either in one run or in several runs.

Currently, three different checkers are available: a general one for checking complex criteria, one for a fast comparison (diff) between two collections, and one for checking the current situation against the installed packages.

The criteria in the general checker consist of a list of requirements. Each requirement states which tests should be applied on which records. Optionally a context and severity may be given, so the failures can be more easily categorized in the viewers. Also the different texts may be adapted to the particular case. In the viewers, the text "The home directory of 'pim' is not owned by him" can be clearer than the default message "uid 0 is not equal to 501".

The general checker uses one or more collections and applies a set of criteria to them. To simplify the criteria, they are most easily expressed as general requirements with exceptions. This means that we write:

```
require('/', perm_le( 'rwxr-xr-x' ), selection=ALL)
require('/tmp', [perm_eq( 'rwxrwxrwt' ), uid_eq(0)])
require('/tmp', perm_le( 'rwxrwxrwx' ), selection=ALL_CHILDREN)
```

The checker applies these requirements so that the more specific tests block the less specific ones, where more specific means that the test is more specific (`perm_eq` is more specific than `perm_le`) or that the object of the requirement is deeper in the tree (`/tmp` is deeper than `/`). Thus, the given requirement are interpreted as: all objects from `/` down (i.e. all objects in the file system) should have permission <= rwxr-xr-x, but `/tmp`

should have permission == rwxrwxrwt and uid == 0 (root!), and all objects from `/tmp` down, but not `/tmp`, should have permission <= rwxwxrwx. Note: permission checks work bitwise.

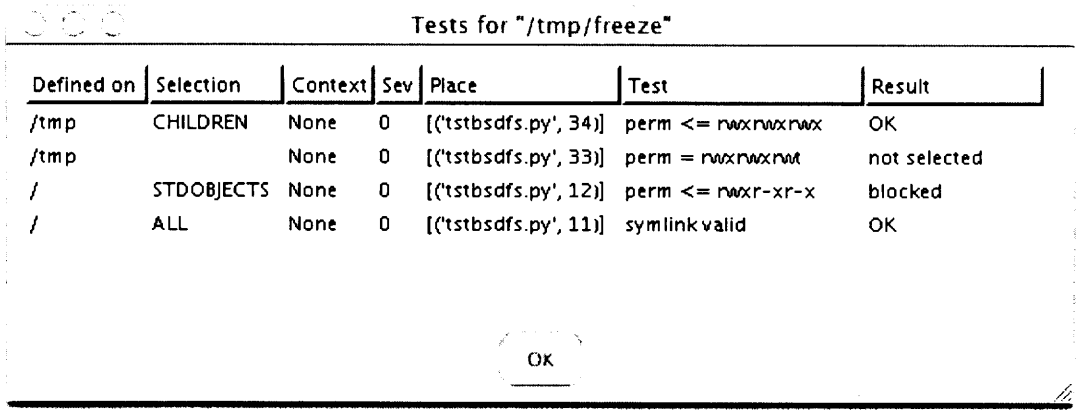


Fig. 1: Test results for `/tmp/freeze`.

The currently available tests consist mainly of simple checks of one field against one value, or against a range or interval. Also some more elaborate tests are defined, for example whether a file is a hard link to another file. New tests can also be defined as logical combinations of simple tests, e.g. `suid_root_exe()` can be defined as `And(perm_eq('rwsr-xr-x'), uid_eq(0), format="not a setuid root executable")`.

Using these basic tests, one might expect that only a static check can be defined, e.g. principally independent of the current situation on the system. But the general checker is more powerful, because the criteria is a Python script that defines the requirements. It is possible to generate requirements based on the contents of some objects. So it is very easy to define that the home directories of the users are their private area on the system. Based on the password file contents, tests are generated that the home directory trees belong to the correct user, and that its root has permissions equal to rwx------. Note that you want to skip entries for nobody and bin, hence the `REAL_USER_SELECTION`.

```
for el in x.elements('Passwd:File', REAL_USERS_SELECTION):
      x.require('FileSystem:' + el.home, [perm_eq('rwx------')])
      x.require('FileSystem:' + el.home,
                  [uid_eq(el.uid), gid_eq(el.gid)], ALL)
```

The checker can be called with more than one collection. Therefore, it is easy to test several systems against each other. The checker can be used e.g. to find both stale accounts and missing accounts on local systems by comparing it to a server. It is allowed that the local systems are running Windows (in various flavors), while the server is a Mac OS X Server. Two fail-over systems should be compared regularly, because most fail-over software depends (indirectly) on configuration parameters that are not forced to be the same (e.g. passwords). And comparing the collections of one system that are created at two time points shows exactly what is changed. This can be used to test if the installation of a package does not interfere with existing packages, and if a de-installation restores the old situation.

In our view, there is a difference between a test failure because the property has an invalid value, and a test failure because the object is not in the collection. Therefore run

time errors are given if the object of a requirement is not in the collection, or if the test cannot be applied.

The checker adds the errors and failures to the collection, so the viewer can show not only the failures, but also the environment, making it easier to interpret the results.

Viewing

The viewers are used to see the results of the collectors and the checkers. The graphical viewer shows an annotated tree of the results. The criteria failures and the run time errors are shown in two separate windows. Errors and failures are also indicated on the objects self. To guide the user to problem areas, the items in the tree are colored to indicate that in that subtree some problems exist. To understand the requirements on one object, it is possible to view the requirements that are applicable to that object, and the result of those requirements.

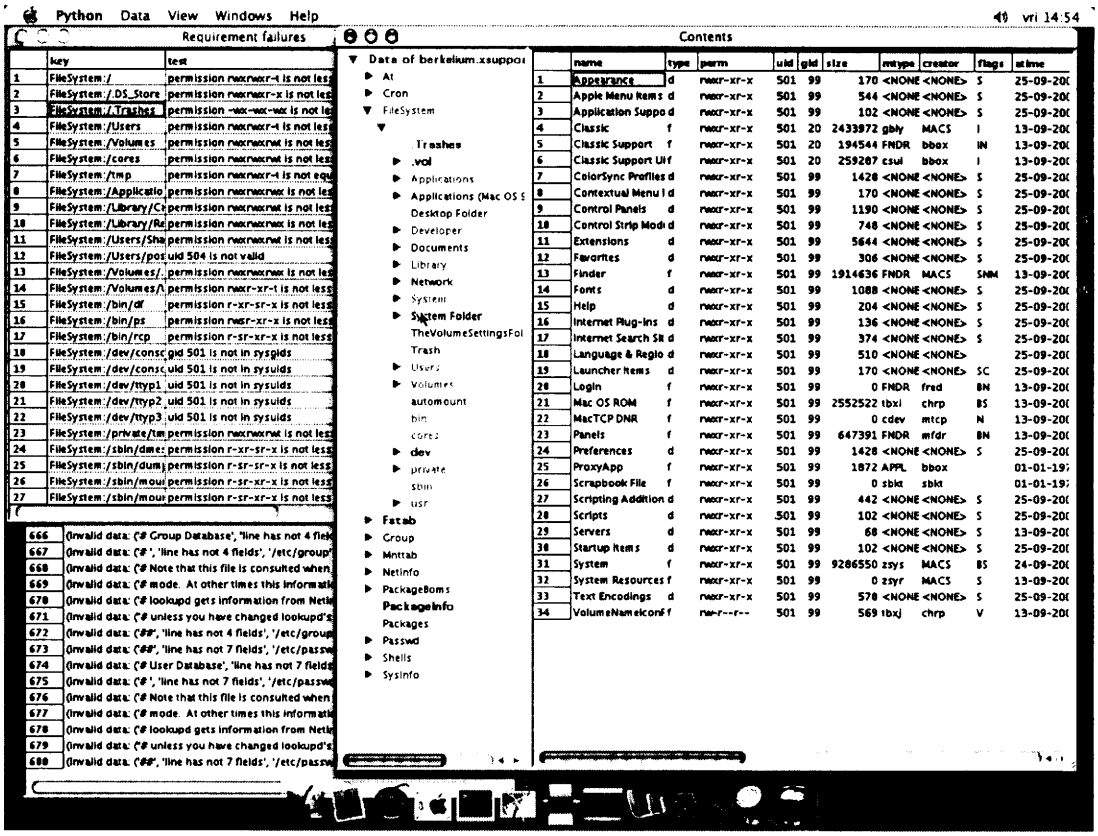


Fig. 2: Annotated tree window and errors window for a Mac OS X collection

By selecting an error or failure, the object is also selected in the annotated tree.

Repairing

For simple failures it can be convenient to repair the failure from within the viewer. Instead of writing a small shell script which may repair too little or too much, or which may even corrupt your system, Xperteyes has the possibility to use the viewer to repair the system.

A repair action can be executed on the objects that are selected in the criteria failures. This repair action should be defined previously in the criteria. The repair action will

only change the system if certain conditions are met: usually the object should still have the same properties as in the collection, but a less restrictive condition may be used, e.g. the contents are not changed.

The person defining the criteria with repair actions should be aware of the dangers of repairing. The person actually applying the repair actions should check that the failures really should be repaired. An example of the dangers is easy. Suppose that a malicious user has hard linked `/etc/passwd` in his home directory as `'.mailrc '`. One of the rules states that all objects in a home directory belong to the user. A repair action may be: set the uid to this user. It is clear that in this case the object should not be repaired, but removed. Very accurate definition of the repair action is necessary, and correct checking is also necessary.

Extending

One of the major design goals was to create an easily extendible tool. This is based on the idea that it should be easy to include some information on the system that is specific for the organization using the system. Suppose that our own company has the policy that all objects in `/xsupport/bin` should be statically linked executables. This information is not collected by default, so we need to get this additional information, and define and apply an additional test.

The additional information is most easily added to the file system collector. An extra field, called 'magic', is defined. A function is defined, which will call the external program `/usr/bin/file` and store the result in the 'magic' field. This function is added to the file system collector in the same way as the checksum and content functions.

In the file system collector configuration the magic is required for all files in `/xsupport/bin`. Now we can run the collector and the collection will contain the file magic of the files in `/xsupport/bin`, as the viewer will show.

A new test needs to be defined, which will test the magic field. This is a straightforward test, having as argument the content of the magic field. In the criteria this test is applied to the children of `/xsupport/bin`. The general checker can now generate failures for this test, and with the viewer we will see if the system satisfies our criteria.

The time estimate for implementing and testing this new feature will be about one day for someone not familiar with the code.

Portability

Xperteyes is currently running on Mac OS X, Linux and Windows. It is implemented in Python with one extension module in C, for compact storage and fast retrieval of large data sets. The graphics are based on the wxPython extension module, which is a portable (Windows, Mac and X11) graphical toolkit.

For the collectors, a few wrappers are written in C to read the system data and convert it to Python structures. This code is the only OS dependent part of the application; all other code is Python and will run on any platform with Python. Also, the data is portable, making it possible to check on a Mac OS server the Windows clients against a Linux server. The data is self-contained, so it is not necessary to use the collection on the same kind of platform.

Package views - a more flexible infrastructure for third-party software

Alistair Crooks
Wasabi Systems
<agc@wasabisystems.com>

Firstly, conventional systems for installation of third party software, including FreeBSD's ports system, NetBSD's pkgsrc, and OpenBSD's ports system, are analysed and compared. In addition, other approaches and infrastructure layouts within the industry are examined.

One of the main problems faced by users of the various systems is the means by which multiple versions of a package, or packages which "conflict" with each other, can be installed at the same time.

Following that, a new system is proposed, which has been implemented in NetBSD's pkgsrc implementation:

- to allow any number of different versions of packages to co-exist at any one time
- to provide support for dynamic packing lists
- to allow the testing of different versions of packages on a single machine at any one time
- to allow more dynamic conflict detection at install time
- whilst continuing to use the existing pkg_install(1) tools

The new system is laid out in detail, including the practical aspects of managing a large number of third party packages across a number of different Operating Systems, all within the same pkgsrc framework.

The advantages and disadvantages of the new infrastructure are discussed, as are the lessons learned from its deployment.

Mr. Crooks is Senior Development Manager for Wasabi Systems, who specialise in providing NetBSD-based solutions for the embedded marketplace. He also heads up the NetBSD packages team, and is an ex-member of the NetBSD core-team. Alistair has over 20 years experience in the IT industry, and has worked in financial institutions, for major computer manufacturers, software houses, and for other users, as well as running his own consulting business for 8 years. He lives in the UK with his wife and children, and has worked in the UK, USA, Germany and the Netherlands.

Package Views - a more flexible infrastructure for third-party software.

Alistair Crooks, Wasabi Systems, agc@wasabisystems.com

7th October 2002

Abstract

Firstly, conventional systems for installation of third party software, including FreeBSD's ports system, NetBSD's pkgsrc, and OpenBSD's ports system, are analysed and compared. In addition, other approaches and infrastructure layouts within the industry are examined. One of the main problems faced by users of the various systems is the means by which multiple versions of a package, or packages which "conflict" with each other, can co-exist at the same time.

To address that, a new system is proposed to allow any number of different versions of packages to co-exist at any one time, and the importance of dynamic packing lists is discussed. The new infrastructure is then described in detail, including the practical aspects of managing a large number of third party packages across a number of different operating systems.

Finally, the lessons learned from its deployment within the NetBSD pkgsrc infrastructure are drawn.

1 Third-party software

In the BSD world, it is not the norm to have software automatically packaged for you. That is the prerogative of operating environments such as Windows and Linux (although different Linux distributors matter to this equation). Because of this, an infrastructure which takes freely-available software and makes it available to others is desirable. This infrastructure must, at a miniumum:

1. Retrieve the software from its home site or a mirror (assuming you are connected in some way to the Internet), or from a CD or other medium.
2. Verify its integrity.
3. Apply any patches.
4. Configure the software for the host operating system, then build and install.
5. Track all installed files to permit easy removal of software using the packaging utilities.
6. Optionally create a binary package that can be installed on other hosts.

Any prerequisite software will also be automatically downloaded, built, and installed.

There are numerous advantages to having an infrastructure which does this automatically:

1. The packages have already been setup to compile and install correctly on your system, so you don't have to worry about porting the software yourself.
2. The latest versions of a program, and its patches are obtained for you, and sorted out so that the software works with NetBSD.
3. It is easy to use, and quick, even over a dialup connection.
4. You can submit additional software to the packaging system, so that others can benefit from your porting work.

5. You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network.

6. The same ease of use and maintenance applies to both binary and source based packages.

7. All packages are installed in a consistent directory tree, including binaries, libraries, man pages, and other documentation.

8. Optional configuration parameters are controlled by a single central config file, including install prefix, acceptable software licenses, and domestic(US) or international encryption requirements.

9. The packages are sorted into categories, providing useful lists of tools to browse through, all guaranteed to work.

10. Pkgsrc knows about primary distribution and mirror sites for source packages, so you can install even when that URL you memorize doesn't work.

This infrastructure helps out people new to the BSD platform by giving them pre-ported software, and helps out the "old lags" too by lifting the burden of having to duplicate the work that others may have done before them.

This, however, is nothing new. The FreeBSD ports collection has been doing this since 1993.

1.1 What is pkgsrc?

In 1997, NetBSD decided to introduce a third-party software infrastructure, and base it on the the FreeBSD ports collection `http://www.freebsd.org/ports/index.html`. The pkg_install tools were imported into the NetBSD CVS repository in June 1997, followed by the basic bsd.port.mk file to the share/ hierarchy, and then the basic pkgsrc infrastructure in early October 1997. For more information on the NetBSD Packages collection, see the relevant documentation on the NetBSD web site a shortcut to the relevant page on the NetBSD web site. `http://www.pkgsrc.org/`

The figures for the growth of pkgsrc are given by Hubert Feyrer in The Growth of the Packages Collection `http://www.netbsd.org/Documentation/software/pkg-growth.html`. There were 3214 packages in the packages collection at the end of September 2002. This compares to over 7000 for FreeBSD, and around 2000 for OpenBSD (although OpenBSD have a "flavors" enhancement to their ports system which reduces the overall number of packages).

One of the problems of bringing over ports from the FreeBSD ports collection has been that NetBSD is primarily a multi-architecture operating system. To the NetBSD people, a "port" means NetBSD running on a different architecture. (17 different processor families, 23 different architectures, 50+ platforms). Hence the name had to change, and the SI unit for NetBSD's third party software collection came to be known as a "package". A place in the CVS repository to house the infrastructure for the packages had to be found, and so that place came to be known as pkgsrc, modelled after the existing basesrc, xsrc, othersrc directories. "pkgsrc" was born.

1.2 How does it differ from the ports collection?

The NetBSD packages team made a number of significant changes. A full list of these changes can be gleaned from the web interface to the NetBSD CVS repository. the web interface to the NetBSD CVS repository `http://cvsweb.netbsd.org/bsdweb.cgi/`.

1. bsd.port.mk and the mtree files were moved to be in the pkgsrc hierarchy, and use relative paths to refer to files within pkgsrc. This allows us to have a number of pkgsrc trees checked out and in use at the same time.

2. Real CONFLICT handling was added to packages.

3. Wildcard and relative matching of package version numbers was added

4. "just-in-time su(1)" functionality was added, so that people can do everything except package installation as unprivileged users

5. pkgsrc was ported to Solaris, and then to Linux and Darwin, so that people can use pkgsrc on those platforms. This used to be done by means of a compatibility layer called Zoularis, but is now done natively, using the othersrc/bootstrap-pkgsrc generic bootstrap kit. Debugging output was improved at the same time. We have a truly generic bsd.pkg.mk, whereby different Operating Systems use abstract values defined in a defs.${OPSYS}.mk, and these abstractions are then used within bsd.pkg.mk. This makes it much easier to port pkgsrc to other operating systems.

6. The specification of default values was made the same across all packages, with a single file which is automatically included in the make(1) process - bsd.pkg.defaults.mk - and differences from the defaults can be placed in /etc/mk.conf. One single file was made which can be included by package Makefiles in order to pick up standard defaults, and also any differences from the norm as specified in /etc/mk.conf - package Makefiles simply .include "../../mk/bsd.prefs.mk" before any make(1) .if ... conditionals.

7. "buildlink" functionality was introduced, which ensures that the correct files are used in the build and linking processes.

8. Manual pages are not specified in a package Makefile - if a package has files, they are all included in the package's PLIST.

9. Simple coarse-grained locking was added to pkgsrc using shlock(1). If a package is being built, subsequent attempts to build the same package will lock, waiting for the first package to finish building.

10. The package tools were given the ability to use digitally-signed packages - if a package has been signed, the user can be prompted whether or not to install a package, depending on whether or not the creator of the binary package is trusted.

11. Message digests of all relevant patch files were generated, so that people using sup or extracting patch files over an existing set of patch files will only get the necessary patches applied. (If the digest doesn't match, the patch file is not applied). Support was added for message digests other than md5 for distfiles and patches, by using the digest package, and support for SHA256 and SHA512 was added to the digest package

12. The ACCEPTABLE_LICENCE feature was added to /etc/mk.conf, to ensure that people only installed packages with whose licence they agreed.

13. Automatic calculation of the effective date of the pkg_install tools is carried out. If the tools are too old, the user will be told this, and how to fix it (typically, by installing the pkg_install package). Full-pathname symbolic links are adjusted in pkg_create(1) to be relative to ${PREFIX}, if appropriate. This helps with binary packages.

14. An xpkgwedge package was added, which makes packages which would normally be installed in ${X11BASE} be installed in ${LOCALBASE}. pkg_info(1) will find out the installed prefix of a package dynamically, rather than guessing at ${X11BASE} or ${LOCALBASE}

15. The MASTER_SITE_SORT definition was added whereby we can sort the MASTER_SITES so that the nearest toplogically get tried first, and FAILOVER_FETCH was added so that, when retrieving distfiles, the distfile digests can be checked, and, if they don't match, the distfile will be considered incorrect, and the next site will be tried.

16. The object format for shared libraries is determined dynamically at package install time rather than using a hard-coded table - this is much more dynamic, and allows NetBSD ports to migrate from a.out to ELF with no appreciable changes to the pkgsrc infrastructure. All binaries and

shared libs are checked after installation to make sure that any shared libs are found correctly by said binaries and other shared libraries.

17. The audit-package package was added, which uses the relational matching of package names to scan a published list of known vulnerabilities, which is maintained by the NetBSD Security Officer, and published on ftp.netbsd.org as the vulnerabilities file `ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/vulnerabilities`, along with a small script to download it. This allows users to be notified automatically if there is a vulnerability in one of their installed packages, and does away with the need for security advisories for packages.

18. "system packages" have been added to the base system, whereby all system utilities and kernels can be treated as packages, and deleted, added, matched, updated at will.

19. The BUILD_DEPENDS semantics were changed to match the existing DEPENDS syntax - the first component is now a pkg_info(1) recognisable package name (with possible relational or alternate matching)

20. Special handling for the installation of rc.d scripts, create users, and install example files has been added

21. A new framework for handling info files generation and installation was added

22. A "replace" target was introduced, which updates a package in place, modifying any packages which use it. There's also an "undo-replace" target

23. A finer-grained INTERACTIVE_STAGE definition was introduced, so that builds can continue better unattended

and many, many more enhancements, improvements and speedups.

OpenBSD have also made a number of separate improvements (they have made many more, these are simply some of the main ones)

1. They were the first to speed up the building process by eliminating the use of .USE macros.

2. They have implemented their "FAKE" functionality to provide staged installations (similar to Debian packages).

3. They implemented "flavors" functionality, whereby a package can be built in a number of ways, for example with or without X11 functionality.

FreeBSD have continued to grow their ports collection, and still have the most packages - near 7000 at the last count.

2 The current situation

The conventional *BSD ways of installing software (NetBSD's pkgsrc, FreeBSD/OpenBSD ports system) install directly into ${LOCALBASE}, possibly overwriting existing files. This has certain disadvantages:

1. There can only be one version of a piece of software installed at any one time. There are numerous occasions when it is desirable to have a newer copy of software to be evaluated, whilst still using the production copy of this. One approach is to install the package to be evaluated by using a different prefix, but there are numerous package management problems with this approach, and it does not scale well.

2. It is often possible that a package overwrites a working version of another unrelated package simply because they contain commands or libraries header files with the same name. Whilst this may seem trivial, and a simple choice has to be made as to the more appropriate package to have installed under these circumstances, it is often more complicated than that. Other packages may demand certain choices be made, which may not be convenient for individual users.

3. Problems can arise when some third party software is upgraded, and a lot of other software depends upon it (libpng, jpeg, zlib). All of the packages which use the updated package have to be re-linked, and the only feasible way to do that is to de-install all of them, with the ensuing problems that that can bring. Various attempts have been made to work around this situation (retiring packages, pkg_hack, "make replace"), but none of these has addressed the fundamental problem.

It is desirable to have a means whereby two packages with the same file system entries can co-exist. As explained above, one method of doing this is to install the newer package into a ${LOCALBASE} in a different location, but this does not scale at all well, and we run into problems with the metadata files in ${PKG_DBDIR}. It is clear that a different approach is needed.

3 Other approaches

Some other approaches to the problems outlined above have been tried:

1. Using separate machines (where they are available) to install newer versions of packages, test their stability and functionality, and then finally deploy them across a network of machines.

2. "Retiring" packages (where shared objects are retained under a differently-named package) will only work properly when the major number of the shared objects are changed on ELF platforms.

3. OpenBSD's "FAKE", a staged installation approach similar to Debian's, will only allow one version of a package to be installed at any one time. In this case, a binary package is created in the staging area, and that binary package is added to the destination. This has the benefit of creating a binary package which can then be installed on other systems, and otherwise manipulated.

4. CMU's depot software `http://andrew2.andrew.cmu.edu/depot/` is a large piece of software which creates a tiered environment for third party software packages. Some consider it too unwieldy for use in a packaging environment, and mandates an interesting bootstrap procedure and difficult management and configuration problems. One of our pkgsrc developers used this for his own packaging system before switching to pkgsrc (on Solaris). Maintenance was the main reason he cited for this.

5. GNU's stow program `http://www.gnu.ai.mit.edu/software/stow/stow.html` is a very useful program, which uses a tiered approach to software installation. Unfortunately, stow is written in Perl, which again provides us with some bootstrap problems. The program is also distributed under the GPL, and we'd rather not go there.

6. Various other packaging efforts `http://www.encap.org/` also use a tiered approach to the installation of software

After much consideration, it was decided that the approaches outlined above could be improved. Some experiments were made with a staged installation approach, similar to OpenBSD's "FAKE" approach, but other problems with this method encountered. Three approaches to installing a package into a staging area were identified:

1. The package's build mechanism already provided a means of installing into a staging area - packages which have been modified for Debian's ${DESTDIR}, for example, and newer X11-based packages which also installed into ${DESTDIR}. This approach was known as the "DESTDIR" approach.

2. A number of wrapper scripts were written, to enable install(1), ln(1), cp(1) and other programs which are used to install packages into ${LOCALBASE} to take the same arguments as at present, but modify these arguments internally to point to the staging area. This approach was

found to be applicable in most circumstances, although we also encountered problems with packages which used GNU libtool, perl and other utilities to install their files, and a surprising number of wrapper scripts had to be written.

3. by setting ${LOCALBASE} to include a specific ${DESTDIR} component, and passing that down to sub-make invocations within the package build and installation procedures.

However, these experiments showed that this approach was simply were papering over the cracks - the base problem (that you can have only one version of a package installed at any one time) still existed, and had not been worked around in any way by this.

4 The aims of package views

Having studied the problem, it was obvious that a better method of installing packages into a destination was necessary. The main aim was that multiple versions of a package should be capable of being installed at any one time. There were also subsidiary aims, too:

1. to allow any number of different versions of packages to co-exist at any one time
2. to allow the testing of different versions of packages on a single machine at any one time
3. to allow more dynamic conflict detection at install time
4. whilst continuing to use the existing pkg_install tools.

4.1 Dynamic Packing Lists

It was subsequently realised that if a package was installed in its own hierarchy, then dynamic PLISTs could also be supported. From its inception, pkgsrc has used a static list of files which constitute the package. This list of files is called a "PLIST", which is short for "Packing LIST". Over the years, the PLISTs have taken up more and more time in package maintenance, requiring manipulation for:

- gzipped or standard manual pages
- shared object and library differences by platform and by object format (ELF or a.out)
- changes to reflect other packages installed on a machine (which may not be desired or necessary)
- the machine architecture
- the version of the operating system software
- the version number of the package itself

If PLISTs could be created at installation time, a lot of this extra maintenance would disappear. Dynamic PLISTs require no manual maintenance, and remove a barrier from anyone wishing to create a pkgsrc entry for a new package. Dynamic PLISTs also mean that the manipulations described above do not have to be performed. There are other packaging systems in existence which use dynamic packing lists (Amdahl's PSF, included in UTS 4.3.3, for example) from which many lessons can be drawn.

5 Package Views

From the basic tenet that multiple versions of a single package need to be installed, it was obvious that a single ${LOCALBASE} directory was insufficient - multiple ${LOCALBASE}s were necessary. It then became obvious that some form of layering would be needed to accomplish this aim. We also observed the way that multiple versions of packages were installed on machines manually by seasoned administrators.

- The basic idea of package views is that a tiered approach, which was later found to be similar to the encap packaging system.
- The basic package is installed into ${LOCALBASE}/packages/${PKGNAME}. This is called the depot directory.
- A custom built shell script is used to build the upper tiers of symbolic links in separate "views", pointing to the files and directories in the depot directory.

Using these ideas, we build up small hierarchies per package. Symbolic links are made to each of the files and symbolic links which constitute a package, and those symbolic links are referenced, rather than the original file within the small hierarchy of the package. For example, with a number of packages installed, the contents of the ${LOCALBASE}/packages directory is shown in Figure 1.

Within each of these "depot" directories, the hierarchy is shown in Figure 2.

As can be seen, the package's metadata files are kept in the depot directory - this is so that the pkg_install utilities work when used with a ${PKG_DBDIR} value of ${LOCALBASE}/packages (so that relational matching of package names and version numbers continue to work). Once the files have been installed in the depot directory, we then create a "view" of that package's entries under ${LOCALBASE}. This is called the default view.

We make a "linkfarm" of symbolic links to the entries under ${LOCALBASE}/packages/${PKGNAME} for each of the files and symbolic links in the package. If there is a package-specific directory in the depot directory, it will be created as a directory in ${LOCALBASE}, provided it does not yet exist. If there is already an entry under ${LOCALBASE} with the same name, that symbolic link is replaced by the new symbolic link. This is not such a drastic move as it is at the present time - since the entry under ${LOCALBASE} is merely a symbolic link, the entry in the other depot directory is not touched in any way.

The linkfarm is created by an extra Bourne shell script, and was written to do the same work as the GNU stow program, except for the folding of directories. The linkfarm script takes the same (long and short) arguments as stow, and performs the same job.

When the linkfarm has been created, a +VIEWS metadata file is added to the depot directory. This file contains the views which have been built on top of the depot directory.

There is one default view, and all packages have a view in the default view.

Any number of other views can also be created. For example, a "devel" view could be created specifically for packages which have to be tested and evaluated before being put into production use. In a similar way, "kde2", "kde3" and "gnome2" views could be created in order to appraise those specific groups of packages. We are occasionally asked about putting all GNU utilities under a separate ${PREFIX} in pkgsrc - with package views, these packages can quite simply be pulled up into a "gnu" view.

It should be noted that all packages, even the X11-based ones, need to install into the same ${LOCALBASE} directory. This means that xpkgwedge is obligatory (xpkgwedge puts a package which would normally be destined to be installed under ${X11BASE} into the normal ${LOCALBASE} hierarchy). This has other benefits too, since xpkgwedge preserves the sanctity of what some consider to be system libraries, and reduces the impact upon the installed package hierarchy when a new version of X11 is installed on the computer, although some re-linking may be necessary.

A package may not be deleted from the depot directory if there are any views of that package in existence. This is to preserve the cleanliness of the views model, to keep a principle of cleaning up after ourselves, and to preserve the sanity of system administrators everywhere. The standard pkg_delete(1) command can be used to delete a view, as can the linkfarm script. pkg_delete(1), and linkfarm(1), can also be used to delete a view itself. pkg_info(1) can be used to view packages in the depot directory or in views.

When the next version of the package comes along, because it has a different package name, it gets installed into a different depot directory. The two different versions exist side by side. If the old view in ${LOCALBASE} still exists, the linkfarm script can be used to delete the old view, before making the new view for the new package version. This ensures that packages linking to the package will pick up the entries in the new version of the package.

At the current time, packages link with prerequisite packages in ${LOCALBASE}. Over time, we may migrate this to link directly to files in the depot directories, so that packages are built with one

```
[15:54:02] agc@sys1 /usr/vpkg/packages 13 > ls -al
total 150
drwxr-xr-x 147 root wheel 4096 May 13 16:09 .
drwxr-xr-x 19 root wheel 512 May 7 20:23 ..
drwxr-xr-x 10 root wheel 512 May 7 15:52 9wm-1.1
drwxr-xr-x 10 root wheel 512 May 7 22:12 GConf-1.0.9
drwxr-xr-x 10 root wheel 512 May 7 17:34 Mesa-3.4.2nb1
drwxr-xr-x 13 root wheel 512 May 7 22:12 ORBit-0.5.15
drwxr-xr-x 10 root wheel 512 Apr 25 09:55 Xaw3d-1.5
drwxr-xr-x 13 root wheel 512 May 13 11:37 a2ps-4.13.0.2
drwxr-xr-x 13 root wheel 512 May 7 16:36 abiword-personal-0.99.5
drwxr-xr-x 13 root wheel 512 Apr 26 19:23 autoconf-2.13
drwxr-xr-x 13 root wheel 512 Apr 26 19:23 automake-1.4.5nb1
drwxr-xr-x 13 root wheel 512 Apr 25 11:15 bison-1.35
drwxr-xr-x 10 root wheel 512 May 7 22:12 bonobo-1.0.18nb1
drwxr-xr-x 10 root wheel 512 May 7 17:34 control-center-1.4.0.4
drwxr-xr-x 13 root wheel 512 May 8 21:40 curl-7.9.6
...  etc ...
```

Figure 1: the contents of the ${LOCALBASE}/packages directory

canonical version, but doing this has other ramifications, such as the ability to have wildcard dependencies on other packages.

6 Practical Aspects of Package Views

6.1 Views

A package's files are always in one canonical location, the depot directory. On top of that, views can be constructed.

- There is a default view, which defaults to ${LOCALBASE}.
- Any number of views can be added.
- The traditional NetBSD pkg_install(1) tools are used, with the addition of the script to manage the symbolic link farms.

6.2 ${LOCALBASE} vs. ${X11BASE}

Traditionally, packages have installed into ${LOCALBASE}, or ${X11BASE}, depending upon a number of issues.

NetBSD's pkgsrc has a utility called xpkgwedge which forces all packages which would normally install into ${X11BASE} into ${LOCALBASE}, thereby keeping the X11 tree "clean".

6.3 ${PREFIX}

With xpkgwedge installed on a computer, all packages now install into ${LOCALBASE}. The floating ${PREFIX} definition is now unnecessary. However, ${PREFIX} is used in most of the packages' own Makefiles to represent the installation prefix.

We thus "move" the ${PREFIX} definition to refer to the depot directory, ${LOCALBASE}/packages/${PKGNAME}. This gives us a simple and easy way to refer to the depot directory from package Makefiles.

```
[16:01:00] agc@sys1 ...vpkg/packages/pth-1.4.1 > ls -al
total 22
-rw-r--r-- 1 root wheel 693 May 7 15:52 +BUILD_INFO
-rw-r--r-- 1 root wheel 374 May 7 15:52 +BUILD_VERSION
-rw-r--r-- 1 root wheel 32 May 7 15:52 +COMMENT
-rw-r--r-- 1 root wheel 224 May 7 15:52 +CONTENTS
-rw-r--r-- 1 root wheel 1063 May 7 15:52 +DESC
-rw-r--r-- 1 root wheel 6 May 7 15:52 +SIZE_ALL
-rw-r--r-- 1 root wheel 6 May 7 15:52 +SIZE_PKG
-rw-r--r-- 1 root wheel 17 May 7 15:52 +VIEWS
drwxr-xr-x 10 root wheel 512 May 7 15:52 .
drwxr-xr-x 147 root wheel 4096 May 13 16:09 ..
drwxr-xr-x 2 root wheel 512 May 7 15:52 bin
drwxr-xr-x 3 root wheel 512 May 7 15:52 etc
drwxr-xr-x 3 root wheel 512 May 7 15:52 include
drwxr-xr-x 2 root wheel 512 May 7 15:52 info
drwxr-xr-x 4 root wheel 512 May 7 15:52 lib
drwxr-xr-x 2 root wheel 512 May 7 15:52 libexec
drwxr-xr-x 25 root wheel 512 May 7 15:52 man
drwxr-xr-x 7 root wheel 512 May 7 15:52 share
[16:01:02] agc@sys1 ...vpkg/packages/pth-1.4.1 >
```

Figure 2: the contents of a "depot" directory

6.4 bsd.pkg.mk internals

At the present time, packages which use GNU configure scripts are passed the item –prefix=${GNU_CONFIGURE_PREFIX} where ${GNU_CONFIGURE_PREFIX} defaults to ${PREFIX}. With package views, as mentioned above, PREFIX is modified to point to ${LOCALBASE}/packages/${PKGNAME}, and so no further internal manipulation of prefices needs to take place.

6.5 Upgrading packages

Previously, an upgrade or update to a package, especially one containing shared libraries and objects, could be an onerous task, especially (on ELF systems) if a shared library major number change was involved. With packages views, the new package is installed alongside the old one. There are now two possible circumstances (it is assumed that ELF platforms are being used, since almost all systems now use the ELF format):

6.5.1 The majority of cases

In the overwhelming majority of cases, the newer version of the package is installed in its own depot directory, the linkfarm in the default view to the older version is deleted, and a new linkfarm to the newer version is created in the default view. No further changes are necessary, and it is possible to try out other packages which use this package, even if shared libraries are involved. Note that, should the newer version of the package not function as intended, it is a simple matter to revert to the older version, by deleting the linkfarm in the default view to the newer version, and adding a linkfarm to the default view for the older version. As we optimise for the most common occurrence in all things, this approach brings huge benefits.

6.5.2 A shared library major number change

Using the existing "overwrite" meachanism, for a few specific and annoying cases, a major number change

for a shared library has meant that those packages, and any other packages which re-use them, have to be re-linked. There have been two memorable occasions over the last year (libpng and libiconv) when this has necessitated a large amount of "make update" work. With package views, this situation does not cause any problems, since the old shared library is still around in its depot directory, and the symbolic link to it still exists from the default view; similarily, the new shared library exists in its depot directory, and a symbolic link to its major version exists in the default view, too:

libwibble.so -> /usr/pkg/packages/wibble-2.0/lib/libwibble.so.2.0

libwibble.so.1 -> /usr/pkg/packages/wibble-1.0/lib/libwibble.so.1.0

libwibble.so.2 -> /usr/pkg/packages/wibble-2.0/lib/libwibble.so.2.0

Whilst the symbolic link to the non-versioned shared library in the default view (libwibble.so) is overwritten, it makes no difference, since that symbolic link is only used for compilation.

7 A Worked Example

7.1 An illustration - the depot directory

The files which constitute the package's entries in the file system are shown in Figure 3.

7.2 An illustration - the default view

The files which constitute the package in the default view are shown in Figure 4.

7.3 The Linkfarms

The symbolic links in the default view, and their targets under the depot directory, are shown in Figure 5.

8 Advantages

With package views, the immediate benefits are the same as the aims:

1. to allow any number of different versions of packages to co-exist at any one time
2. to allow the testing of different versions of packages on a single machine at any one time
3. to allow more dynamic conflict detection at install time
4. whilst continuing to use the existing pkg_install tools, and
5. to provide support for dynamic packing lists

and, quite unexpectedly, other advantages were gained:

1. it is immediately obvious to which package a file or directory belongs
2. many additional views can be built up - package views are scalable
3. pkg_delete(1) deletes links in the views as well as the package itself
4. multiple conflicting packages (not just multiple versions of one package) can be installed at the same time
5. development packages can be tested and evaluated on the same machine on which they will eventually run

This is portable to any system on which pkgsrc runs - NetBSD, Solaris, Darwin, and Linux. FreeBSD, Irix, Digital Unix and HP/UX are currently in the works, although the generic bootstrap kit should work on any POSIX-compliant system.

Users can migrate to package views simply by setting an /etc/mk.conf variable definition. For cleanliness, it would be better to move to a complete package views system at one time, and so a pkgsrc flag day is on the cards. In reality, the current "overwrite"

```
[16:42:15] agc@sys1 /usr/vpkg/packages 339 > env PKG_DBDIR=/usr/vpkg/packages pkg_info -L
pth
Information for pth-1.4.1:
Files:
/usr/vpkg/packages/pth-1.4.1/bin/pth-config
/usr/vpkg/packages/pth-1.4.1/bin/pthread-config
/usr/vpkg/packages/pth-1.4.1/include/pth.h
/usr/vpkg/packages/pth-1.4.1/include/pthread.h
/usr/vpkg/packages/pth-1.4.1/lib/libpth.a
/usr/vpkg/packages/pth-1.4.1/lib/libpth.la
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14.21
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.a
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.la
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14.21
/usr/vpkg/packages/pth-1.4.1/man/man1/pth-config.1
/usr/vpkg/packages/pth-1.4.1/man/man1/pthread-config.1
/usr/vpkg/packages/pth-1.4.1/man/man3/pth.3
/usr/vpkg/packages/pth-1.4.1/man/man3/pthread.3
/usr/vpkg/packages/pth-1.4.1/share/aclocal/pth.m4
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/ANNOUNCE
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/AUTHORS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/COPYING
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/HACKING
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/NEWS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/README
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/SUPPORT
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/TESTS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/THANKS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/USERS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/pthread.ps
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/rse-pmt.ps
[16:42:27] agc@sys1 /usr/vpkg/packages 340 >
```

Figure 3: the contents of the package in the "depot" directory

```
[16:42:27] agc@sys1 /usr/vpkg/packages 340 > pkg_info -L pth
Information for pth-1.4.1:
Files:
/usr/vpkg//bin/pth-config
/usr/vpkg//bin/pthread-config
/usr/vpkg//include/pth.h
/usr/vpkg//include/pthread.h
/usr/vpkg//lib/libpth.a
/usr/vpkg//lib/libpth.la
/usr/vpkg//lib/libpth.so
/usr/vpkg//lib/libpth.so.14
/usr/vpkg//lib/libpth.so.14.21
/usr/vpkg//lib/libpthread.a
/usr/vpkg//lib/libpthread.la
/usr/vpkg//lib/libpthread.so
/usr/vpkg//lib/libpthread.so.14
/usr/vpkg//lib/libpthread.so.14.21
/usr/vpkg//man/man1/pth-config.1
/usr/vpkg//man/man1/pthread-config.1
/usr/vpkg//man/man3/pth.3
/usr/vpkg//man/man3/pthread.3
/usr/vpkg//share/aclocal/pth.m4
/usr/vpkg//share/doc/pth/ANNOUNCE
/usr/vpkg//share/doc/pth/AUTHORS
/usr/vpkg//share/doc/pth/COPYING
/usr/vpkg//share/doc/pth/HACKING
/usr/vpkg//share/doc/pth/NEWS
/usr/vpkg//share/doc/pth/README
/usr/vpkg//share/doc/pth/SUPPORT
/usr/vpkg//share/doc/pth/TESTS
/usr/vpkg//share/doc/pth/THANKS
/usr/vpkg//share/doc/pth/USERS
/usr/vpkg//share/doc/pth/pthread.ps
/usr/vpkg//share/doc/pth/rse-pmt.ps
[16:42:41] agc@sys1 /usr/vpkg/packages 340 >
```

Figure 4: the contents of the package in the default view

```
[16:42:41] agc@sys1 /usr/vpkg/packages 341 > ls -al `pkg_info -qL pth`
lrwxr-xr-x 1 root wheel 43 Apr 24 09:28 /usr/vpkg//bin/pth-config -> /usr/vpkg/packages/pth-1.4.1/bin/pth-config
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//bin/pthread-config -> /usr/vpkg/packages/pth-1.4.1/bin/pthread-config
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//include/pth.h -> /usr/vpkg/packages/pth-1.4.1/include/pth.h
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//include/pthread.h -> /usr/vpkg/packages/pth-1.4.1/include/pthread.h
lrwxr-xr-x 1 root wheel 41 Apr 24 09:28 /usr/vpkg//lib/libpth.a -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.a
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//lib/libpth.la -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.la
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//lib/libpth.so -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.so
lrwxr-xr-x 1 root wheel 45 Apr 24 09:28 /usr/vpkg//lib/libpth.so.14 -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//lib/libpth.so.14.21 -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14.21
lrwxr-xr-x 1 root wheel 45 Apr 24 09:28 /usr/vpkg//lib/libpthread.a -> /usr/vpkg/packages/pth-1.4.1/lib/libpthread.a
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//lib/libpthread.la -> /usr/vpkg/packages/pth-1.4.1/lib/libpthread.la
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//lib/libpthread.so -> /usr/vpkg/packages/pth-1.4.1/lib/libpthread.so
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//lib/libpthread.so.14 -> /usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14
lrwxr-xr-x 1 root wheel 52 Apr 24 09:28 /usr/vpkg//lib/libpthread.so.14.21 -> /usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14.21
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//man/man1/pth-config.1 -> /usr/vpkg/packages/pth-1.4.1/man/man1/pth-config.1
lrwxr-xr-x 1 root wheel 54 Apr 24 09:28 /usr/vpkg//man/man1/pthread-config.1 -> /usr/vpkg/packages/pth-1.4.1/man/man1/pthread-config.1
lrwxr-xr-x 1 root wheel 43 Apr 24 09:28 /usr/vpkg//man/man3/pth.3 -> /usr/vpkg/packages/pth-1.4.1/man/man3/pth.3
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//man/man3/pthread.3 -> /usr/vpkg/packages/pth-1.4.1/man/man3/pthread.3
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/aclocal/pth.m4 -> /usr/vpkg/packages/pth-1.4.1/share/aclocal/pth.m4
lrwxr-xr-x 1 root wheel 51 Apr 24 09:28 /usr/vpkg//share/doc/pth/ANNOUNCE -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/ANNOUNCE
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/AUTHORS -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/AUTHORS
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/COPYING -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/COPYING
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/HACKING -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/HACKING
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//share/doc/pth/NEWS -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/NEWS
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/doc/pth/README -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/README
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/SUPPORT -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/SUPPORT
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//share/doc/pth/TESTS -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/TESTS
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/doc/pth/THANKS -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/THANKS
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//share/doc/pth/USERS -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/USERS
lrwxr-xr-x 1 root wheel 53 Apr 24 09:28 /usr/vpkg//share/doc/pth/pthread.ps -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/pthread.ps
lrwxr-xr-x 1 root wheel 53 Apr 24 09:28 /usr/vpkg//share/doc/pth/rse-pmt.ps -> /usr/vpkg/packages/pth-1.4.1/share/doc/pth/rse-pmt.ps
[16:43:05] agc@sys1 /usr/vpkg/packages 342 >
```

Figure 5: the target symbolic links of the package view

functionality and "pkgviews" functionality can coexist until such time as migration to package views has taken place.

In all, the package views approach is scalable in practice (see similar papers on the infrastructure.org web site `http://www.infrastructure.org/`, and from experience of other highly-experienced system administrators).

9 Disadvantages

Of course, there are disadvantages to this approach:

1. Some people think that the linkfarms are unruly, unsightly and ugly.

2. A minimal amount of extra space is used to provide the linkfarm. The early versions of package views had code to use "hard" links rather than symbolic links to achieve the same effect. This was possible, since it is highly likely that a file and its link will reside on the same file system. Where this approach failed was in configuration files, which may be edited by people using popular editors which create a new file rather than a "hard" link to a file when the editing session is saved. In all, however, some extra space is used to store the symbolic link information, but, in the whole scheme of things, with falling disk costs and increasing disk capacities, it is no more than a fraction of a percentage of the total disk space used, and so can be discounted for all practical purposes.

In all, with different versions of packages to be installed side by side, more disk space in general will be needed (this is more of a consequence than a disadvantage), which may not always be appropriate (NetBSD still runs on a number of systems, like the VAX and acorn26, where directly attached disk space is at a premium). One suggestion for this is to use NFS or cheaper, mass-produced IDE discs (where possible).

10 Conclusions

The advantages of being able to have two different versions of a package installed at any one time are immense. It is now possible to try out new versions of packages without compromising the existing version. The move to dynamic packing lists will simplify pkgsrc entry creation for everyone, and reduce the amount of maintenance which has to be performed on the current packages, including all the special cases for different operating systems and object formats. In addition, package views allow us to detect conflicts at package install time, rather than by specifying this as a static definition in a package Makefile, and resolve the conflicts in a non-destructive way. The existing package tools can continue to be used, and the symbolic link farms, whilst ugly, give an immediate idea of the package to which a file entry belongs. Whilst using package views, a direct increase in the amount of disk space which will be used is only to be expected. The utility value of the advantages far outweigh the disadvantages.

11 Future directions

At present, package views are implemented on a CVS branch within the NetBSD CVS repository. We intend to take the following steps within pkgsrc:

1. Make xpkgwedge the default for all packages, having first made sure via a bulk build that all packages are xpkgwedge-friendly

2. Introduce package views by merging the **pkgviews** branch in the NetBSD pkgsrc CVS repository with the trunk. At the current time the default is not to use package views - they are only used if the definition **PKG_INSTALLATION_TYPE** is set to **pkgviews.** The default value for **PKG_INSTALLATION_TYPE** is **overwrite.**

3. When that has been done, we will switch over to dynamic PLISTs in pkgsrc. This will be done

by using the **PLIST_TYPE** definition to **dynamic**. The default value for **PLIST_TYPE** is **static**.

4. Monitor reaction to package views and dynamic PLISTs, and to improve upon it where possible

References

[FreeBSD] http://www.freebsd.org/ports/index.html - FreeBSD Ports

[Pkgsrc] http://www.pkgsrc.org/ - A shortcut to the pkgsrc area on the NetBSD website.

[Feyrer] http://www.netbsd.org/Documentation/software/pkg-growth.html - The Growth of the packages Collection

[NetBSD] http://cvsweb.netbsd.org/bsdweb.cgi/ - a web interface to the NetBSD CVS Repository

[CMU] http://andrew2.andrew.cmu.edu/depot/ - The Depot Configuration Management Project

[GNU] http://www.gnu.ai.mit.edu/software/stow/stow.html - GNU Stow

[Encap] http://www.encap.org/ - The Encap Archive

[Infrastructures] http://www.infrastructures.org/ Infrastructures.org - Best Practices in Automated Systems Administration and Infrastructure Architecture

[Vulnerabilities] ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/vulnerabilities - A List of Known Vulnerabilities in Packages

Clustering NetBSD

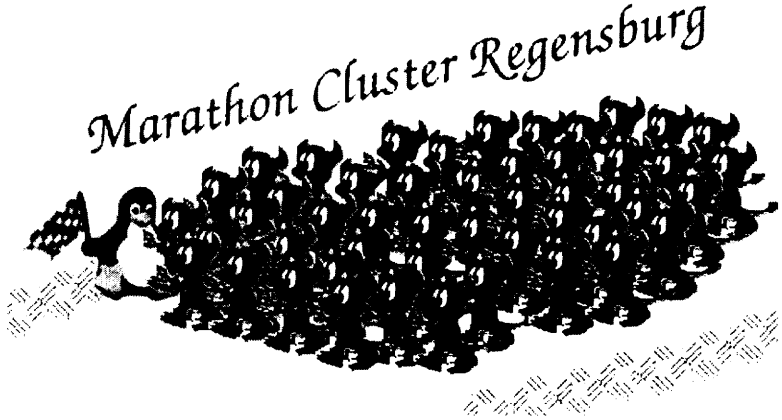
Hubert Feyrer
The NetBSD Project
<hubertf@netbsd.org>

In last year's Regensburg city marathon, each of the 5000+ runners was able to view a personal video of him/herself reaching the goal. A cluster of 45 machines running the NetBSD operating system turned 5 hours of video material into single mpeg snippets.

Topics discussed in the presentation are:

- deployment of the operating system and applicationi software on the cluster machines
- details on the tasks performed by the cluster - decomposing the input mpeg stream(s), re-assembling images for each runner based on the time he went through the goal
- experiences gained during the cluster project - application based, plus lots of graphs and images from monitoring the cluster
- facts numbers - lots of them :)

After studying computer science with an emphasis on operating systems, Hubert Feyrer started working as system and network engineer at the CS department of the University of Applied Science (FH) Regensburg. His daytime work includes administrating the CS department's Unix machines and IPv6 infrastructure. In his spare time, he contributes to the NetBSD project in 1993. Work areas there include the NetBSD Packages System and 3rd Party Software and documentation as well as advocacy and PR. Other interresting projects include evaluation of NetBSD on Toshiba laptops, lectures on Unix system administration at the FH Regensburg as well as a NetBSD-based cluster of 45 machines for rendering videos during the Regensburg city marathon.



Reaching the Goal with the Regensburg Marathon-Cluster

– A NetBSD Cluster Project –

Hubert Feyrer <hubert@feyrer.de>

October 7, 2002

Abstract

This paper discusses the technical setup and execution of a video rendering cluster based on Open Source software. First, deployment of the cluster clients is discussed, followed by computing steps performed by the cluster - splitting a big video stream into single images and rendering individual videos from the images. Details are given on the hardware and software used as well as optimizations made. Various experiences gained by the cluster project are listed before showing some performance graphs displaying the cluster under load. Lists of facts, numbers and links sum up the whole project.

Contents

1 Introduction

Last summer, R-KOM and the University of Applied Science (Fachhochschule, FH) Regensburg, Germany, took their share of the Regensburg city marathon by putting an individual video and image of each runner reaching the goal on the Internet. A cluster of 45 machines rendered the more than five thousand videos.

Being a sponsor of the Sports Experts Marathon, R-KOM contributed by putting the video and images of each runner completing the course up on the Internet. Due to increased demands on quality, the company's machines were not fast enough to render all the images and videos in a reasonable amount of time. With contacts to the Fachhochschule Regensburg, the department of computer science offered a sufficient number of machines as well as support for setting them up, and a collaboration was made .

The university of applied sciences (Fachhochschule) Regensburg provided 45 machines, including installation and management of the nodes, data storage, and also helped optimizing and tuning the Unix-based software created by employees of R-KOM based on freely available components.

The software consisted of components to split video streams into single pictures as well, as rendering video sequences from single pictures based on the time of arrival of each runner. A fully automated, scriptable processing was important here for the 5.500 runners reaching the goal.

Preparations of the Regensburg Marathon Cluster started on early saturday evening by installing the cluster machines under supervision of Hubert Feyrer of the computer science department of the Fachhochschule Regensburg as well as several students, who helped deploying the machines. Adjusting and tuning the cluster software took the rest of the pre-marathon evening.

After the last runner reached the goal on Sunday afternoon, processing of the video material provided by the local TV company TVA started under the supervision of Jürgen Mayerhofer, R-KOM. The material was used to render video sequences and pictures of each runner's completion of the course.

Using NetBSD, a Unix/Linux-like Open Source operating system, on the cluster clients and other Open Source software to control and calculate the MPEG-animations of the marathon videos and pictures gave a solid base for a software project of this size.

2 Setup of the cluster client machines

The computer science department of the FH Regensburg provided three public rooms full of machines for the cluster, with a total of 57 computers. 15 of the machines were running Solaris, which we were not allowed to change. The remaining machines were available for re-installing, and we chose NetBSD as client operating system, as all the software we needed was easily available through the NetBSD 3rd party software collection, the system was simple

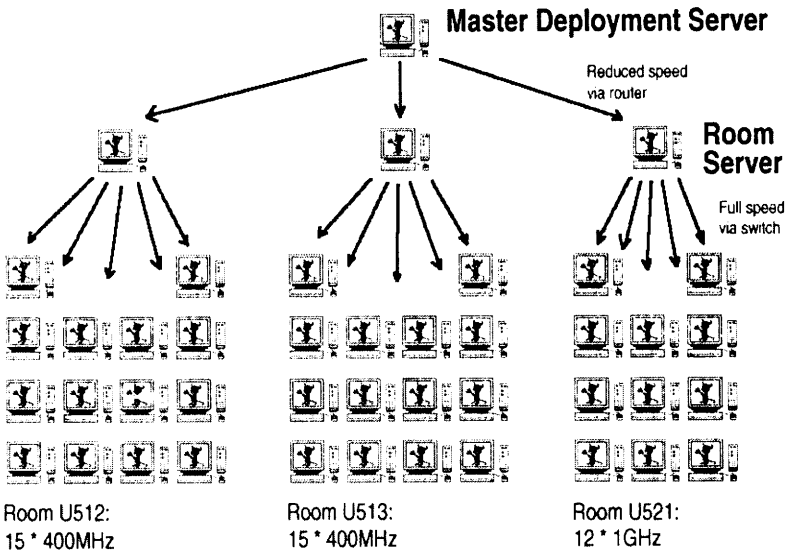
to install on the client machines and we had a lot of know-how for NetBSD available. The hardware consisted of Dell OptiPlex PCs with various configurations of RAM and CPU. In two of the rooms (U512, U513) were 15 machines each with PII-400MHz, 64 MB RAM and 4GB HD each, the other one (U521) had 12 machines with PIII-1GHz, 256MB RAM and 10GB HD each.

For the cluster setup, we installed one of the PII-400 machines with NetBSD and added all the necessary programs (dumpmpeg, mpeg_encode) from the NetBSD Packages Collection. We had a single account for the whole cluster project, which had it's home directory mounted from a NFS server and we also used NFS for temporary diskspace used during the calculations. The NFS server used was the Unix server of the computer science department, a Sun Ultra 10 with a 300MHz CPU, 1024MB RAM and 120GB harddisk (Arena hardware IDE-to-SCSI RAID).

For monitoring purposes we also installed an SNMP agent, the rstat service as well as the 'tload' program, which gives a console-based overview of the machine's load. Various programs needed by dumpmpeg and mpeg_encode completed the client installation.

After the cluster client was configured and running properly, the harddisk-image of the client machine was stored on the master deployment server of the FH Regensburg using the "g4u" harddisk image cloning software. The 4GB harddisk image was compressed to about 650MB in that process, deploying that 4GB image to both the 4GB as well as 10GB disks of the clients wasn't a problem with g4u.

For the deployment of the client, the image was first installed on one machine in each of the available rooms (using g4u again). After rebooting these machines and starting up NetBSD, the just-installed client machines were setup themselves as "room-server". For that, the client's harddisk image was put into a prepared FTP area, and after that, the remaining clients in that room were able to retrieve the image from their nearest room-server. As deployment from the master-server was via a router, deployment within the various rooms was much faster as the machines operated in a pure 100MBps switched network, with no slow router in between.

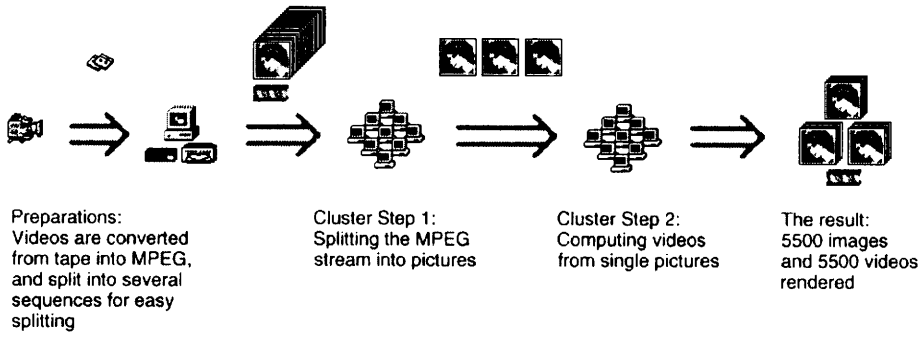


3 Computation performed by the cluster

The computation performed by the Regensburg Marathon Cluster consisted of two steps:

1. splitting the video sequences into single images
2. merging single images into individual video sequences

The following image gives an overview of the whole process:



3.1 Preparations

At the goal of the marathon, two video cameras placed by the local TV station TVA recorded the whole event of the runners reaching the goal. When the video tape of one camera reached the end after about 90 min, the second camera was switched on, with an overlap of about five minutes. That way, we got about five hours of video material, in Sony Betacam format.

Next, the four videotapes were converted into MPEG sequences using a Sony Betacam VCR unit as well as a PC running Windows 98 that had a Hauppage PVR card with hardware MPEG encoder. Each MPEG sequence consisted of 11 minutes of film material with 25 frames per second, a resolution of 352x288 pixel and a color depth of 24 bit, which led to about 110 MB per sequence. The sequences were then FTP'd to a PC that was connected via a crossed TP cable and that ran RedHat Linux 7.1. There, the sequences were archived to CD using mkisofs and cdrecord.

3.2 Step 1: Splitting the sequences

The fast 1GHz machines from the FH were used for splitting the MPEG sequences. After inserting a CD that contained a sequence, the Script "create_dir.pl" split it into single images and wrote them to the NFS server. Each of the 11 minutes long MPEG video sequences were split into about 16.500 single images in JPEG format by the program "dumpmpeg" in about 45 minutes. Each JPEG image was about 24KB in size, which lead to about 400MB of JPEG image data for each MPEG sequence.

The program "dumpmpeg" that was invoked by the "create_dir.pl" script was modified for the Marathon Cluster, as it was only able to write BMP images initially, which would have led to too big images and thus wasted storage space, plus the software used in the second step of the cluster required JPEG as input. Because of this, the source code of the open source program dumpmpeg was modified so that after saving 250 images (10 seconds of video) they were converted from BMP to JPEG using the netpbm tools. Conversion with netpbm tools is rather slow, but there were no alternatives available in the SDL- and smpeg-libraries used by dumpmpeg that allowed direct saving in JPEG format. Optimizing the many fork(2)- and exec(2)-calls away by using routines from the netpbm-tools was not possible due to a tight timeframe for the project.

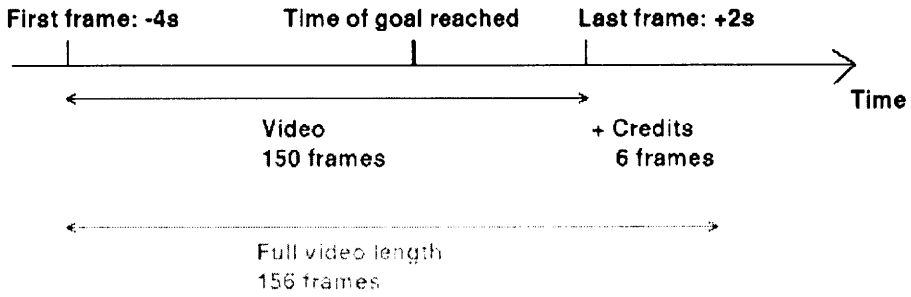
The second change that was made to dumpmpeg was the directory, in which the program created the images. Instead of placing all images into a single directory, 250 images were each placed in their own directory, improving access time.

3.3 Intermediate step: Enter image data into image database

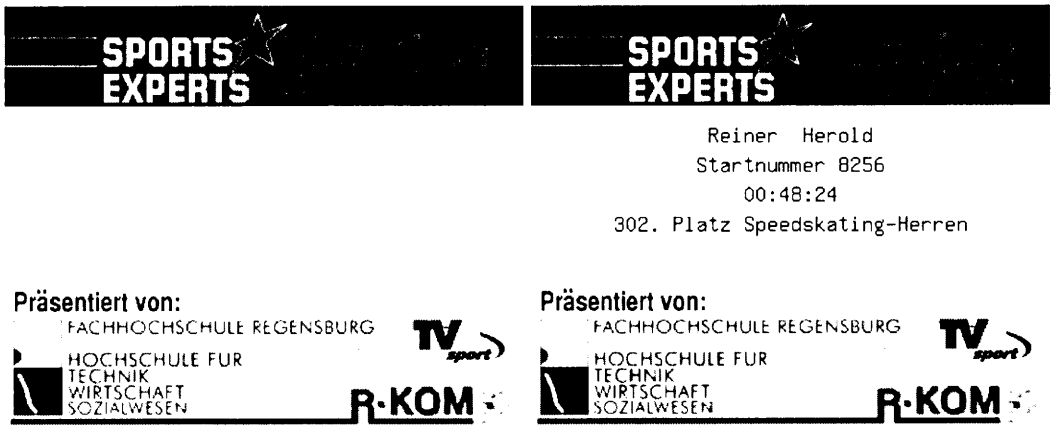
For the further processing in step 2, exact start- and end-time of each MPEG sequence was needed, as well as the corresponding filenames of the first and last JPEG image. These data were stored in a MySQL database on the job control machine. Furthermore the actual frame rate was calculated, as the VCR device didn't always provide a constant 25 images per second, e.g. due to heat and resulting mechanical inaccuracies. A tiny difference could increase massively over five hours of video material, leading to useless results when the exact images were needed for a certain time.

3.4 Step 2: Creating the videos

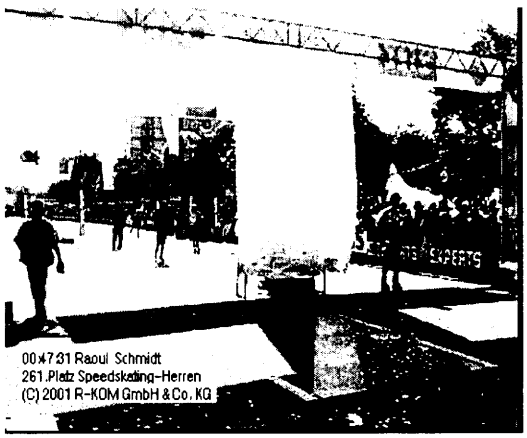
After the MPEG sequences were split into single images and stored on the NFS server in the first step, they were merged back into little videos in the second step. The goal was that each runner can watch a picture of themself reaching the goal giving their start number, and that each runner gets his individual MPEG video, rendered by the Marathon Cluster. Of the 7.000 runners who started the marathon, about 5.500 reached the goal. There were three disciplines: marathon (42km), half-marathon (21km) as well as speedskating (21km). For each discipline, separate lists of results for men and women existed, recording the time when they passed the goal. Based on this time and the corresponding image, we advance two seconds to the last image of the video, and from there we went back 150 frames that were copied into a work directory. These 150 frames are equal to six seconds of animation with 25 frames each.



Finally, six more frames were copied into the working directory that all had the same contents: Name of the runner, his start number, time, place as well as discipline were painted into a prepared mask displaying logo of the sponsors of the videos, using the program "convert" (part of ImageMagick). These six frames are then displayed at the end of the video:



Besides the images for the video showing the runner reaching the goal, the image of the exact time of him reaching the goal is also copied, and name of the runner, time, place and discipline are added via "convert":



Rendering of the video from the single images in the temporary working directory was done with the program "mpeg_encode". A config file described location of the working directory, which images to use for the video, and which client machines to use for rendering the video.

mpeg_encode first starts by calculating a few images on each of the client machines, to get an estimation on how fast the machines are. After that, the remaining images are distributed among the cluster nodes according to this speed estimate. The nodes read the images via NFS, and write rendered parts of the videos back via NFS, where the main process will pick them up and assemble them into the resulting MPEG file. All this is performed automatically by the freely available program without any need for manual interaction or tuning, which saves a lot of time and prevents possible errors.

The config file used by mpeg_encode was available for each of the four subclusters, so that we were able to render videos for each distinct discipline on a dedicated subcluster - depending on which one hat CPU time available. The job scheduling and distribution of discipline lists among the subclusters was done manually. There were six disciplines altogether:

Marathon women:	148 Finishers
Marathon men:	1268 Finishers
Halfmarathon women:	893 Finishers
Halfmarathon men:	2469 Finishers
Speedskating women:	202 Finisher
Speedskating men:	521 Finisher

The results was available in a separate list for each discipline in the form of a CSV file, which were split into ASCII files using a perl script. This was used as an input for another perl script "mpeg250_pic.pl". For each runner, the input file contained the name, place as well as the time of them reaching the goal. The image database described above (see 3.3) was used to calculate the sequence in which the runner was, and from that the corresponding exact image of the runner reaching the goal was calculated. As described above, this served as a reference point for the 150 frames used for the individual videos. After the images including the ones for the credits images were copied to the temporary working directory, the client machines were used to render the individual MPEG for one runner. This individual MPEG as well as the exact image of the runner reaching the goal were then archived to a separate directory.

The program mpeg_encode which is started via "mpeg250_pic.pl" then started the job on the various (sub)clusters via rsh. rsh is used to prevent expensive authentication as used by ssh. As the rendering of a single MPEG was between 3 and 8 seconds, the overhead of SSH authentication - which is about 2 seconds - would have been to big, without any gain at all.

Example videos can be found at

- http://www.feyrer.de/marathon-cluster/video-50.mpg
- http://www.feyrer.de/marathon-cluster/video-2.mpg

4 Experiences gained from the Marathon-Cluster

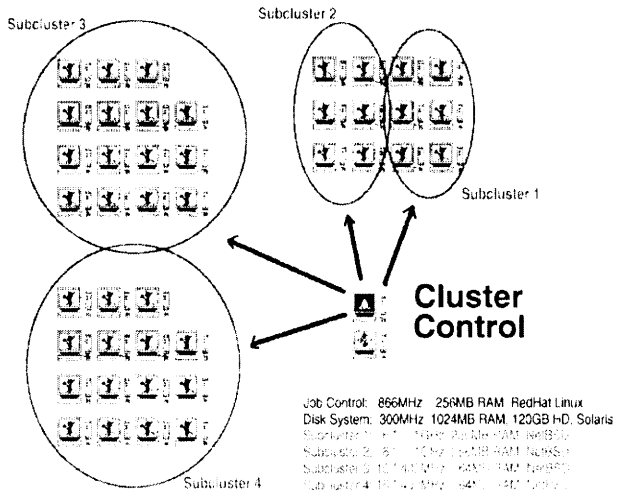
Deployment of the client machines took longer than expected. Installing a single client machine via g4u from the master server took about 30 minutes, copying the ca. 650MB

big image again after that (to prepare it as room server) took the same time. The following deployment to all machines from a single room took rather long, as 11 / 14 machines were fighting for bandwidth to the room server, plus all three rooms were connected via a single 100MBit 3Com switch, not one per room! A network design based on separate switches for each room would probably improved deployment times here.

dumpmpeg works on NetBSD and Linux, but not on Solaris/x86 using identical hardware. The cluster software was tested by R-KOM on Linux only, and it worked without problems there. On the Solaris based FH machines, dumpmpeg - which is based on SDL and smpeg - dumped core sporadically. Debugging with gdb showed that blocks of memory were overwritten, as the crashes happened in malloc(3). We guessed that the malloc(3) implementation by NetBSD and Linux are different than Solaris', so that possible overwritten memory blocks are treated less gracefully in the later case. In that case, the problem would have been located and fixed in the SDL- or smpeg-sources, but this would have taken more time than we had. The problem here was not Solaris itself, but the result was that we were not able to use the Solaris machines for the cluster, and we lost 15 valuable machines. With a bit more time for preparation and better tests, such bad effects can be avoided.

dumpmpeg ran longer then expected. The test sequence of 18 minutes length that was used for tuning and configuring the cluster took about 60 minutes to split on one of the 1GHz machines. Running the dumpmpeg process on several machines in parallel achieved big speed improvement, but there were also some shortages for network and disk resources. Especially reading and splitting the MPEG sequences was with eight hours about three hours longer than estimated.

mpeg_encode cannot render on an unlimited number of machines. A sequence of 156 images cannot be computed on more than about 15 machines. When trying to use more machines, errors are printed and the program hangs. To achieve maximum parallelism, the scripts that created the temporary working directories, filled them with images and ran mpeg_encode were changed to do so for different subclusters. The config files for mpeg_encode had to be adjusted to give work to several subclusters:

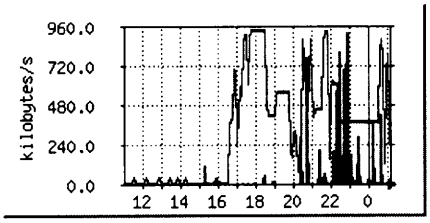


mpeg_encode sometimes stops after printing "Wrote 160 frames". There's no obvious

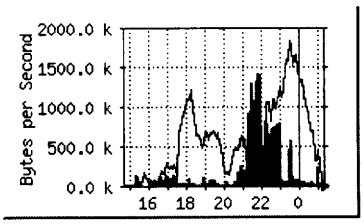
reason, and a quick view into the source code didn't lead to any enlightenment. Aborting the script was the fastest way of recovery here. The input list of runners to process just had to be edited so noone's processed multiple times.

5 Various images & graphs

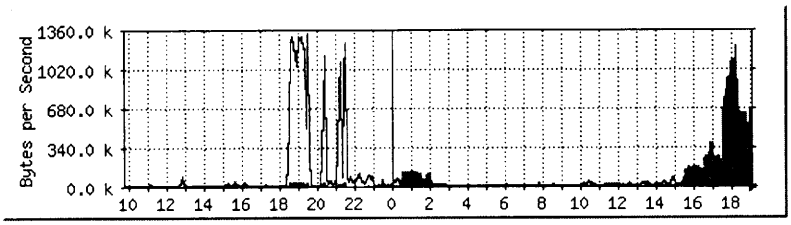
- Disk utilization of the NFS server shows that splitting of the sequences (blue) began at about 4:30pm, at 1:15am all sequences were split. Writes (green) show when the videos for the disciplines who went through the goal first - speedskaters - were rendered starting 10pm:



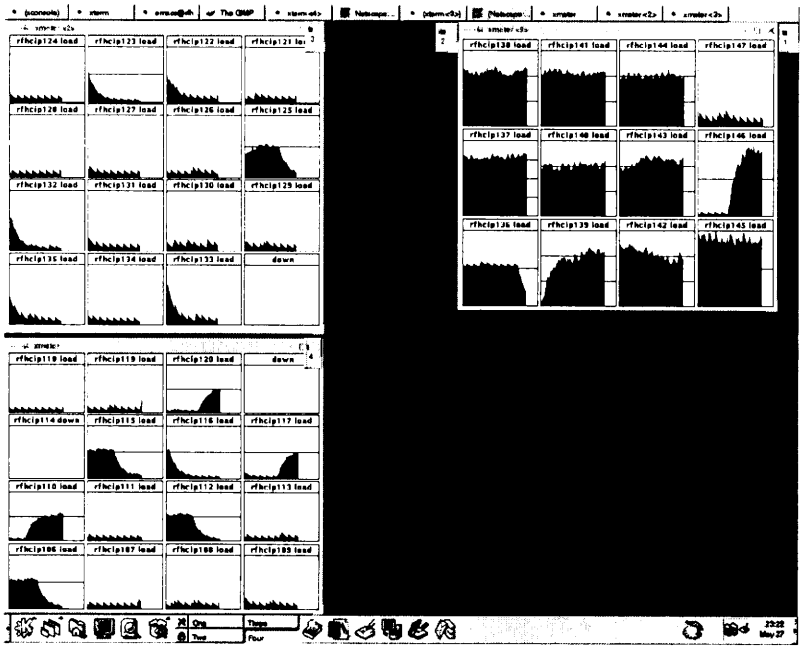
- Looking at the network traffic of the NFS server, it's very clear at which times the split images were written to disk (blue), and when they were read again for processing them for the speed skaters (green).



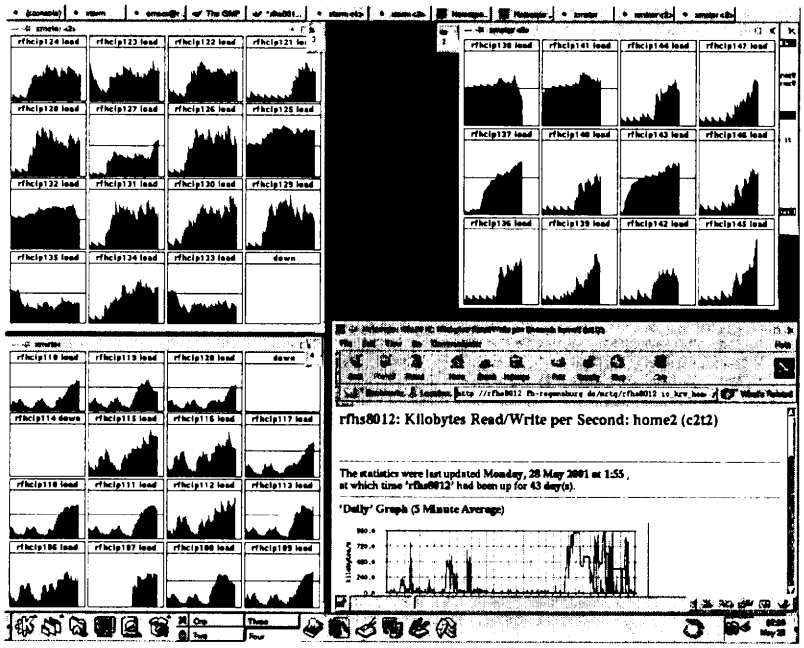
- Network traffic between the cluster machines and the control machine: while there were mostly read operations (blue) for the disk image during deployment of the clients on saturday between 18h and 22h, the cluster itself started producing data (green) on sunday at about 15h:



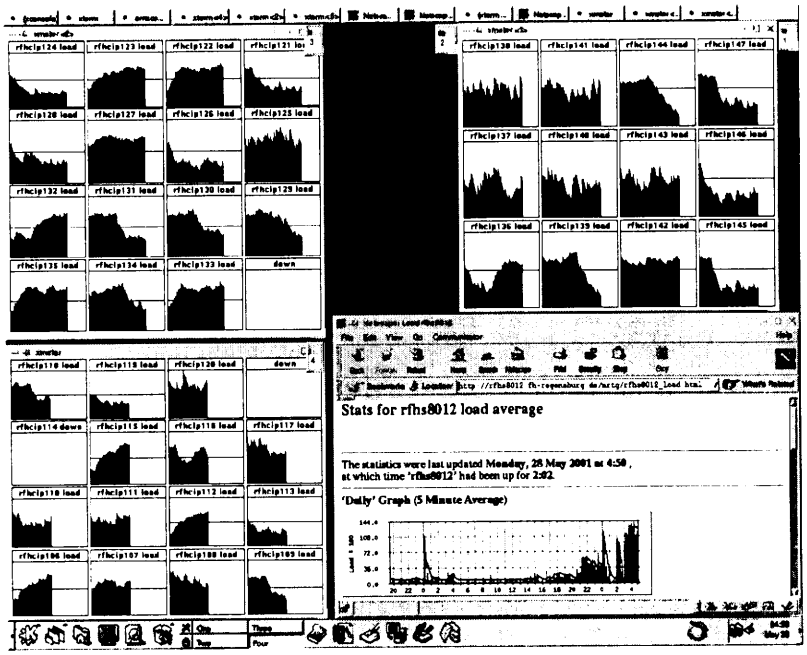
- Sunday, early afternoon: while the subcluster 3 and 4 (left side, yellow numbers) are still waiting for data, the GHz machines in subclusters 1 and 2 (upper right) are already splitting MPEG sequences into single images.



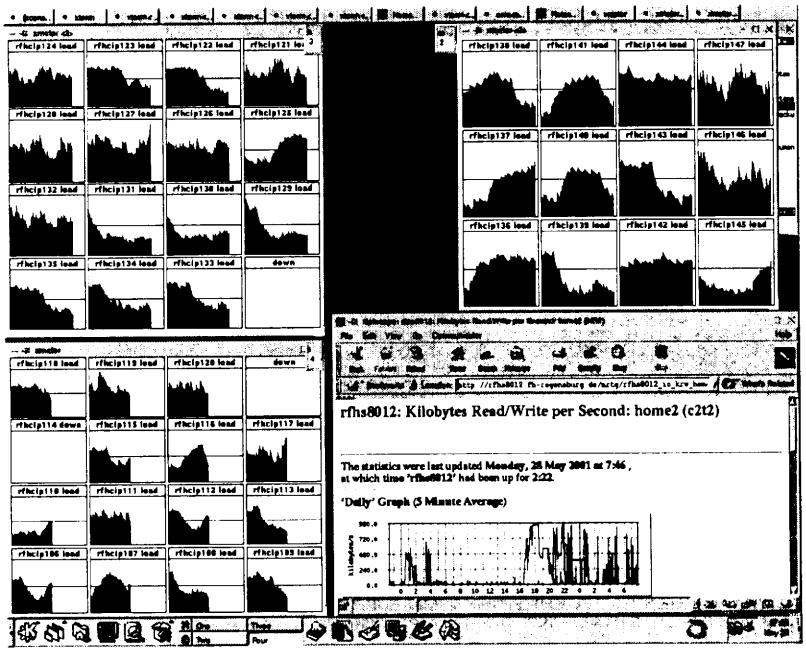
- Monday 2am: The last sequence is split, from now the second step can be run on all machines in all subclusters in parallel:



- Monday 5am: The NFS server running Solaris 2.6 had to be rebooted at 3am due to mysterious NFS/RPC/NIS problems. After that, the whole cluster is running with full steam again. Three hours from now the machines have to be available for students again!

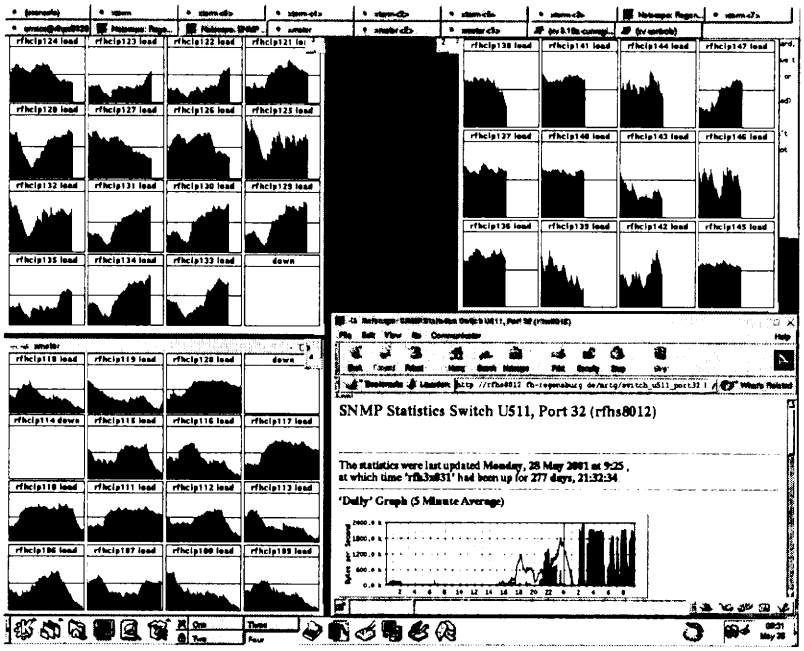


- Monday 5:30am: We just lost all the scripts by a lazily typed "`rm tempfiles *`" (note the second space!). A backup made the previous evening saved the project.



- Monday, 8am: In the past hours, several interrupts stopped the work up to an hour. Reasons were again mysterious phenomenons in the NFS/automounter area. (Or maybe booting network components - who knows?). Monday 8am there are still 900+500 halfmarathon- and about 500 marathon-videos to render, distributed over all subclusters. There are no lectures in the computer rooms until 11:30am, so we have a bit more time.

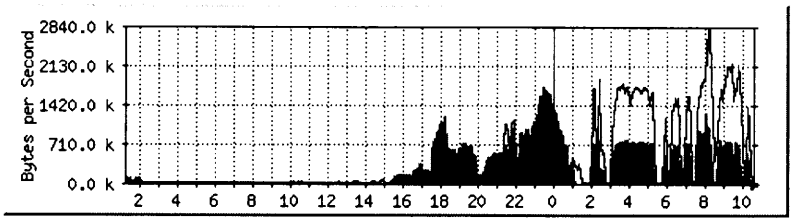
- Monday, 9:35am: After various NFS outages the clients run again with full load, the end is in sight!



- Monday, 10:40: Done! All result-lists are processed and spot checks on the resulting images and videos show good results.

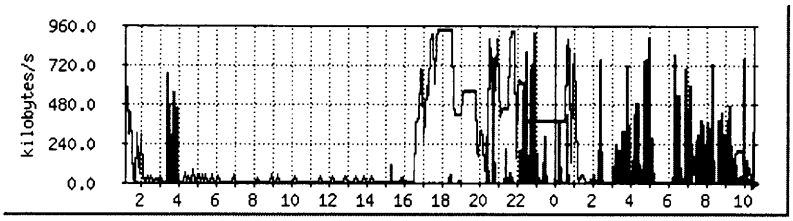
Here's an overview of all the performance graphs of the Regensburg Marathon Cluster:

- Traffic between control machine and cluster machines:



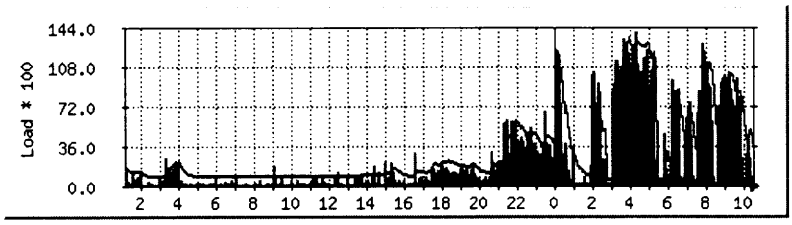
The two steps can be easily distinguished here: writing the images from about 5pm to 1am, and both writing and reading from about 2am until the end at 10:40am.

- Disk utilization of the NFS server:



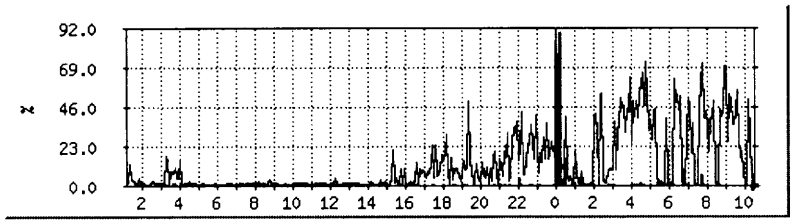
The difference between the first and second step is very obvious here. While there were only write operations while splitting the MPEGs into images (blue), they were read again for rendering the videos in the second step (green). While the first step was still running between 9pm and 11pm, the second step was running in parallel for rendering the speedskaters' videos.

- The system load (load average) of the NFS server:

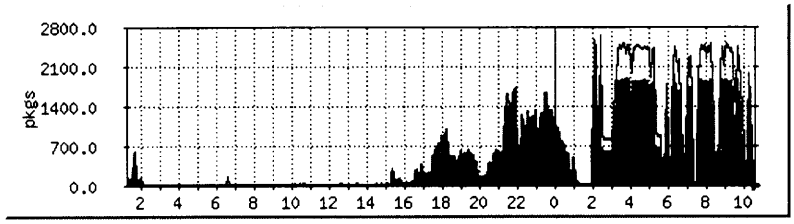


Step one (until about 2am) didn't put too much load on the NFS server, the high load at about midnight results from various cron job. Starting at about 2am the load increases due to parallel read- and write-jobs in the second computing-step of the cluster. Obvious are various interrupts at 3am, 5:30am and 8:30am.

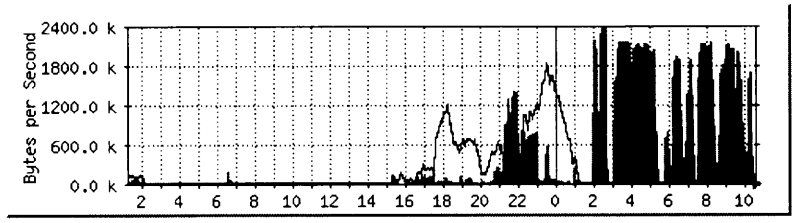
- The CPU utilization of the NFS server:



- Network utilization of the NFS server:



- Network traffic of the NFS server measured at the switch:



6 Facts

6.1 Subclusters

U512:	15	* 400MHz,	64MB RAM,	NetBSD 1.5.1_BETA2
U513:	15	* 400MHz,	64MB RAM,	NetBSD 1.5.1_BETA2
U521:	6	* 1000MHz,	256MB RAM,	NetBSD 1.5.1_BETA2
U521:	6	* 1000MHz,	256MB RAM,	NetBSD 1.5.1_BETA2

6.2 Cluster Control

noon:	1	* 866MHz,	256MB RAM,	RedHat 7.1
rfhs8012:	1	* 300MHz,	1024MB RAM,	Solaris 2.6, 120GB HD

6.3 Numbers

- Date of the Regensburg city marathon: Sunday May 27th 2001
- Female participants marathon: 148
- Male participants marathon: 1.268
- Female participants half-marathon: 893
- Male participants half-marathon: 2.469
- Female participants speedskating: 202
- Male participants speedskating: 521
- Participants overall: 5.501
- Available computers: 57, of which 15 were not usable (due to SDL/Solaris), and one had a broken floppy disk drive (=> no deployment possible)
- Computers participating in the cluster: 41
- Number of images after step #1: 669.936
- Number of reboots of the NFS server: 2
- Number of reboots per cluster client: 0
- Average size image of the runner reaching the goal (JPEG): 27 kB
- Average size video of the runner reaching the goal (MPEG): 987 kB
- Number of images of runners reaching the goal (JPEG): 5.501

- Number of videos of runners reaching the goal (MPEG): 5.501
- Time for deployment of the cluster: about 4 h (Sa 6pm to 10pm)
- Time for tuning and configuration of the cluster: about 4 h (Sat 10pm to Sun 2am)
- Time for setting up the VCR and other preparations: about 3.5 h (Sun 2pm to 5:30pm)
- Time for reading and splitting MPEG sequences (step 3): about 9 h (Sun 5:30pm to Mon 1am)
- Time for rendering videos and images: about 9 h (Mon 1am to 10:40am)
- Number of video tapes: 4
- Overall running time of video tapes: 5 h
- Number of MPEG sequences: 33
- Length per MPEG sequence: 11 min
- Uptime Hubert Feyrer: 22 h
- Uptime Jürgen Mayerhofer: 27 h

6.4 Software

- SDL 1.2.0, for dumpmpeg (http://www.libsdl.org/)
- smpeg 0.4.3, for dumpmpeg (http://www.lokigames.com/development/smjpeg.php3)
- dumpmpeg 0.6 (modified!): Converting MPEG->JPEGs (http://sourceforge.net/projects/dumpmpeg)
- netpbm 9.7, for dumpmpeg (http://netpbm.sourceforge.net/)
- mpeg_encode 1.5b: Converting JPEGs -> MPEG (http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_encode.html)
- perl 5.6.0: Scripting for job scheduling (http://www.perl.com/)
- gimp 1.2.1: Postprocessing of credits image, documentation (http://www.gimp.org/)
- Image Magick 5.2.7: Postprocessing of credits image and image of the runner reaching the goal (http://www.imagemagick.org/)
- tload 2.0.6 (modified): Monitoring cluster machines (ftp://people.redhat.com/johnsonm/procps/)
- xmeter 1.15: Monitoring cluster machines
- g4u 1.6: Client image deployment (http://www.feyrer.de/g4u/)
- NetBSD 1.5.1_BETA2/i386: Operating system of the cluster machines (http://www.netbsd.org/)

6.5 Participants

- Hubert Feyrer, Department of Computer Science , FH Regensburg:
  Design, implementation and execution of deployment of cluster clients, adaption of cluster software, documentation
- Jürgen Mayerhofer, R-KOM, Regensburg:
  Design and implementation of the cluster, coordination of videos and MPEG encoding
- Oliver Melzer, working student R-KOM & student FH Regensburg:
  Design and implementation of the cluster, coordination of videos and MPEG encoding
- Daniel Ettle, student of computer science (technical emphasis), FH Regensburg:
  Execution deployment
- Christian Krauss, student of computer science (technical emphasis), FH Regensburg:
  Execution deployment
- Tino Hirschmann, student of computer science (technical emphasis), FH Regensburg:
  Execution deployment
- Fabian Abke, student of computer science (technical emphasis), FH Regensburg:
  Execution deployment
- Udo Steinegger, Cable & Wireless, Munich:
  Team assistant

7 Links

- http://www.stadtmarathon-regensburg.de/:
  Web page of the Sports Experts Marathon
- http://www.stadtmarathon-regensburg.de/ergebnis_info.ph3:
  Results of the Sports Experts Marathon, including form for retrieving images and videos by starting number of the runner.
- http://www.r-kom.de/rkom-content-stadtmarathon2001.htm:
  Description of the intended project by R-KOM in the Marathon news paper
- http://www.feyrer.de/marathon-cluster/:
  Web site of this documentation about the actually performed project of the Regensburg Marathon Cluster.
- http://www.netbsd.org/:
  NetBSD, a free Unix/Linux-like operating system not only for PCs, which was used on the client machines of the marathon cluster.

Monitoring the world with NetBSD

Alan Horn
Inktomi Corp.
<ahorn@inktomi.com>

There are many papers and publications on monitoring your network and systems. Most of the time they fail to present a monitoring strategy that is both practical and general. Whilst I don't wish to restate what is already generally known, I do feel that most sites are lacking in a really solid monitoring framework.

At my current site and at previous locations, I have used NetBSD along with a range of free tools to build a customized system for proactive notifications and failure management. I would like to talk about the evolution of this system over the years, the decisions I took and why I took them, ending with a look at the current deployment and some thoughts for the future.

My aim is to try and present advice that folks can take back and drop into place at their sites with a very minimum implementation time and be getting a meaningful return very quickly. I want to distill a lot of different documentation into a set of notes that people can use almost as a set of design philisophies along with the practicum.

Alan currently works at Inktomi in their US headquarters as a Systems Architect. Prior to working in two different countries for Inktomi, he was a security consultant with Internet Security Systems, and their commercial deployment manager for EMEA. In a slightly earlier life he was a senior administrator at Dreamworks Feature Animation, where he was responsible for designing and building the production network, and maintaining systems security as well as daily senior admin type operations. Before that he mostly played with backup media :) Alan is an experienced technologist with a very broad range of knowledge and skills. His specialties include systems architecture and network design, security management and response and developing bespoke, robust code to support these endeavours.

Monitoring the World with NetBSD

Alan Horn
Inktomi Corp
Alan.Horn@inktomi.com

Jennifer Davis
California Institute of Technology
Jennifer.Davis@caltech.edu

Abstract

Through presenting a set of guidelines and freeware monitoring tools, we hope to prevent your enterprise from experiencing embarrassing and costly mistakes. Part of building any system, whether from a fresh start, or adding to a preexisting architecture, requires this kind of planning, although the depth depends on the complexity of your environment. This will help prevent system degradation and public embarrassment as well as improve perceived system performance.

1. Introduction

System environments are becoming more complex as customer needs and requirements increase. Keeping up with competitors in this negative market requires that money is not wasted on trivial matters. As budgets are tightened, and hiring additional IT techs is frozen, managing the growing spider web of systems and networks becomes more difficult. Time is consumed by tracking down problems, security patches, and basic system management. With an effective monitoring solution, administrators can free up their time for more important tasks.

1.1 Define the problem.

In the standard environment, monitoring comes last after purchasing adequate equipment, setting up the system and required services, and making the system live.

Sometimes, monitoring is never considered, as the overworked administrator is charged with all of the previous tasks, little resources to accomplish them, and the demand that it all be finished yesterday. Monitoring becomes important when the company financial controller (Mr. Smiley) realizes how much money the company has lost because of unforeseen outages in the last year. This is when the IT department is charged with setting up a low cost monitoring solution.

1.2 What is monitoring?

What do we mean by our desire to set up a monitoring solution? If we reference http://www.webster.com

v. mon·i·tored, mon·i·tor·ing, mon·i·tors [Latin, from monre, to warn]
v. tr.
1. To check the quality or content of (an electronic audio or visual signal) by means of a receiver.
2. To check by means of an electronic receiver for significant content, such as military, political, or illegal activity: monitor a suspected criminal's phone conversations.
3. To keep track of systematically with a view to collecting information: monitor the bear population of a national park; monitored the political views of the people.
4. To test or sample, especially on a regular or ongoing basis: monitored the city's drinking water for impurities.
5. To keep close watch over; supervise: monitor an examination.
6. To direct.

From this definition, we can define a comprehensive solution to our problem. Monitoring comprises regularly sampling some sort of content, systematically tracking the state of that content, and warning the appropriate parties when necessary.

1.3 What should we be monitoring?

Determining what should be monitored is a decision that should be made by analyzing individual environments. For critical out facing machines, monitoring the world may be the only solution. For stand-alone work machines, the system may not need to be monitored at all. The difference depends on what people consider important, why they consider it important, and who those people are.

The main types of monitoring are uptime or availability, performance, and security. The goals of monitoring are to make the job easier, more manageable, and efficient, and to fix problems before they are seen. If you work for a profit making company, ultimately you assist in 'increasing shareholder value'.

Monitoring is more than the world of bits and bytes. It can also involve the physical environment in which your systems live.

Monitoring should not replace redundancy of systems and services. Redundancy prevents complete outages allowing continued service, with possible degradation during a failure situation. Monitoring complements redundancy by alerting the IT department to fix the degraded state of services. For example, a RAID 5 disk array can suffer the loss of one disk, and still function. If the monitoring system alerts the system administrator immediately that a failure has occurred, then the disk can be replaced quickly without any loss of service or data on the array.

2.0 Selecting your tools.

The tools you select for monitoring should be dependable, stable, and consistent to the purpose to which you put them. As the rest of the company adopts the monitoring solution, the tools need to have a sufficiently rich set of features that provide flexibility to this expansion. The tools should be of a clean design, and readily understandable with a modicum of effort from others once your strategy has been implemented. For some, the tools must be quickly implemented as previous outages have made setting up a monitoring solution now a crisis.

2.1 One OS to rule them all

Although NetBSD is the authors' preferred OS, the techniques and applications discussed will work on other operating systems. The important factors for determination of OS are knowledge of OS, comfort levels at the basic levels of administration, integration into the existing environment, and standing political issues within your company.

There are several reasons to choose NetBSD over other operating systems. NetBSD provides a stable clean design and implementation and is well documented. It has a great network stack. It also has a comprehensive base Unix system with good analysis tools for simple monitoring, and readily available packages in the pkgsrc system. NetBSD runs on many platforms, as well as cheap commodity hardware. The final personal reason is personal comfort with the OS, and brand loyalty.

2.2 Pkgsrc system – an overview

Similar to the FreeBSD ports collection, the NetBSD pkgsrc system is a very good source of tools that is growing daily. Pkgsrc is not installed by default, but it is easily set up. Instructions for obtaining it are found in the references.

To see what packages are available in pkgsrc specifically for monitoring, check the net/, security/, and sysutils/ subdirectories.

```
$ cd /usr/pkgsrc/net
$ less */DESCR
```

This will show you the descriptions of every package within the net subsection.

To install a pkgsrc package (e.g. nocol) :

```
$ su
# cd /usr/pkgsrc/net/nocol
# make
# make install
```

Package binaries are typically installed in /usr/pkg/bin or /usr/pkg/sbin. Modify your $PATH environment variable as appropriate to include these new paths within your executable path.[1]

If you have problems when installing a package, contact the maintainer of the package. The person responsible for the maintenance of each individual package, is listed in the top-level Makefile in each pkgsrc package directory (e.g. /usr/pkgsrc/net/nocol/Makefile).

[1] Paths can be set on an individual per-user level, or you can modify /etc/csh.cshrc and /etc/profile to set the additional path for every user on the system.

3. Monitoring for availability

3.1 Definition

Availability is more than a system being up, and a service running. Availability means that individuals can get to what they need, when they need it, retrieve what it offers, and avail themselves of the service. For example, just because on start of Apache the "httpd started"message pops up, and the process logs show that http is up, and running, this does not indicate whether a user can actually reach a the document root, or any other page on the site. Another example, just because the Oracle ora_* processes are up and running, doesn't indicate whether that database is accepting connections and relaying the data needed.

3.2 Strategies

The best method of monitoring for availability, is to emulate as closely as possible the normal 'request operation'. A 'request operation' (RO) is how a user of the system, process on the system, or a process on another system accesses a specific service.

Examples of Access Tests for different services
web server - access the root document. (http://my.webserver.org)

name server – get the start of authority record (SOA) for a given zone, with additional tests confirming that the A records and PTR records for critical hosts are served correctly.

Critical web application – suite of tests, each element confirming a different stage or aspect of the web application is performing correctly.

Make informed decisions about the frequency of RO queries. By querying ROs too frequently, monitoring can induce additional load that degrades service performance. By not querying ROs enough, monitoring can miss outages that occur between your ROs. A good monitoring system will include tunable timing parameters such as these.

Depending on the tool chosen for each RO, tests are made on the exit status of the tool's run or a parse of the RO's output.

3.3 Tools

Monitoring focuses on the testing of a condition, and warning if that condition is not satisfied. One method of obtaining monitoring tools, is to subvert the function of a tool whose primary purpose is not monitoring.

Availability monitoring tools

ping – Ping is a basic function to test network connectivity

fping – Ping's big brother, fping will send ICMP connectivity checks to any number of hosts (specified on the command line or via an input file). It checks the hosts asynchronously with timeouts, meaning that it can be used in scripts very easily, and is generally a better choice than ping.

Snips (formerly known as nocol) – Snips is one of the 'uber-monitors'. It comes with a variety of testing tools to check against specific services. It has the ability to notify on alerts. It also has a simple control interface with four levels of alerting based on how often a check fails, with the alerts being configurable based on the levels. Snips contains built in monitors for:
ICMP ping
RPC portmapper
OSI ping
Ethernet load
TCP ports
Name server
Radius server
Syslog messages
Mailq
NTP
UPS (APC) battery
Unix host performance
BGP peers
SNMP variables
Data throughput

Nagios – Formerly known as netsaint, nagios is similar to snips in concept, but with additional features. Nagios has a very flexible alerting and notifications system, a nice GUI interface, and tactical display. It has a complete set of small binaries for standard service checks. Nagios has standard plugin check binaries for:
Dig disk space dns fping game hpjd http https ide-smart imap ldap load mrtg mrtgtraf mysql nagios nntp nt nwstat overcr pgsql ping pop procs radius real smtp snmp ssh swap tcp time udp ups users

vsz

There are also sample perl script checks for: breeze disk_smb flexlm ifoperstatus ifstatus ircd log netdns ntp oracle rpc sensors wave

Tkined – tkined is a TK application based on Scotty. It can be useful interactively for discovering networks, and live monitoring, but requires extensive customization to work in the background.

Lynx – Lynx is a simple text based web client. lynx-dump can grab raw documents in text for you to parse with additional code.

Wget – wget is a tool for grabbing both http and ftp files from servers.

ftp – The native NetBSD ftp client can be placed in non-interactive scripts to retrieve a particular dataset. The exit status of the command can be checked (0 means everything was successful).

Bigbrother – Bigbrother is another 'uber-monitor' similar to nagios. It's been around longer and has a far wider range of user-contributed plug-ins for monitoring lots of different services.

Bigsister – Bigsister is a clone of big brother developed to add different features, and also to improve performance by avoiding shell scripts.

Perl scripts using Net:: modules – With some creativity a perl script can be written to check for almost any network service. There are some limitations when testing crypto-based services.

Built-in software testing tools for a given device – For example on a Solaris system, A1000 raid array supporting software has a tool called 'healthck' which can be used to test for problems with the raid array. Other devices and packages will have tools that may be used similarly.

4. Monitoring for performance

4.1 Definition

Performance is harder to fully monitor as you may not have control over all machines involved in the transaction, or the network. To the user, acceptable performance generally means 'after I perform a specified action, the response occurs in a reasonable amount of time. For example, just because a web server's http process is showing up in process stats, this doesn't indicate anything about the fact that the server is taking 2 minutes to respond to each query. Another example, is email delivery. Just because sendmail is up and running, this doesn't give the administrator any idea if mail is getting processed, or whether time critical emails are getting delivered in time. Although the coverage from the system performing the actions to the server sitting in your data center may not be completely your responsibility, you can confirm that your responsibilities are functioning to their maximum performance.

4.2 Strategies

Along with ensuring the availability of a service, system, or network, there may be extended information you can or should gather about the performance of that availability. The information you should know or gather before setting up each individual monitor is the knowledge of the baseline performance i.e. normal operation of system or service. Also spend time predicting the failure modes and methods of degradation in performance for the object you are monitoring.

With this information, figure out the performance counters to monitor, and at what thresholds to alarm. You can choose to never alarm, and you will still have a body of historical information to analyze later.

At a minimum, any system providing a service should have the underlying system performance monitored; CPU load, disk space, I/O of all kinds. These basics will allow you to monitor system changes, which will provide data towards upgrading system resources, or the system itself as needed. Forewarned, you can kill out of control processes before they cause degraded performance. If you don't want to monitor these basic metrics, regularly take a few performance baselines so that in a crisis you have a record of what the system should look like. Individual services may also have metrics that you can monitor such as response time, resources used, etc.

System performance (and to a large extent service performance), cannot always easily be monitored across the network. SNMP is one popular solution, but it has drawbacks :

o Generally requires a complex daemon to run on a port.

o Susceptibleness to security issues[2].
o Overcomplication due to attempting to satisfy for all eventualities.
o The simple in SNMP (Simple Network Management Protocol) is not in reference to the configuration.

SNMP is probably the best choice for monitoring systems such as network switches and routers that do not have a feature rich operating system that you can access. SNMP is not the best choice for servers. Instead, use available local system tools, and either write a tight piece of code to send specific information back to the centralized monitoring host periodically or on demand, or avail yourself of a tool that performs this function.

Depending on the OS of the machine you are monitoring, different tools with varying ease of installation, and use will be available to you.

4.3 Tools

Sampling of tools for performance purposes
ps – With ps you can display process status on a host, and other useful information depending on the options available on your OS, such as cpu time, memory used by a process, owner of the process, etc.
df – df is helpful in determining filesystem usage.
uptime - system uptime and runq load
iostat – iostat reports information about I/O load.
vmstat - vmstat reports information about virtual memory. One of the best sources for determining what ails your server, cpu problems, excessive swapping, etc.
netstat – netstat provides you with various information about network stats. netstat –a provides information about all sockets. netstat –rn provides information about routing tables.
mrtg – mrtg is a simple tool for monitoring and graphing traffic load on network links
rrdtool – rrdtool is mrtg++, a round robin database tool developed based upon MRTGs graphing and logging features. Will display any time series. If you need to view data samples over time, this is the way to store it for analysis.
cricket – cricket is a rrdtool frontend focused on SNMP stats gathering for monitoring network hardware.
flowscan – flowscan is another rrdtool frontend using cflowd to gather Cisco flow data.
smokeping – smokeping is another rrdtool frontend that presents network latency in an MRTG style graph.
perl - Naturally, you can write your own tools in perl using the rich set of Net:: and other modules. Scripts can be written in shell, but you may also consider writing the glue scripts in perl too

[2] In the widely implemented SNMPv1, administrative relationships known as communities, are defined for SNMP entities. As long as you know the name of this community, you can access that particular SNMP community. This community name is used as a pseudo password to gain access to an SNMP device. To compound the problem, with every SNMP packet, the pseudo password is passed in clear text.

5. Monitoring for security

5.1 Definition

Beyond monitoring for availability and performance, you should monitor for security reasons. The material necessary to describe monitoring for security would provide the source for an entire paper by itself. If security was not included, our monitoring solution would not be complete. By venturing into the shallows of monitoring security, hopefully you will have some direction as to what depths you wish to further navigate.

When we monitor for security, we monitor for three things; Confidentiality, Integrity and Availability. The classic definitions are : [Ref : CISSP Definitions]

Confidentiality
Prevent the intentional or unintentional unauthorized disclosure of a message's contents.

Integrity
Prevent modifications to data by unauthorized personnel or processes, unauthorized modifications to data by authorized personnel or processes, and that the data is internally and externally consistent.

Availability
Ensure reliable and timely access to data or computing resources by the appropriate personnel.

For example, databases containing sensitive data like personnel salaries should be kept confidential, with maintained integrity, while still being available to those persons and processes that are required to create biweekly paychecks.

The depth of your security monitoring depends on the resources you have available to do the analysis. Quantitative security analysis involving estimates of yearly loss per vulnerability takes a long time and is costly. Qualitative analysis (per situation) is easier to perform.[3]

5.2 Strategies

As in the strategies with monitoring performance, determine the baseline of the system. Figure out what the operation norms are. In some respects security is easier since it is more rigid. If we need to ensure that certain files do not change, then we use a tripwire/md5 type solution and compare to our prepared table of hashes.

Some systems come with a predefined set of daily security tests (/etc/security.conf on *BSD systems), the output from these tests can be parsed and a flag raised if need be.

System logs can be analyzed, and patterns alerted on.

Places where changes are made to critical points in your security infrastructure (external access points, LDAP servers, NIS servers, Windows PDC) should be monitored closely as they affect your entire infrastructure if security fails.

Security requires a greater understanding to implement fully. Generally vendors will not share your security model, some unable to grasp that you would care about security at all which results in writing glue code to accomplish your security needs.

5.3 Tools

Sampling of tools for security purposes

nmap – nmap is a ubiquitous tool for scanning networks to see what ports are listening. It is very useful as an early stage in network mapping exercises, and can be used in a script to check for changes to the baseline. (e.g. if a new port suddenly starts listening, you can be notified)

nessus – A security audit tool, nessus performs a range of vulnerability tests against a host via the network.

md5 - MD5 is a hashing tool, producing a small digital fingerprint for any given file. By storing a list of hashes for critical files, you can see when a file has changed unexpectedly.

swatch - 'Simple log watcher', swatch watches syslog files, scans for patterns, and then runs an alert shell script when a match occurs.

/etc/security - *BSD security check shellscript with output that can be parsed and flagged.

portsentry – portsentry binds to a specified set of ports on a host, waits for connection attempts to those ports, and then performs a set of actions (e.g. drop the IP via a null host route, log the connection attempt to syslog which can then be picked up and acted upon by swatch).

logsentry – Similar to swatch, logsentry has more features and understands common log patterns by default. It is also quicker to deploy.

snort – Snort is a packet signature analyzer. It watches packets arriving at a host (or network port if used as a perimeter sniffer), compares those packets to a list of signatures, profiles potential attacks and reports on them.

tcpwrappers – tcpwrappers will monitor connections to daemons (typically started from inetd.conf) and log to syslog. Swatch may be used to alert.

ipfilter/ipchains/ipfw - At a very low level, packets that match specific rules can be logged and/or counted using one of these tools. Counts can usually be displayed with an 'ipfstat' type tool, logs
with an 'ipmon' type tool.

perl - Perl can be the glue code to hold the various other tools together, or to create a new specialised tool.

6. Monitoring setup and configuration

6.1 Centralized monitoring host

Previously, we have mentioned the centralized monitoring host when discussing the tools available for monitoring without explaining what a centralized monitoring host is. The centralized monitoring host , which we will name central, is the system that ties together all of the individual

[3] Security Risk Analysis
http://www.security-risk-analysis.com/introduction.htm

monitoring tools. It should provide the following features :

o Use stable, predictable code. This machine will be left unattended for weeks or even months at a time. You need to be sure that it will still be working as expected when you return
o Implement redundancy features with failover.
o Have configurable alerting of individuals and groups based on different criteria. (time-based, threshold-based)
o Have a variety of standard testing tools for standard services
o Accept input in a standard form from third party software/tools
o Be intelligent about alerting, having the ability to distinguish between a host/service outage, and a network outage.
o Be well written and try to conform to the Unix philosophy of 'small tools to do simple tasks very well'
o Store historical logging of check data for future analysis

In addition, the following features although not required would be optimal:

o Be easy to configure, or have tools to assist in configuration
o Have a nice web display for the command-line challenged (GUIs have their place, this may be one of them)
o Have a nice configurable web display for the company officials to see graphically how the money spent on this monitoring solution is actually saving money.
o A historical reporting function

6.2 The 'uber-monitor'

Quite a few 'uber-monitor' tools have been written, although they are not all currently maintained, or updated. Each has strengths and weaknesses that may determine whether you use one or the other of them in your operating environment. The focus of this paper will be on nagios, as we find it to be the most feature full on average for our needs. Nagios has drawbacks in that it is not very easy to configure, but that problem is actively being worked upon. It is sufficiently lightweight to flex into different designs, yet having a feature set that makes it an asset at the same time.

The principles described should translate to other software tools as well. Determination of tools is a matter of personal choice as stated at the beginning of the paper.

Some alternatives to look at include :
(as mentioned on the Nagios website at http://nagios.sourceforge.net/docs/1_0/about.html#othermonitors)

Angel Network Monitor
Autostatus
Big Brother
HiWAyS
MARS
Mon
Netup (French)
NocMonitor
NodeWatch
Penemo
PIKT
RITW
Scotty/TKined
Spong
Sysmon

6.3.0 An Overview of Nagios

Nagios is primarily a notification tool. By itself it does not do any actual monitoring. Nagios comes with a suite of plugin binaries and scripts to do the monitoring. This means that nagios can focus on the specific task of notification.

Nagios uses basic-auth mechanisms to determine who is allowed to access a given feature. It has several levels of access control within the application, allowing you to define who can view certain info, or modify certain info, as well as other controls. By default owners can only view the service/host checks that they would normally be notified about.

Nagios requires a web server to be installed if you want to run and administer it via the web interface. We favor apache, although any web server that allows CGI should work fine.

Because the basic auth involves sending passwords via the web, it is recommended that you use SSL (https) for the web server and not plain http.

6.3.1 Installation of nagios

Nagios is not yet in pkgsrc. Once this happens the installation becomes considerably less involved.

Install nagios.
1. Create a nagios user and group (both named 'nagios')

2. Install gmake from pkgsrc

```
cd /usr/pkgsrc/devel/gmake
make install
```

3. Download nagios.
http://www.nagios.org/download/

4. Unpack the download file

```
gzcat nagios-1.0b6.tar.gz | tar vfx -
```

5. Configure and make nagios

```
cd nagios-1.0b6
./configure
gmake all
gmake install  * installs into /usr/local/nagios
gmake install-init  * installs init script into
/etc/rc.d
gmake install-commandmode * installs and sets
permissions on external commands file
gmake install-config  *installs *SAMPLE*
config files into /usr/local/nagios/etc
```

6. Download nagios plugins.
http://www.nagios.org/download/

7. Unpack the download file.

```
gzcat nagiosplug-1.3-beta1.tar.gz | tar vfx -
```

8. Configure and make the plugins.

```
cd nagiosplug-1.3-beta1
./configure
gmake
gmake install  *installs plugins into
/usr/local/nagios/libexec
```

9. Configure your web server as per the nagios documentation. This will vary depending on your existing web server installation. Make sure you have an SSL enabled web server. Mod_ssl works great with Apache, is easy to set up, and has a utility for creating your own certificates.

6.3.2 Editing the Nagios configuration files

You can put all of your nagios configurations into one big file. In practice it is easier to separate different configuration groups into separate files. The main nagios.conf file contains include statements to inherit the other files. This file is the one you specify from the command line when starting nagios.

Although nagios comes with a set of well commented sample files, configuration files are still somewhat bewildering to anyone who has not used the software before. The most common basic actions are shown in the examples below. Following these examples through should give you a minimal working nagios installation monitoring one service on one host.

Set up nagios.cfg (using csh syntax).

1. Make copies of the sample files.

```
cd /usr/local/nagios/etc
foreach i (*-sample)
cp $i ${i:r}.cfg
end
```

2. You do not need to change most of the settings, but you should probably set the values of admin_email and admin_pager in nagios.cfg.

Add a host check.

1. Edit hosts.cfg, by removing everything below the 'generic host template'
2. Add a host check for the host 'foo' by inserting these lines at the end of the file:

```
define host{
    use             generic-host   ; Name of host
template to use

    host_name           foo
    alias             foo server
    address             w.x.y.z
    check_command         check-host-alive
    max_check_attempts     10
    notification_interval  120
    notification_period    24x7
    notification_options   d,u,r
    }
```

(NOTE - w.x.y.z is the machines actual IP address, substitue accordingly)

Replace foo with a name of one of your mailservers that runs SMTP.

3. Save the file.

Add a service check.

1. To monitor sendmail/smtp daemon on "foo", edit services.cfg, delete every entry except the one that looks like :

```
# Service definition
define service{
    use                     generic-service    ; Name
of service template to use
    host_name               novell1
    service_description     SMTP
    is_volatile             0
    check_period            24x7
    max_check_attempts      3
    normal_check_interval   3
    retry_check_interval    1
    contact_groups          novell-admins
    notification_interval   120
    notification_period     24x7
    notification_options    w,u,c,r
    check_command           check_smtp
    }
```

Replace host_name novell1 with host_name foo

Notice the two lines contact_groups and check_command, we'll deal with more with those in the next examples.

Configure contacts

If you want nagios to contact you, you need to tell it whom to contact. Let's configure a contact group and the contacts in that group.

1. Edit contacts.cfg.
2. Change the nagios admin definition to have the right email and pager addresses for the person or group that maintains the nagios host.
3. Replace the 'John Doe' example with a real person (probably yourself if you're just starting out).
4. Add any additional contacts in as new records similar to the John Doe example.
5. Save that, and edit contactgroups.cfg
6. Remove every record, except for the 'novell-admins' record (since that is the name listed in our previous service definition, you could of course change it to something else as long as they both match).
7. In the members field, add all of your contacts that you want to be notified for outages of this service, comma separated list.

Configure hostgroups

Hostgroups are a way of grouping functionally similar hosts into groups, for example DNS servers,
mail servers, etc.

1. Edit hostsgroups.cfg
2. Replace the record that reads :

```
# 'novell-servers' host group definition
define hostgroup{
    hostgroup_name  novell-servers
    alias           Novell Servers
    contact_groups  novell-admins
    members         novell1,novell2
    }
```

With :

```
# 'mailserver' host group definition
define hostgroup{
    hostgroup_name  mailservers
    alias           Mail Servers
    contact_groups  novell-admins
    members         foo
    }
```

Edit dependencies

Dependencies are an advanced feature of nagios that allow you to define services and/or hosts as being dependent on other services and/or hosts. This reduces the noise when a given service fails, since monitoring and notifications on the dependent services can be silenced automatically.

At this stage, we will not use this, but the example file comes with some dependencies defined.

1. Edit dependencies.cfg, and comment out every entry by making sure that every line starts with a #.

Edit Escalations

This is another advanced feature that we do not wish to define yet. Basically this is the feature that makes managers very happy, allowing very precise definitions of escalation coverage, at what point different groups get notified about a failure etc... embedding service level agreements in the monitoring.

For now, we shall comment this out as we did with dependencies.

At this point, we can check nagios operation by running /etc/rc.d/nagios reload. You should see something like :

```
gilgamesh# /etc/rc.d/nagios reload
Running configuration check...done
Starting network monitor: nagios
  PID TT STAT    TIME COMMAND
20095 ?? Ss   0:00.02
/usr/local/nagios/bin/nagios -d
/usr/local/nagios/etc/nag
gilgamesh#
```

You can check that things are operating normally without going to the web, simply by looking directly in the status file in /usr/local/nagios/var/status.log (or wherever you have installed in nagios).

Configuring Nagios Remote Plugin Executor

Earlier an alternative to SNMP for monitoring system counters via the network was mentioned. Nrpe is the tool for doing that. The nrpe has two components. One is a small daemon that sits on the monitored host, one is an active client check that runs from central. The daemon has a config file which defines basic IP based access control (which host can get to the daemon and ask it for information, typically only central is defined, perhaps one other IP for redundancy), and then you define a list of strings that you associate with a system command.

Download nrpe.
http://www.nagios.org/download

The build process is in two parts, first compile the check_nrpe client on central.

Download, untar and compile the source on central

```
./configure
make all
cd src
```

Configure nagios commands.cfg to know about check_nrpe

Add the following into commands.cfg :

```
define command {
      command_name    check_remote
      command_line $USER1$/check_nrpe
$ARG1$ -c $ARG2$
}
```

$ARG1$ will be the hostname argument in services.cfg, $ARG2$ will be the name of the check passed to the nrpe daemon on the monitored host.

Compile the nrpe daemon on the remote host (may be a different OS), or copy the compile from central if it's the same OS. I find it easiest to install into /usr/local/nagios on the remote host as well.

/usr/local/nagios/bin/nrpe
/usr/local/nagios/etc/nrpe.cfg
/usr/local/nagios/libexec/check_foo (the plugins you need to use to run the check, either home written in perl/sh, or compiled from nagios plugins)

Define an nrpe.cfg on the remote host, containing the checks you want to execute.

```
# IP address of the monitoring host (hideout)
allowed_hosts=w.x.y.z
#
# list of commands that may be executed locally
#
command[woprdisk]=cd /net/wopr; cd /;
/usr/local/nagios/libexec/check_disk -w 2%
 -c 1% -p /net/wopr
```

(in this case we're using an NFS mount to check how full a NetAPP filesystem has become, it will report a warning at 98% full, and a critical at 99% full. These values were chosen based on the total size of filesystem (> 1TB), typically you would warn earlier on smaller filesystems.)

So say we put an entry like :
command[rootdisk]=/usr/local/nagios/libexec/chec
k_disk -w 10% -c 5% -p /

Startup nrpe on the monitored host, either from inetd, or via an rc script.

Going back to central, define a service check thus in services.cfg :

```
define service {
        host_name               foo
        service_description     foo root disk usage
        check_command
check_remote!foo!rootdisk
        is_volatile             0
        check_period            24x7
        max_check_attempts      3
        normal_check_interval   3
        retry_check_interval    1
        contact_groups          novell-admins
        notification_interval   120
        notification_period     24x7
        notification_options    w,u,c,r
        }
```

Restart nagios with /etc/rc.d/nagios reload. Now NRPE will check root disk status on foo periodically.

6.3.3 Additional Nagios tools

Nagios Administration Tool (NAGAT) - A web based solution written in PHP for configuring nagios host,service checks etc..

Nagios Service Check Acceptor (NSCA) 2.1 - A two part client/server tool (similar to NRPE) used in the other direction, allowing remote clients to submit asynchronous events (such as security alerts) to a daemon listening on central. The daemon then pushes those events into nagios as a PASSIVE check.

nagios_statd - Perl/Python plugin that lets you check remote host information such as load, users, filesystems etc.

NTray 0.91 - Handy NT app that sits in your system tray and retrieves info from the nagios status file and gives red green lights for you to watch.

Remote Execution Layer (REL)- A layer for providing alternate transport between client and server for NRPE and NCSA. This gets around modifying firewalls to work with nagios. Currently it sends results via email into nagios.

remote_ctl - Perl CGI for easily enabling/disabling service checks remotely using wget or a web browser.

6.3.4 Nagios gotchas

Process space

Nagios can fire off enough checks at one time on the central host that it runs into per-user process limits. Always increase the process limit for the nagios users as you increase your use of the application.

Always do restart

When modifying nagios config files, always use RCS as a matter of course, but never stop and start the daemon, always use /etc/rc.d/nagios restart. The reason for this is the restart option will cause a configtest
against the files before any action is taken. This means that even if you have caused an error in your config files, the running daemon will not go away, and you will get the chance to fix your config files at your leisure without interrupting service.

Use dependencies and parents

Parents are how nagios attempts to model the network and distinguish between network and host/service outages. For every host you define, you should define a parent as the first hop on a traceroute from that host to central. This way, if a network fails and you can't get to a critical server subnet from central, you won't get a number of pages about the hosts, since nagios will know that it's a network outage.

Similarly for dependencies, dependencies actually allow you to monitor more complex systems and reduce the number of 'noise' notifications you get when a failure occurs.

6.4 Monitoring the monitor.

Quis custodiet ipsos custodes - The watched shall watch the watchers

If central fails for whatever reason, you have lost the ability to monitor everything. Unless you are also monitoring central itself, you will never know this as a failure in your monitoring solution as the failure mode is no alerts which is the same as if everything is running smoothly.

A second box should be setup (monitored by central, naturally) whose purpose is to monitor central. It needs to have at least the ability to

inform someone if/when central fails. It should probably complain very loudly when this occurs. Pager notifications to everyone are appropriate in this situation.

Generally the two boxes will not fail at exactly the same time, so you will get some sort of notification about a failure.

6.5 Notifications

You will need two methods of sending notifications. The general day to day method is alerts via emails to email accounts, or pager/sms gateways. The second method should be 'out of band'. Out of band means a method that is not in
any way subject to the same set of failure modes as the original method. If your email server fails, you want to be able to get notifications, but if you rely solely on email to receive your alerts, you will never receive the alert.

Now, you can of course take this requirement to extremes, but generally speaking, it's acceptable to have a separately powered machine, connected
via a phone line that doesn't go through your main company exchange. This machine will be used to send notifications (to a pager usually), via a dialup connection to the pager provider, or to an ISP somewhere.

Again the key here is not to be totally redundant, but to have enough redundancy to be able to see the problems as they occur and be able to react in a timely manner.

7. Monitoring Overlap

As you can see, there is a lot of overlap between the different monitoring categories. Separation of categories is provided to make it easier to understand the different components required in monitoring your environment.

By designing your monitoring tests to complement one another it's possible to make early informed judgments about where problems lie. Try to emulate the sort of questions you would ask in debugging a problem. By understanding what a series of failures occurring at the same time really means, you can jump straight to the underlying problem and fix it more quickly.

8. The physical world

The worst failures often arise outside of the computer hardware or software. Failure of power, UPS, or air conditioning can be catastrophic. A/C failure can go unnoticed for several hours on a weekend, driving up data center temperatures into triple digits creating lumps of metals out of your expensive critical machines.

It is a simple matter to build (or buy) temperature probes which can then report back to the central host. Place several of these around your data center and you then have a handy temperature monitoring system. Graph it with rrdtool over time and you can see if there is a gradual increase and if you have to respecify the A/C coverage. You may also be able to take advantage of the built-in temperature probes in (for example) some Cisco hardware. See references for websites that either sell or have plans for this.

Good quality UPS systems will often have a serial connection which changes state when the UPS operates. This can be monitored, or have an action associated with it (for example, shutting down critical servers gracefully in the time left with power, to save data).

Electronic entry systems generally come with their own software, but if you choose to roll this into your monitoring system it can be done with a little creativity. For example, if you really need to be notified whenever someone enters a particular location for example, or between certain hours. Your imagination is the key.

9.0 Implementation

9.1Case Study I

Divisional monitoring

This system was setup to augment existing monitoring systems in the division, various scripts using OS native tools like ping, and df. It was originally intended as an interesting experiment and cool hobby for the designer, something to play and learn with. Although more of a personal monitoring tool than a production monitoring solution, it did get used by other members of the team. It monitored Irix, and Solaris systems as well as Fore systems network equipment.

Hardware

Single old Pentium 133Mhz box (lethe), running NetBSD1.3, installed with nocol from pkgsrc.

Machine was located on the same subnet and network media as most of the services it was to monitor. Single network interface (10bT).

Relatively new to NetBSD at the time, I found it a much better operating system to work with, than the other operating systems in use at the organization. Irix was going through a period of flux in its OS development, 32bit versus 64bit, n32 versus o32. Compiling was a nightmare sometimes, especially with Open Source software. Solaris was clunky. Nocol was chosen, as it best met needs at the time from the packages that were researched online, and from pkgsrc.

Implementation

It took a few weeks to configure, as this was a first attempt at this sort of service. Approximately 40 Unix servers were monitored with 'rpcpingmon', and 15 managed switches and routers were monitered with ippingmon. Critical servers' performance statistics were monitored more closely with 'hostmon'. Some experimentation with syslogmon occured, but at the time, a separate system using swatch by itself was in use.

Notifications were via email and pager, and also via a console ('netconsole') running on the network administrator's workstation.

After setup, I examined the system every time new systems were installed into the environment. I also reevaluated the configuration every few months to fine tune the system.

The system scaled reasonably well, although some slowdown was noted on lethe as more checks were added. There was no provision in the system to understand network outages, and no redundant notification paths. Given the environment this was acceptable however, since most of the admins were frequently online and watching stuff anyway.

9.2 Case Study II

Enterprise wide monitoring setup

This system was setup from scratch in a place with no monitoring. It was intended to completely monitor all critical services within the enterprise. At this time it is still evolving

Hardware

Three desktop PCs (Pentium II-400, 128MB RAM, 4 or 9GB hard drives), each with a vanilla NetBSD installation (1.5.x).

PC1 (hideout) - The main nagios host, some sample configuration files shown in Appendix B.

PC2 (pageboy) - The phone dialer. Disconnected completely from the network for security reasons (the dialing process will bring up an IP stack), and connected via serial cables to PC1 and PC3. Communications via UUCP. This is the host that will send out pages via an external service. The connected analogue phone line doesn't go through the main building phone switch.

PC3 (hwatch) - Hideout's watcher. This is the second monitor host whose only purpose is to monitor PC1 and scream if it goes down. Always sends pages via pageboy. Arguably this function could be rolled into pageboy.

With many years of NetBSD experience now, NetBSD was the obvious choice because it's great and has never failed at any of the tasks I've attempted with it. I'm very comfortable with the OS. Nagios was chosen, as it seems to be the best of the monitoring packages available currently. that.

Implementation

Setting up the initial install of nagios took an afternoon. The longest part of configuring nagios, was gathering information from other groups to make it useful to them. The basic system monitoring was up within hours, and configured more completely in about 3 weeks. The first couple of days, I configured nagios to monitor the infrastructure (DNS, mail, NIS, LDAP, etc..), the rest of the time was adding in hosts that others needed, using nrpe to gather system specific statistics.

The system currently monitors 72 hosts and 126 services. No noticeable performance impact has been seen on hideout, notifications (based on time tuning) are generally immediate, and problems are noted and worked upon before anyone in the user community sees them

Since setup, I regularly check the system weekly to make changes. It is gradually getting into people's awareness as something to consider as part of a deployment of new services.

I've written specialized nagios plugin client code, to check our high uptime web content.

It has assisted our organization by allowing us to respond to failures before they become catastrophic, or impacting business profits. Our group has a better reputation, with more personnel having faith in the IT department's ability to prevent and handle emergencies.

Setting up a similar system

Generally there are very few limitations unless you're really deploying the system worldwide, at which point your better option is to scale hierarchically and have slave nodes reporting back to a master reporting station. Current desktops have more than enough resources in every area to perform as a central monitoring host. The current enterprise setup is not suffering under its current load. Figuring out the ideal system really depends on how far you want to scale.

If you need to buy a system, a mid range rack mount or desktop will be fine. A laptop will work in a pinch. High disk I/O isn't absolutely necessary (although it might make response time better when you need to maintain the system).

My ideal setup would probably be an older model athlon on a stable chipset board with 512MB ram and 18GB mirrored disk. That would be ample for a standalone station.

If you are at the point of needing slave nodes, then use a similar setup, using a current model athlon, bigger disks with hardware raid, and gigE connections for the master node.

Future work

When the world of the dot-coms has money once again it will be nice to build a monitoring solution with more powerful machines, along with the usual hardware redundancy features. In choosing the hardware (they are currently compaq PCs...) we will be more easily able to raidframe disk mirroring etc...

Conclusions

We defined monitoring as a sampling of some sort of content, systematically tracking the state of that content, and warning the appropriate parties when needed. By investing time and resources on preparing a monitoring solution at the outset of enterprise architecture, catastrophe will be adverted when AC in your data center fails, the company's website becomes unreachable or a distraught recently fired HR employee attempts to trash the systems. A monitoring solution does not replace the need for redundancy in your systems, or having reliable backups. As system environments become more complex, monitoring becomes more important. With the lag in resources for increasing manpower versus the need for more systems to handle load, a good monitoring solution is the only way to keep on top of system performance. Running NetBSD, one of the most stable and secure operating systems available, with the very configurable nagios, you will have built a monitoring solution that will withstand most system and network administrator's nightmares. Instead of Mr. Smiley calling you up questioning the cost benefits of your setup, and monetary losses of downtime, you will be prepared with information to backup decisions on buying system/network resources, and be prepared when emergency strikes to minimize downtime.

The intent of this paper was to impart design skills to help you enhance your own monitoring solutions, and get you started with a basic monitoring framework if monitoring is a new topic for you. The authors are happy to field questions via email, and contract work is always welcome.

References

Online copies of this paper, sample perl tools source code and sample Nagios configuration files may be obtained from http://www.deorth.org/papers/monitoring

Mr Smiley http://www.userfriendly.org

Obtaining pkgsrc http://www.netbsd.org/Documentation/software/packages.html

Fping
cd /usr/pkgsrc/net/fping && make install.
Or, sources can be obtained from
ftp://ftp.uu.net/usenet/comp.sources.unix/volume26/fping/

SNIPS http://www.netplex-tech.com/software/snips/
Nagios and associated plugins and contributions http://www.nagios.org
TKined (and in kpgsrc) http://wwwhome.cs.utwente.nl/~schoenw/scotty/
Lynx (and in kpgsrc) http://lynx.browser.org/
Wget (and in pkgsrc) http://www.gnu.org/software/wget/wget.html
Big brother http://www.bb4.com
Big sister http://bigsister.graeff.com/
MRTG (and in kpgsrc) http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool
Cricket http://cricket.sourceforge.net/
Flowscan http://net.doit.wisc.edu/~plonka/FlowScan/
Smokeping http://people.ee.ethz.ch/~oetiker/webtools/smokeping/
Other front ends to rrdtool http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/rrdworld/
Nmap http://www.nmap.org/nmap/index.html
Nessus http://www.nessus.org
MD5 Part of the NetBSD base OS, src for compiling on other operating systems is available from ftp://ftp.cert.dfn.de/pub/tools/crypt/md5/01-README
Swatch http://oit.ucsb.edu/~eta/swatch/
Portsentry http://www.psionic.com/abacus/portsentry/
Logsentry http://www.psionic.com/products/logsentry.html
Snort http://www.snort.org/
TCPwrappers ftp://ftp.porcupine.org/pub/security/index.html

Acknowledgements

We'd like to thank the BSDCon committee for giving us the opportunity to share our thoughts with the BSD community.

Alan would specifically like to thank his places of employment Inktomi and Dreamworks for providing the work environment to experiment and explore innovative ways of providing better system performance through monitoring. Also, he would like to thank David Brownlee for introducing him to the one OS, NetBSD, and the many people in the NetBSD community who have always been helpful and instructive through the years.

Timecounters: Efficient and precise timekeeping in SMP kernels

Poul-Henning Kamp
The FreeBSD Project
<phk@FreeBSD.org>

The FreeBSD timecounters are an architecture-independent implementation of a binary timescale using whatever hardware support is at hand. The timecounter timescale converts to other time representations using cheap multiply and shift operations and provides for sufficient precision and resolution to cater for all future needs. The math and implementation will be described, including the features which support NTP PLL/FLL time synchronization, lockless multi-cpu operation and on-the-fly change in hardware support.

Poul-Henning Kamp believes that UNIX is the best OS ever made so far, he is convinced we can still make it better and he has been trying to since the early eighties. Ever since Minix 1.0 came out, Poul-Henning has been running UNIX on his laptop, and via 386BSD he came to FreeBSD where he sat on the Core Team from 1994-2000. Poul-Henning has been release engineer for a number of FreeBSD releases, written, rewritten and cleaned up many pieces of FreeBSD kernel, written a memory allocator, a password scrambler, the beerware license and generally been having a good time. Poul-Henning lives in Denmark with his wife, his son, his daughter about ten FreeBSD computers and one of the worlds most precise NTP clocks. He makes a living as an independent contractor doing all sorts of magic with computers and network.

Timecounters: Efficient and precise timekeeping in SMP kernels.

Poul-Henning Kamp
The FreeBSD Project

ABSTRACT

The FreeBSD timecounters are an architecture-independent implementation of a binary timescale using whatever hardware support is at hand for tracking time. The binary timescale converts using simple multiplication to canonical timescales based on micro- or nano-seconds and can interface seamlessly to the NTP PLL/FLL facilities for clock synchronisation. Timecounters are implemented using lock-less stable-storage based primitives which scale efficiently in SMP systems. The math and implementation behind timecounters will be detailed as well as the mechanisms used for synchronisation.

Introduction

Despite digging around for it, I have not been able to positively identify the first computer which knew the time of day. The feature probably arrived either from the commercial side so service centres could bill computer cycles to customers or from the technical side so computers could timestamp external events, but I have not been able to conclusively nail the first implementation down.

But there is no doubt that it happened very early in the development of computers and if systems like the "SAGE" [SAGE] did not know what time it was I would be amazed.

On the other hand, it took a long time for a real time clock to become a standard feature:

The "Apple][" computer had neither in hardware or software any notion what time it was.

The original "IBM PC" did know what time it was, provided you typed it in when you booted it, but it forgot when you turned it off.

One of the "advanced technologies" in the "IBM PC/AT" was a battery backed CMOS chip which kept track of time even when the computer was powered off.

Today we expect our computers to know the time, and with network protocols like NTP we will usually find that they do, give and take some milliseconds.

This article is about the code in the FreeBSD kernel which keeps track of time.

Time and timescale basics

Despite the fact that time is the physical quantity (or maybe entity ?) about which we know the least, it is at the same time [sic!] what we can measure with the highest precision of all physical quantities.

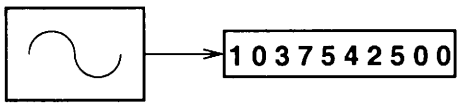
The current crop of atomic clocks will neither gain nor loose a second in the next couple hundred million years, provided we stick to the preventative maintenance schedules. This is a feat roughly in line with to knowing the circumference of the Earth with one micrometer precision, in real time.

While it is possible to measure time by means other than oscillations, for instance transport or consumption of a substance at a well-known rate, such designs play no practical role in time measurement because their performance is significantly inferior to oscillation based designs.

In other words, it is pretty fair to say that all relevant timekeeping is based on oscillating phenomena:

sun-dial	Earths rotation about its axis.
calendar	Ditto + Earths orbit around the sun.
clockwork	Mechanical oscillation of pendulum.
crystals	Mechanical resonance in quartz.
atomic	Quantum-state transitions in atoms.

We can therefore with good fidelity define "a clock" to be the combination of an oscillator and a counting mechanism:



Oscillator + Counter = Clock

The standard second is currently defined as

> The duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium 133 atom.

and we have frequency standards which are able to mark a sequence of such seconds with an error less than $2 \cdot 10^{-15}$ [DMK2001] with commercially available products doing better than $1 \cdot 10^{-14}$ [AG2002].

Unlike other physical units with a conventionally defined origo, longitude for instance, the

ephemeral nature of time prevents us from putting a stake in the ground, so to speak, and measure from there. For measuring time we have to rely on "dead reckoning", just like the navigators before Harrison built his clocks [RGO2002]: We have to tally how far we went from our reference point, keeping a running total at all times, and use that as our estimated position.

The upshot of this is, that we cannot define a timescale by any other means than some other timescale(s).

"Relative time" is a time interval between two events, and for this we only need to agree on the rate of the oscillator.

"Absolute time" consists of a well defined point in time and the time interval since then, this is a bit more tricky.

The Internationally agreed upon TAI and the UTC timescales starts at (from a physics point of view) arbitrary points in time and progresses in integral intervals of the standard second, with the difference being that UTC does tricks to the counting to stay roughly in sync with Earths rotation [1].

TAI is defined as a sequence of standard seconds (the first timescale), counted from January 1st 1958 (the second timescale).

UTC is defined basically the same way, but every so often a leap-second is inserted (or theoretically deleted) to keep UTC synchronised with Earths rotation.

Both the implementation of these two, and a few others speciality timescales are the result of the combined efforts of several hundred atomic frequency standards in various laboratories and institutions throughout the world, all reporting to the BIPM in Paris who calculate the "paper clock" which TAI and UTC really are using a carefully designed weighting algorithm [2].

[1] The first atomic based definition actually operated in a different way: each year would have its own value determined for the frequency of the caesium resonance, selected so as to match the revolution rate of the Earth. This resulted in time-intervals being very unwieldy business, and more and more scientists realized that that the caesium resonance was many times more stable than the angular momentum of the Earth. Eventually the new leap-second method were introduced in 1972. It is interesting to note that the autumn leaves falling on the northern hemisphere affects the angular momentum enough to change the Earths rotational rate measurably.

[2] The majority of these clocks are model 5071A from Agilent (the test and measurement company formerly known as "Hewlett-Packard") which count for as much as 85% of the combined weight. A fact the company deservedly is proud of. The majority of the remaining weight is assigned to a handful of big custom-design units like the PTB2 and NIST7.

Leap seconds are typically announced six to nine months in advance, based on precise observations of median transit times of stars and VLBI radio astronomy of very distant quasars.

The perceived wisdom of leap-seconds have been gradually decreasing in recent years, as devices and products with built-in calendar functionality becomes more and more common and people realize that user input or software upgrades are necessary to instruct the calendar functionality about upcoming leap seconds.

UNIX timescales

UNIX systems use a timescale which pretends to be UTC, but defined as the count of standard seconds since 00:00:00 01-01-1970 UTC, ignoring the leap-seconds. This definition has never been perceived as wise.

Ignoring leap seconds means that unless some trickery is performed when a leap second happens on the UTC scale, UNIX clocks would be one second off. Another implication is that the length of a time interval calculated on UNIX time_t variables, can be up to 22 (and counting) seconds wrong relative to the same time interval measured on the UTC timescale.

Recent efforts have tried to make the NTP protocol make up for this deficiency by transmitting the UTC-TAI offset as part of the protocol. [MILLS2000A]

Fractional seconds are represented two ways in UNIX, "timeval" and "timespec". Both of these formats are two-component structures which record the number of seconds, and the number of microseconds or nanoseconds respectively.

This unfortunate definition makes arithmetic on these two formats quite expensive to perform in terms of computer instructions:

```
/* Subtract timeval from timespec */
t3.tv_sec = t1.tv_sec - t2.tv_sec;
t3.tv_nsec = t1.tv_nsec -
             t2.tv_usec * 1000;
if (t3.tv_nsec >= 1000000000) {
    t3.tv_sec++;
    t3.tv_nsec -= 1000000000;
} else if (t3.tv_nsec < 0) {
    t3.tv_sec--;
    t3.tv_nsec += 1000000000;
}
```

While nanoseconds will probably be enough for most timestamping tasks faced by UNIX computers for a number of years, it is an increasingly uncomfortable situation that CPU clock periods and instruction timings are already not representable in the standard time formats available on UNIX for consumer grade hardware, and the first POSIX mandated API, `clock_getres(3)` has

already effectively reached end of life as a result of this.

Hopefully the various standards bodies will address this issue better in the future.

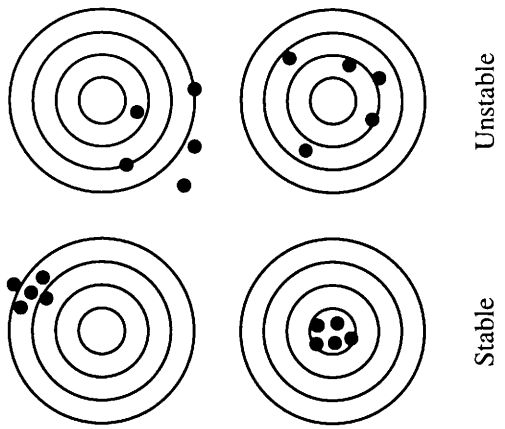
Precision, Stability and Resolution

Three very important terms in timekeeping are "precision", "stability" and "resolution". While the three words may seem to describe somewhat the same property in most uses, their use in timekeeping covers three very distinct and well defined properties of a clock.

Resolution in clocks is simply a matter of the step-size of the counter or in other words: the rate at which it steps. A counter running on a 1 MHz frequency will have a resolution of 1 microsecond.

Precision talks about how close to the intended rate the clock runs, stability about how much the rate varies and resolution about the size of the smallest timeinterval we can measure.

From a quality point of view, Stability is a much more valuable property than precision, this is probably best explained using a graphic illustration of the difference between the two concepts:

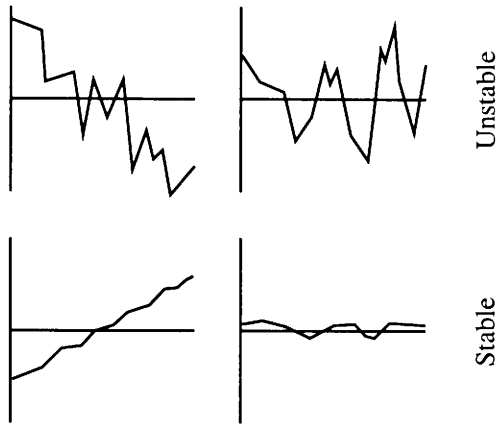


In the top row we have instability, the bullet holes are spread over a large fraction of the target area. In the bottom row, the bullets all hit in a very small area.

On the left side, we have lack of precision, the holes obviously are not centred on the target, a systematic offset exists. In the right side we have precision, the bullets are centred on the target [3].

[3]We cannot easily get resolution into this analogy, the obvious representation as the diameter of the bullet-hole is not correct, it would have to be the grid or other pattern of locations where the bullet could possibly penetrate the target material, but this gets too quantum-mechanical-oid to serve the instructional purpose.

Transposing these four targets to actual clocks, the situation could look like the following plots:



On the x-axis we have time and on the y-axis how wrong the clock was at a given point in time.

The reason atomic standards are such a big deal in timekeeping is that they are incredibly stable: they are able to generate an oscillation where the period varies by roughly a millionth of a billonth of a second in long term measurements.

They are in fact not nearly as precise as they are stable, but as one can see from the graphic above, a stable clock which is not precise can be easily corrected for the offset and thus calibrated is as good as any clock.

This lack of precision is not necessarily a flaw in these kinds of devices, once you get into the $10 \cdot 10^{-15}$ territory things like the blackbody spectrum at the particular absolute temperature of the clocks hardware and general relativistic effects mostly dependent on the altitude above earths center has to be corrected for [4].

Design goals of timecounters

After this brief description of the major features of the local landscape, we can look at the design goals of timecounters in detail:

Provide timestamps in timeval and timespec formats,

This is obviously the basic task we have to solve, but as was noted earlier, this is in no way the performance requirement.

on both the "uptime" and the POSIX timescales,

The "uptime" timescale is convenient for time

[4]This particularly becomes an issue with space-based atomic standards as those found on the "Navstar" GPS satellites.

intervals which are not anchored in UTC time: the run time of processes, the access time of disks and similar.

The uptime timescale counts seconds starting from when the system is booted. The POSIX/UTC timescale is implemented by adding an estimate of the POSIX time when the system booted to the uptime timescale.

using whatever hardware we have available at the time,

Which in a subtle way also implies "be able to switch from one piece of hardware to another on the fly" since we may not know right up front what hardware we have access to and which is preferable to use.

while supporting time the NTP PLL/FLL discipline code,

The NTP kernel PLL/FLL code allows the local clock and timescale to be synchronised or syntonised to an external timescale either via network packets or hardware connection. This also implies that the rate and phase of the timescale must be manoeuvrable with sufficient resolution.

and providing support for the RFC 2783 PPS API,

This is mainly for the benefit of the NTPD daemons communication with external clock or frequency hardware, but it has many other interesting uses as well [PHK2001].

in a SMP efficient way.

Timestamps are used many places in the kernel and often at pretty high rate so it is important that the timekeeping facility does not become a point of CPU or lock contention.

Timecounter timestamp format.

Choosing the fundamental timestamp format for the timecounters is mostly a question of the resolution and steer-ability requirements.

There are two basic options on contemporary hardware: use a 32 bit integer for the fractional part of seconds, or use a 64 bit which is computationally more expensive.

The question therefore reduced to the somewhat simpler: can we get away with using only 32 bit ?

Since 32 bits fractional seconds have a resolution of slightly better than quarter of a nanosecond (.2328 nsec) it can obviously be converted to nanosecond resolution struct timespec timestamps with no loss of precision, but unfortunately not with pure 32 bit arithmetic as that would result in unacceptable rounding errors.

But timecounters also need to represent the clock period of the chosen hardware and this hardware might be the GHz range CPU-clock. The list of clock frequencies we could support with 32 bits are:

$$\begin{aligned} 2^{32}/1 &= 4.294 \text{ GHz} \\ 2^{32}/2 &= 2.147 \text{ GHz} \\ 2^{32}/3 &= 1.432 \text{ GHz} \\ &\ldots \\ 2^{32}/(2^{32}-1) &= .999 \text{ Hz} \end{aligned}$$

We can immediately see that 32 bit is insufficient to faithfully represent clock frequencies even in the low GHz area, much less in the range of frequencies which have already been vapourwared by both IBM, Intel and AMD. QED: 32 bit fractions are not enough.

With 64 bit fractions the same table looks like:

$$\begin{aligned} 2^{64}/1 &= 18.45 \cdot 10^9 \text{ GHz} \\ 2^{64}/2 &= 9.223 \cdot 10^9 \text{ GHz} \\ &\ldots \\ 2^{64}/2^{32} &= 4.294 \text{ GHz} \\ &\ldots \\ 2^{64}/(2^{64}-1) &= .999 \text{ Hz} \end{aligned}$$

And the resolution in the 4 GHz frequency range is approximately one Hz.

The following format have therefore been chosen as the basic format for timecounters operations:

```
struct bintime {
    time_t  sec;
    uint64_t frac;
};
```

Notice that the format will adapt to any size of time_t variable, keeping timecounters safely out of the "We SHALL prepare for the Y2.038K problem" war zone.

One beauty of the bintime format, compared to the timeval and timespec formats is that it is a binary number, not a pseudo-decimal number. If compilers and standards allowed, the representation would have been "int128_t" or at least "int96_t", but since this is currently not possible, we have to express the simple concept of multiword addition in the C language which has no concept of a "carry bit".

To add two bintime values, the code therefore looks like this [5]:

```
uint64_t u;

u = bt1->frac;
bt3->frac = bt1->frac + bt2->frac;
bt3->sec  = bt1->sec  + bt2->sec;
if (u > bt3->frac)
    bt3->sec += 1;
```

[5] If the reader suspects the '>' is a typo, further study is suggested.

An important property of the bintime format is that it can be converted to and from timeval and timespec formats with simple multiplication and shift operations as shown in these two actual code fragments:

```
void
bintime2timespec(struct bintime *bt,
                 struct timespec *ts)
{

    ts->tv_sec = bt->sec;
    ts->tv_nsec =
      ((uint64_t)1000000000 *
      (uint32_t)(bt->frac >> 32)) >> 32;
}

void
timespec2bintime(struct timespec *ts,
                 struct bintime *bt)
{

    bt->sec = ts->tv_sec;
    /* 18446744073 =
      int(2^64 / 1000000000) */
    bt->frac = ts->tv_nsec *
      (uint64_t)18446744073LL;
}
```

How timecounters work

To produce a current timestamp the timecounter code reads the hardware counter, subtracts a reference count to find the number of steps the counter has progressed since the reference timestamp. This number of steps is multiplied with a factor derived from the counters frequency, taking into account any corrections from the NTP PLL/FLL and this product is added to the reference timestamp to get a timestamp.

This timestamp is on the "uptime" time scale, so if UNIX/UTC time is requested, the estimated time of boot is added to the timestamp and finally it is scaled to the timeval or timespec if that is the desired format.

A fairly large number of functions are provided to produce timestamps, depending on the desired timescale and output format:

Desired Format	uptime timescale	UTC/POSIX timescale
bintime	binuptime()	bintime()
timespec	nanouptime()	nanotime()
timeval	microuptime()	microtime()

Some applications need to timestamp events, but are not particular picky about the precision. In many cases a precision of tenths or hundreds of seconds is sufficient.

A very typical case is UNIX file timestamps: There is little point in spending computational resources getting an exact nanosecond timestamp, when the data is written to a mechanical device which has several milliseconds of unpredictable delay before the operation is completed.

Therefore a complementary shadow family of timestamping functions with the prefix "get" have been added.

These functions return the reference timestamp from the current timehands structure without going to the hardware to determine how much time has elapsed since then. These timestamps are known to be correct to within rate at which the periodic update runs, which in practice means 1 to 10 milliseconds.

Timecounter math

The delta-count operation is straightforward subtraction, but we need to logically AND the result with a bit-mask with the same number (or less) bits as the counter implements, to prevent higher order bits from getting set when the counter rolls over:

$$\Delta Count = (Count_{now} - Count_{ref})\ BITAND\ mask$$

The scaling step is straightforward.

$$T_{now} = \Delta Count \cdot R_{counter} + T_{ref}$$

The scaling factor $R_{counter}$ will be described below.

At regular intervals, scheduled by `hardclock()`, a housekeeping routine is run which does the following:

A timestamp with associated hardware counter reading is elevated to be the new reference timecount:

$$\Delta Count = (Count_{now} - Count_{ref})\ BITAND\ mask$$

$$T_{now} = \Delta Count \cdot R_{counter}$$

$$Count_{ref} = Count_{now}$$

$$T_{ref} = T_{now}$$

If a new second has started, the NTP processing routines are called and the correction they return and the counters frequency is used to calculate the new scaling factor $R_{counter}$:

$$R_{counter} = \frac{2^{64}}{Freq_{counter}} \cdot R_{NTP}$$

Since we only have access to 64 bit arithmetic, dividing something into 2^{64} is a problem, so in the name of code clarity and efficiency, we sacrifice the low order bit and instead calculate:

$$R_{counter} = 2 \cdot \frac{2^{63}}{Freq_{counter}} \cdot R_{NTP}$$

The R_{NTP} correct factor arrives as the signed number of nanoseconds (with 32 bit binary fractions) to adjust per second. This quasi-decimal number is a bit of a square peg in our round binary hole, and a conversion factor is needed. Ideally we want to multiply this factor by:

$$\frac{2^{64}}{10^9 \cdot 2^{32}} = 4.294967296$$

This is not a nice number to work with. Fortunately, the precision of this correction is not critical, we are within an factor of a million of the 10^{-15} performance level of state of the art atomic clocks, so we can use an approximation on this term without anybody noticing.

Deciding which fraction to use as approximation needs to carefully consider any possible overflows that could happen. In this case the correction may be as large as ± 5000 PPM which leaves us room to multiply with about 850 in a multiply-before-divide setting. Unfortunately, there are no good fractions which multiply with less than 850 and at the same time divide by a power of two, which is desirable since it can be implemented as a binary shift instead of an expensive full division.

A divide-before-multiply approximation necessarily results in a loss of lower order bits, but in this case dividing by 512 and multiplying by 2199 gives a good approximation where the lower order bit loss is not a concern:

$$\frac{2199}{512} = 4.294921875$$

The resulting error is an systematic under compensation of 10.6PPM of the requested change, or $1.06 \cdot 10^{-14}$ per nanosecond of correction. This is perfectly acceptable.

Putting it all together, including the one bit we put on the alter for the Goddess of code clarity, the formula looks like this:

$$R_{counter} = 2 \cdot \frac{2^{63} + 2199 \cdot \frac{R_{NTP}}{1024}}{Freq_{counter}}$$

Presented here in slightly unorthodox format to show the component arithmetic operations as they are carried out in the code.

Frequency of the periodic update

The hardware counter should have a long enough period, ie, number of distinct counter values divided by frequency, to not roll over before our periodic update function has had a chance to update the reference timestamp data.

The periodic update function is called from `hardclock()` which runs at a rate which is controlled by the kernel parameter *HZ*.

By default HZ is 100 which means that only hardware with a period longer than 10 msec is usable. If HZ is configured higher than 1000, an internal divider is activated to keep the timecounter periodic update running no more often than 2000 times per second.

Let us take an example: At HZ=100 a 16 bit counter can run no faster than:

$$2^{16} \cdot 100Hz = 6.5536MHz$$

Similarly, if the counter runs at 10MHz, the minimum HZ is

$$\frac{10MHz}{2^{16}} = 152.6Hz$$

Some amount of margin is of course always advisable, and a factor two is considered prudent.

Locking, lack of ...

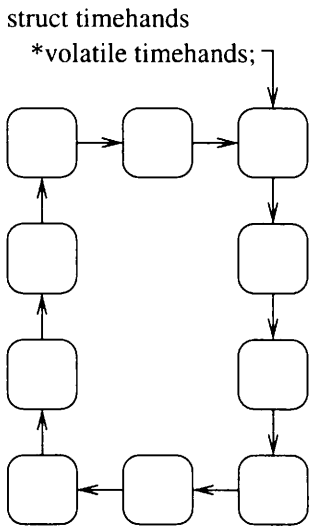
Provided our hardware can be read atomically, that our arithmetic has enough bits to not roll over and that our clock frequency is perfectly, or at least sufficiently, stable, we could avoid the periodic update function, and consequently disregard the entire issue of locking. We are seldom that lucky in practice.

The straightforward way of dealing with meta data updates is to put a lock of some kind on the data and grab hold of that before doing anything. This would however be a very heavy-handed approach. First of all, the updates are infrequent compared to simple references, second it is not important which particular state of meta data a consumer gets hold of, as long as it is consistent: as long as the $Count_{ref}$ and T_{ref} are a matching pair, and not old enough to cause an ambiguity with hardware counter rollover, a valid timestamp can be derived from them.

A pseudo-stable-storage with generation count method has been chosen instead. A ring of ten "timehands" data structures are used to hold the state of the timecounter system, the periodic update function updates the next structure with the new reference data and scaling factor and makes it the current timehands.

The beauty of this arrangement lies in the fact that even though a particular "timehands" data structure has been bumped from being the "currents state" by its successor, it still contains valid data for some amount of time into the future.

Therefore, a process which has started the timestamping process but suffered an interrupt which resulted in the above periodic processing can continue unaware of this afterwards and not suffer corruption or miscalculation even though it holds no locks on the shared meta-data.



This scheme has an inherent risk that a process may be de-scheduled for so long time that it will not manage to complete the timestamping process before the entire ring of timehands have been recycled. This case is covered by each timehand having a private generation number which is temporarily set to zero during the periodic processing, to mark inconsistent data, and incremented to one more than the previous value when the update has finished and the timehands is again consistent.

The timestamping code will grab a copy of this generation number and compare this copy to the generation in the timehands after completion and if they differ it will restart the timestamping calculation.

```
do {
     th = timehands;
     gen = th->th_generation;
     /* calculate timestamp */
} while (gen == 0 ||
    gen != th->th_generation);
```

Each hardware device supporting timecounting is represented by a small data structure called a timecounter, which documents the frequency, the number of bits implemented by the counter and a method function to read the counter.

Part of the state in the timehands structure is a pointer to the relevant timecounter structure, this makes it possible to change to a one piece of hardware to another "on the fly" by updating the current timehands pointer in a manner similar to the periodic update function.

In practice this can be done with sysctl(8):

```
sysctl kern.timecounter.hardware=TSC
```

at any time while the system is running.

Suitable hardware

A closer look on "suitable hardware" is warranted at this point. It is obvious from the above description that the ideal hardware for timecounting is a wide binary counter running at a constant high frequency and atomically readable by all CPUs in the system with a fast instruction(-sequence).

When looking at the hardware support on the PC platform, one is somewhat tempted to sigh deeply and mutter "so much for theory", because none of the above parameters seems to have been on the drawing board together yet.

All IBM PC derivatives contain a device more or less compatible with the venerable Intel i8254 chip. This device contains 3 counters of 16 bits each, one of which is wired so it can interrupt the CPU when the programmable terminal count is reached.

The problem with this device is that it only has 8bit bus-width, so reading a 16 bit timestamp takes 3 I/O operations: one to latch the count in an internal register, and two to read the high and low parts of that register respectively.

Obviously, on multi-CPU systems this cannot be done without some kind of locking mechanism preventing the other CPUs from trying to do the same thing at the same time.

Less obviously we find it is even worse than that: Since a low priority kernel thread might be reading a timestamp when an interrupt comes in, and since the interrupt thread might attempt to generate a timestamp also, we need to totally block interrupts out while doing those three I/O instructions.

And just to make life even more complicated, FreeBSD uses the same counter to provide the periodic interrupts which schedule the `hardclock()` routine, so in addition the code has to deal with the fact that the counter does not count down from a power of two and that an interrupt is generated right after the reloading of the counter when it reaches zero.

Ohh, and did I mention that the interrupt rate for hardclock() will be set to a higher frequency if profiling is active ? [6]

It hopefully doesn't ever get more complicated than that, but it shows, in its own bizarre and twisted way, just how little help the timecounter code needs from the actual hardware.

The next kind of hardware support to materialise was the "CPU clock counter" called "TSC" in

[6] I will not even mention the fact that it can be set also to ridiculous high frequencies in order to be able to use the binary driven "beep" speaker in the PC in a PCM fashion to output "real sounds".

official data-sheets. This is basically a on-CPU counter, which counts at the rate of the CPU clock.

Unfortunately, the electrical power needed to run a CPU is pretty precisely proportional with the clock frequency for the prevailing CMOS chip technology, so the advent of computers powered by batteries prompted technologies like APM, ACPI, SpeedStep and others which varies or throttles the CPU clock to match computing demand in order to minimise the power consumption [7].

Another wiggle for the TSC is that it is not usable on multi-CPU systems because the counter is implemented inside the CPU and not readable from other CPUs in the system.

The counters on different CPUs are not guaranteed to run syntonously (ie: show the same count at the same time). For some architectures like the DEC/alpha architecture they do not even run synchronously (ie: at the same rate) because the CPU clock frequency is generated by a small SAW device on the chip which is very sensitive to temperature changes.

The ACPI specification finally brings some light: it postulates the existence of a 24 or 32 bit counter running at a standardised constant frequency and specifically notes that this is intended to be used for timekeeping.

The frequency chosen, 3.5795454... MHz[8] is not quite as high as one could have wished for, but it is certainly a big improvement over the i8254 hardware in terms of access path.

But trust it to Murphys Law: The majority of implementations so far have failed to provide latching suitable to avoid meta-stability problems, and several readings from the counter is necessary to get a reliable timestamp. In difference from the i8254 mentioned above, we do not need to any locking while doing so, since each individual read is atomic.

An initialization routine tries to test if the ACPI counter is properly latched by examining the width of a histogram over read delta-values.

[7]This technology also found ways into stationary computers from two different vectors. The first vector was technical: Cheaper cooling solutions can be used for the CPU if they are employed resulting in cheaper commodity hardware. The second vector was political: For reasons beyond reason, energy conservation became an issue with personal computers, despite the fact that practically north American households contains 4 to 5 household items which through inefficient designs waste more power than a personal computer use.

[8]The reason for this odd-ball frequency has to be sought in the ghastly colours offered by the original IBM PC Color Graphics Adapter: It delivered NTSC format output and therefore introduced the NTSC colour sync frequency into personal computers.

Other architectures are similarly equipped with means for timekeeping, but generally more carefully thought out compared to the haphazard developments of the IBM PC architecture.

One final important wiggle of all this, is that it may not be possible to determine which piece of hardware is best suited for clock use until well into or even after the bootstrap process.

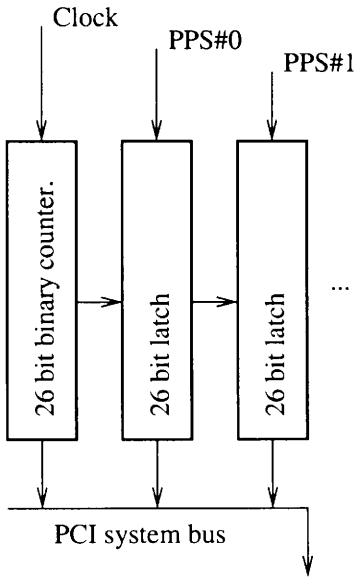
One example of this is the Loran-C receiver designed by Prof. Dave Mills [MILLS1992] which is unsuitable as timecounter until the daemon program which implements the software-half of the receiver has properly initialised and locked onto a Loran-C signal.

Ideal timecounter hardware

As proof of concept, a sort of an existentialist protest against the sorry state describe above, the author undertook a project to prove that it is possible to do better than that, since none of the standard hardware offered a way to fully validate the timecounter design.

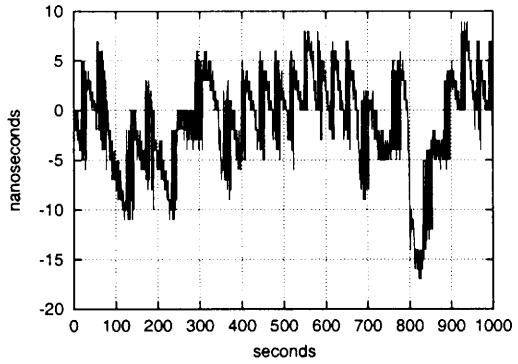
Using a COTS product, "HOT1", from Virtual Computers Corporation [VCC2002] containing a FPGA chip on a PCI form factor card, a 26 bit timecounter running at 100MHz was successfully implemented.

In order to show that timestamping does not necessarily have to be done using unpredictable and uncalibratable interrupts, an array of latches were implemented as well, which allow up to 10 external signals to latch the reading of the counter when an external PPS signal transitions from logic high to logic low or vice versa.



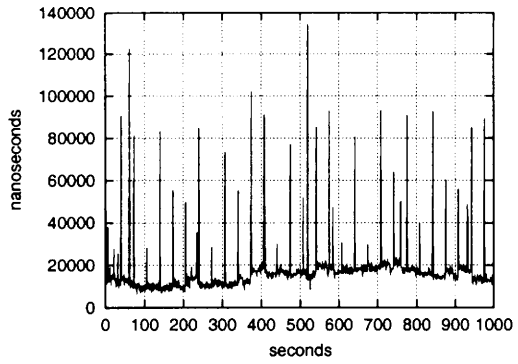
Using this setup, an standard 133 MHz Pentium based PC is able to timestamp the PPS output of

the Motorola UT+ GPS receiver with a precision of ± 10 nanoseconds ± one count which in practice averages out to roughly ± 15 nanoseconds[9]:



It shold be noted that the author is no hardware wizard and a number of issues in the implementation results in less than ideal noise performance.

Now compare this to "ideal" timecounter to the normal setup where the PPS signal is used to trigger an interrupt via the DCD pin on a serial port, and the interrupt handler calls `nanotime()` to timestamp the external event [10]:



It is painfully obvious that the interrupt latency is the dominant noise factor in PPS timestamping in the second case. The asymetric distribution of the noise in the second plot also more or less entirely invalidates the design assumption in the NTP PLL/FLL kernel code that timestamps are dominated by gaussian noise with few spikes.

[9]The reason the plot does not show a very distinct 10 nanosecond quantization is that the GPS receiver produces the PPS signal from a clock with a roughly 55 nanosecond period and then predicts in the serial data stream how many nanoseconds this will be offset from the ideal time. This plot shows the timestamps corrected for this "negative sawtooth correction".

[10]In both cases, the computers clock frequency controlled with a Rubidium Frequency standard. The average quality of crystals used for computers would totally obscure the curves due to their temperature coefficient.

Status and availability

The timecounter code has been developed and used in FreeBSD for a number of years and has now reached maturity. The source-code is located almost entirely in the kernel source file kern_tc.c, with a few necessary adaptations in code which interfaces to it, primarily the NTP PLL/FLL code.

The code runs on all FreeBSD platforms including i386, alpha, PC98, sparc64, ia64 and s/390 and contains no wordsize or endianess issues not specifically handled in the sourcecode.

The timecounter implementation is distributed under the "BSD" open source license or the even more free "Beer-ware" license.

While the ability to accurately model and compensate for inaccuracies typical of atomic frequency standards are not catering to the larger userbase, but this ability and precision of the code guarntees solid support for the widespread deployment of NTP as a time synchronization protocol, without rounding or accumulative errors.

Adding support for new hardware and platforms have been done several times by other developers without any input from the author, so this particular aspect of timecounters design seems to work very well.

Future work

At this point in time, no specific plans exist for further development of the timecounters code.

Various micro-optimizations, mostly to compensate for inadequate compiler optimization could be contemplated, but the author resists these on the basis that they significantly decrease the readability of the source code.

Acknowledgements

The author would like to thank:

Bruce Evans for his invaluable assistance in taming the evil i8254 timecounter, as well as the enthusiastic resistance he has provided throughout.

Professor Dave Mills of University of Delaware for his work on NTP, for lending out the neglected twin Loran-C receiver and for picking up the glove when timecounters made it clear that the old "microkernel" NTP timekeeping code were not up to snuff [MILLS2000B].

Tom Van Baak for helping out, despite the best efforts of the National Danish Posts center for Customs and Dues to prevent it.

Corby Dawson for helping with the care and feeding for caesium standards.

The staff at the NELS Loran-C control station in Bø, Norway for providing information

about step-changes.

The staff at NELS Loran-C station Eiðe, Faeroe Islands for permission to tour their installation.

The FreeBSD users for putting up with "micro uptime went backwards".

References

[AG2002] Published specifications for Agilent model 5071A Primary Frequency Standard on http://www.agilent.com

[DMK2001] "Accuracy Evaluation of a Cesium Fountain Primary Frequency Standard at NIST." D. M. Meekhof, S. R. Jefferts, M. Stephanovic, and T. E. Parker IEEE Transactions on instrumentation and measurement, VOL. 50, NO. 2, APRIL 2001.

[PHK2001] "Monitoring Natural Gas Usage" Poul-Henning Kamp http://phk.freebsd.dk/Gasdims/

[MILLS1992] "A computer-controlled LORAN-C receiver for precision timekeeping." Mills, D.L. Electrical Engineering Department Report 92-3-1, University of Delaware, March 1992, 63 pp.

[MILLS2000A] Levine, J., and D. Mills. "Using the Network Time Protocol to transmit International Atomic Time (TAI)". Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting (Reston VA, November 2000), 431-439.

[MILLS2000B] "The nanokernel." Mills, D.L., and P.-H. Kamp. Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting (Reston VA, November 2000), 423-430.

[RGO2002] For an introduction to Harrison and his clocks, see for instance http://www.rog.nmm.ac.uk/museum/harrison/ or for a more detailed and possibly better researched account: Dava Sobels excellent book, "Longitude: The True Story of a Lone Genius Who Solved the Greatest Scientific Problem of His Time" Penguin USA (Paper); ISBN: 0140258795.

[SAGE] This "gee-wiz" kind of article in Dr. Jobbs Journal is a goot place to start: http://www.ddj.com/documents/s=1493/ddj0001hc/0085a.htm

[VCC2002] Please consult Virtual Computer Corporations homepage: http://www.vcc.com

Using SCTP with Partial Reliability for MPEG-4 Multimedia Streaming

Marco Molteni
Cisco Systems
<mmolteni@cisco.com>

SCTP (Stream Control Transmission Protocol) [1,2], is a new IP transport protocol. It is reliable and connection-oriented as TCP, but has a set of new features that make it well suited for a wide class of applications, among them multimedia streaming. SCTP is record-oriented, providing built-in framing capabilities, and can multiplex multiple data streams into one connection, thus avoiding head-of-line blocking and providing a natural framework for multi-streamed systems like MPEG-4. It can provide ordered or unordered delivery. Further, a sender that implements Partial Reliability SCTP (PR-SCTP) can optionally choose the retransmission behavior on a per packet basis, in a continuos spectrum from TCP-like reliability with multiple retransmissions to UDP-like unreliability with no retransmission at all. In any case PR-SCTP retains all the benefits of SCTP: TCP-friendly congestion control and congestion avoidance, plus multi-homing fail-over capabilities.

The MPEG-4 standard is becoming a popular format for streaming multimedia on the Internet. MPEG-4 encodes the bitstream in groups of different frame types (I, P and B-frames), where the I-frame is independent, while the P and B-frames depend on the I-frame in the group. This means that loosing an I-frame (for example due to network congestion) forces the player to skip to the next frame group, with a noticeable worsening of the video quality. The transport protocol utilized by MPEG is RTP, which in turn is layered on top of UDP. RTP, being concerned with real-time traffic like multimedia streaming, does not provide reliable delivery.

Lately an implementation of SCTP has been imported in the IPv6/IPsec stack developed by KAME, providing kernel-level SCTP for all *BSD flavours [4].

This paper describes our work on FreeBSD to modify the open source MPEG-4 streamer and player found in MPEG4IP [5] to utilize PR-SCTP as transport instead of RTP/UDP, enforcing differentiated Partial Reliability per frame type. Various congestion scenarios show the improved playout quality of the client player, due to the increment of the number of I-frames that PR-SCTP allows to salvage from congestion.

[1] RFC 2960.
[2] draft-stewart-tsvwg-prsctp-00.txt
[3] http://www.sctp.org/
[4] http://www.mpeg4ip.net/

I discovered Unix at University, and I have fallen in love with BSD since then. After graduating in Computer Science, I left Italy for California, where I worked for SRI International. I am now based in France and work for Cisco Systems. In my various experiences I have enjoyed hacking on FreeBSD and more exotic platforms. Among my interests are network and system programming, network protocol design, computer security and system administration. I like spending my spare time creating welded sculptures designed by my wife Francesca.

Using SCTP with Partial Reliability for MPEG-4 Multimedia Streaming

M. Molteni and M. Villari*
Cisco System Technology Center
06410 Sophia Antipolis, France.
e-mail: {mmolteni,mvillari}@cisco.com

Abstract

The MPEG-4 standard encodes a video stream in 3 different frames, I, P and B, where the P and B frames depend on the I frame. Loosing an I frame is especially bad. In this paper we exploit the Partial Reliability features of the SCTP transport protocol to selectively retransmit I frames in presence of congestion, obtaining a better quality of the decoded stream.

1 Introduction

The Stream Control Transmission Protocol (SCTP) is a relatively new IP transport protocol. It is reliable, connection and message-oriented, and has a set of new features that make it well suited for a wide class of applications. SCTP can provide ordered or unordered delivery, and when both sides implement Partial Reliability SCTP (PR-SCTP), the sender can choose the retransmission behavior on a per packet basis, in a continuos spectrum from TCP-like reliability with multiple retransmissions to UDP-like unreliability with no retransmission at all, always retaining TCP-friendly congestion control and congestion avoidance.

The MPEG-4 standard is becoming a popular format for streaming multimedia on the Internet. MPEG-4 encodes the bitstream in groups of different frame types (I, P and B frames), where the I frame is independent, while the P and B frames depend on the I frame in the group. This means that loosing an I frame (for example due to network congestion) causes a noticeable worsening of the video quality of all the frame group. The transport protocol utilized by MPEG is RTP/UDP. RTP, being concerned with real-time traffic, does not provide reliable delivery.

Lately a Cisco implementation of SCTP has been imported in the IPv6/IPsec stack developed by KAME, providing kernel-level SCTP for the operating systems derived from BSD Unix.

This paper describes our preliminary work on FreeBSD to modify two open source programs, a MPEG-4 streamer and a MPEG-4 player, to utilize PR-SCTP as transport instead of RTP/UDP, enforcing differentiated Partial Reliability per frame type.

Our preliminary results show that under congestion scenarios there is improved playout quality of the video stream, due to the increment of the number of I-frames that PR-SCTP allows to salvage from congestion.

2 Background

In this section we briefly describe the various fields involved in our work, highlighting key concepts. The reader is referred to the bibliography for deeper treatment.

*on leave from University of Messina.

2.1 SCTP

The Stream Control Transmission Protocol (SCTP) [13] is a relatively new IP transport protocol. It is reliable and connection-oriented as TCP, and has a TCP-friendly congestion control.

While TCP is byte stream oriented, SCTP is message oriented (i.e. it preserves message boundaries as, for example, UDP). This means that an application that needs message boundaries doesn't have to provide its own framing as is the case with TCP.

SCTP can multiplex multiple data "streams" into one SCTP association. Each message is associated with a stream number, and messages belonging to the same stream are delivered in order. However, while one stream may be blocked waiting for the next in-sequence message, delivery from other streams may proceed, avoiding head-of-line blocking. Also, the packet delivery can be ordered or unordered, with stream granularity.

Further, when both endpoints implement Partial Reliability SCTP (PR-SCTP) [14], the sender can choose the retransmission behavior on a per message basis. As of today the only partially reliable service specified is the *timed reliability* service, but different notions of partial reliability can be introduced and the niceness of the extension is that the receiver doesn't have to know which kind of partial reliability is performed by the server.

Timed reliability means that the user can specify the lifetime of a message: when the lifetime is expired and the message hasn't been acked yet, the sender stops the retransmission efforts and drops the packet. In the protocol control plane, the sender will send a forward TSN (Transmission Sequence Number), telling the receiver to move its cumulative ack point forward. The effect of moving the ack point forward is to consider the skipped messages as received and acked.

2.2 MPEG

The Motion Pictures Experts Group (MPEG) released in 1998 the MPEG-4 standard [1]. Looking at the previous standards, MPEG-1 can encode up to 1.5 Mbps (low bit rate), whereas MPEG-2 can go up to 15 Mbps (high bit rate). MPEG-2 is used in applications such as DVD video and cable or satellite broadcast. MPEG-4 embodies several video codecs, such as Dvix and Xvid, which are capable of a ten times reduction of the bit rate in both areas (low and high bit rate) keeping the same quality. The MPEG-4 visual standard has been explicitly optimized for three bit rate ranges: below 64 Kbps, 64–384 Kbps and 384–4 Mbps. MPEG-4 provides also features for the animation of faces and synthetic bodies.

The MPEG-4 scene consists of a number of audio and video media objects. Several of them are typically background, like audio clips or static images. The information for each streaming media object are brought within one or more elementary streams.

The entire MPEG-4 standard includes specifications on hundreds of features, but no particular application needs to, or is expected to, support all of those features. *Profiles* and *Levels* define what an application supports. A Profile defines the features and qualitative functionality, and the Level specifies the quantitative complexity of the functions within a Profile.

The basic object in MPEG-4 is a VOP, Video Object Plane, which can have any shape. A conventional video frame is represented by a VOP with a rectangular shape, and in this paper we use the term *frame* and *VOP* equivalently.

The MPEG encoding considers three kinds of frames: Intra-VOP (I), Predicted (P) and Bidirectional interpolated (B) frames. I frames are coded independently from other frames; the compression is typically spatial. P frames are coded having as reference the time preceding P or I frame. B frames are

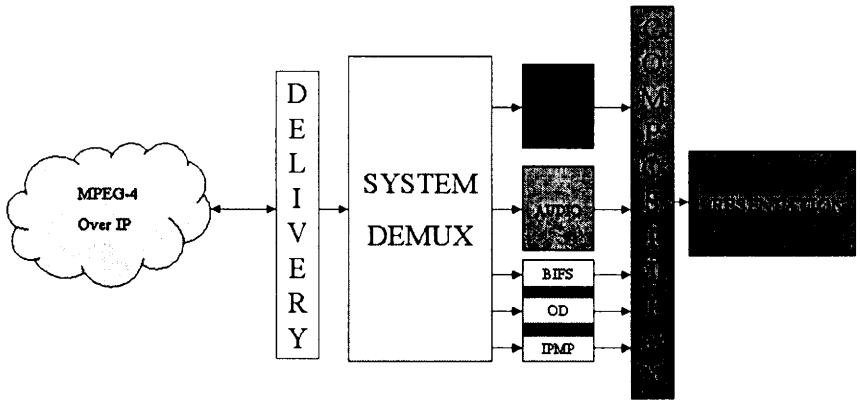


Figure 1: MPEG-4: The data streams corresponding to audio/video signals are stored separately. They are composed in an audiovisual and integrated presentation only to the receiver.

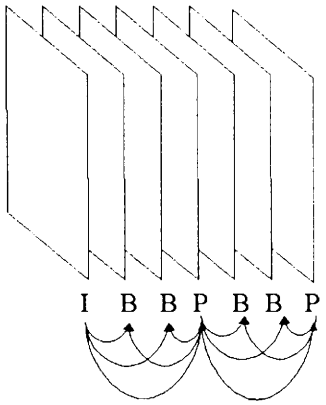


Figure 2: Group of frames (GOV), showing the dependency between frames

coded having as reference the time adjacent I or P frames. B frames are never a reference for other frames. P and B frames are temporal compressed, and used to perform motion estimation and compensation.

A GOV (Group of VOPs) is a set of I, P and B frames; its sequence and length can vary during encoding. A GOV, as depicted in Figure 2, always begins with a I frame, which is also the only I frame in the GOV. The next I frame in the stream is the beginning of the next GOV. An important part of the information in a GOV resides in the I frame.

3 Related Work

There is a lot of research going on to improve the quality of video streaming over best effort networks, trying to introduce intelligent retransmission.

The Internet Draft [8] describes new RTP payload formats to enable multiple and optional selective retransmissions in RTP. These are especially applicable to environments where enhanced RTCP feedback is available. These payload formats can be used to separate the media stream according to prioritization of packets or according to the status of the transmission (i.e. transmission or retransmission).

Feamster [6] realizes a complete environment to test Selective Reliability RTP (SR-RTP). He uses RTP over UDP with an extension to enable a feedback communication between server and client. He has developped a streaming application to delivery high quality video. He introduces also a post-processing phase to recover some of the packet loss, and analyzes the results with peak signal-to-noise ratio (PSNR).

The work by Raman [10] is an implementation and evaluation of the Image Transport Protocol (ITP) for image transmission over lossy or wireless networks. They ver-

ify the quality at Application Level Framing (ALF); as measure they consider the evolution of the PSNR. ITP runs over UDP and incorporates receiver-driven selective reliability. ITP enables a variety of new receiver post-processing algorithms such as error concealment that further improve the interactivity and responsiveness of reconstructed images.

4 MPEG-4 over PR-SCTP

[7] describes how MPEG-4 in an IP network is trasported over the real-time transport protocol (RTP) [11]. RTP itself is normally transported over UDP[1].

UDP is best effort as IP, and since RTP concerns itself with real-time data, a RTP packet lost in the network will not be retransmitted by the sender. This is normally what is wanted, because there is no point in receiving a time sensitive message after its useful lifetime has expired, and so retransmission would be useless for the receiver and potentially bad for the overall network congestion.

Applications like streaming video use time-sensitive messages, but to avoid temporary network delays utilize a few seconds buffering. MPEG-4 encodes the video stream in three different frames types, I, P and B, as explained in section 2.2. Loosing an I frame is strongly worse than loosing a P or a B frame. The idea at the basis of our and other works is to somehow give reliability (by means of retransmission) to what is really important, I frames.

In our case we obtain I frames reliability by exploiting a native feature of the SCTP transport protocol: partial reliability, instead of having the application adding this on top of RTP/UDP as is done in other approaches.

[1] It is also possible to transport RTP over TCP, in the so-called interleaved mode of RTSP (Real Time Streaming Protocol) [12].

As in implementation 2 and 3 we analize the signal quality information from the application level with PSNR, as detailed in section 6.

Nonetheless, reliability by itself is useless if the message is received after its validity is expired, and bandwidth-heavy if used for all messages without regards to the importance. SCTP naturally solves both problems, because

- the granularity of the retransmission is per message, and so only messages containing I frames can be tagged for retransmission
- the efforts in retransmissions are tunable, and in our case the *timed reliability* maps directly to the time-sensitiveness of video streaming

Also, since MPEG-4 takes care of message reordering, we asked SCTP for unordered delivery.

From an implementation point of view, as detailed in section 5, we took an existing streaming server and player and replaced the UDP sockets with SCTP ones, trying to perturb as little as possible the surrounding code.

When the server was to send an I frame, we set the time to live of the message to a tunable value, to be determined by experimentation. As default we used 2000 ms. Conversely, when the server was to send a P or B frame, we set the time to live to the special value `SEND_EXACTLY_ONCE`.

We open the SCTP socket in the so-called UDP-style (a UDP-style SCTP socket has nothing to do with UDP unreliability, it just refers to one of the two models offered by the SCTP socket API[2]) because it maps well to the UDP interface that the server expects and because it is the only model that allows us fine-grained control over the per message SCTP behaviour.

[2] See section 2.1 for further details.

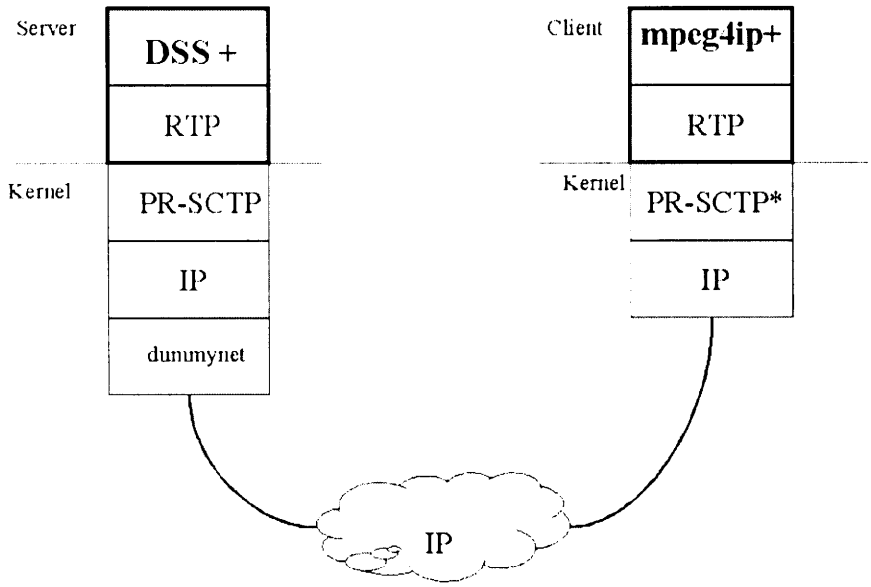


Figure 3: Client Server scenario

Since we used SCTP instead of UDP to encapsulate RTP, the payload available for RTP was smaller, and we had to take this into consideration when creating the *hint* track of the MPEG-4 file to be read by the server.

Figure 3 depics the components of our scenario. On the server side, DSS+ is the Darwin Streaming Server as modified to use SCTP. Dummynet is used to introduce random packet losses. On the client side, mp4player+ is the mp4player as modified to use SCTP. PR-SCTP* on the client is to remember that the SCTP receiver doesn't have to know which reliability algorithm is used by the sender.

5 FreeBSD Implementation

We used a FreeBSD 4.6-RELEASE machine plus various KAME snapshots. The original intent was to use one machine as the server and various machines as the clients, but due to time constraints and a few bugs we finally settled to use only one machine and the loopback interface.

We used dummynet [9], a network emulator and traffic shaper included in FreeBSD, to introduce random packet drops in the test network.

5.1 KAME/SCTP

Randall Stewart and Peter Lei, both from Cisco, wrote a SCTP kernel implementation [5] for the various BSDs, which is now part of the KAME source code [3]. Installing a KAME snapshot and adding the line `options SCTP` to the kernel configuration file is enough to enable SCTP support, providing the SCTP socket API as defined in [15].

Since the SCTP kernel implementation is actively developed, we kept following both the KAME snapshots and the SCTP patches against the snapshot, until October 2002.

5.2 Darwin Streaming Server

The Darwin Streaming Server (DSS) [2] is mostly written in C++. It starts from a generic socket class, from which a UDP socket class is inherited. On top of that, since RTP/UDP requires two sequential ports (even and even + 1), it has "socket

pairs" and various other gadgets. We replaced the RTP socket with a SCTP socket, leaving the companion RTCP socket as UDP.

We then teached to the SCTP socket that it had to send its packets with Partial Reliability, depending on the frame type: P and B frames had to be send just once (as plain RTP), while I frames[3] had to be eventually resent. Since the socket class didn't had the notion of frame type, we had to find out where in the code we could grab this information and how to pass it all the way down to the socket.

5.3 MPEG4IP mp4player

The MPEG4IP [4] mp4player is written in C++ and relies on the UCL (University College London) RTP library, written in C. As in the Darwin Streaming Server case, we replaced the RTP UDP socket with a SCTP socket. Since RTP/UDP requires two sequential ports (even and even + 1) the mp4player code first gets two UDP sockets and binds them to two sequential ports, then closes them and calls immediately the RTP library passing the starting port as parameter. This approach didn't work well with our trick of replacing the UDP socket with a SCTP one, so we had to modify both the mp4player and the RTP library. In the player we don't close the socket any more, and the library is extended to accept already initialized sockets instead of doing the initialization by itself.

We also modified some of the decoder plugins, as detailed in section 5.4.

5.4 Tools

The mp4player uses plugins for the decoding, among them MPEG-4 ISO and Xvid. We modified both plugins to gather and log various informations related to timings, sequence number, frame size, etc, and to write to file a full dump of the raw video stream (in YUV format) just decoded.

We also wrote a tool, `psnr`, to analyze two raw video dumps, generated by the plugins, and to calculate the peak signal-to-noise ratio (PSNR) between each corresponding frame, as described in section 6. The same tool is also able to extract a frame form the YUV dump and export it in PPM format.

6 Experimental Results

In this section we provide an analysis of the experimental results we obtained. We would like to point out that these results are preliminary and require further testings and analysis. We made tests from both high quality (MPEG-2 DVD tracks converted to medium quality 700 kbps) and low quality (200 kbps) video streams, and we present here the results for the low quality case.

We used peak signal-to-noise-ratio (PSNR) in Equation 1 as method to evaluate in an objective manner the quality of video streams. PSNR is widely utilized in literature to evaluate the quality of a generic image before and after a lossy compression. Although it is not the best technique to synthetize the human visive perception, it provides a reasonable level of objectivity.

We compared the same video sequence with and without packet drops in the UDP and SCTP case. The dummynet traffic shaper [9] allowed us to vary the drop rate from 2^{-8} (ca 0.004) to 2^{-3} (ca 0.125).

Figure 4 shows the differences in PSNR for UDP (upper picture) and SCTP (lower picture) in the case of a 2^{-6} drop rate. On the x axis there is the frame number, on the y axis there is the PSNR in dB. The 100 dB peak actually means infinity, when the two frames are the same. The more the peaks, the higher the quality of the video stream. The graphs show that the SCTP case has a bigger number of peaks.

Also in our experiments we noted that for high packet drop rates (2^{-3}) the SCTP case

[3] called Key Frames in DSS.

$$PSNR = 20\log_{10}\frac{255}{(\frac{1}{N_1 N_2}\sum_{x=0}^{N_1-1}\sum_{y=0}^{N_2-1}[f(x,y)-f`(x,y)]^2)^{1/2}} \quad (1)$$

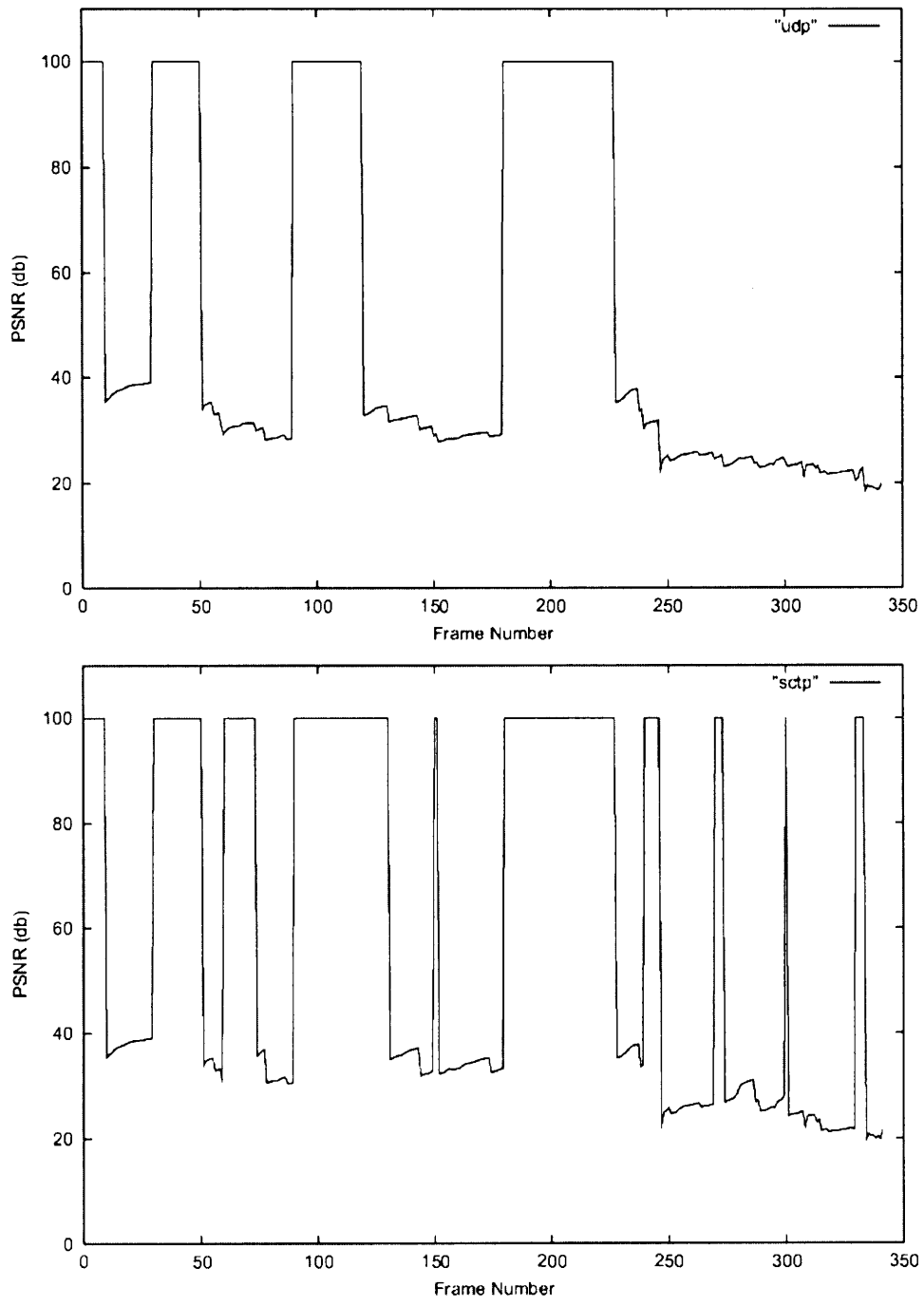


Figure 4: PSNR for UDP and SCTP.

is actually worse than UDP. Our gut feeling is that the SCTP congestion control mechanism gets triggered, while UDP happily keeps sending packets at full throttle worsening the congestion. Further experimentation is needed to replicate the behaviour and to give a proper explanation.

7 Conclusions and Future Work

In this paper we showed that using Partial Reliability SCTP to optionally retransmit MPEG-4 I frames may result in improved quality of the decoded video stream. Further work is required to better identify and qualify the congestion scenarios that might be improved by the usage of PR-SCTP.

For the future, we plan to address some of the shortcomings and limitations of the present work, among them we will utilize different hosts for the server and the clients, and we will do more tests with deeper analysis.

We also plan to make the various components we used more robusts. We had many crashes, and while some of them may be explained by the perturbations introduced by our modifications, some others seems to be more general, and related to insufficient input validation.

References

[1] Iso/iec 14496-1 final draft international standard mpeg-4 systems (oct. 1998).

[2] `http://www.developer.apple.com/darwin/projects/streaming/`.

[3] `http://www.kame.net/`.

[4] `http://www.mpeg4ip.net/`.

[5] `http://www.sctp.org/`.

[6] Nicholas Feamster. Adaptive delevery of real-time streaming video. Master's thesis, Massachusetts Institute of Technology, 2000.

[7] Y. Kikuchi et al. RFC 3016: Rtp payload format for mpeg-4 audio/visual streams, 2000.

[8] A. Miyazaki et al. *RTP Payload Formats to Enable Multiple Selective Retransmissions.* draft-ietf-avt-rtp-selret-05.txt, 2002.

[9] L. Rizzo. Dummynet and forward error correction, 1998.

[10] M. Srinivasan S. Raman, H. Balakrisman. An image transport protocol for the internet. In *2000 International Conference on Network Protocols.*, 2000.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 1889: RTP: A transport protocol for real-time applications, 1996.

[12] H. Schulzrinne, A. Rao, and R. Lanphier. RFC 2326: Real time streaming protocol (rtsp), 1998.

[13] R. Stewart et al. RFC 2960: Stream control transmission protocol, 2000.

[14] R. Stewart et al. *SCTP Partial Reliability Extension.* draft-stewart-tsvwg-prsctp-00.txt, 2002.

[15] R. Stewart et al. *Sockets API Extensions for Stream Control Transmission Protocol (SCTP).* draft-ietf-tsvwg-sctpsocket-05.txt, 2002.

All For One Port, One Port For All

Bram Moolenaar
Stichting NLnet Labs
<bram@moolenaar.net>

The BSD systems each offer a different ports system. FreeBSD has the largest number of ports, OpenBSD ports provide a few extra features and NetBSD uses pkgsrc. Even though these ports systems originate from the same root, enough changes have been included to make them incompatible. The result is that a port has to be written and maintained for each BSD variant. The number of ports available greatly depends on the willingness of someone to create and maintain a port while a lot of work is repeated over and over again by the different maintainers. Obviously this is an unwanted situation.

One serious attempt has been made to reunite the ports systems: OpenPackages. Unfortunately, this project has stalled. There are several reasons why this route is very unlikely to succeed. The main one is that none of the BSD distributions wants to switch to a different system that suddenly makes all their existing ports unusable. Political and personal preferences add enough controversy to make any attempt at merging fail.

An alternative is to introduce a new system that exists next to the traditional ports systems. This provides the possibility for a gradual shift towards a better solution. One by one the ports can be transferred to this new system. There is no risc of making existing ports obsolete. Each maintainer is free to stick with the traditional system or choose to start using the new one, with the advantage of providing the port for all BSD systems at once.

The A-A-P project is providing this solution. A-A-P uses a recipe that is similar to the structure of a makefile, but much more powerful and flexible. A recipe is easier to understand than the tricks a port includes to use "make" for something it wasn't designed for. One of the strong advantages of A-A-P is that it handles the details of the packaging system. This is important, because many developers have expressed they are struggling with packaging their application. A-A-P takes care of downloading required files and tools, thereby avoiding the need to edit a script and run "cvsup". This is quicker and more reliable. Many ports done with A-A-P will also work on Linux and other Unix-like systems. Further advantages of using A-A-P for ports will be explained, while the disadvantages and potholes will not be concealed.

www.a-a-p.org

Bram Moolenaar has worked on open-source software for more than ten years. He is mostly known as the creator of the text editor Vim. Currently he is working on a project called A-A-P, which is about creating, distributing and installing (open source) software. His background is in computer hardware, but these days mostly works on software. He still knows on which end to hold a soldering iron though. In the past he did inventions for digital copying machines, until open-source software became his full-time job. He likes travelling, and often visits a project in the south of Uganda. Bram founded the ICCF Holland foundation to help needy children there. His home site is www.moolenaar.net.

All For One Port, One Port For All

Bram Moolenaar
Stichting NLnet Labs
<Bram@A-A-P.org>

The ports system provides a convenient way to install an application from source code. With just a few commands the files for the latest version are downloaded, build and installed. A port specifies patches that need to be applied, allows tweaking features and handles dependencies on other components. These useful features of the ports system have increased the popularity of BSD distributions.

Each BSD distribution has their own ports system. Although they all originate from the same root, incompatible features have been added. This requires a port to be done and maintained for each system separately. Since there are thousands of ports, the amount of duplicated work is significant.

Attempts to reunite the ports systems have failed so far. Examining the reasons for this makes clear that the chances for each BSD system to drop their own solution and use a common ports system are very small. The development of solutions that replace the existing ports systems have stalled.

A possible solution is introducing a new system that exists side by side with the traditional ports system. This allows a gradual shift, moving ports to the new system one by one. Since the ports files of new system do not need to be backward compatible, there is a lot of freedom to make choices for a better and more powerful implementation. The goal that it must co-exist with the traditional ports systems makes sure it avoids the pitfalls that stopped previous reuniting attempts from being successful.

A first version of this new system has been implemented. To avoid the complicated mix of Makefile and shell script the recipe format of the A-A-P project has been used. This first implementation shows the advantages and possibilities of the proposed solution, but also the problems that still need to be solved.

1. HISTORY

The first available code for the ports system was written by Jordan K. Hubbard. The date mentioned in the file, still present in derived works, is August 20 1994 [FreeBSD]. This version did not handle dependencies or downloading of files. Development went fast though, by January 1995 there were already 150 ports [Asami].

In the following years the OpenBSD and NetBSD projects have split off a version from FreeBSD and started adding their own features.

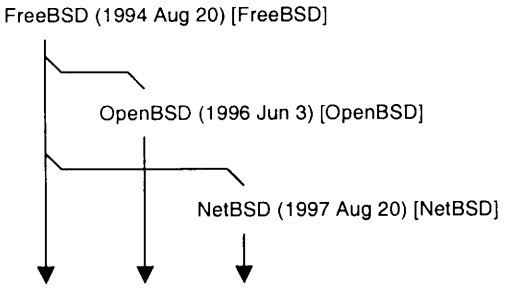


figure 1. History of BSD ports systems

This figure does not show various exchanges of modifications. All three systems have been enhanced over time, which can be seen in the CVS logs [FreeBSD] [OpenBSD] [NetBSD].

NetBSD uses the name "pkgsrc" for their ports system and uses "package" for what others call a "port". To keep it simple we will use the term "port" here for dealing with sources, "package" for an installable set of files with binaries and "source package" for a port that includes the required source files and patches.

Zoularis is the name used for an adaptation of NetBSD pkgsrc to other systems, such as Solaris [Zoularis]. It is not different from pkgsrc in functionality, therefore we will not mention it below.

NetBSD started pkgsrc because the FreeBSD system had a few small problems that needed to be solved (i386 centric, fixed install directory). They apparently didn't try very hard convincing the FreeBSD people to fix this. According to Hubert Feyrer: "it was easier for us to just make the changes we wanted". And FreeBSD apparently wasn't interested in including all the NetBSD improvements either.

OpenBSD first forked off their version with the intention to feed back changes to FreeBSD. A few years back OpenBSD intentionally improved their ports system separately from FreeBSD. Partly to clean up the Makefiles and also to add useful features. Currently there does not appear to be an intention to resync with FreeBSD.

It is clear that incompatibilities between the three ports systems were not added intentionally or because of technical reasons, but are the result of lack of motivation to work together on one system. Being able to quickly change their own version instead of having to convince someone else to include changes also appears to play an important role.

2. BENEFITS OF ONE PORT SYSTEM

The separation into three different ports systems causes several problems:

- Maintenance of each port has to be done three times. This can vary from simple changes, such as updating the version numbers and file names, to handling more complicated issues.
- Bugs and security problems that are not solved by the maintainer of the original sources have to be fixed three times.
- Tricks required to port an application to a BSD system have to be figured out three times.
- Dependencies have to be figured out three times.
- If one BSD system adds a nice feature to their ports system, it is not directly available to the others.

Would there be one ports system, there are several benefits:

- For each port there are more eyes to detect a bug and more hands to fix it. This makes the average reliability of the software considerably higher.
- Ports will be updated quicker and more ports will be available.
- Fewer people are required for maintenance.
- Each BSD system will become more powerful and attractive.
- When a user is making a choice for one of the three BSD systems he no longer needs to exclude systems that do not provide a port he requires. He can make the choice on more relevant issues.

One thing that will not change is the effort required for testing. Each port still needs to be tried out on many different systems.

3. FAILED SOLUTIONS

Despite the obvious benefits from using one ports system, the attempts to bring the BSD systems back together failed so far.

The most promising attempt to make a ports and packages system that should replace all others is Open Packages [OpenPackages]. It was created with the goal to reunite. But the project has stalled, the last news item is dated July 2001. Why? Talking with the developers reveals several reasons.

At first the project consisted of a good idea for reuniting the three solutions. Then the contributors got carried away and wanted to do much more. It became too much work and developers started dropping out. An attempt to redefine the project and split it up in manageable pieces has been made, but there do not appear to be developers who will do the work. Out of the six modules five have no project leader. Possible reasons are:

- Developers do not find the extra features that Open Packages offers important enough to spend a lot of time on.
- Implementing the features is quite complicated. There is much "hidden knowledge" in the existing ports systems and few comments or documentation to explain why certain choices were made. Not many people have the skills to do the work.
- The people who do have the required skills prefer working on their own project (esp. the maintainers of the ports systems of the BSD distributions).

Another alternative is OpenPKG [OpenPKG]. This project appears to promise more than what it can actually do. It says it is portable, but it uses GNU bash shell scripts and an incompatible version of rpm. It also uses many specific system utilities. You can say it could be made portable. Its Linux background and lack of co-operating with the existing packages system make it unattractive to replace the BSD ports systems. When used side-by-side it does not handle dependencies between the two systems.

There are a few other ports and/or package systems, but they do not come close to being a useful replacement for the BSD ports systems. [Install Tools]

4. HOPELESS SOLUTIONS

Instead of switching to a new system, a solution would be to have the existing systems come back together. This would require that features from all three existing systems are included into a common version, with the result that the port files are identical. There are several reasons why this is very unlikely to happen.

The current situation is that FreeBSD is the most popular system and has the largest number of ports, while OpenBSD and NetBSD offer more features. This creates a deadlock: FreeBSD is doing well and does not appear to be interested to change their ports system. NetBSD and OpenBSD have more features, thus do not want to use the FreeBSD implementation.

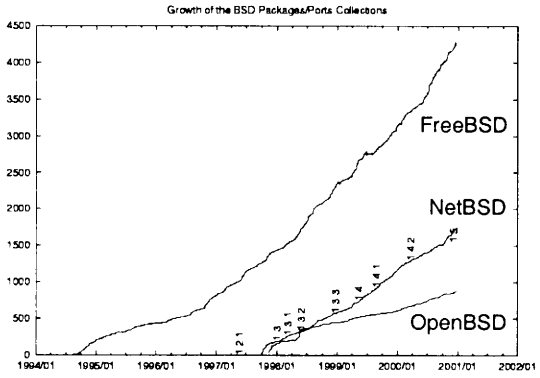


figure 2. Growth of number of BSD ports [Feyrer]

This graph shows the growth of the ports systems over a long period. Recent figures: FreeBSD 7523, NetBSD 3163, OpenBSD 1965. The OpenBSD ports system uses flavours, which reduces their count by an unknown number.

Another important reason is that the developers of each BSD system do not like the idea of giving up the control of their code. They like to be able to decide what goes in and what does not. The releases happen asynchronously, freezes sometimes make it difficult to include updates. Thus a release of one BSD distribution gets in the way of development of the others.

The ports systems are not compatible. Reuniting them means that thousands of ports need to be adjusted. Even when some of the work can be automated, the testing will still be a huge amount of work. None of the BSD systems will want to take this effort in between two minor releases. And doing it for a major release means two versions of each port have to be maintained for a longer time, since a new major release exists besides the previous, stable release branch for more than a year.

How about creating a ports system that is backward compatible with each of the existing ports systems? This could start in such a way that a port contains a big "if" statement, separating the code for each system. Then gradually common

parts can be collected until the system-specific part is made very small. The problem with this is that it would be very complicated to implement. A Makefile is not a programming language. The combination of BSD specific Makefile syntax with shell scripts and the make program re-invoking itself leads to very ugly code. This solution probably leads to the worst code quality possible. And it seems to be impossible to find people who are motivated and have the skills to do this work. Even more so than for the stalled Open Packages project.

5. A NEW SOLUTION

If reuniting is impossible, and existing efforts do not lead to a useful ports system, then what will? What we need is a new application that offers a path for a gradual shift towards a unified ports system. That is the only way to avoid having to rewrite all the ports at once. The existing ports systems will continue to exist, the new solution has to co-operate with it.

In the following we call the existing ports systems "traditional" and the proposed solution the "new" system. Similarly we will talk about a "traditional port" for the existing ports systems and use "new port" for the proposed solution.

Once a port has been made for the new system, it should work on all BSD systems and possibly others. This makes it attractive to create ports for the new system. It is not necessary to transfer all traditional ports to the new system, thus there is no big threshold to start using it.

The required co-operation with the traditional ports systems implies that dependencies can be handled in both directions. A new port can depend on a traditional port and the other way around. And the registration of installed packages must use the existing package system to avoid the need for two sets of commands to find out what packages are currently installed.

This all sounds very nice. The question is how this will be implemented. Using make (BSD or GNU) has the disadvantage of resulting in ugly solutions when it gets complicated. Quoting Jordan Hubbard: "FreeBSD ports is essentially implemented as some very impressive but hairy BSD make(1) macros and can be a little opaque and non-extensible from the perspective of someone looking to extend or re-factor parts of the system" [Hubbard].

Inventing something new specifically for use in the new ports system has the disadvantage of yet-another-file-format. A script language like Python or Perl would work, but still requires adding a lot of application specific functions and variables, thus creating a new file format anyway. And it would be very different from the current ports files, which would discourage quite a few people. Examples are Cons (uses Perl) [Cons] and SCons (uses Python) [SCons].

The solution proposed here uses the A-A-P recipe. This is not the only possible solution, but one that has a good chance of being successful. An alternative might be DarwinPorts [DarwinPorts]. More about that in section 12.

6. INTRODUCING A-A-P

Understanding the proposed solution requires knowing the basic idea of A-A-P recipes.

The A-A-P project provides a portable framework for developing, distributing and installing software [A-A-P]. What is relevant for this paper is the A-A-P recipe. It was specifically designed to replace Makefiles and shell scripts. It provides much more functionality and avoids the dependency on shell commands with specific options and features. A recipe is often portable to many different systems, including MS-Windows.

A recipe is like a Makefile in many ways. The base is formed by specifying the dependencies between files. The build commands of a dependency are used to produce a target file from sources. Everyone using Makefiles will quickly understand the structure of a recipe. Here is an example for compiling a C program:

```
myprog : main.c version.c extra.c
    :do build $source
```

The ":do" command invokes a build action. It detects that the sources are C code and decides to use a C compiler. How the compiler will be invoked depends on the system. This is automatically detected or comes from a configuration file. This separates the system-independent specification of what needs to be done from the system-dependent details. It is also still possible to invoke a shell command when portability is not required.

Some of the advantages of using the A-A-P recipe instead of a Makefile for building a program:

- Signatures are used instead of timestamps, this avoids problems with networked file systems

and files unpacked from an archive. Timestamps can still be used when desired.

- Dependencies on included files are handled automatically. There is no need for running "make depend".
- Building for different systems from one set of sources is handled automatically, a separate directory is used for intermediate results (object files) of each system.
- Rules for line continuation are flexible, backslashes are not often needed. This avoids the most common mistakes. The amount of indent is used to indicate where a command ends. Example:

```
SOURCE =    main.c
            version.c
            extra.c
TARGET =    myprog
```

This actually does almost the same as the previous example. The SOURCE and TARGET variables are turned into a dependency automatically.

For often-used tasks the built-in commands can be used. These are similar to common shell commands, but with extra features. For example, the ":copy" command can copy files specified with a URL (http:, ftp:, scp:, etc.). This greatly reduces the use of specific system commands and improves portability. Some of the built-in commands are:

`:copy`	copy files and directories
`:move`	rename/move files and directories
`:mkdir`	create a directory or directory tree
`:delete`	delete a file or directory tree
`:cat`	concatenate files
`:include`	include a recipe file
`:child`	read a sub-project recipe file
`:execute`	execute a recipe
`:system`	execute a shell command
`:update`	execute build commands when a target is outdated

These commands can not only be used in build commands, but also at the top level. This makes it possible to use a recipe like a shell script. This is one of the advantages of an A-A-P recipe over using a Makefile that is important when implementing a ports system.

For control flow and expressions Python script can be used. This provides a powerful and well-defined syntax that combines pretty well with the Makefile-like syntax of the recipe (e.g., comments start with "#" and amount of indent is significant). Python libraries provide functionality to handle almost any task and make it possible to avoid system-specific code. Example:

```
SUBDIR = sub
USE_SUBDIR ?= 0
@if USE_SUBDIR:
    SOURCE += `glob(SUBDIR + "/*.c")`
```

The line starting with "@" in this example is a Python command. The USE_SUBDIR variable can be set by a non-Python assignment and is used by the Python command. Thus the variables available in the Python code and the non-Python code are the same. In the last line a Python expression in backticks is used. The Python glob() function expands wildcards and the resulting list of files is appended to the SOURCE variable.

The A-A-P recipe has uploading and downloading functionality built-in. For example, a file can be downloaded automatically by specifying where it is to be obtained:

```
EXTRA_SOURCE = extra.c {refresh =
    ftp://ftp.foo.org/pub/files/extra.c}

xfoo : $SOURCE $EXTRA_SOURCE
```

In this example, if the "xfoo" target is updated and the file "extra.c" does not exist locally, it will automatically be downloaded. The text between { and } specifies an attribute. Attributes provide a generic mechanism to attach meta information to a file name.

Uploading is done in a similar way. A recipe like this is used to update the A-A-P web site:

```
FILES =  `glob("*.html")`
         `glob("images/*.png")`

:attr {publish =
    scp://vimboss@vim.sf.net/vim/%file%}
    $FILES
```

The file names are assigned to FILES using the Python glob() function. The destination of the files is added by attaching the "publish" attribute to the file names. Executing this recipe with "aap publish" will cause each file with a "publish" attribute to be uploaded. The uploading is skipped for files that did not change since the last upload.

A-A-P is a generic tool, a sort of a super-make. You can use it to develop software, distribute files, download, install, etc. More information can be found on the web site [A-A-P].

7. PORTS WITH A-A-P

Since A-A-P recipes are powerful and still resemble Makefiles, they form an excellent base for implementing a new ports system. When using a recipe for a port file, in many cases it will not be necessary to use the extra A-A-P features and the new port mostly looks like a traditional port. When more complicated tasks are to be performed, the recipe file offers the functionality in a nice way. A first implementation of this new ports system has been made.

A user of the traditional ports system usually performs these steps:
1. become root
2. update the whole ports tree with cvsup
3. build and test: "cd group/appname" "make" "make test"
4. install: "make install"

Using a new port the usual steps are:
1. download or update a port recipe (one file)
2. build and test: "aap", "aap test"
3. try it out: "aap install DESTDIR=$HOME"
4. install: "aap install"

The new port works in a similar way as the traditional port. Dependencies will be handled where needed, files are downloaded and patches applied. The most important differences of using a new port are:
- It can be run anywhere, it does not need to happen in /usr/ports.
- There is no need to obtain the whole ports tree before installing one port. Only parts that are actually used will be updated.
- Not doing the downloading and building as root is much more secure. For installing a package (also for dependencies) the root password must be entered once.
- To try out a port it can be installed for one user.
- Updating to a new version is simply done with "aap refresh".

Not all of this is easily implemented and quite a few choices need to be made. The most important issues will be discussed in the following sections.

8. USING PACKAGES

There are two basic methods for installing a port:
1. The port directly installs the files to their final location. A binary package can be created after this. Recording the port as being installed is done separately from the actual install.
2. The port installs the files into a temporary directory. This is often called a "fake install". A binary package is created from these files. The binary package is then installed and registered as being installed.

The second method has many advantages. It avoids accidentally overwriting existing files. The first method is actually impossible when a binary package is to be created without installing it, an already installed package using the same files would be corrupted. The second method also makes sure that installing the port gives the same results as installing the binary package.

A disadvantage of the second system is that for some ports it involves extra work to make the installation put the files in the temporary directory instead of /usr/local. This is a small price to pay, therefore the choice was made to use the second method.

Since the package administration is not the same on all systems, A-A-P leaves the work of installing the binary package to existing system tools.

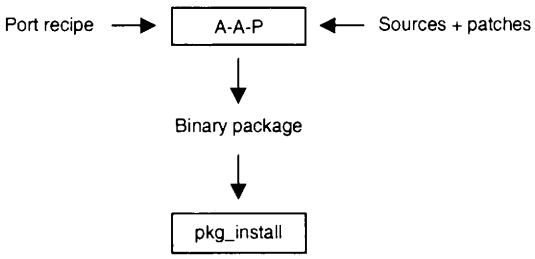


figure 3. Connection between A-A-P and the package system

A disadvantage of this system is that not all package tools support sufficient features. Desired features are:
- Dependency handling on a range of package versions and with wildcards.
- Possibility to install two versions of the same package at the same time.
- Support for the sequence: Install version 1.1, install version 1.2, verify that version 1.2 works well, delete version 1.1.

These are generic problems and separate from the porting issue. They should be solved in the package tools. Adding another set of package commands next to the existing ones is not a good idea, since the existing commands will not be aware of packages installed with the new commands. This must really be solved by

improving the existing package tools. Until this has been implemented the new ports system will accept the limits of the existing commands. Some issues could be handled by adding a pre-install script to the package, e.g., for handling dependencies with wildcards. However, this causes new problems, the time would be better invested in improving the package system.

A source package is nothing more than an archive containing the port recipe with all required source files and patches. No downloading will be needed then. Otherwise the building and installing works just like using the port recipe.

9. DEPENDENCIES

The dependency checking is split up in two parts:

1. Verifying the dependencies can be met. This happens before archives and patches are downloaded, so that wastefully downloading something that will not work is avoided.
2. Installing ports and/or packages that are required happens just before they are needed. This reduces cyclic dependency problems.

There is no need to update all ports before installing one. To figure out the dependencies only the port recipe has to be obtained. Normally, when a recipe can be found on the system that meets the dependency it is used. It is also possible to specify that the latest version of the recipe must be obtained.

There is no need to specify the directory (e.g., "editors/emacs20-mule-devel") for a dependency. This has always been confusing, especially for ports that exist in more than one place or are moved. A unique name is required anyway. This also allows including a version number in the directory name or adding a subdirectory with versions, so that several versions of a port can co-exist. A simple, automatically generated index file is used to locate an application locally. The same can be done on a server that provides ports for downloading.

Alternatively, unofficial ports can be obtained from various locations. This is especially useful for ports under development that depend on ports that have not been committed yet. For stable ports this should not be used.

As mentioned above, the dependencies that can be specified in a package are not always sufficient. It might be necessary that the dependencies in the binary package specify fixed version numbers, thus are less flexible. Therefore the dependencies of the ports recipe will be used when the recipe is available.

Besides the dependencies on ports and packages that need to be installed, the A-A-P recipe also offers features to check for installed tools and decide how to build the application. This automatic configuration is useful to reduce the number of dependencies for applications that do not use autoconf and do allow specifying optional features when building.

10. BACKWARDS COMPATIBLE

To be able to work properly side-by-side with the existing ports system, the dependency of a new style port on a traditional port must be handled. This is not different from dependencies between traditional ports. A-A-P will invoke the traditional ports tools. The knowledge of how this is done on different systems is build into A-A-P. The user will only need to do a bit of configuration if he is not using the standard setup.

The dependency of a traditional port on a new port requires a bit more work, since the traditional port does not know about the existence of the new ports system. A wrapper port is required to make the connection. Most of this wrapper is the same for all wrapper ports, since the actual building is done with the port recipe. There is no need to specify items like MASTER_SITES, for example. What the wrapper port still needs to do:

- Specify the required items, such as port name and version number.
- Specify the dependencies. Not all of them need to be included, the recipe can also handle them. Including them in the wrapper port has the advantage that several tools will be able to find them.
- Specify a dependency on A-A-P itself, so that it will be installed when necessary.

```
# A-A-P port recipe for Vim
AAPVERSION =      1.0

PORTNAME =        vim
PORTVERSION =     6.1
MAINTAINER =      Bram@vim.org

CATEGORIES =      editors
PORTCOMMENT =     Vim - Vi IMproved, the text editor
PORTDESCR << EOF
This is the description for the Vim package.
A very nice editor indeed.
You can find all info on http://www.vim.org.
                  EOF

# Where to obtain an update of this recipe from.
AAPROOT =         http://www.a-a-p.org/vim
:recipe {refresh = $AAPROOT/main.aap}

WRKSRC =          vim61                              # Vim does not use vim-6.1
DEPENDS =         gtk>=1.2<2.0 | motif>=1.2          # GTK 2.0 does not work yet
BUILDPROG =       make

# This is used when CVS is available
CVSROOT ?=        :pserver:anonymous@cvs.vim.sf.net:/cvsroot/vim
CVSMODULES =      vim
CVSTAG =          vim-6-1-003

# This is used when CVS is not available or when disabled with "CVS=no".
MASTER_SITES =    ftp://ftp.vim.org/pub/vim
PATCH_SITES =     $MASTER_SITES/patches
DISTFILES =       unix/vim-6.1.tar.bz2
                  extra/vim-6.1-lang.tar.gz
PATCHFILES =      6.1.001 6.1.002

#>>> automatically inserted by "aap makesum" <<<
do-checksum:
     :checksum $DISTDIR/vim-6.1.tar.bz2 {md5 = 7fd0f915adc7c0dab89772884268b030}
     :checksum $DISTDIR/vim-6.1-lang.tar.gz {md5 = ed6742805866d11d6a28267330980ab1}
     :checksum $PATCHDISTDIR/6.1.001 {md5 = 97bdbe371953b9d25f006f8b58b53532}
     :checksum $PATCHDISTDIR/6.1.002 {md5 = f56455248658f019dcf3e2a56a470080}
#>>> end <<<
```

11. EXAMPLE

The example shows some of the A-A-P port recipe features. Most of the variables are the same or similar to the traditional ports. A few items deserve an explanation:

- AAPVERSION indicates the version of A-A-P this recipe was written for. When the version of A-A-P actually used is older it will produce an error. When it is newer it will behave like the indicated version would.
- PORTCOMMENT and PORTDESCR are included in the recipe. Only one file needs to be downloaded to obtain a port.
- The ":recipe" command specifies where an update of the port recipe is available. The command "aap refresh" will get it.
- DEPENDS specifies that either GTK or Motif is required, both for building and running Vim. For GTK the version must be 1.2 or higher, but below version 2.0. Motif version 1.2 and higher is accepted. If neither is currently installed the first package mentioned is installed, in this case GTK, with the highest acceptable version that can be found.
- Vim is configured and build with "make", this is specified by setting BUILDPROG. If configure would have to be run first a "pre-build" target is to be defined. This allows the port maintainer to perform the configuration exactly as he wants to, without the need to know about special variables.
- CVSROOT indicates the files are available through CVS. This is the preferred method to obtain the source files, because it includes all the latest patches. "CVS=no" can be used to disable using CVS.
- When CVS is not used the DISTFILES are downloaded. The checksums are also included in the port recipe. This is done by the port maintainer with the command "aap makesum".
- There is no list of installed files. It is generated automatically by doing a fake install and finding the files ending up in the fake root. For Vim this works as expected. For other applications it might be required to specify the files explicitly.

12. WILL IT WORK?

The big question is whether the proposed solution will actually catch on and a substantial number of ports will become available. Will A-A-P succeed where others have failed? The above text has explained that there is no fundamental showstopper. But the solution is not without disadvantages:

- Python is required. Not everybody likes it, the performance is less than with a C program and it is not a standard part on all systems.
- Yet another tool to learn to use.
- It does not solve the problems with packages.

There are many advantages:

- Using Python is much better than a mix of BSD make and shell script.
- No tricky solutions are needed, such as how a different master site list is selected by adding ":2" to the file name; the sites to be used for a file can be specified directly with an attribute.
- It is easy to use several versions of a port (stable, current, alpha).
- Only the actual install on the system needs to be done by root.
- Ports can work on many Unix systems.
- A-A-P is still under development, this provides the possibility of adding up all knowledge of existing ports systems. There is much freedom to specify the ports recipe format in a good way.

The proposed new ports system with A-A-P looks more attractive than other solutions. Especially the possibility to use it side by side to a traditional ports system, this allows users to try it out and get used to it. Still, whether it will attract a substantial audience remains unpredictable. When the A-A-P recipe is used for other purposes (developing and distributing software) it also becomes more likely that a ports system based on it will be successful. This should become clear the coming year.

An alternative for using A-A-P might be DarwinPorts [DarwinPorts]. This project also decided that a script language is needed to avoid the problems with Makefiles, they chose TCL. The file format looks more different from a traditional port than the A-A-P recipe, but not as much as Cons or SCons. What makes it interesting is that the "father of BSD ports" Jordan Hubbard is involved in DarwinPorts. However, it is still new and currently only working for Mac OS X 10.2. Support for FreeBSD is planned and the people behind OpenPackages recently expressed they will join with DarwinPorts. The main drawback of DarwinPorts is that it does not co-operate with the existing ports and packages systems. It registers installed packages in its own way, storing TCL procedures instead of shell scripts. Making the switch from the traditional package system to DarwinPorts will be difficult.

13. CONCLUSION AND CURRENT STATUS

The proposed solution is to create a new ports system with A-A-P. This system has enough similarities with the traditional ports systems to avoid a steep learning curve and at the same time offers many improvements. This solution does have a good chance of providing a united ports system for the BSD systems. The possibility to use it next to the existing ports systems avoids many of the problems that made other solutions fail.

A-A-P is still under development. Version 1.0 is expected spring 2003. The author of this paper will be working full-time on A-A-P. This means the project will not stall. The speed of developments will depend on contributions from others.

The A-A-P ports system currently works for a few examples. Before a large number of ports are to be made, the syntax of the port recipe must be ascertained. This requires that useful features from the various ports systems are included and the consistency of the result is checked. Before it can be used for stable systems a lot of testing is required. Thus there is still quite a lot of work to be done.

In between the writing of this paper and the presentation on the European BSD conference 2002 more progress will have been made, an update will be given in the presentation. Further progress will depend on reactions on this paper.

REFERENCES

[A-A-P]	The A-A-P project: http://www.A-A-P.org
[Asami]	Usenix 1999 presentation by Satoshi Asami: http://www.usenix.org/events/usenix99/full_papers/asami/asami.pdf http://people.freebsd.org/~asami/presen/usenix99/html/index.html
[Cons]	http://www.dsmit.com/cons/
[DarwinPorts]	http://www.opendarwin.org/projects/darwinports/
[Feyrer]	NetBSD packages growth compared to FreeBSD and OpenBSD, made by Hubert Feyrer: http://netbsd.org/Documentation/software/pkg-growth.html
[FreeBSD]	FreeBSD CVS log for bsd.port.mk: http://www.freebsd.org/cgi/cvsweb.cgi/ports/Mk/bsd.port.mk
[Hubbard]	DarwinPorts FAQ: http://www.opendarwin.org/projects/darwinports/faq.php
[Install Tools]	Overview of tools: http://www.A-A-P.org/tools_install.html
[NetBSD]	NetBSD CVS log for bsd.pkg.mk (long!): http://cvsweb.netbsd.org/bsdweb.cgi/pkgsrc/mk/bsd.pkg.mk
[OpenBSD]	OpenBSD CVS log for bsd.port.mk: http://www.openbsd.org/cgi-bin/cvsweb/ports/infrastructure/mk/bsd.port.mk
[OpenPackages]	http://www.openpackages.org/
[OpenPKG]	http://www.openpkg.org/
[SCons]	http://www.scons.org/
[Zoularis]	http://www.netbsd.org/zoularis/

RELEVANT LINKS

FreeBSD CVS log for ports/INDEX with Asami' s song texts:
http://www.freebsd.org/cgi/cvsweb.cgi/ports/INDEX
FreeBSD porters Handbook: http://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook
OpenBSD: "Building an OpenBSD port" http://www.openbsd.org/porting.html
OpenBSD: "Important differences from other BSD projects" http://www.openbsd.org/porting/diffs.html
NetBSD packages collection (pkgsrc): http://www.netbsd.org/Documentation/software/packages.html
NetBSD pkgsc documentation (well written, mentions differences from FreeBSD):
ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/Packages.txt
NetBSD bsd.pkg.mk: ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/mk/bsd.pkg.mk

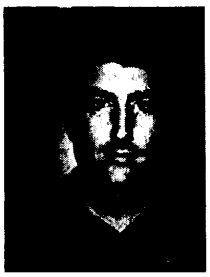
BIOGRAPHY

Bram Moolenaar has worked on open-source software for more than ten years. He is mostly known as the creator of the text editor Vim. Currently he is working on a project called A-A-P, which is about creating, distributing and installing (open source) software. His background is in computer hardware, but these days mostly works on software. He still knows on which end to hold a soldering iron though. In the past he did inventions for digital copying machines, until open-source software became his full-time job. He likes travelling, and often visits a project in the south of Uganda. Bram founded the ICCF Holland foundation to help needy children there. His home site is www.moolenaar.net.

Advanced VPN support on FreeBSD systems

Riccardo Scandariato
Politecnico di Torino
<scandariato@polito.it>

Currently, the VPN support offered by FreeBSD is quite limited: it provides a way to establish tunnels but it does not consider the problems of multiple VPNs concurrently deployed on the same machine. Our implementation enables the provisioning of VPN services on FreeBSD by extending its routing capabilities. Primarily, our implementation provides support for multiple IP routing tables (one for each VPN) in order to avoid conflicts between overlapping address spaces of different VPNs. User-space programs (route, zebra, etc) can select the proper instance when accessing/updating a routing table through a modified routing sockets interface. At the kernel level, the different VPN flows are identified on per-interface basis. Once a packet is recognized as belonging to a given VPN, the forwarding routine selects the proper routing table to look up. We also modified several user-level applications to allow the exploitation of the new routing infrastructure, such as "route", "netstat", "zebra", and "ifconfig".



Riccardo Scandariato received his master degree in telecommunication engineering at Politecnico di Torino on December 2000. Currently he is a PhD student at the Control and Information Engineering Department of Politecnico di Torino.

Advanced VPN support on FreeBSD systems

Riccardo Scandariato, Fulvio Risso
Politecnico di Torino, Italy
{scandariato, risso}@polito.it

Abstract— **Currently, the Virtual Private Network (VPN) support offered by FreeBSD is quite limited: it provides a way to establish tunnels but it does not consider the problems of multiple VPNs concurrently deployed on the same machine. Our implementation enables the provisioning of VPN services on FreeBSD by extending its routing and forwarding infrastructure. We adopted the virtual router approach, by adding support for multiple routing tables. Forwarding kernel modules have also been modified accordingly. We also improved several user-level applications (e.g. route, ifconfig, zebra) to allow the exploitation of the new routing infrastructure.**

Keywords— **Provisioned IP VPN, virtual router, GRE tunnel, FreeBSD**

I. Introduction

THE Internet, originally born as an academic-based infrastructure, is rapidly evolving toward a generic network in which academics, business, and several other worlds are coexisting. From the pure networking perspective (i.e. we do not intend to take into account any application issue), one of the problems of the nowadays public IP networks is the lack of support for IP private addresses. At a glance, supporting a private addressing schema on a public IP network seems to be a non-sense. However, from the perspective of companies with a wide area network infrastructure, this is a strong requirement since the IP public network is becoming a way to connect together their branch networks around the world (and saving money). This is the well-know topic under the name of Virtual Private Networks (VPN), i.e. networks that use a public IP infrastructure to connect together several pieces with private addressing (and with the need of secured communications). The biggest issue in VPN support is that the IP protocol did not foresee the need of multiple overlapping addresses spaces, so that applications like VPNs introduce a high degree of complexity in the management of the IP network. The idea of a VPN is not a novelty in the networking world. The novelty is that, so far, companies created their private network by using a data-link infrastructure (for example point to point links, or X.25 / Frame Relay / ATM accesses) provided by a telecom provider. That infrastructure was simply a private network (i.e. a network that allowed only the employees to access to the resources of the company) build on top of a public network (the public telephone network instead of the public X.25 network or whatever). In other words, the basic technology changes while the underlying concept of VPNs does not.

In order to support efficiently VPN services, there are two main options:

- the presence of the VPN is relegated to the access side of the network. This means that any VPN is hidden in the core (i.e. in the public part of the IP network) and the complexity is inserted into the edge routers. These routers must map the private address space into a public space, deliver packets to the proper destination, and then map the packets back.
- the presence of the VPN is well known inside all the network (backbone included), so that the routers must be aware that overlapped address spaces exist and they must be able to cope to this problem.

The first option is far simpler, but (in general) it does not allow creating an optimized virtual network from the topological viewpoint. The second option is more complex, but the routing into the network can be highly optimized. The present solutions (MPLS, tunneling, ...) chose the first method to deal with these problems. In our mind, however, these solutions cannot be used to configure plug and play networks, nor can be used to create large virtual infrastructures.

This work wants to present the lessons learned by modifying the forwarding path of a software router (FreeBSD 4.4) in order to support VPNs in the network backbone. Our effort was devoted to change the forwarding mechanism in order to support overlapped address spaces. Each router is then able to find the proper route to each packet by checking at the destination address (contained into the packet) and an additional parameter identifying the VPN the packet belongs to. However, this choice implies several other components to be modified in order to support the VPN into the backbone. Routing protocols, for example, need to be modified in order to be aware of what VPN they are currently computing the best path. Therefore, several other components (detailed in the following) have been modified in order to provide seamless VPN support.

The rest of the paper is organized as follows. Section II introduces the functionalities that are needed for concurrent VPN service provisioning. Section III describes the modification introduced into the FreeBSD kernel and into some user-space applications in order to implement such functionalities. Section IV discusses the related work, and, finally, the conclusive remarks and further work are presented in Section V.

II. Objective

As mentioned in Section I, VPN services can be implemented either on top of access routers only, or cooperatively afforded by all backbone routers. In the first case ([1], [2]), VPN traffic is identified when entering the backbone network, and then delivered to the opposite network edge by means of tunnels. Core routers are unaware of VPNs since they forward VPN tunneled traffic by looking up the destination address of the outer IP header (see later). All the VPN-specific work is done by access routers sitting at the backbone edge. For instance, routing information about VPN destination is exchanged only between access nodes. On the other case, VPN traffic is tagged at access node and sent through the core without encapsulation. All the core nodes forward packets using the couple (destination address, VPN_ID tag). This means that routing information about VPN destinations must be available to all the backbone nodes,

which have to maintain separate routing table (one for each tag, i.e one for each VPN).

In the first case, VPN packets are forwarded edge-to-edge across tunnels, which traverses multiple physical link (see tunnels between `freebsd` and `dante` traversing a third node in Figure 1). Obviously, since core nodes (e.g. `core001` in Figure 1) look up packets by using only the outer destination address, VPN packets (and tunnels) follows the physical paths governed by the "real" IP network (hereafter called the *base* network, in opposition to *virtual* networks). In other words, a single hop on the virtual network implies multiple hops on the base networks. This causes many problems if traffic engineering or QoS guarantees must be applied to tunnels, since the routing instances of the two types of networks (base and virtual ones) are unrelated. However, this solution simplifies the management of the core side.

The second case is logically equivalent to a network where all the backbone nodes (both access and core) play the role of VPN routers, and where tunnels are established on each physical link. Hence, a single hop on the virtual network coincides with an hop on the base network [1]. Hence, all the conventional techniques for traffic engineering and QoS can be applied to VPNs. The drawback is a huge complexity in the core side.

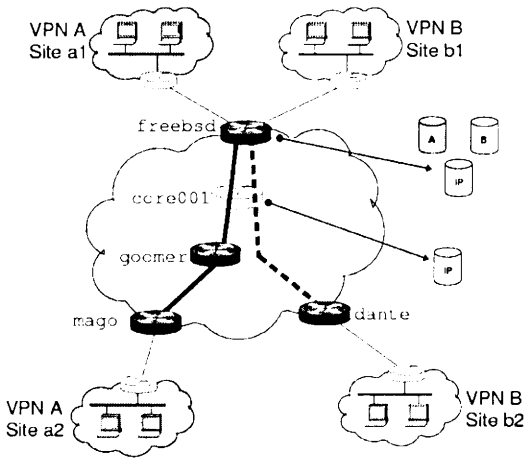


Fig. 1. Backbone network provisioning multiple VPNs

Our solution tries to merge the benefits of both the above approaches. Figure 1 shows a sample scenario that can be deployed by adopting our implementation. The figure depicts four customer sites connected to a provider network (the bigger cloud in the middle). The backbone is made of VPN routers (see darker nodes) and VPN-unaware routers (e.g. node `core001`). All the nodes providing access to client sites (`freebsd`, `mago`, `dante`) must be VPN nodes. Further, also some nodes in the core (`goomer`) can be promoted to be VPN router. VPN routers run the modified version of FreeBSD providing multiple table support and virtualized forwarding, as described in Section III. Having VPN routers in the core gives an increased flexibility for VPN deployment. For instance, a central node can be used as tunnel concentrator, in order to apply traffic filters to a given VPN, or to merge traffic originated in different VPNs. We recall that packet are available in clear, i.e. not tunneled, only at tunnel termination points, hence, such operation can be performed only by a VPN router. In Figure 1, a pure edge-based adopted was adopted for `VPN B` (see the dashed edge-to-edge tunnel between access nodes labeled as `freebsd` and `dante`), while `VPN A` uses an intermediate VPN node for traffic delivery (see the couple of tick tunnels between `freebsd`, `goomer`, and `mago`). Note that all VPN routers are maintaining a dedicated routing table for each VPN they are serving, additionally to the base network IP table. Hence, `freebsd` has to maintain 2+1 tables, while other VPN nodes are maintaining 1+1 tables. Non-VPN nodes just maintain the IP base table.

As highlighted by Figure 3 an Figure 2, access and core nodes fulfill different tasks. This are detailed in next two sections.

A. *Access router functionalities*

Figure 2 shows the detail of the `freebsd` VPN router. It is placed at the network edge and it has three physical interfaces:

- `eth0` is an Ethernet interface connecting client site VPN_A.a1 (site a1 of VPN A). This site is using the 10.0.1.0/24 address space.
- `eth1` is a second Ethernet interface connecting client site VPN_B.b1. This site is using the 10.0.1.0/24 address space too.
- `eth2` attached to the backbone core and has a public IP address.

Bounded to eth2, there are two pseudo-interfaces (also called virtual interfaces), that can be created dynamically:

- `gif0` is a GRE (Generic Routing Encapsulation, [3]) interface. This interface represents the tunnel end-point used to forward traffic of VPN A. The GRE interface is assigned (by means of the `ifconfig` UNIX command) with a private address out from the VPN A address space.
- `gif1` is another GRE interface. This interface represents the tunnel end-point used to forward traffic of VPN B. The GRE interface is assigned with a private address out from the VPN B address space.

The binding between virtual interface and the physical interface is done by means of the `gifconfig` BSD command, which configures the source and destination addresses to be used in the outer header.

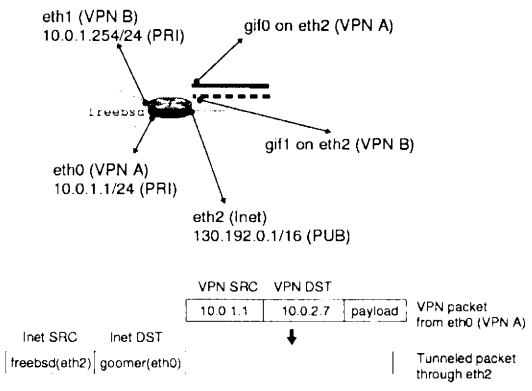


Fig. 2. VPN access router managing multiple VPNs

[1] Actually, while in the first case networks are layered (being virtual networks on top of the base networks) in the second case networks are sided (being the base network just one of the existing parallel networks).

In order to properly serve the two VPN client sites, the node must implement the following functionalities:

- *Routing virtualization.* The node must have multiple IP routing tables (one for each VPN) in order to support the overlapping address spaces within the two different VPNs. Each routing table contains path informations about all the other VPN destinations. Each route contains the address of a peer tunnel end-point (i.e. a `gif` on a remote tunnel-connected VPN router) as next hop. In our solution, each routing table is updated by a dedicated routing protocol instance (`zebra-OSPF`) running on top of the virtual network. This means that routing advertisement are tunneled too.
- *Incoming traffic identification.* The node must associate each packet incoming from the `eth0` and `eth1` interfaces to the corresponding VPN. To this aim, our solution "colors" the ingress physical interface by tagging them with a VPN identifier. This solution is straightforward to implement. However it imposes some limitations, as described in Section V.
- *Forwarding virtualization.* Upon reception of a packet from the client site, the forwarding module must be able to select the proper routing table according to the identified VPN. After having looked up the correct table, the forwarder must sent out the packet on one of the tunnels that have been configured for the given VPN. Selection of the right tunnel is determined by the VPN-level routing information.
- *Tunneling.* As shown in the lower part of Figure 2, once an outgoing tunnel has been selected, the corresponding `gif` module is responsible of encapsulating (i.e. of adding the outer header) the packet before transmission. This functionality (differently from all the above items) is already available in FreeBSD, and was not implemented.

Note that above we explained the case of a packet arriving from the client site. Obviously, the same operations must be pursued in case of delivery to the site of a packet arriving from the core.

B. Core node functionalities

Figure 3 shows the detail of the `goomer` VPN router. It is placed in the network core and it has two physical interfaces connecting him to other routers of the backbone. These interfaces are assigned with addresses out from the provider public address space.

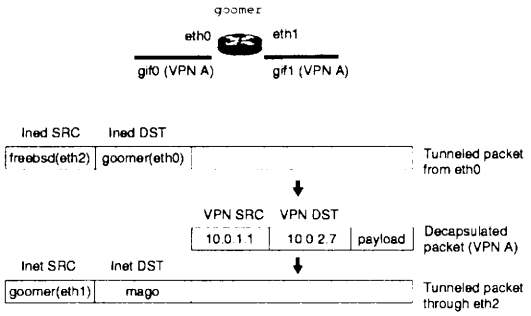


Fig. 3. VPN core router acting as a tunnel switch

The router also has two tunnel end-points, both terminating tunnels that belong to the VPN A:

- `gif0` represents the tunnel end-point used to forward traffic of VPN A to/from `freebsd`. The GRE interface is assigned with a private address out from the VPN A address space.
- `gif1` represents the tunnel end-point used to forward traffic of VPN A to/from `mago`. This interface is assigned with a private address out from the VPN A address space too.

Note that `gif0` is bounded to `eth0`, while `gif1` is bounded to `eth1`.

The node acts as a tunnel switch, that is it receives VPN traffic from a (incoming) tunnel and forwards it to another (outgoing tunnel). This means that the node has to decapsulate the incoming VPN packet, looking up the destination of the inner header towards the proper table, and then encapsulating the packet again (but with a new outer header). This operation is described in the lower part of the Figure 3.

With respect to the access case, the difference is in the traffic identification operation that is much more simpler, since the VPN traffic already arrives in a tunneled manner. Hence, a tag applied to the tunnel (actually the tunnel end-point) will serve the aim perfectly. Differently from Section II-A, this solutions does not involve any limitation, since end-point can be created dynamically in any desired number (up to the kernel configured limit). The only problem (as shown in Section V) is that the current tunneling implementation does not support more than one configured tunnel between a couple of IP addresses.

III. Implementation

This section outlines the strategy adopted to integrate the VPN functionalities described in Sections II-B and II-A into the FreeBSD 4.4 operating system. Since we refer frequently to the internals of FreeBSD, the reader unfamiliar with the networking architecture of a BSD-like system can refer to [4] and [5]. The detailed description of all the kernel/application modification can be found in [6], or directly in the source code [7].

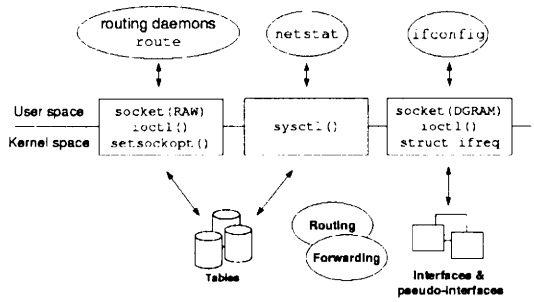


Fig. 4. Roadmap of VPN functionalities

Figure 4 gives the roadmap of modifications introduced by our implementation. Actually, the total amount of modified lines of code is very small, since our aim was to implement our solution in the most simple and clean way. Further, we tried to realize the most harmonic solution, with respect with the existing (i.e. original) FreeBSD code. Improvements were introduced both in the kernel space, and to application programs in the user space. Consequently to the introduction of new features in the kernel, we also modified the interface provided by some system calls, in order to provide applications with the new kernel

capabilities.

At the kernel level, the most important modification introduced the support for the on-demand creation of multiple routing tables, and for the tagging of interfaces. This features are exported by modified versions of the `socket()` and `ioctl()` system calls (an the modified versions of the related structures and ancillary functions). These features are exploited by utility programs (such as `route` and `ifconfig` respectively), and routing applications (such as `zebra-ospfd`, [8]), that were both modified. We also upgraded the `sysctl()` system call, providing bulk access to the routing table. This call is mainly used by the `netstat` program to get all the routing table at once. Hence we modified `netstat`, which is now capable of accessing the different tables. Obviously, the main part of the work was dedicated to the modification of routing mechanisms (table management) and forwarding functions (`ip_input()`, `ip_forward()`) in the kernel space.

The following sections detail the introduced variants to the FreeBSD system. In particular, Sections III-A and III-B present the improvements to the routing and forwarding modules respectively, while Section III-C presents the improvements we made to the user-space routing applications.

A. *Multiple tables*

FreeBSD supports many network protocols, such as IPv4, IPv6, IPX, etc. Since each one uses a single (and peculiar) addressing scheme, the protocol is internally identified by means of its *address family* number. For instance, the constant `AF_INET` identifies the IPv4 protocol, while the constant `AF_OSI` identifies the OSI protocol. The kernel assigns a dedicated routing table to each protocol (family), and tables are implemented as Patricia's trees, which are data structures optimized for longest-prefix-match searches. Tables are stored in a the `rt_tables[AF_MAX+1]` array, where `AF_MAX` is a constant representing the number of defined address families (i.e. the number of supported transport protocol). Each element of the array contains a pointer to the radix node of the corresponding tree-based table: for instance `rt_tables[AF_INET]` points to the IPv4 routing table. Tables are created and initialized at system startup by the `route_init()` function, which iteratively calls the `rn_inithead()` function, once for each family.

In order to support multiple routing tables for the `AF_INET` family, we defined an additional array as follows

```
(sys/socket.h) #define VPN_MAX 100
(net/route.h)  struct radix_node_head *
               vpn_rt_tables[VPN_MAX+1];
```

The maximum number of tables (and hence VPNs) is is limited by the `VPN_MAX` constant, since the array is statically allocated. To support a higher number of VPNs, the kernel must be recompiled with a different constant value. This choice is due to efficiency reasons. Further, the first element is not used, since the zero value is reserved to identify the base IP table. This design trick considerably reduces the number of modifications to the original kernel code (as explained later). Note that table structures are not created at startup time as above: initially, the vpn table array is empty, and Patricia's trees are created and initialized on-demand when a new VPN must be supported by the local node.

Once the multiple table support was introduced, we modified the interface providing the read/write access to the tables. User space programs communicate with kernel functions that manage the routing tables by means of *routing messages* exchanged through *routing sockets*. Routing messages are data structures defined in kernel headers that the programs fill in accordingly to the operation they want to execute on the table (e.g. route add, route delete, read, etc.). Routing sockets are created through the standard `socket()` system call, by specifying proper arguments. To allow the selection of the target table to which the operations must be executed, we introduced a new field (`so_vpnid`) in the `socket{}` data structure, as shown in Figure 5.

```
(sys/socketvar.h)
struct socket {
  short   so_state;  /* internal state */
  caddr_t so_pcb;    /* control block */
  ...
  u_int   so_vpnid;  /* VPN_ID -  ADDED */
}
```

Fig. 5. Socket data structure with VPN identifier

The `so_vpnid` field is initialized to zero by the `socreate()` function, when a new socket is requested through the `socket()` call. If the field is unmodified, all the routing messages will affect the base IP table (recall that VPN 0 is reserved). Thus, to select a table, the application program must set the `so_vpnid` field to a non-zero value, corresponding to the desired target table. To this aim, a modified version of the `ioctl()` system call is provided, which accepts the new `SIOCSVPNID` (set) and `SIOCGVPNID` (get) arguments. Alternatively, the VPN_ID can be set/read by means of a modified version of the `setsockopt()/getsockopt()` calls, respectively (which accept the new `SO_VPNID` socket level option). A sample code showing the use of the modified socket interface is provided in Figure 6.

At the kernel level, we modified the functions that process routing messages. Messages are first received by the `route_output()` routine. In case of read requests (`RTM_GET` is specified in message headers, similar to line 7 of Figure 6), it calls the `rnh_lookup()` function; otherwise, if the message requests a table modification (`RTM_ADD`, `RTM_DELETE`), it calls the `rtrequest()` routine. This latter, selects the target table on the basis of the address family of the route to be added or deleted.

We modified the default behavior of the `route_output()` function. Since, the `rtrequest()` cannot infer the VPN_ID from its input arguments, we were obliged to redefine the function as `vpn_rtrequest()`, which receives the VPN_ID as its last arguments. If the message is directed to the base table, this argument is zero. On the contrary, the `route_output()` function receives (as input argument) the pointer to the socket that transmitted the routing messages. Hence, it can extract the VPN_ID from the socket structure and can pass it to the `vpn_rtrequest()`. This latter selects the proper table corresponding to the received

```
1.  unsigned int vpnid = 5;
2.
3.  struct {
4.    struct  rt_msghdr header;
5.    char    body[512];
6.  } msg;
7.  msg.header.rtm_type = RTM_ADD;
8.
9.  int s = socket(PF_ROUTE, SOCK_RAW, 0);
10. ioctl(s, SIOCSVPNID, &vpnid);
11. // Alternatively ...
12. // setsockopt(s, SOL_SOCKET, SO_VPNID,
13. //            &vpnid, sizeof(vpnid));
14.
15. write(s, (char *)&msg, sizeof(msg));
```

Fig. 6. Sample code adding a route to VPN table no. 5

VPN_ID and executes the requested operation. Note that, if the table does not exists yet, the function dynamically creates and initializes a new Patricia's tree. To limits the number of modified line of code in function redefinition, we used a macro as shown in Figure 7. The adoption of macro redefinition, together with the association of the VPN zero to the base network, made modifications simpler and clearer. This strategy was used extensively throughout the code.

```
(bar.h)
1. // void foo(int, int);
2. void vpn_foo (int, int, u_int);
3. #define  foo(a, b) (vpn_foo(a, b, 0))

(bar.c)
1. void
2. //foo(a, b)
3. vpn_foo(a, b, vpnid)
4.   int a;
5.   int b;
6.   u_int vpnid;
7. {
8.   ...
9. }
```

Fig. 7. Example of code modification

Routing messages can also be generated by the kernel itself upward the applications, e.g. when a network interface goes down. Another example is the static configuration of a route through the `route` command. In this case, the kernel is responsible for the notification of the table update event to the routing daemons. These messages are sent to all the applications that have an open routing socket for the same transport protocol of the modified table. Applications specify the protocol they are interested in via the third argument of the `socket()` system call (zero means all protocols). The upward messages are processed by the `route_output()` routine, which in turn calls the `raw_input()` function. As above, we redefined this latter as `vpn_raw_input()`, which dispatches the messages according to both the protocol and the VPN_ID of open routing sockets.

B. Forwarding virtualization

Figure 8 sketches the kernel functions processing IP packets during forwarding. When a network interface receives an IP packet, it places the packet in the input queue. When the `ip_input()` function is scheduled, it fetches a packet from the queue head and processes it. For instance, the function analyzes the presence of eventual IP options, and if the packet is directed to a non local destination, it is passed to the `ip_forward()` routine for delivery. Packets are stored in a data structure called `struct mbuf` that contains both the packet data and related information, such as the ingress interface that received the packet. If the packet is tunneled, `ip_input()` passes the `mbuf` to the `gif_input()` routine, which decapsulates the packet (i.e. it strips the outer header off) and replaces the ingress interface in `mbuf` by putting the proper `gif` interface that terminates the tunnel, in place of the physical interface (e.g. `eth0`). Finally, `gif_input()` queues the packet again. Next time, the `ip_input()` will pass the decapsulated packet directly to the `ip_forward()`, and the `mbuf` will point to the `gif` ingress interface. The `mbuf` is available to both the `ip_input` and the `ip_forward()` routines as input argument, hence both can obtain a pointer to the ingress interface (being it a physical interface for access VPN router, a virtual one for VPN core nodes).

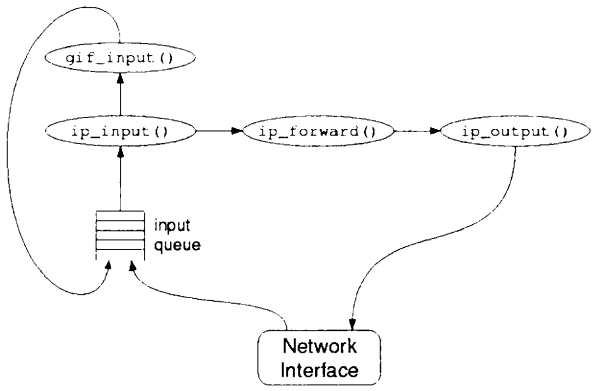


Fig. 8. Forwarding module in FreeBSD

The `ip_forward()` invokes the `rtalloc_ign()` to lookup the (base) routing table, and then sends the packet to the proper output interface. Transmission is mediated by the `ip_output()` that, for example, decrements the packet TTL and finally invokes the interface driver transmission routine.

To virtualize the forwarding process, we modified the default behavior of the `ip_forward()` routine. First, we added the VPN_ID to the interfaces by inserting an additional field to the `ifnet{}` data structure, as illustrated in Figure 9. We instructed the `if_attach()` routine (which is called to initialize all the interfaces, even for the dynamically created ones) to set the `if_vpn` field to zero. We modified the `ioctl()` system

```
(net/if_var.h)
struct ifnet {
  char    *ifname;    /* name, e.g. eth, gif */
  u_short if_index    /* numeric abbreviation */
  ...
  u_int   if_vpnid;  /* VPN_ID -  ADDED */
}
```

Fig. 9. Interface data structure with VPN identifier

call (and the `ifioctl()` ancillary function), which now accepts the `SIOCSIFVPNID` and `SIOCGIFVPNID` parameters to set/get the interface VPN_ID field. The modified version of `ifconfig` (which now accepts the `vpnid` switch), uses this system call, as shown in Figure 10.

```
1.  struct ifreq ifr;
2.  int    s, vpnid;
3.  char   ifname[16] = "eth0";
4.
5.  s = socket (AF_INET, SOCK_DGRAM, 0);
6.  vpnid = 5;
7.
8.  /* specify interface */
9.  strcpy(ifr.ifr_name, argv[2]);
10.
11. /* set interface VPN-ID */
12. ifr.ifr_vpnid = vpnid;
13.
14. ioctl(s, SIOCSIFVPNID, (caddr_t)&ifr);
```

Fig. 10. Code sample in `ifconfig` to set the VPN_ID on `eth0`

The `ip_forward()` was modified in order to call the `vpn_rtalloc_ign()` if the VPN_ID of the ingress interface is set to a non-zero value. The `vpn_rtalloc_ign()` (and the ancillary functions) is a redefined version of the standard `rtalloc_ign()`, which selects the correct table by using the `if_vpnid` field before looking up for the next hop. As a final result, packet forwarding is done by jointly considering both the destination address and the VPN_ID.

C. Routing daemons

This point can be seen as less important compared to the previous ones because it does not involve the modification of the operating system. Indeed, it involves the modification of an external software that cooperates with the OS to compute the best path to the destination which, in our case, varies according to the VPN_ID.

From this perspective, there are two models available in the literature. In the *piggyback* model a single routing daemon is able to compute the best path for all the VPNs. The routing daemon must be heavily modified since it must exchange, in its routing message with the peer routers, the VPN_ID of each destination. Vice versa, in the *virtual router* model, each router keeps several routing daemons active on the same machine. These daemons are completely independent (*ships in the night* approach) and each routing daemon exchanges only routing informations related to its VPN with the other peers.

The first model is probably more efficient, but it requires non-trivial modifications in the routing protocols. Vice versa, the second model allows the deployment of off-the-shelf daemons, with minimal modifications (you must assure that each routing daemon receives only the messages related to it). Our prototype uses a modified version of the Zebra [8] daemon that has a new starting parameter (the VPN_ID), which is used only to interact with the operating system (update routes or query for information). The remaining part of the daemon (routing messages, etc.) are kept unchanged. This modification allows the routing daemon to update on the part of the routing table that is related to its VPN, while to interact to unmodified daemons.

Since the network will have several routing messages flowing on it, each routing daemon bounds only to the virtual interfaces that are marked as belonging to the selected VPN: in other words, each tunnel carries only the routing messages that are related to the VPN it belongs to. Each router could have up to $N_{VPN} + 1$ routing daemons on it: the standard one (bound to all its physical interfaces) for the base network, and one addictional daemon for each locally served VPN.

IV. Related work

Several Internet Service Providers already have VPN provisioning in place; the most important router vendors have their solutions, and also the IETF community is working on that, trying to standardize a general solution. However, all the solutions available nowadays are based on the paradigm "VPNs at the edge, traditional IP routing in the backbone". Also solutions based on MPLS [9] can be seen as belonging to this paradigm, since VPNs are supported by creating a new set of label switched paths between ingress and egress routers so that multiple VPNs never share the same path.

Although non-existing in the marketplace, the idea of changing the router forwarding path in order to support VPNs natively has been examined by several projects in the literature. Among the others, the most important one is the Virtual Network Service project (VNS, [10]).

Although the technical solutions adopted in VNS seems to be quite similar to ours, the purpose of VNS is different. VNS was born to provide a private, secure, quality of service guaranteed channel between two end points. To do that, it uses IPSec (tunnel mode) in order to encapsulate the original IP packet. It follows that the intermediate routers on the path do not know the real (and private) address of the packet. Since the original IP address is hidden, the backbone routers should not need to know the VPN_ID of the packet. However, VNS provides Quality of Service guarantees on a per-VPN basis; therefore intermediate routers must know the VPN_ID of the packet. VNS, for instance, inserts the VPN_ID into an optional field of the outer IP header of the encrypted packet. Moreover, VNS modifies the routing protocols in order to support different paths (from the same couple ingress-egress routers) according to the VPN_ID. Therefore, the forwarding path of each router has to be modi-

fied in order to take into account both the destination address of the IPSec tunnel and the VPN_ID of the packet. It follows that VNS implements both a modified forwarding path (through multiple routing tables addressed by the VPN_ID) in order to support per-VPN routing and a mechanism to specify the VPN of each IP packet. Therefore VNS could support overlapped address spaces as well as we do, although this was not an objective of the project.

From the association between packets and VPNs, our present implementation uses a statically defined mapping between interfaces (also virtual, like tunnels) and VPNs, but this can be changed to a more sophisticated method (the one in VNS, or the MPLS tag, or even other ways) without changing the mechanisms that are used to forward IP traffic[2]. In other words, VNS wants to provide a way to create end-to-end VPN services; our project focuses particularly on the forwarding path and it wants to demonstrate an alternative way to create a VPN-aware IP network.

V. Conclusions

This paper presented the design and the implementation of an advanced support for provisioned virtual private networks, based on the FreeBSD operating system. The introduced new features allow FreeBSD to be adopted as an open-source mean for developing concurrent VPNs. In our implementation we modified the kernel functionalities (e.g. `socreate()`, `rtrequest()`, `ip_forward()`), the system calls providing an interface toward the kernel (e.g. `ioctl()`, `setsockopt()`, `sysctl()`), and many user-space applications (e.g. `ifconfig`, `route`, `netstat`, `zebra`).

The status of the current implementation is complete, and test-bed was ran at Politecnico di Torino, demonstrating that the implementation was working fine. Performances are not reported in this paper because there are absolutely no differences prior and after our modifications. In fact, FreeBSD already has support for multiple routing tables because it can handle several network-level protocols (IP, IPX, etc.) at the same time. The overhead of our VPN support can be seen like another network protocol, which results in a longer `switch` instruction into the forwarding path. The results confirms that the same operating system forwards the same number of packets with or without our modifications.

However some issues need to be further investigated. The main problem concerns the ARP module of standard FreeBSD. Since ARP entries are cached within the IP base routing table, this creates a conflict when a VPN access router is connected to multiple sites that are using the same address space. The virtualization of ARP caches and ARP lookups is needed (similarly to the virtualization of tables and table lookups, as described in Section III) in order to make the implementation more flexible. The second issues relates to the standard GRE module. By now, it is not possible to define more than one tunnel bounded to the same couple of physical interfaces between two peer VPN routers. Obviously, this hampers the applicability of our implementation, since it is not possible to deploy two parallel tunnels for two distinct VPNs without using different physical (i.e. outer) addresses. Such problem could be overcome by patching the GRE support, to integrate the adoption of the GRE key field. By mapping the key field to the `gif` VPN identifier, the parallel tunnels would be still distinguishable.

Besides this issues, further work can be undertaken in many area. For instance, the Zebra support for multiple routing table could be improved. Our current implementation requires the instantiation of a `zebra` router manager daemon (and a corresponding `ospfd` routing daemon) for each defined table. It would be more manageable to have a single router manager and let the routing daemons to specifies the table of interest for the routing updates. This requires a deeper modification (with respect of the current status), since the communication protocol between the daemons and the manager should be extended.

A second major improvement concerns the identification of VPN traffic at the access side. Currently, all packets incoming from a tagged physical interface are associated to a single VPN. In case of a client site belonging to multiple VPN, the site access router must be connected to multiple interfaces of the provider access router. Further, the site access router must be able to distribute client packets of different VPNs towards the different interfaces it is attached to. This requires additional capabilities from the site access router, hence hampering the transparency for the client. To this aim, a more sophisticated approach could be used for traffic identification. Multiple traffic filters could be applied to the access interface (instead of a single tag, which is logically equivalent to a single wild-card filter). Filters could be used to identify the membership of a packet to a given VPN on the basis of protocol fields of the TCP/IP headers. This would also allow a greater granularity to the traffic identification operation. This solution would allow the site access router to be attached to a single interface toward the backbone network, and would simplify its task: it should have a single default route for all non local traffic (rather than distributing packets on several outgoing interfaces).

Finally, other possible evolutions are the support of IPSec tunnels to allow secure VPNs when needed and the integration of a QoS module (e.g. ALTQ [11]) with the virtual forwarder, to allow QoS-based forwarding on per-VPN basis.

[2]In this case the current implementation of the virtual routing daemon must be changed as well because it relies on different interfaces (i.e. tunnels) to distinguish the routing messages belonging to different VPNs.

Acknowledgments

The authors would like to thank Angelo Calafato for his great job in the implementation of the prototype.

References

[1] B. Gleeson, et al., *A Framework for IP Based Virtual Private Networks*, IETF RFC 2764, Feb. 2000
[2] R. Callon (ed.), et al., *A Framework for Layer 3 Provider Provisioned Virtual Private Networks*, IETF Internet Draft, Apr. 2002
[3] D. Farinacci, et al., *Generic Routing Encapsulation (GRE)*, IETF RFC 2748, Mar. 2000
[4] G. R. Wright, W. R. Stevens, *TCP/IP Illustrated, vol 2: The Implementation*, Addison-Wesley, 1995
[5] S. J. Leller, et al., *The Design and the Implementation of the 4.3BSD UNIX Operating System* Addison-Wesley, 1989
[6] A. Calafato, *Architectural Choices for Developing Virtual Networks* (in Italian), Master thesis, Politecnico di Torino, Jan. 2002
[7] FreeBSD 4.4 patches, On-line at http://softeng.polito.it/freebsd/
[8] Zebra Project Page, On-line at http://www.zebra.org
[9] E. Rosen, et al., *BGP/MPLS VPNs*, IETF RFC 2547, Mar. 1999

[10] L.K. Lim, et al., Customizable Virtual Private Network Service with QoS, Computer Networks, vol. 36, no. 2-3., pp. 137-151, Jul. 2001
[11] K. Cho, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX based Routers*, Usenix 1998, New Orleans, Louisiana, USA

A shared write-protected NFS root file system for a cluster of diskless machines

Ignatios Souvatzis
Bonn University, CS Department, Chair V
<ignatios@cs.uni-bonn.de>

Managing three diskless network clients can be done manually. Manually managing ten is still possible, but tedious. Manually managing hundreds is close to impossible.

When we got ten disk- (and head-)less network computers of a new type that we wanted to use as computing nodes for a parallel virtual machine for a practical course, I decided to set them up with a shared root file system.

However, a BSD root file system has to be unique and writable for every client machine for a couple of reasons (for example to write log files, create communication sockets for some daemons, set device node ownership at login and logout and change timestamps), so that a single writable shared root file system does not work.

As an alternate solution, I placed most writable directories onto (virtual) memory file systems. The program area and configuration files need only to be exported by the server for read only access. This way, the system programs, system libraries, and the configuration are protected against malicious users, even if they should gain root privileges on the client machine.

This presentation elaborates on the problems I encountered and the solutions I implemented, using stock NetBSD 1.5 as the client operating system, with just a small script and a few configuration lines added.

Finally, I compare my solution to different possibility, namely using a kernel embedded root file system.

Ignatios Souvatzis is "System Programmer" (in reality, a nonlinear combination of system administrator, tape operator, kernel hacker, and user advisor) at Chair V of the Computer Science Department at the University of Bonn. He is also a NetBSD key developer. His main tasks have been some device drivers, a new ARP system, and maintaining the Amiga port. Sometimes, those assignments overlap. He studied Physics and Astronomy in Bonn. He has used nearly everything running VMS from the 11/780 to MicroVAX, and everything running Ultrix from the DECstation 2100 to the 5000/260 and even a CDC Cyber 172 and a Convex to do astronomical data reduction. Earlier at University, he (ab)used lots of different systems, from IBM 360 to 4331 to PC to solve the eight queens problem, has written test programs for 8085 and UNIBUS control boxes for the new accelerator at the physics department. In his second University year, he was introduced to Unix (on a Z8000 box) at a small software company. He seldom admits that he was teaching an introduction to BASIC at an adult education center in late 1981/early 1982 (but it paid driving home for the weekends).

A shared write protected root filesystem for a cluster of network clients

Ignatios Souvatzis*

Abstract

A method to boot a cluster of diskless network clients from a single write-protected NFS root file system is shown. The problems encountered when first implementing the setup and their solution are discussed. Finally, the setup is briefly compared to using a kernel-embedded root file system.

1 Introduction

- Managing three diskless network clients can be done manually.
- Manually managing ten is still possible, but tedious.
- Manually managing hundreds is close to impossible.

When we got ten disk- (and head-) less network computers of a new type that we wanted to use as computing nodes for a parallel virtual machine for a practical course[1], we decided to set them up with a shared root file system.

However, a BSD root file system has to be unique and writable for every client machine for a couple of reasons, so that a single writable shared root file system does not work.

As an alternate solution, we placed most writable directories onto (virtual) memory file systems. The program area and configuration files need only to be exported by the server for read only access. This way, the system programs, system libraries, and the configuration are protected against malicious users, even if they should gain root privileges on the client machine.

This presentation elaborates on the problems we encountered and the solutions we implemented, using stock NetBSD 1.5 as the client operating system, with just a small script and a few configuration lines added.

*Universität Bonn, Institut für Informatik, Römerstr. 164, D-53117 Bonn, Germany; e-mail ignatios@cs.uni-bonn.de

2 System Environment

2.1 File- and Bootserver

SUN UltraSparc-10, originally running Solaris 2.6 (now Solaris 8), located in a secure room. However, the problem and its solution do not depend on the machine size and operating system, as long as the clients' root file systems are mounted via something similar to NFS.

2.2 Clients

Originally 10 Digital Network Appliance Reference Design (DNARD) machines, with 64 MB of RAM, no disk, no keyboard/monitor attached, running NetBSD 1.4, later 1.5.3 (soon 1.6), located in a secure room.

We believe that the solution could be applied, with some modification, to other Unix-like operating systems, although certain features of NetBSD did help a lot.

2.3 Social environment

The machines are to be used by 10 to 20 students for the duration of a half-year course. They should be able to login from home. Even if we successfully limit logins to them, we can't really trust them not to break into the system when they can.

3 Why a shared, read-only root?

3.1 Easier administration

We want to install and configure on the server (or just one machine, under special circumstances) and at most have to reboot the (other) clients.

3.2 Saving disk space

NetBSD-1.5.3/arm32 needs about 20 megabytes of disk space in the traditional root file system (see figure 1).

```
server 196 # du -ks bin sbin etc
5795    bin
12760   sbin
606     etc
```

Figure 1: NetBSD-1.5.3/arm32 root filesystem usage

While the total size (200 megabytes for 10 machines) does not look like much today, even assuming 100 machines, disk space was more expensive when we started, and *reliable* disk space, especially *backed up* disk space, has not become cheaper as fast as unreliable ("desktop") IDE disks.

3.3 Server security

The root filesystem *has to* be exported with (client-side) root access rights. Leaving it writable allows a malicious client to permanently change (at least) its own configuration.

(Securing user data against read-out or modification from a manipulated client is beyond the scope of this work.)

4 Problems found

4.1 Booting

- The client has to learn its name and network address.
- The client has to learn the name of its NFS swap file.
- The client has to learn the name of the per-client filesystems on the server.

4.2 During operation

Files on the root file system that are written on traditional installations, or that are tied to a single machine, include:

- System log files, which are written to all the time.
- `*.pid` - files, written (at least) during startup of server processes.
- Sockets (`/dev/log`, `/dev/printer`, ...), which are created by the server processes that use them to communicate with their clients.
- Device nodes (of terminal-like devices, including pty, mice, keyboards) change their owner at each login and logout.
- The ssh host key is stored in the root file system.
- The package system database, normally `/var/db/pkg`. If we made `/var` per-machine, we'd need a per-machine `/usr/pkg`, too. Otherwise we would need to move the database to a shared location.

4.3 During shutdown

`/etc/nologin` is created by the first machine shutting down. It will prevent login on the other clients.

This is fine for a coordinated shutdown of the whole cluster, but not when booting a single client machine to test, say, a new kernel.

5 Methods

5.1 Some problems aren't

- As all our clients are equal in configuration and don't carry any permanent state, there is no point to make them distinguishable in a cryptographically secure way. So we just use the same set of host keys for sshd on all the machines.
- Network configuration for IPv4 is done by DHCP anyway (when booting the machines), so we can also use it to learn the client name (and a few other parameters). For IPv6, we're using stateless IPv6 autoconfiguration.
- We configured syslog to send all logged events to the server.

5.2 DHCP server setup

Here, nothing special is needed. The clients are set up with fixed addresses and names assigned to them and learn the name of the kernel to boot from and the location of the root file system (figure 5).

5.3 Sockets and `*.pid` - files

During boot time, a small memory file system is created and mounted on `/var/run`. (BSD mfs stores its data in the address space of the `newfs_mfs` process, so it's pageable[2]). We added this to `/etc/fstab` (figure 2) and marked `/var/run` as a filesystem to be mounted very

```
server:/BSD/root2/1.5     /           nfs ro                0 0
server:/BSD/root/vargames /var/games  nfs rw                0 0
swap                      /tmp        mfs rw,-s=32768       0 0
swap                      /var/run    mfs rw,-i=512,-s=256  0 0
```

Figure 2: /etc/fstab

```
rc_configured=YES
shroot_pfx_var=server:/BSD/root/var-
shroot_pfx_swap=server:/BSD/swap/
critical_filesystems_beforenet="/var /var/run"
update_motd=NO
rpcbind=YES # nfs
domainname=nis.doma.in # NIS
ypbind=YES
amd=YES amd_dir=/var/amdroot
nfs_client=YES
ip6mode=autohost
defaultroute=10.10.10.10
sshd=YES postfix=YES ntpd=YES
lpd=YES lpd_flags=-s
inetd=YES       # ntalk, (c)fingerd
```

Figure 3: /etc/rc.conf

```
> mount
server:/BSD/root2/1.5 on / type nfs (read-only)
server:/BSD/root/var-client on /var type nfs
server:/BSD/swap/client on /swap type nfs
mfs:34 on /dev type mfs (asynchronous, local)
mfs:37 on /etc type mfs (asynchronous, local, union)
mfs:1000 on /var/run type mfs (asynchronous, local)
server:/BSD/root/vargames on /var/games type nfs
mfs:1085 on /tmp type mfs (asynchronous, local)
pid1121@client:/home on /home type nfs
studsrv:/opt/export/home/stud/user on \
        /var/amdroot/studsrv/opt/export/home/stud/user \
        type nfs (nodev, nosuid)
server:/export/home/2 on /var/amdroot/server/export/home/2 \
        type nfs (nodev, nosuid)
```

Figure 4: The filesystems at run-time

```
option domain-name "my.doma.in";
option domain-name-servers 10.10.10.2, 10.10.10.1;
deny unknown-clients;
use-host-decl-names on;
subnet 10.10.10.0 netmask 255.255.255.0 {
        option routers 10.10.10.3;
        group {
                option lpr-servers server.my.doma.in;
                server-name "server";
                next-server server;
                filename "netbsd-SHARK-1.5.3_20020114";
                option root-path "/BSD/root2/1.5";
                host client {
                        hardware ethernet 10:20:30:40:50:60;
                        fixed-address client.my.doma.in;
                }
                ...
        }
}
```

Figure 5: DHCP server configuration file excerpt

early in `/etc/rc.conf` (figure 3). This is used for the following files:

- `*.pid` files - often in `/etc` - are created in `/var/run` by all daemons integrated into NetBSD or installed from the package system.
- `/dev/log` was moved to `/var/run/log` by changing the syslogd code; this is the standard location in NetBSD nowadays.
- `/dev/printer` was moved to `/var/run/printer`; this is the standard location in NetBSD.

5.4 Device nodes

We create a memory file system for `/dev`. `/dev` on the server only needs `/dev/console` (for the benefit of `/sbin/init`).

At boot time, we populate `/dev` by running the MAKEDEV script, which is installed in `/sbin` in our setup. This takes about 20 seconds. Should we use slower machines, we could tune the amount of devices created - currently, we run `sh MAKEDEV all`.

The code to do this–as all special code needed–lives inside a small script called `shroot` (figure 6).

NetBSD-1.6 /sbin/init does all of this automatically, when no /dev/console is found. (This was implemented to make CD-ROM and MS-DOS filesystem demonstration installations possible.) After upgrading, we should be able to remove the `/dev/console` *on the exported root file system and remove the lines in* `shroot` *that handle* `/dev`.

5.5 Swap, `/var`

During boot time we run `/bin/hostname` to find out how we're called - the kernel has learned it using DHCP. Using this name, we synthesize the server-side names of the swap file and `/var` filesystem to mount.

`/var` is per-machine to allow per-machine spool files for outgoing e-mail, printing and similar services.

5.6 Remaining files written to `/etc`

Some of the files on `/etc` can be configured not to be changed (e.g., `/etc/motd`). However, there are a few that can't be easily handled without code and functionality change, like `/etc/nologin`.

As a simple catch-all to this problem, we create a small memory file system and union-mount it over the NFS `/etc`. This is handled in `/etc/rc.d/shroot`, too (fig. 6).

5.7 Where to place the code?

As mentioned already, we concentrated all special startup script code needed in the `shroot` script.

Obviously, this script has to run early in the boot process (before device nodes, `/var/run` etc. are accessed). Traditional

```
#!/bin/sh

# PROVIDE: shroot
# REQUIRE: root
# BEFORE: mountcritlocal

# shared root setup.

. /etc/rc.subr

name="shroot"
start_cmd="shroot_start"
stop_cmd=":"
required_files="/sbin/MAKEDEV /sbin/MAKEDEV.local"

shroot_start () {

    hostname=`/bin/hostname`

    case "$shroot_pfx_var" in
        "") ;;
        *) /sbin/mount -t nfs ${shroot_pfx_var}${hostname} /var ;;
    esac
    case "$shroot_pfx_swap" in
        "") ;;
        *) /sbin/mount -t nfs ${shroot_pfx_swap}${hostname} /swap\
            && /sbin/swapon /swap ;;
    esac

    /sbin/mount -t mfs -o -i=256 -o -s=512 swap /dev
    /sbin/mount -t mfs -o -i=256 -o -s=512 -o union swap /etc
    /bin/chmod 755 /dev /etc
    echo -n "creating device nodes...";
    /bin/cp /sbin/MAKEDEV /sbin/MAKEDEV.local /dev
    (cd /dev; sh MAKEDEV all)
    echo done.
}

load_rc_config $name
run_rc_command "$1"
```

Figure 6: /etc/rc.d/shroot

`/etc/rc.local` is much too late. For NetBSD-1.4, we had hooked the equivalent script up in `/etc/netstart.local`; without that, we would have had to find a suitable place in `/etc/rc`, or among the zillions of SVR4 or Linux startup scripts.

The explicit startup script dependencies first implemented in NetBSD-1.5 [3] made the task easy: the `shroot` script is placed into the directory /etc/rc.d/ and states explicitly that it wants to be run after the root file system is there, but before critical local file systems are mounted (see figure 6).

This is important, because `/var/run` has to be mounted after shroot mounts `/var`!

Depending on applications, some subdirectories of `/var` have to be mounted from a shared NFS volume - e.g. `/var/mail` or `/var/games`. This is handled normally in `/etc/fstab` or using the automounter.

Figure 4 shows the run-time file system table. Note that `/usr` is embedded in the root file system! As it is never written by the clients, there is no point to seperate it from root.

5.8 Software Installation with the `pkg`-system

We are using a shared directory tree for the third-party packages installed through the NetBSD package system. The package database is moved inside `/usr/pkg`, so that it is shared, too.

During software installation, we give a single chosen client (which temporarily gets a keyboard and monitor) write access to the server, and revoke it afterwards.

The environment variable `PKG_DBDIR` has to be set to `/usr/pkg/libdata/pkgdb` for installation as well as any other use of the `pkg_xxx` tools. Fortunately, this is all that is needed to make the package system tools happy. `pkg_info` should be usable by the students to find out what software is installed.

However, from a security and performance viewpoint, it would be better to have cross-install tools and to run them on the server.

6 Why no embedded root file system?

An alternate method we've considered is to embed a root file system in the client kernel. This leads to the same security benefits outlined in section 3.3. However, there are two drawbacks:

- An embedded filesystem is completely RAM based, non-pageable, during execution. MFS, on the other side, is virtual memory based.

  To make this work at all, the part of the root file system actually inside the kernel has to be carefully tuned. The chosen method allows to use a stock NetBSD installation, with only the `shroot` script added and some configuration files changed.

- Both changing the kernel and changing some configuration file require to embed the kernel file system anew into the kernel file and reboot all clients.

  This is inconvenient, especially when only the configuration of some short-running component was to be changed.

  The chosen setup allows to change all of `/etc` on the server and have it immediately available, if so desired.

7 Summary

A method to make the root file system for network clients read-only and shared has been presented. Administration can mostly happen on the server. Client break-ins would not affect the system files on the server. The installation uses a mostly normal NetBSD-1.5.3 installation, with a single script added and some configuration files in `/etc` changed.

References

[1] Bonn University CS Dept., parallel systems student lab home page:
http://theory.cs.uni-bonn.de/info5/system/parlab/

[2] NetBSD `mount_mfs`(8) and `newfs_mfs` manual pages

[3] NetBSD `rc`(8), `rcorder`(8) and `rc.conf`(8) manual pages

Using BSD for current and next generation voice telephony services

David Sugar
Open Source Telecom
<dyfet@ostel.com>

This presentation will cover how to use GNU Bayonne with BSD related operating systems, including particularly FreeBSD and NetBSD, which are what I am most familiar with. A presentation of issues related to telephony drivers and voice telephony hardware support for BSD and what is needed to make voice telephony services work under BSD will be provided.

During this presentation I will introduce the telephony application server of the GNU Project, "GNU Bayonne". This presentation will cover what GNU Bayonne is and how it fits into the overall strategy of enterprise and carrier class telephony solutions for current and next generation IP based telephone networks using free software running on free operating systems, such as xBSD systems, GNU/Linux, etc.

GNU Bayonne is composed of three servers and each of these (Bayonne, Olorin, and Babylon) will be discussed in detail, including their architecture and how to configure, operate, and create applications for GNU Bayonne servers to produce real-world telephony solutions for xBSD operating systems. The scripting language of GNU Bayonne will be covered, as well as how to integrate GNU Bayonne servers with traditional scripting languages such as tcl, Perl, Python, and Guile.

GNU Bayonne will also be demonstrated live running under FreeBSD (4.5) as part of this presentation. Examples and scripted applications used for real world applications people can deploy using GNU Bayonne under xBSD, such as voice mail and debit calling, will also be demonstrated.

I am one of the founders of and Chief Technology Officer for Open Source Telecom Corporation (http://www.ostel.com). I am also the primary author of and active maintainer for a number of packages that are part of the GNU project, including GNU Common C++, GNU ccScript, GNU ccRTP, and GNU ccAudio, as well as the GNU telephony application server, GNU Bayonne. Furthermore, I maintain the FreeBSD ports for these packages. I also serve as the voluntary chairman of the FSF's DotGNU steering committee (http://www.dotgnu.org), and have served as the communities elected representative to the International Softswitch Consortium.

GNU Bayonne: telephony services for freely licensed operating systems

David Sugar <*sugar@gnu.org*>
http://www.gnu.org/software/bayonne

Abstract

GNU Bayonne is a middle-ware telephony server that can be used to create and deploy script driven telephony application services. These services interact with users over the public telephone network. What we are hoping to do is enable, using commodity PC hardware and CTI cards running under GNU/Linux and FreeBSD, which often are available from numerous vendors, to create carrier applications like Voice Mail and calling card systems, as well as enterprise applications such as unified messaging. GNU Bayonne can be used to provide voice response for e-commerce systems and has been used in this role in various e-gov projects. GNU Bayonne can also be used to telephony enable existing scripting languages such as perl and python.

1 Introduction

Even without considering all the various reasons of why we must have Free Software as part of the telecommunications infrastructure, it is important to consider what the goals and platform needs are for a telephony platform. Historically, telephony services platforms had been the domain of real-time operating systems. Recent advances in CTI hardware has made it possible to offload much of this requirement to hardware making it practical for even low performance systems running efficient kernels to provide such services for many concurrent users.

Telephony services are usually housed in phone closets or other closed and isolated areas. As such, remote maintainability, and high reliability are both important platform requirements as well. The ability to integrate with and use standard networking protocols is also becoming very important in traditional telephony, and certainly is a key requirement for next generation platforms.

So we can summarize; low latency/high performance kernels, remote manageability without the need for a desktop environment, high reliability, and open networking protocols. This sounds like an ideal match for BSD or Linux kernel based systems.

However, when we looked further into this question and architected GNU Bayonne, we also decided threading was important. Threading offers some interesting design advantages, but, equally important, it provides a means of better utilizing SMP hardware. We found it important to scale up to solutions that can support voice processing on a full DS-3 circuit using a single server. Threading represents some challenges in current BSD systems as I will elaborate further in this paper, so we initially chose to focus primarly on GNU/Linux systems rather than BSD.

Our goal for GNU Bayonne 1.0 was primarily to make telephony services as easy to program and deploy as a web server is today. We choose to make this server easily programmable through server scripting. We also desired to have it highly portable, and allow it to integrate with existing application scripting tools so that one could leverage not just the core server but the entire platform to deliver telephony functionality and integrate with other resources like databases.

GNU Bayonne, as a telephony server, also imposes some very real and unique design constraints. For example, we must provide interactive voice response in real-time. "real-time" in this case may mean what a person might tolerate, or delay of 1/10th of a second, rather than what one might measure in milliseconds in other kinds of real-time applications. However, this still means that the service cannot block, for, after all, you cannot flow control people speaking.

Since each vendor of telephony hardware has chosen to create their own unique and substantial application library interface, we needed GNU Bayonne to sit above these and be able to abstract them. Ultimately we choose to create a driver plug-in architecture to do this. What this means is that you can get a card and API from Aculab, for example, write your application in GNU Bayonne using it, and later choose, say, to use Intel telephony hardware, and still have your application run, unmodified. This has never been done in the industry widely because many of these same telephony hardware manufacturers like to produce their own middle-ware solutions that lock users into their products.

2 Supporting Libraries

To create GNU Bayonne we needed a portable foundation written in C++. I wanted to use C++ for several reasons. First, the highly abstract nature of the driver interfaces seemed very natural to use class encapsulation for. Second, I found I personally could write C++ code faster and more bug free than I could write C code.

Why we choose not to use an existing framework is also simple to explain. We knew we needed threading, and socket support, and a few other things. There were no single framework that did all these things except a few that were very large and complex which did far more than we needed. We wanted a small footprint for GNU Bayonne, and the most adaptable framework that we found at the time typically added several megabyte of core image just for the runtime library.

GNU Common C++ (originally APE) was created to provide a very easy to comprehend and portable class abstraction for threads, sockets, semaphores, exceptions, etc. This has since grown into it's own and is now used as a foundation of a number of projects as well as being a part of GNU.

In addition to having portable C++ threading, we needed a scripting engine. This scripting system had to operate in conjunction with a non-blocking state-transition call processing system. It also had to offer immediate call response, and support several hundred to a thousand instances running concurrently in one server image.

Many extension languages assume a separate execution instance (thread or process) for each interpreter instance. These were unsuitable. Many extension languages assume expression parsing with non-deterministic run time. An expression could invoke recursive functions or entire subprograms for example. Again, since we wanted not to have a separate execution instance for each interpreter instance, and have each instance respond to the leading edge of an event callback from the telephony driver as it steps through a state machine, none of the existing common solutions like tcl, perl, guile, etc, would immediately work for us. Instead, we created a non-blocking and deterministic scripting engine, GNU ccScript.

GNU ccScript is unique in several ways. It is step executed, and is non-blocking. Statements either execute and return immediately, or they schedule their completion for a later time with the executive. A given "step" is executed, rather than linearly. This allows a single thread to invoke and manage multiple interpreter instances. While GNU Bayonne can support interacting with hundreds of simultaneous telephone callers on high density carrier scale hardware, we do not require hundreds of native "thread" instances running in the server, and we have a very modest CPU load.

Another way GNU ccScript is unique is in support for memory loaded scripts. To avoid delay or blocking while loading scripts, all scripts are loaded and parsed into a virtual machine structure in memory. When we wish to change scripts, a brand new virtual machine instance is created to contain these scripts. Calls currently in progress continue under the old virtual machine and new callers are offered the new virtual machine. When the last old call terminates, the entire old virtual machine is then disposed of. This allows for 100% uptime even while services are modified.

Finally, GNU ccScript allows direct class extension of the script interpreter. This allows one to easily create a derived dialect specific to a given application, or even specific to a given GNU Bayonne driver, simply by deriving it from the core language through standard C++ class extension.

3 TGI support and plug-ins

To be able to create useful applications, it is necessary to have more than just a scripting language. It requires a means to be extended so that it can incorporate database access libraries or other functions that fall outside of the scope of the scripting language itself. These extensions should be loaded on demand

only when used, and should be specified at runtime so that new ones can easily be added without the need to recompile the entire server.

To support scripting extensions we have the ability to create direct command extensions to the native GNU Bayonne scripting languages. These command extensions can be processed through plug-in modules which can be loaded at runtime, and offer both scripting language visible interface extensions, and, within the plug-in, the logic necessary to support the operation being represented to the scripting system. These are much more tightly coupled to the internal virtual machine environment and a well written plug-in could make use of thread pools or other resources in a very efficient manner for high port capacity applications.

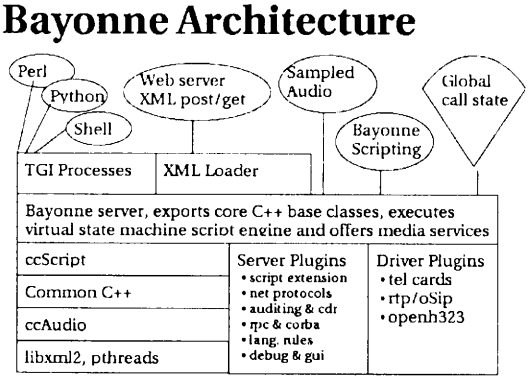


Figure 1: Architecture of GNU Bayonne

When writing command extensions, it is necessary to consider the need for non-blocking operations. GNU Bayonne uses ccScript principally to assure non-blocking scripting, and so any plug-in must be written so that if it must block, it does so by scheduling a state operation such as "sleep" and performs potentially blocking operations in separate threads. This makes it both hard and complex to correctly create script extensions in this manner.

While GNU Bayonne's server scripting can support the creation of complete telephony applications, it was not designed to be a general purpose programming language or to integrate with external libraries the way traditional languages do. The requirement for non-blocking requires any module extensions created for GNU Bayonne are written highly custom. We wanted a more general purpose way to create script extensions that could interact with databases or other system resources, and we choose a model essentially similar to how a web server does this.

The TGI model for GNU Bayonne is very similar to how CGI works for a web server. In TGI, a separate process is started, and it is passed information on the phone caller through environment variables. Environment variables are used rather than command line arguments to prevent snooping of transactions that might include things like credit card information and which might be visible to a simple "ps" command.

The TGI process is tethered to GNU Bayonne through stdout and any output the TGI application generates is used to invoke server commands. These commands can do things like set return values, such as the result of a database lookup, or they can do things like invoke new sessions to perform outbound dialing. A "pool" of available processes are maintained for TGI gateways so that it can be treated as a restricted resource, rather than creating a gateway for each concurrent call session. It is assumed gateway execution time represents a small percentage of total call time, so it is efficient to maintain a small process pool always available for quick TGI startup and desirable to prevent stampeding if say all the callers hit a TGI at the exact same moment.

4 Bayonne Architecture

As can be seen, we bring all these elements together into a GNU Bayonne server, which then executes as a single core image. The server itself exports a series of base classes which are then derived in plug-ins. In this way, the core server itself acts as a "library" as well as a system image. One advantage of this scheme is that, unlike a true library, the loaded modules and core server do not need to be relocatable, since only one instance is instantiated in a specific form that is not shared over arbitrary processes.

When the server comes up, it creates gateways and loads plug-ins. The plug-ins themselves use base classes found in the server and derived objects that are defined for static storage. This means when the plug-in object is mapped through dload, it's constructor is immediately executed, and the object's base class found in the server image registers the object with the rest of GNU Bayonne. Using this method, plug-ins in effect automatically register themselves through the server as they are loaded, rather than through a separate

runtime operation.

The server itself also instantiates some objects at startup even before main() runs. These are typically objects related to plug-in registration or parsing of the configuration file.

Since GNU Bayonne has to interact with telephone users over the public telephone network or private branch exchange, there must be hardware used to interconnect GNU Bayonne to the telephone network. There are many vendors that supply this kind of hardware and often as PC add-on cards. Some of these cards are single line telephony devices such as the Quicknet LineJack card, and others might support multiple T1 spans. Some of these cards have extensive on-board DSP resources and TDM busses to allow interconnection and switching.

GNU Bayonne tries to abstract the hardware as much as possible and supports a very broad range of hardware already. GNU Bayonne offers support for /dev/phone Linux kernel telephony cards such as the Quicknet LineJack, for multiport analog DSP cards from VoiceTronix and Dialogic, and digital telephony cards including CAPI 2.0 (CAPI4Linux) compliant cards, and digital span cards from Intel/Dialogic and Aculab. We are always looking to broaden this range of card support.

At present both voice modem and OpenH323 support is being worked on. Voice modem support will allow one to use generic low cost voice modems as a GNU Bayonne telephony resource. The openh323 driver will actually require no hardware but will enable GNU Bayonne to be used as an application server for telephone networks and softswitch equipment built around the h323 protocol family. At the time of this writing openh323 support is slated for release as part of GNU Bayonne 1.1.

5 GNU Bayonne and XML Scripting

Some people have chosen to create telephony services through web scripting, which is an admirable ambition. To do this, several XML dialects have been created, but the idea is essentially the same. A query is made, typically to a web server, which then does some local processing and spits back a well formed XML document, which can then be used as a script to interact with the telephone user. These make use of XML to generate application logic and control much like a scripting language, and, perhaps, is an inappropriate use of XML, which really is designed for document presentation and inter- exchange rather than as a scripting tool. However, given the popularity of creating services in this manner, we do support them in GNU Bayonne.

GNU Bayonne did not choose to be designed with a single or specific XML dialect in mind, and as such it uses a plug-in. The design is implemented by dynamically transcoding an XML document that has been fetched into the internal ccScript virtual machine instructions, and then execute the transcoded script as if it were a native ccScript application. This allows us to transcode different XML dialects and run them on GNU Bayonne, or even support multiple dialects at once.

Since we now learn that several companies are trying to force through XML voice browsing standards which they have patent claims in, it seems fortunate that we neither depend on XML scripting nor are restricted to a specific dialect at this time. My main concern is if the W3C will standardize voice browsing itself only to later find out that the very process of presenting a document in XML encoded scripting to a telephone user may turn out to have a submarine patent, rather than just the specific attempts to patent parts of the existing W3C voice browsing standard efforts.

Currently GNU Bayonne impliments a "BayonneXML" dialect as a model XML plugin. This dialect demonstrates a range of functionality similar to "CallXML". We have had offers from various sources to fund specific development of W3C spec compliant XML dialects, but so far none of these offers have ever reached the point where a check was cut. It would take considerable time and talent to finish GNU Bayonne XML work, and none of the people activily using it have pushed for XML support. As such, it has received a lower profile in the list of features we wish to currently work on.

6 Current Status

The 1.0 release of GNU Bayonne was released on September 1st. This release represents several years of active development and has been standardized in how it operates and how it is deployed. This release is part of the GNU project and has been packaged for use with many GNU/Linux distributions. With this release we have had a stable platform for developing

GNU Bayonne applications and for considering future development.

GNU Bayonne does not exist alone but is part of a larger meta-project, "GNUCOMM". The goals of GNUCOMM is to provide telephony services for both current and next generation telephone networks using freely licensed software. These services could be defined as services that interact with desktop users such as address books that can dial phones and softphone applications, services for telephone switching such as the IPSwitch GNU softswitch project and GNU oSIP proxy registrar, services for gateways between current and next generation telephone networks such as troll and proxies between firewalled telephone networks such as Ogre, realtime database transaction systems like preViking Infotel and BayonneDB, and voice application services such as those delivered through GNU Bayonne.

7 GNU Bayonne and FreeBSD

GNU Bayonne is successfully used with GNU/Linux systems today. It is widely used in many areas ranging from commercial carriers in Europe to state governments in the United States. We do not believe telephony should be restricted to any one platform, however, and, even from the beginning, choose to make GNU Bayonne highly portable.

The core libraries that compose GNU Bayonne are all highly portable, and in particular, are built on a single abstract interface library, GNU Common C++. GNU Common C++ offers portable threading and socket support, and has been ported to many platforms, including FreeBSD, as well as some non-Unix platforms. These libraries also have active support and are distributed with build files for directly building BSD style "ports" collection entities. As such it is very simple to build a "port" of Common C++, ccAudio, or ccScript. One just does "make ports" from the master Makefile after using ./configure.

I do not activily package GNU Common C++, ccAudio, or ccScript for FreeBSD just as I do not activily provide Debian package for the GNU/Debian distribution, or for any other target OS. At one time I did many of these things, but I found it became too overwhelming to both manage and build releases, and to personally provide binary and build packages for every target platform. We also do not have a large enough group of active developers where one can simply work on packaging. In fact, we now depend on the broader GNU/Linux community and vendors for packaging of GNU Bayonne for specific GNU/Linux distributions and for patches when needed for specific distributions.

I actually do on occassion develop under FreeBSD. I actually authored the original FreeBSD port builds for GNU Bayonne's supporting libraries and tested them on my FreeBSD development system at home. I also did an experimental build of GNU Bayonne under FreeBSD and we are hoping to demonstrate it at EuroBSD. However, we have ran into several issues in building GNU Bayonne under FreeBSD related to threading.

The primary issue with FreeBSD threading boils down to two issues; one, that not all blocking system calls behave as cancellation points, and two, that "immediate" cancellation is not at all supported in FreeBSD's native libc_r threading environment. This causes a number of abhorent behaviors on the test server I have built under FreeBSD (4.6), which at this time almost works correctly.

There is also the port of the "LinuxThreads" package available for FreeBSD. I recently modified GNU Common C++ to support this as an optional build choice under FreeBSD. The "LinuxThreads" package, I gather, uses the FreeBSD version of "clone()", and impliments posix threads the way that Linux glibc does. The question I have, and I hope to resolve by talking with active FreeBSD developers, is if it makes sense to use and require "LinuxThreads" for supporting GNU Bayonne in the future, or if it makes more sense to work out the issues that prevent the native threading library from working.

The final challenge we have is that there is very limited computer telephony hardware choices available for xBSD systems today.

8 EuroBSD conference goals

One goal we have is to generate more interest in the BSD community in general about telephony and in particular about GNU Bayonne. We are looking for help from the BSD community in several areas. One area is to find a person to help coordinate distribution and updates of the FreeBSD "ports" collections for GNU Bayonne and dependent packages.

We also wish to discuss the various issues related to threading and various BSD platforms. In that the current GNU Bayonne developers have fairly limited exposure to xBSD events, there is a lack of full understanding of these issues in xBSD and how they relate to the upcoming 5.0 release of FreeBSD.

Ideally we would like to have contributions from the BSD community as part of GNU Bayonne. In part, such contributions would be particularly helpful in making GNU Bayonne better able to build and be used on the various BSD systems. However, contributions can take many forms, not all of which are coding.

Finally, we wish to further interest computer telephony oem's to support BSD platforms, There are many performance advantages to the BSD kernel, and many reasons it would be useful for such vendors to target development under BSD as well as under the Linux kernel. Some vendors, such as Voicetronix, already activily do this with freely licensed drivers that work on FreeBSD as well as Linux kernels. Most cti vendors neither have freely licensed drivers nor choose to support BSD at all.

9 Future Development

While XML is not an immediate area of active development, we are working in several important areas for the next major release. These include designing support for building a complete script driven PBX/telephone switch with GNU Bayonne. Such a system would offer direct application integration with a complete GNU Bayonne hosted phone system that can be used by a small office. This work is initially possible with the new line of Voicetronix PBX cards and their freely licensed drivers for both GNU/Linux and FreeBSD systems.

We are also looking to expand support for additional telephony hardware. In particular, we are interested in completing support for the Zapata telephony interfaces for the 1.1 release. These drivers are available for both GNU/Linux and FreeBSD systems and are freely licensed. The Zapata line includes support for digital (T1) voice resource cards as well as analog telephony hardware. I see this as the first oppertunity for FreeBSD systems to participate in high density digital telephony solutions such as those we use GNU/Linux for with GNU Bayonne today with commercial carriers.

Certainly, one of our most important goals is making GNU Bayonne fully available for BSD systems, and I hope that will finally be accomplished as part of the upcoming 1.1 release.

10 Acknowledgments

There are a number of contributors to GNU Bayonne. These include Matthias Ivers who has provided a lot of good bug fixes and new scheduler code. Matt Benjamin has provided a new and improved TGI tokenizer and worked on Pika outbound dialing code. Wilane Ousmane helped with the French phrasebook rule sets and French language audio prompts. Henry Molina helped with the Spanish phrasebook rule sets and Spanish language audio prompts. Kai Germanschewski wrote the CAPI 2.0 driver for GNU Bayonne, and David Kerry contributed the entire Aculab driver tree. Mark Lipscombe worked extensively on the Dialogic driver tree. There have been many additional people who have contributed to and participated in related projects like GNU Common C++ or who have helped in other ways.

Porting NetBSD to JavaStation-NC

Valeriy Ushakov

<uwe@netbsd.org>

This paper summarizes experience in porting NetBSD to JavaStation-NC, a network computer class machine built on the microSPARC-IIep processor. microSPARC-IIep is a sun4m with an integrated PCI controller. This makes it unique amongst sparc32 systems as other sun4c and sun4m models are SBus based. It is sufficiently similar to sun4m to reuse a lot of the existing code and sufficiently different to require a non-trivial porting effort. Generic and flexible machine-independent infrastructure and drivers provided by NetBSD were crucial to completing the port in short time despite the author's lack of any previous experience with NetBSD kernel internals.

Low-level support for CPU timers, interrupt related code and the like was implemented. Existing memory management code for sun4m was directly applicable. OpenFirmware support in the kernel and the boot loader, already borrowed from the sparc64 port but never tested, was completed. After machine-dependent parts of PCI framework were implemented, the driver for Happy Meal Ethernet was available for free. EBus support was borrowed from the sparc64 port, though differences in OpenFirmware prevented sharing of the EBus driver proper, but sparc and sparc64 can share drivers' EBus attachment code. The driver for CS4231 audio was reworked to support EBus attachment and is shared with sparc64.

Existing sparc framebuffer, keyboard and mouse drivers don't support the NetBSD machine-independent Workstation Console subsystem yet, but for the JavaStation-NC with its PS/2 keyboard and mouse using Workstation Console code was a natural choice, only the framebuffer driver had to be written. A simple driver for the InteGraphics Systems IGA1682 video card was developed and the plan is to support the CyberPro2010 card in this driver as well, thus providing a framebuffer driver for NetBSD/netwinder.

More work on the IGS driver is required to support acceleration and other models of InteGraphics cards. The Xigs server from XFree86 that currently supports only CyberPro 5050 seems like an ideal starting point for X support, though no work has been done yet.

Support for other microSPARC-IIep based systems would be nice, but requires access to the hardware. Besides several esoteric JavaStation prototypes the other widely available machine based on this processor is the CP1200 Compact PCI board. The same processor is also used in the SunRay appliance, though feasibility of SunRay support is questionable due to firmware issues and extremely small memory.

Valeriy started with 2.9BSD on a PDP-11. His involvement with NetBSD started with the sparc port and has been expanding along with his small but growing collection of hardware. He became a NetBSD developer in 2001. His day job involves working with Java AWT, so hacking device drivers is a refreshing change.

Porting NetBSD to JavaStation–NC

Valeriy E. Ushakov
<uwe@ptc.spbu.ru>

Abstract

Porting NetBSD to a new platform that has its CPU already supported is simplified by clean interfaces of the NetBSD kernel and a wide range of machine–independent drivers for different buses and devices. This paper summarizes experience in porting NetBSD to the JavaStation–NC and gives an overview of machine–dependent code that was necessary.

1. Introduction

The JavaStation–NC is a network computer class machine built on the microSPARC–IIep processor. The microSPARC–IIep is a sun4m with an integrated PCI controller [3]. This makes it unique amongst 32-bit sparc systems as other sun4c and sun4m models are SBus-based. It is sufficiently similar to sun4m to reuse much of the existing code and sufficiently different to require a non-trivial porting effort. Generic and flexible machine–independent infrastructure and drivers provided by NetBSD were crucial to completing the port in short time despite the author's lack of any previous experience with NetBSD kernel internals.

It is hoped that this experience will be interesting to people who need a modern OS for the device they develop or to enthusiasts who want to port BSD to their favorite gadget. Since NetBSD already supports almost all modern (and not so modern) processors, chances are that most of the hard work, like MMU and cache support, is already done. Most likely there are already machine–independent device drivers for some of the devices found in the target platform, thus further reducing your porting time and costs.

Porting NetBSD to a completely new platform is described in [2] that also outlines key features of NetBSD that contribute to its great portability.

2. The target machine

In late 1990s Sun was aggressively pushing the concept of "Network Computer". It developed several network computer class machines of which only two were more or less widely available in the wild — JavaStation-1, codename "Mr. Coffee", and JavaStation–NC, codename "Krups"[1]. All JavaStations were shipped with JavaOS.

Mr. Coffee is a "chimeric" machine. It is a straight sun4m, except equipped with commodity PS/2 keyboard and mouse. It uses the Sun TCX framebuffer, but the video connector is standard 15-pin VGA D-SUB so that it can be used with any PC monitor.

For Krups, Sun used the microSPARC–IIep processor, where 'e' stands for "embedded" and 'p' stands for "PCI". Being "embedded", the microSPARC–IIep is very low–heat, so Krups has no fan, making it a dead-silent machine. Its integrated PCI controller makes the microSPARC–IIep unique amongst other 32-bit sparc machines, but in all other respects it is a SPARC v8. Krups uses a PCIO chip that provides "Happy Meal" Ethernet and EBus (8-bit peripheral bus). The latter is used to connect PS/2 keyboard and mouse, the serial port, and CS4231 audio.

NetBSD/sparc already had complete support for SPARC v8 MMU and caches, so when the Krups port was started, most of the hard stuff was already there. As far as peripherals are concerned, Krups is sufficiently similar to PCI-based Sun Ultra machines, so the plan was to reuse as much device support code from NetBSD's sparc64 port as possible.

3. Firmware and boot loader

The first thing that was needed was a boot loader. Like Ultra machines Krups uses Open Firmware (OFW). Fortunately, NetBSD/sparc port already borrowed OFW support from the sparc64 port, though there were few minor bugs and missing bits here and there because JavaStations are probably the only 32-bit sparcs with OFW and so the OFW support had just never been tested in the 32-bit sparc port (other 32-bit sparcs use Open Boot PROM (OBP), a predecessor of OFW).

The nasty surprise was that OFW in Krups has many quirks. The worst one was that OFW, in violation of the standard, was located at `f000.0000` — the address at which the NetBSD/sparc kernel expects to be loaded. Nor-

[1] This paper will refer to JavaStations by their codenames for brevity and clarity.

mally, OFW is located in high virtual addresses, so the NetBSD/sparc kernel has a fundamental assumption that it has the space between its own end and the beginning of OFW at its disposal. As a workaround the kernel was relocated to a lower address and memory bootstrap code was tweaked to start its heap past OFW. This is a kludge, but it required changing only few lines in a couple of files and all was ready to proceed with porting, leaving the question of how to properly deal with this situation to a better time. This work was actually done on a Mr. Coffee that also had OFW with this problem, but had the benefit of having a minimal working support already, so it was easy to test the kernel relocation on it with otherwise working kernel.

There were also a few other problems, mostly related to device nodes and their properties. To avoid polluting the kernel with numerous special cases and workarounds for OFW quirks, the boot loader was modified to "patch" the OFW before loading the kernel. Since OFW is a full-fledged Forth environment, this was easily achievable with small pieces of forth code that boot loader passes to OFW to execute.

Linux took a different path. Pete Zaitcev implemented PROLL, a small OBP simulator that provided to the kernel the device tree and a minimal subset of OBP entry points that Linux sparc kernel used. PROLL completely replaces machine's OFW. However having OFW around is quite handy to be able to inspect hardware state interactively, so for the NetBSD port, the case was decided in favor of OFW despite the necessary kludges described above.

4. Overall port plan

When the boot loader is written the next milestone is to mount the root filesystem. There are two popular choices. One possibility is to embed a minimal root filesystem into the kernel, the other is to mount root from NFS server. For both approaches, some sort of console support is required, and for the diskless boot, a network driver is also necessary. The latter approach is often attractive because once the network driver is complete the system will be able to boot multiuser directly with both root and swap on NFS, thus passing two milestones in one leap.

NetBSD/sparc can use firmware for its console input and output. As noted in the previous section OFW support was completed during the work on the boot loader and as that code is shared by the boot loader and the kernel the console was functional even before the kernel can do anything useful. That was very helpful during development of early kernel bootstrap code.

Happy Meal Ethernet at PCI is supported by the `hme(4)` driver, so diskless boot was chosen for the next milestone. Thus the only things missing were the most low level code to deal with interrupts, timers and the like, and the machine–dependent parts of the PCI framework.

5. Low level code

The microSPARC–IIep has totally different system registers to raise software interrupts, report pending interrupts, control system and processor timers, etc. While learning the intricacies of system operation and writing support for those low level things was, perhaps, the most interesting part of the project, it is also the most boring part of the project to describe, so this section will only give a short summary of things done.

While some assembly hacking was required, only three short assembler routines were written. `sparc_interrupt4m` — the interrupt trap handler. `raise()` — the function to raise a software interrupt. `microtime()` — the function that reports current time in μs.

There is, actually, another assembler routine that needs to be written but hasn't been yet — the routine to handle non-maskable interrupts that indicate system malfunction. But it is not necessary for the *normal* system operation after all, so it was postponed to some later time.

Bootstrap code and the initial autoconfiguration process were tweaked to reflect new CPU variant support. Details of mapping between PCI and physical address spaces and interrupt routing were encapsulated in a driver that provided usual `bus_space(9)` interface [1]. Finally, kernel clocks that use system and processor counters were implemented.

6. PCI framework

NetBSD has a machine–independent PCI framework that needs only few typedefs and functions provided by the machine–dependent code[2]. These are usually declared in the port's `<machine/pci_machdep.h>` header file. Please refer to `pci(9)` and `pci_intr(9)` for function signatures.

[2] Section 9 of the NetBSD manual does not (yet) fully document what types and functions must be provided by machine–dependent PCI code. This section is intended to summarize the current situation.

An important type that the port must define is `pci_chipset_tag_t`. It is a chipset tag for the PCI bus. Effectively, it describes a root for a hierarchy of PCI buses. The chipset tag is passed to almost all machine–dependent functions described below.

Since the microSPARC–IIep has an integrated PCI controller there is no need to provide for different possible PCI chipsets, and the chipset tag just carries some private data. But e.g. on Alpha the chipset tag also contains pointers to functions that implement machine–dependent methods for each PCI chipset that can be found in Alpha machines.

6.1. Autoconfiguration

`pci_attach_hook()`
: The hook called right before each pci bus is attached during autoconfiguration.

`pci_bus_maxdevs()`
: Returns a maximum number of devices for the given PCI bus.

`pci_enumerate_bus()`
: Necessary if the port needs some special bus enumeration. For the microSPARC–IIep, it is a macro that just calls machine–independent `pci_enumerate_bus_generic()`.

6.2. Device tags

`pcitag_t`
: Configuration tag describing the location and function of the PCI device. Opaque to the PCI framework. On sparc, the `pcitag_t` is a 64-bit integer that encodes OFW device node for this PCI device and the tuple <*bus*, *device*, *function*> in a form used for PCI configuration accesses.

`pci_make_tag()`
: Construct `pcitag_t` value for bus, device, function.

`pci_decompose_tag()`
: Return bus, device, function for the PCI tag.

6.3. Conf space access

`pci_conf_read()` and `pci_conf_write()` are used to access PCI configuration space. The microSPARC–IIep uses standard mode 1 configuration accesses so implementation of these function is straightforward.

6.4. Interrupt manipulation

`pci_intr_handle_t`
: A handle describing an interrupt source. Opaque to the PCI framework that uses the following functions to manipulate interrupts.

`pci_intr_map()`
: The function takes a pointer to `struct pci_attach_args` and maps it to a `pci_intr_handle_t`.

`pci_intr_string()`
: Returns a string describing interrupt source that the driver can use if it wishes to refer to it in an attach or error message.

`pci_intr_establish()`
: Actually establish the interrupt handler for `pci_intr_handle_t` mapped with `pci_intr_map`. Returns a cookie that can be passed to `pci_intr_disestablish()`.

`pci_intr_disestablish()`
: Disestablish the interrupt handler previously established with `pci_intr_establish()`.

`pci_intr_evcnt()`
: Returns the event counter that is the parent for all interrupt–related counters associated with the given PCI bus hierarchy. Refer to `evcnt`(9) for the description of the NetBSD generic event counter framework.

7. First boot

After the steps outlined in preceding sections were completed, the Krups was able to boot multiuser off the NFS. Of course it lacked a lot of device drivers, most annoying was the lack of driver for the time–of–day clock that in Krups is connected via EBus, but nonetheless the machine was self-hosting at this point.

It took about a month from the beginning of the project to the first boot. However it should be noted that this was author's very first experience with both NetBSD kernel programming and with programming something *that* low–level. A seasoned NetBSD hacker could have probably done it in under a week.

While clean interfaces of the NetBSD kernel that support code portability were crucial in completing the Krups port very quickly, they also allowed this small project to contribute back to the NetBSD more then just yet another platform support. The remainder of this article gives some ex-

amples of code that was developed for Krups, but was immediately useful for other platforms.

8. Audio

Device drivers in NetBSD are split into bus–independent code that drives the device and bus–specific attachment code. Bus–independent code uses `bus_space(9)` abstraction layer [1]. While EBus is commonly found in PCI-based Ultra machines and NetBSD/sparc64 has a driver for it, unfortunately the driver can not be used for Krups because OFW properties of EBus bus node and its children are very different and often incomplete in Krups. However it is desirable to share the EBus–specific drivers' attachment code between two sparc ports.

A notable example is `audiocs(4)`, a driver for CS4231 audio that is found under both SBus and EBus in sparc and sparc64 machines. At the time the driver supported only SBus and only playback. Some rudimentary EBus support was written for sparc64 but was far from complete. Also some SBus specific code was polluting the machine–independent part of the driver.

The driver was refactored so that bus–specific details are removed from machine–independent code and EBus playback support was completed. At that point it turned out that the driver internal interfaces allow to add capture support almost trivially. This is a good smaller–scale example of how clean interfaces of the NetBSD kernel contribute to its unparalleled portability by greatly simplifying development.

The refactored driver was developed on JavaStations, Mr. Coffee (SBus) and Krups (EBus), and when it was complete the sparc64 port automatically got full CS4231 support as well.

9. Graphic card

The graphic chip in Krups is IGA 1682 from Integraphics Systems (now Tvia). Fortunately, the good folks at Tvia kindly provided technical docs for it. It was decided to use the machine–independent "workstation console" subsystem (`wscons(9)`) for it. The rest of the NetBSD/sparc port doesn't use wscons yet, but wscons is intended as the standard console subsystem, and there was no existing code for the IGA 1682 at the time, so it made sense to write the new driver to support the intended standard. As a side note — compare this to Mr. Coffee, that uses Sun TCX framebuffer for which the driver already exist. For Mr. Coffee it was faster to develop PS/2 keyboard and mouse drivers that conformed to old Sun interfaces. That made Mr. Coffee supported with stock `Xsun(4)` binary.

For writers of framebuffer drivers NetBSD provides generic raster operations (`rasops(9)`) that implement text rendering and unaccelerated blitting. The driver shall implement `wsdisplay(9)` interface so that that upper layers of wscons can attach to it. With completion of the drivers, Krups got a real console.

More recent Integraphics chips, CyberPro2000 series, are also used in several other machines that NetBSD runs on. Matt Thomas provided a Corel Netwinder machine for developing CyberPro support in the driver. It turned out that only minimal extensions were required. The biggest problem was that a complete chip init is required for Netwinder, a task that on Krups is performed by the firmware and so can be skipped in the driver. Details of the chip initialization (on which Integraphics docs are extremely scarce) were mostly learned from the Forth code of Krups firmware.

10. Acknowledgements

Pete Zaitcev, who did the Linux port to the microSPARC–IIep, kindly provided a lot of hints on hardware operation. Matthew Green, Eduardo Horvath and Paul Kranenburg provided valuable insights into obscure corners of the low level sparc code. Eduardo Horvath and Jason Thorpe helpfully clarified details of generic NetBSD kernel interfaces, the PCI subsystem in particular, and patiently replied to numerous questions. Martin Husemann has done a lot of testing and debugging for Krups in general and for `audiocs(4)` on sparc64 as well.

References

[1] Chris Demetriou. `bus_space(9)` manual page. Originally in NetBSD 1.3, 1997.

[2] Frank van der Linden. Porting NetBSD to the AMD x86-64: a case study in OS portability. In *Proceedings of the BSDCon 2002 Conference.* Usenix, 2002.

[3] Sun Microelectronics. *microSPARC™–IIep User's Manual.* Part number #802-7100-01. Sun Microsystems, April 1997.

Mac OS X on a budget

Gerald Wilson

<gww@stonehill.org.uk>

Many BSD users may have an interest in Mac OS X, but cannot justify buying a new Macintosh computer. They need a solution for a tight budget.

The presenter has previously demonstrated Mac OS X running on a home-built system, constructed mostly from old parts, with a total parts cost of EUR 600. The presenter has built several such systems, including a server which cost EUR 300 in parts. While it is not difficult to create a Mac OS X installation on a system not supported by Apple, there are numerous traps and gotchas to avoid.

This session describes the process of creating a Mac OS X system on a budget:

- What processors and motherboards are supported?
- What disk configurations are needed?
- Which peripherals work and which don't?
- What software is needed for unsupported systems?
- How much will it cost?
- Which second-hand systems give best results?

... and so on.

Gerald Wilson has worked for more than twenty years in technical computing, as programmer, designer, and technical manager. In that time, Gerald has worked on control systems for laboratories and factories, avionics systems, military communications systems, and most recently naval command systems. Over the years he has also spent many happy hours as network and systems manager for technical development teams. Gerald has an unhealthy depth of knowledge about Macintosh computers, a working knowledge of UNIX, and can install Windows at a pinch. Gerald thinks that Dr Edgar David Villanueva Nunez should be the next Secretary General of the United Nations.

Mac OS X on a Budget
Gerald W Wilson

1 Introduction

1.1 Preamble

This is a practical paper, about a practical subject, for practical people. If you get nervous around things like screwdrivers and Lego, you'd best stick to writing software.

This paper is aimed at those who wish to run **Mac OS X**, but whose budget to do so is limited. It should help those who are familiar with traditional Macintosh hardware and software, but have little experience of Mac OS X; and more particularly it should help those who are more familiar with PCs than Macs. It is not intended as a source of information about older Mac OS issues, although it draws on sources of data about those.

My favourite quotation of this year comes from Bill Gates. In the continuing courtroom drama of the antitrust trial between the US authorities and Microsoft Corporation, Gates himself gave testimony under oath as an expert witness in April 2002. Here [reference 1] is Gates, speaking as Microsoft's "Chief Software Architect", stating his opinion of the doubtful value of modular coding practices:

"In a purely theoretical world, one could imagine developing modest software programs in such a way that any module could be swapped out in favour of a similar module developed by a third party ... In the commercial world, it is hard to see what value such replace-ability would provide even if it is achieved."

To anyone with the slightest education in software engineering, this view is bizarre. Fortunately, the UNIX philosophy has been to construct operating systems out of "replaceable modules" from the very start, and Mac OS X is no exception. Consider the layered structure of Mac OS X, as Apple originally portrayed it in January 2000.

Mac OS X is – of course – derived from the NeXTStep operating system which Apple acquired when it bought NeXT for Christmas in 1996. As the foundation layer for Mac OS X, Apple has therefore adopted BSD, running over the Mach microkernel. While Apple has borrowed from various flavours of BSD, it has chosen to create its own particular variant, named "Darwin", to provide the BSD reference base for Mac OS X. Darwin, together with some other Apple software packages, is released as Open Source.

The benefits of founding Mac OS X on Open Source Software are the same benefits seen by all Open Source projects. The disadvantage, from Apple's commercial point of view, is that OS X can be hacked – hacked, not cracked – which allows it to be used in ways Apple has not intended. Oh well – that just goes with the Open Source territory...

So, in this paper, I shall cover two things:

- For those who prefer to play safe, how to choose a budget Mac for their OS X needs.
- For those who like to live dangerously, how to run OS X on unsupported hardware, and what kinds of issue that brings.

All recent Macs run the traditional Mac OS well, but Mac OS X makes different demands. Some recent Macs are not well suited to Mac OS X and are better avoided; while some older machines can be tricked out for Mac OS X with good results.

At the present time there are two reasons why OS X requires Mac hardware:

1 **Technology Constraints**: While Apple's internal labs undoubtedly test versions of Mac OS X for other hardware architectures, nothing is yet released, nor is likely to be before 2004. If you wish to explore Mac OS X right now, you need a Mac.

2 **Licencing Constraints**: Apple's software licence for Mac OS X states clearly that it is for use on an "Apple-badged computer". You have been warned...

1.2 Wealth Warning

Please be clear. Apple makes excellent computer products, which are good value when taking into account the quality of design, quality of construction, and added worth of the operating system and bundled applications. If you wish to run Mac OS X, and your budget is plentiful, your best value is – without doubt – to buy a new Macintosh system from your nearest Apple supplier, and enjoy the support and warranty that brings. However, "value" is not the same as "I can afford it". There are many reasons why you might not want a brand new system, but still want to run Mac OS X. Here are some:

- **Poverty**: You may be short of money, but still wish to use OS X. Maybe buying or upgrading a used system will get you what you want at a price you can afford.
- **Cash-flow**: Your organization may not permit the level of spending needed to buy a whole new Macintosh. Maybe you can build it up a piece at a time – "salami tactics" – in order to get the system you want.
- **Prototyping**: You may need to experiment with OS X before you can justify a full purchase. Maybe you can build a prototype system on lesser hardware to confirm whether or not the system will be right for you.
- **Networking**: You may wish to evaluate the network facilities in OS X. Even if your budget runs to one new Mac, it may not cover two or more needed for valid network tests. Maybe you can acquire the extra OS X nodes for your tests at budget prices.
- **Configuration**: At any given time, Apple makes only certain machines in certain form factors. If the current range can't supply your needs, maybe you can meet them by configuring an older model, to get a system with the mix of interfaces you want.
- **Availability**: You may already have an older machine capable of running OS X. Maybe you can convert it to run OS X for much less than a new machine.
- **Compatibility**: You may have a group of machines in a workgroup or a school, which you wish to keep at a compatible standard. Maybe you need to know what you can do to convert them all, economically, to run compatible versions of OS X.
- **Learning**: A good way to learn about technology is to pull it apart and put it together again – and make sure it still works afterwards. To build greater knowledge of OS X, maybe you wish to take this approach in order to learn from the experience.
- **Challenge**: Admit it to yourself. You just like exciting challenges, and you've already climbed K2 and Denali this week. An unsupported OS X install is calling for you.

Be clear again: by following these guidelines, you should be able to run Mac OS X for less **cash**, but you will still have to pay – mainly in **time** and **effort**. No free lunch!

Spend with care. It is too easy to waste money by spending it a little at a time, when better value would have been to find funding for all you need. Think before you start.

1.3 Acknowledgements

Much of this would have been impossible without the work of **Ryan Rempel**.

Ryan determined, early on, to use the available knowledge about NeXT Step and Rhapsody (the precursors to Mac OS X) to maintain the availability of OS X on older unsupported Macintosh hardware. Backed by **Other World Computing** [reference 2], Ryan has worked tirelessly for two years releasing the patches needed to get OS X running on older machines, and is constantly extending the reach of his knowledge and the range of supported equipment. Give the man a Nobel Prize.

Ryan's early efforts were complex to follow, but led to a simple, Open Source utility now called XPostFacto. If you use XPF, please help Ryan by paying to join his support list.

Dan Knight runs the "Low End Mac" web-site [reference 3], which is an excellent portal into information about Mac-related topics, and in particular data for older Mac hardware. For understandable reasons, Low End Mac has only recently started to promote OS X.

Mike Breeden runs the "Accelerate Your Mac" web-site [reference 4], which acts as a portal for information about Mac hardware upgrades and tune-ups.

There are numerous web-sites now covering Mac OS X, Darwin and their BSD heritage. I will not reference them here, since they are still somewhat volatile. Just search the web.

1.4 What does "Budget" mean?

1.4.1 The Price of New Macs

For pricing in this paper, I have used UK prices, less local sales tax, converted to Euro at the rate of about £1 = €1.6. For rough comparisons, €1 = $1.

In early October 2002, typical UK prices for new budget Macs are shown in Table 1:

Table 1: Typical UK prices for new budget Macs

Model	processor	Screen	RAM(MB)/ Disk(GB)	Optical Drive	Graphics	Price in €uro
iMac	G3/600	15" CRT	128/40	CD-ROM	Rage 128	880
eMac	G4/700	17" CRT	128/40	CD-RW	GeForce 2MX	1,120
iBook	G3/600	12"TFT	128/20	CD-ROM	Radeon Mobility	1,360
iMac	G4/700	15"TFT	128/40	CD-RW	GeForce 2MX	1,360

These are Apple's base models at time of writing. Each is pre-loaded with OS X. In each category, spending more can get you more RAM, a larger disk, a faster processor and a more capable optical drive. Extra RAM is the best way of improving OS X performance. Other improvements will give you more capability, but little improved performance.

The bargains here are the eMac and the iBook. Each is outstanding in its way. If these will suit your needs, stop reading now, buy the one you want (more RAM!) and that's it.

1.4.2 The Price of Used Macs

I assume that "budget" means that you wish to run OS X for significantly less than €1,000. How much less depends on your needs and wants. How much a used budget Mac costs will depend on where you live. In the USA, with easy access to eBay, the prices are likely to be less than in Europe and other territories.

In early October 2002, typical UK prices for used budget Macs are as follows:

PowerBook G3/266	128MB/2 GB/CD	€800
iMac G3/266	160MB/6 GB/CD	€450
PowerMac 7600/132	64MB/2GB/CD	€150

Each of these can be coaxed into running Mac OS X. None of these will give stellar performance, but each will get you going for less money. Remember though that OS X itself will cost you at least €100 to obtain.

2 Understanding Apple's Technology

2.1 The Basics: Supported and Unsupported Installs

Whatever your motive, if you're doing Mac OS X on a budget you need to know your technology better than the average Macintosh user.

For Mac OS X, a "supported" system means one which Apple itself lists as approved for use with OS X. In practice, this means every Mac of any kind originally shipped with a PowerPC G3 or G4 processor, with one exception: the original PowerBook G3. However, there are some constraints, such as the graphics support. This varies with the version of OS X you use. To understand the issues, you need to know what graphics chipset you plan to use, and what OS X can do with that chipset.

An "unsupported system" is therefore anything else which can be persuaded to run Mac OS X by trickery. Apart from RAM and disk, there are four essential requirements:

- a processor Mac OS X will recognise;
- PCI architecture;
- Open Firmware;
- An optical drive from which to load OS X.

2.2 Recognised Processors

In the early 1990s, Apple, IBM and Motorola (AIM) agreed to co-operate to miniaturise IBM's RISC workstation processor architecture ("POWER") as a single scaleable microprocessor family to be called PowerPC. In contrast with the intel x86 architecture, PowerPC is naturally big-endian, although it can be run little-endian as well. The first PowerPC chip was the 601, and was a hybrid designed to link older and newer technologies. The true PowerPC (desktop) families began with the 603 and 604 series.

The 603 started life as a low-dissipation chip intended for budget and portable use. Apple used it successfully for PowerBooks and home computers. The 603 has a limited number of execution units, and lacks support for Symmetric Multi-Processing.

The 604 started life as a high-dissipation chip intended for workstations and servers. Apple used it as their principal cpu for professional computers. The 604 has many execution units, good floating-point performance, and in-built support for SMP.

From the 603, AIM derived the 750 family, known by Apple as "G3". This is a good performer, but like its parent lacks support for SMP.

From the 604, AIM derived the 7400 family, known by Apple as "G4". Motorola, who pioneered these designs, added an efficient vector-processor to the G4, like the SIMD components added to the Intel Pentium and AMD K6 architectures. Motorola's formal name for the vector processor is "Altivec". Because the Altivec SIMD unit can handle groups of four floating-point operands, using dedicated registers, it can provide close to a four times speed-up for code compiled to exploit it. See NASA's opinion [reference 5].

At time of writing, it is expected that Motorola will introduce a more advanced processor ("G5"?) in 2003, and expected that IBM will launch a 64-bit PowerPC micro-processor, complete with Altivec-compatible vector unit, in late 2002. However, neither of these is likely to appear in Macintosh computers this year (2002).

The officially recognized processors are therefore the G3 and G4. Unofficially, Mac OS X can be persuaded to run on the 603 and 604 as well. Remember, processors develop variants as they evolve throughout their product life. For example, the 604e (larger caches, higher clock-speeds) works better with OS X than the original 604.

2.3 The available versions of Mac OS X

2.3.1 What Apple has released

Ignoring its predecessors (NeXTStep and Rhapsody) Apple has so far made four public releases of OS X. All of these use a graphics library which Apple calls Quartz (derived from Adobe's Portable Document Format), with a look-and-feel which Apple calls Aqua.

Public Beta: released September 2000. This has little value. It is time-limited, and too flaky to be of interest except for museums. Recycle the CD as a drinks coaster.

10.0.x, aka Cheetah: released March 2001: Last update (to 10.0.4) released June 2001. This is of little practical use. While it will run stably, it lacks many important features, and differs significantly in interface from the later versions. However, it still has a value. Cheetah can be updated free-of-charge to OS X 10.1. You will need to obtain a copy of the update media, but the licence allows this. (I have several registered copies of OS X 10.0, but Apple itself only supplied me one copy of 10.1 with which to update them all.)

10.1.x, aka Puma: released October 2001: last update (10.1.5) released May 2002, with security patches released up to August 2002. This is a stable and useable version of OS X. It is the baseline version for many OS X applications (such as Microsoft Office for X). If fully patched, it is reasonably brisk and secure, and its underlying UNIX can be used for X-Windows applications and UNIX command-line applications. It provides decent support for Java 2 (at v1.3.1). It runs well on G3 and G4 cpus (supported) and runs fairly well on 603 and 604 cpus (unsupported).

10.2.x, aka Jaguar: released August 2002: currently updated to 10.2.1. This is, in general, significantly improved over OS X 10.1.x. Many things work faster. Jaguar includes a new graphics scheme called "Quartz Extreme". This improves 2D graphics performance by drawing the screen image as an OpenGL scene, which can then be rendered more swiftly for display by a suitable OpenGL-capable graphics card. However, Jaguar at time of writing will only run on G3 and G4 cpus. Further, Jaguar is based on a later version of Darwin, including the compiler suite gcc 3.x and its libraries. Hence some code built for Puma will not work correctly on Jaguar until it has been recompiled.

2.3.2 What you should aim for

OS X 10.1.5: stable, reliable and secure; based on gcc 2.95; 2D Quartz acceleration down as far as the Rage Pro family; can use 603 and 604 cpus (unsupported).

OS X 10.2.x: newer and smarter, but not yet fully stable; based on gcc 3.x; Quartz Extreme when used with suitable graphics card; only for G3 and G4 cpus.

Apple's policy for selling Jaguar is to sell only the full version, not an upgrade from Puma (or Cheetah). This means that anyone who has bought a copy of Jaguar has a full licence for Puma which they can re-sell. Under EU law, a person who has bought a software licence is entitled to sell that licence separately from its original hardware. So a person who bought a Mac originally loaded with Puma is entitled to sell on that licence if they buy a copy of Jaguar to replace it. Hence, in the EU at least, there should be a ready market in pre-owned licences for Puma for use on Apple-badged computers.

2.3.3 PreBinding and OS X

In Mac OS X, prebinding is the process by which the system optimizes the dynamic links between applications on the system's boot volume and system libraries, to make those applications launch and load faster. Cheetah did not exploit prebinding. It is also compromised in many other ways. Puma introduced prebinding to speed up application launch times, and it is an effective improvement. There is no good reason to stick with 10.0, since if you own a legitimate licence for 10.0 you can update to 10.1 without charge.

An irritating consequence of Puma's prebinding is that every major installation of an OS X patch or an application causes the installer to run an extensive "Optimisation" phase, which is boring but necessary. In Mac OS X 10.2, Apple modified the approach to prebinding to eliminate this effect. On Jaguar, if the system discerns that prebinding information has become out-of-date, it automatically updates it to bring it back into line.

2.4 Graphics and Displays

For several years, Apple used only graphics chipsets made by ATI. More recently, Apple selected nVidia as a second source of graphics hardware. While OS X's Quartz is supported on anything from ATI Rage II onwards, some features demand better hardware. The cut-off points for each chipset family are these:

ATI Rage II: Basic OS X display; no acceleration features;

ATI Rage Pro: Basic 2D acceleration and QuickTime acceleration (10.1.5 or later);

ATI Rage 128: Full 3D acceleration;

ATI Radeon AGP: Quartz Extreme acceleration;

Nvidia GeForce 2MX (or better) AGP: Quartz Extreme acceleration.

In the professional desktop Mac systems, the graphics hardware can be replaced or upgraded via AGP or PCI slots, but all other types of Mac have fixed graphics hardware. Hence, if you need a particular graphics capability, you must ensure that the Mac you buy can handle it. This can be confusing. Each graphics chipset family has variants for different purposes, and it is sometimes unclear just what a particular Mac contains.

Mac OS X is very demanding of screen area. The large, "photographic" icons occupy many pixels, and the desktop can seem cramped on a small screen area. While OS X will support screen areas as small as SVGA (800x600) you would be unwise to contemplate a screen area smaller than XGA (1024x768) for everyday work.

2.5 External Interfaces

Macs have never had parallel interfaces. Apple's standard external interfaces are these:

Serial: Introduced in the original 1984 Macintosh, and improved over the years. Still found in some supported models, so has limited support in OS X.

ADB: The **Apple Desktop Bus** - an early daisy-chain bus mainly for mice and keyboards - still found in some supported models, so has limited support in OS X.

SCSI: The mainstay Apple expansion port for many years, and still found in some supported models, so has limited support in OS X.

Ethernet: Built into all supported Macs, so fully supported in OS X. Most supported models have Fast Ethernet or Gigabit-over-UTP.

USB: Introduced with the first iMacs in summer 1998. Now standard in all Macs, so fully supported in OS X.

1394: Apple invented this technology, under the brand "FireWire". Introduced in desktop models in early 1999, and now standard in all Macs, so supported in OS X.

WiFi: Apple pioneered this in conjunction with Lucent under the brand-name "Airport" Introduced with the first iBooks in late 1999, and now a standard option for all Macs.

2.6 Form Factors

In May 1998, at Apple's World-Wide Developer's Conference, Steve Jobs announced a new and simplified product policy for Apple. From that time on, Apple would focus on creating only two kinds of computer – the desktop and the portable – for two kinds of user – the professional and the consumer. Eventually the range consolidated like this:

Table 2: Apple's Product Policy

	Consumer	Professional
Desktop	iMac, eMac	Power Mac G3, G4
Portable	iBook	PowerBook G3, G4

The only significant variations to the range since have been the Power Macintosh G4 Cube and the Xserve. The Cube, available for only a year, was marketed as a "Designer Workstation" but has limited expansion. Its graphics is on an AGP card, so can be upgraded. The Xserve is designed as a 1U rack-mount system, mainly for use as a server. These designs are too specialized to be covered here.

2.7 RAM, MotherBoards and Chipsets

Mac OS X demands RAM. While it will run in 64 MB, it is then too slow for everyday use. In practice less than 128 MB is unusable for work, while 256 MB or more is highly desirable. This is partly because of the extravagant way in which OS X's Aqua graphics scheme uses RAM to buffer screen images.

The need for RAM seriously reduces the usefulness of some older Macs with OS X. Some machines have upper limits on RAM which are too low for comfort; others can accommodate plenty of RAM, but of a type which is rare and expensive.

Hence your choice of a used Mac for use with OS X must take account of the availability and cost of the RAM you will need. All recent Macs usable for Mac OS X take some kind of standard PC-compatible RAM (PC66, PC100, PC133 or DDR), as either normal SDRAM DIMMs, or as low-profile SO-DIMMS intended for use in notebooks.

For the desktop Macs with PCI slots, the capabilities of the machine are defined more by its motherboard than by the name on the case. Here is a short summary of the PCI slot motherboards able to run Mac OS X. For more detailed discussions, consult the Power Macintosh hardware pages maintained by NetBSD [reference 6] and OpenBSD [reference 7].

2.7.1 TNT, Nitro, and Tsunami – Old World ROM

These boards were launched as the "PowerSurge" range, and share many common features. The maximum specified bus speed is 50 MHz, although, because of the way the RAM is timed, this gives memory speed equivalent to a Pentium PC at 66 MHz. RAM comes as 168pin Fast Page Mode (or EDO) DIMMs, which are rare and expensive. The cpu daughter-board fits in a slot, similar to a Pentium II, and can be exchanged. There are two SCSI buses - one fast for internal use and one slow for external peripherals - and the boards have built-in Ethernet. The three-slot version (TNT, Nitro) has built-in graphics and was used in the Power Mac 7500, 7600, 7300, 8500 and 8600 series. The six-slot version (Tsunami) was used in the 9500 and 9600, and has some important differences. It lacks built-in graphics, so needs a suitable PCI graphics card. It has twelve rather than eight DIMM slots, and its Level-2 cache is soldered on, rather than being slot-fitted.

These boards can be temperamental about the combination of cpu, cache and RAM fitted to them. In theory, the memory manager can interleave the RAM to improve bus throughput, but in practice the DIMMs must be carefully matched for this to work without problem. However, once set up and working the boards are reliable.

2.7.2 Gossamer – Old World ROM

These boards were the first Apple G3 boards created for professional desktop machines. They were fitted in Power Macintosh G3 systems as desktops, minitowers, and all-in-one systems for education users. The nominal bus speed is 66 Mhz, but there are hacks to take the board to 75 Mhz or 83 MHz if you fit PC100 RAM and the rest of the board can keep up. These are the last desktop motherboards made by Apple with built-in graphics.

There are several revisions of the board, which give variations in the Boot ROM, IDE behaviour, and graphics performance. Select with care.

2.7.3 Yosemite – New World ROM

These boards were fitted in the Blue-and-White Power Macintosh G3 series. Though superficially similar to Gossamer, the boards have a faster bus at 100 MHz, support for FireWire and USB, and improved support for IDE drives.

Again, there are several revisions of the boards, which can affect hardware capability.

2.7.4 Yikes

A variant of Yosemite adapted to take the G4 cpu instead of the G3.

2.7.5 Sawtooth, and beyond

With the introduction of the full Power Macintosh G4 series, Apple created their first board with a true AGP slot for graphics. Over the years, the board has been progressively enhanced while keeping the same essential characteristics, gaining faster bus speeds, faster disk interfaces, and improved network interfaces.

3 Selecting a Used Budget Machine

3.1 General Criteria

Armed with all this information, what now should you do? I will assume that you do not want an Xserve, since it is too new and too specialized to be considered a budget item. Likewise, I shall assume you aren't after the Power Macintosh Cube, because if you are, you already know what you want. Instead, I assume you have one of four aims in mind:

- An affordable portable Mac OS X machine
- A low risk, low specification, supported desktop machine
- A low risk, high specification supported desktop machine
- A high risk unsupported machine

Here, though, you meet a significant problem. Apple revises the specifications for its models every few months. The specifications published on the Internet are very ragged – even Apple's own data. Once you have selected your style of Mac, and identified your candidate machines, you need to check the specifications carefully, and compare several source of data, to ensure that what you get will do precisely what you want.

3.2 Case 1 – Affordable Portable

The PowerBook G4 series, with its wide screen, and stylish Titanium case, cannot be considered a budget item. There are reports of poor Airport performance.

Hence a budget Mac OS X notebook is a G3. While the first iBooks (the curvy coloured models) have some good features, their screens support only SVGA which limits their use for OS X work. The new style white iBooks – "iceBooks" – introduced in May 2001, are much preferred. Late versions of the PowerBook G3 are all suitable machines, each with their better and weaker points. All these Mac G3 portables have built-in modems and built-in Ethernet (or Fast Ethernet). Table 3 lists affordable Mac OS X portables.

For a good mobile balance between size, weight, strength, battery life, and wireless networking, any of the **Lombard**, **Pismo**, and **iceBook** ranges can be tailored to suit your needs. The (black) Lombard and Pismo have expansion bays, which can be used for alternative media; while the (white) iceBooks are less flexible. Note that to add Wifi to Lombard and WallStreet you need a compatible PC card, such as a Lucent Orinoco.

Table 3: Typical Specifications of affordable G3 portables

Model	Speed cpu /bus	Max RAM (MB)	Disk Fit (GB)	Optical Fit	Graphics	Comments
WallStreet	233+/ 66+	128	2	CD-ROM	Rage LT Pro	Compromised: Best avoided
WallStreet II	233+/ 66/	192	4	CD-ROM	Rage LT Pro	Good basic model; Strongly built; 2 PC card slots.
Lombard	333+/ 66	384	4	DVD	Rage LT Pro	Slim and modern; USB built-in; Limited expansion 1 PC card slot
Pismo	400+/ 100	512	6	DVD	Rage 128 Mobility	Full featured; FireWire and USB; Airport Option; 1 PC card slot.
iBook Rev A	300+/ 66	320	3+	CD-ROM	Rage Mobility	Screen SVGA max.
iBook Rev B	366+/ 66	320	10	CD-ROM DVD	Rage 128 Mobility	Screen SVGA max.
i(ce)Book 2001	500+/ 66+	640	10+	CD-ROM DVD CD-RW	Rage 128 Mobility	Light and compact 12" XGA screen; FireWire and USB; No PC card slot; Airport Option.
i(ce)Book 2002	600+/ 100	640	20+	CD-ROM DVD CD-RW	Radeon Mobility	12" and 14" versions.

3.3 Case 2: Low Specification Supported Machine

3.3.1 The iMac option

Here the obvious candidates are the CRT iMacs. Technically speaking, the original iMacs were PowerBooks adapted to fit a desktop CRT case. As such, the early iMacs share similar strengths and weaknesses to the PowerBook G3s to which they are related. All iMac models support XGA display. All have USB ports and built-in Fast Ethernet. Early models have tray-loading CD-ROM drives and use SO-DIMM memory designed for portables. Later models have slot-loading drives, and use standard PC100 (or faster) SDRAM, with options for FireWire, Airport, and more capable optical drives.

The Rev A to D (Tray-loading) iMacs can be upgraded to faster speeds using parts from third-party manufacturers, such as Sonnet. Thus a Rev A model can be converted to run at 600 Mhz, with an added FireWire port. Likewise there are upgrades to change the original tray-loading CD-ROM for a CD-RW. Only you can decide whether such a radical improvement is worthwhile for an older machine.

You may need to update Firmware to run Mac OS X on an early iMac. Firmware updates can cause third-party RAM to become unrecognized. Check specifications before you try.

Table 4: Typical Specifications of CRT iMacs

Version	Speed cpu /bus	Max RAM (MB)	Disk Fit (GB)	Optical Fit	Graphics	Comments
Rev A	233 /66	128	4	CD-ROM	Rage IIc	Limited: Best avoided
Rev B	233 /66	256	4	CD-ROM	Rage Pro	Good basic model
Rev C	266 /66	256	6	CD-ROM	Rage Pro Turbo	Now comes in colours.
Rev D	333 /66	256	6	CD-ROM	Rage Pro Turbo	
Kihei	350+ /100	512	6+	CD-ROM DVD	Rage 128	Faster models have extra options for FireWire, Airport and more capable optical drives.
2000	350+ /100	1024	10+	CD-ROM DVD	Rage 128 Pro	
2001	400+ /100	1024	10+	CD-ROM DVD CD-RW	Rage 128 Ultra	Adds CD-RW versions

So your CRT iMac options boil down to three:

- For least hassle, carefully study specs, and buy the model which has what you want;
- For least money, buy an early model with enough RAM and use it exactly as is;
- For maximum flexibility, buy a Rev B, C, or D iMac and enhance it with upgrades to the standard you want.

3.3.2 The Desktop Option

The alternative to the iMac is to use a G3 desktop machine, either as a Beige case G3 (Gossamer) or a Blue-and-White (Yosemite). Since these take standard PC parts, like PC66 or PC100 SDRAM and IDE drives, they are easy to enhance for use with OS X.

Table 5: Desktop G3 Configurations

Version	Speed cpu /bus	Max RAM (MB)	Disk Speed	Optical Fit	Graphics	Comments
Gossamer Rev A	233+ /66	384	?	CD-ROM	Rage II+	Limited: Best avoided
Gossamer Rev B	266+ /66	384	?	CD-ROM	Rage Pro	Good Basic model
Yosemite	300+ /100	1024	33	CD-ROM	Rage 128 PCI	Revised coloured case; Adds: UDMA-33, FireWire and USB, Fast Ethernet.

3.4 Case 3: High Specification Supported Machine

The high-specification supported machines are, by definition, all G4 Minitowers. Since these can hold multiple hard drives, there is no standard disk fit. Any individual machine might have been built-to-order to a bespoke configuration. Likewise, the graphics cards may vary from standard. Hence select with care.

These systems are still highly regarded by professionals working in graphics and audio. They can be expensive to buy as used items. Dual processor systems are rare.

Dual processor models have been the 450DP, 500DP, 533DP, 800DP & 1 GhzDP.

Table 6: desktop G4 Configurations

Version	Speed cpu /bus	Max RAM (MB)	Disk Speed	Optical Fit	Graphics	Comments
Yikes	350+ /100	1024	33	CD-ROM	Rage 128 PCI	Limited options; Best avoided
Sawtooth	400+ /100	1536	66	DVD	Rage 128 AGP	Excellent basic model Adds Airport support; Adds UDMA-66
Mystic	450+ /100	1536	66	DVD-R	Rage 128 Pro AGP	Adds dual-processors; Adds Gigabit Ethernet; Adds Radeon option
Digital Audio	466+ /133	1536	66	CD-RW, DVD-R	Rage 128 Pro AGP	Adds 4th PCI slot; Adds GeForce 2MX
Quick silver	733+ /133	1536	66	CD-RW, DVD-R	GeForce 2MX or 3	Revised cpu variants
Quick silver 2002	800+ /133	1536	66	CD-RW, DVD-R	Radeon or GeForce 4	Revised cpu variants

3.5 Case 4: Unsupported Options

3.5.1 Processor upgrades.

With very few exceptions, Apple's policy has never been to support cpu upgrades in its computers. Apple expects you to run it exactly as you bought it until one of you dies.

With very few exceptions, the policy of the Mac hacking community has been to hack Mac hardware to within an inch of its life in order to stretch the bounds of what is possible, and to extend the life of machinery beyond natural reason.

The result is that numerous dedicated Mac hackers have devised ways over the years to run Mac operating systems on unsupported machines. Special credit is due to those who have run Mac OS 8 on an SE/30, and converted the Color Classic to be a Power Mac.

There are three obvious ways to create an unsupported OS X machine:

1 **Simple cpu upgrade**: Take a supported system, but change the cpu for a newer or faster processor.
2 **Unsupported system**: Take an unsupported system, and install OS X on it as is.
3 **Unsupported system with cpu upgrade**: Take an unsupported system, and change its cpu to a G3 or G4, before installing OS X in the new configuration.

3.5.2 CPU upgrade in Supported System

Several companies provide cpu upgrades to permit this. Some have an unreliable commercial history. Upgrades are available for systems based on the Gossamer, Yosemite, Sawtooth, and many later motherboards, Upgrades are also available for various models of iMac and PowerBook. There can be issues caused by incompatibilities between G3 and G4 cpus, and issues of incompatibility with system ROMs or other hardware. When they work, the results can be spectacular, but only you can decide whether or not you are prepared to spend the money and risk creating an unsupported machine this way.

The fullest collection of data on these options is at [reference 4]. Study before you buy.

3.5.3 Original Processor in Unsupported System

To do this you need to use Ryan Rempel's freeware utility called "XPostFacto". At time of writing, Ryan is maintaining active development of XPF. Rather than reproduce all his data, it is best to refer you to his web-site [reference 2], which has up-to-date information. Unsupported installs are available for systems based on the TNT, Nitro and Tsunami motherboards. Ryan has had limited success with other consumer Macs and with some older PowerBooks, but at time of writing I would not recommend these for serious work.

Except for the original Power Mac 7500, these systems all shipped with versions of the 604 cpu. Hence, at time of writing, you can only use Puma, not Jaguar, on these systems.

3.5.4 New Processor in Unsupported System

To do this you must again use XPostFacto, but you must also deal with added complications:

a **Enabling the Level-2 cache**: The original level-2 cache fitted to these boards was a special DIMM, fitted in a special slot, connected direct to the system's main bus. Any newer cpu (G3, G4 or newer) will have a backside or processor-direct cache. Mac OS X needs to be told about the presence and configuration of this cache. There are freeware utilities to do this.

b **G3/G4 speculative execution**: The G3 and G4 have more aggressive policies for speculative execution of code than did the 603 and 604. To ensure successful operation in a PowerSurge motherboard, the NVRAM of the board needs to be patched to match. There are utilities to do this.

c **Fallback if your cpu fails**: Since you configure both the board and the OS X kernel to match the fitted processor, it is important to decide what you will do if the cpu card fails. You do not want to be left with a failed system whose data you can no longer retrieve because your original cpu will not work in the current configuration. A simple policy is to ensure that your system is dual-boot, so that you can revert to Mac OS 9 for emergency repairs.

Despite these complications, with careful choice of parts you can create an unsupported system able to run Mac OS X at speeds close to a recent G4 system. Likewise, you can create a system able to do things an iMac can't do, such as route between multiple physical Ethernets. Only you can decide whether this is worth time, effort and money.

4 Worked Examples

4.1 Dual-booting, Partitioning, and the Reasons Why

For budget users of Mac OS X, I recommend that you install OS X and Mac OS 9 on separate partitions (or separate drives if you have them). This simplifies the set-up of dual-booting arrangements. Unless you are supremely confident that you will never want to boot your system in the traditional Mac OS, set it up as dual boot, so that you have access to an alternative environment for reconfiguration, repair and recovery if need be. I shall assume these partitions are called "System Disk 9" and "System Disk X".

Here are two examples of budget Mac OS X in everyday use.

4.2 Supported Example – budget Portable running Jaguar

For business presentations, I use a Pismo PowerBook, known as "ratmobile".

The later G3 PowerBooks and iceBooks make excellent Mac OS X portables. Many users favour the Pismo, because of its combination of features:

- USB and FireWire (though not the fastest FireWire in the world);
- Built-in Fast Ethernet and Airport;
- Hot-swap Expansion bays for storage devices;
- Relatively easy access to internal hard drive and RAM.

In the UK, a used Pismo costs about €1,350. Ratmobile arrived with 192 MB RAM and an internal Hard Disk Drive of 6 GB. For first experiments, I partitioned this drive into 4 GB for System Disk X and 2 GB for System Disk 9. Once I had checked ratmobile's behaviour under Puma, I bought and installed a larger HDD of 20 GB, now partitioned as 8 GB for System Disk X, 8 GB for Work Space and the rest for System Disk 9. Ratmobile is now running OS X 10.2 (Jaguar). My first attempt was to update the already installed OS X 10.1.5. While the result worked, it seemed temperamental. Instead, I tried the alternative Jaguar option of "archive and install", which captures the user directories but re-installs the OS from scratch. From experience, I would recommend this approach.

Any similar G3 PowerBook, with not less than 128 MB RAM, and not less than 4 GB Hard Disk, should run either Puma or Jaguar satisfactorily.

4.3 Unsupported Example - Dumpster Desktop running Puma

At the other budget extreme, meet Escargot, a rock-bottom system running at warp factor zero. In the UK, the parts for a system like this can be bought used for around €200.

4.3.1 Escargot's Hardware

Escargot is an experiment in how slow a Mac OS X system can get and still not be technically dead: Escargot is a Power Macintosh 7500 (TNT) motherboard, in its standard desktop case, but fitted with a 150 MHz 604 processor card taken from another system. Escargot has an unexciting configuration for its type: 64 MB RAM, 256K Level-2 cache, 2 MB VideoRAM, and two internal 50-pin SCSI drives. The smaller disk of 1.2 GB is split into 1 GB for System Disk 9 running 9.1, and the rest as spare. The other disk of 2 GB is devoted to OS X, running 10.1.5. As can be seen from demonstration, Escargot works perfectly well – albeit slowly. I could make Escargot slightly slower by substituting a 120 MHz 604 (if I had one), reducing the RAM, or removing the L2 cache – but the machine is suffering enough already.

Escargot has three optional additions in its three PCI slots. One is a Belkin USB card (sold for PCs). One is a similar Belkin FireWire card (also sold for PCs). The last is a DEC DE500 fast Ethernet card, originally fitted to a Digital PC. In each case, when running under OS X, the card just works.

4.3.2 Escargot's System Software

At time of writing, there is no way to run Jaguar on the 603 or 604 processors. The best you can do is 10.1.5, and it is important to reach that patch-level. The reason is security. Apple has released a variety of security patches to Puma during the last year, many dealing with issues detected in Open Source components like Apache and OpenSSH, so it is desirable to patch to the maximum. (This is starting to remind me of Solaris...)

Table 7: Full Install and Patch Sequence for Puma

Step	Aim	Needs	Details
1	Format HDD	Apple's Drive Setup Utility	Boot from System Disk 9; Reformat System Disk X as "Mac OS Extended" format.
1	Install 10.0	OS X 10.0 Install CD; XPostFacto	Insert CD. Start XPF. Using XPF, reboot from CD to install on to System Disk X.
2	Configure 10.0	Personal Details	When machine restarts automatically, key in personal details and internet settings.
3	Install 10.1	OS X 10.1 Upgrade CD; XPostFacto	Reboot from System Disk 9. Insert CD. Start XPF. Using XPF, reboot from CD to UPGRADE the installation on System Disk X.
4	Update "Software Update"	SecUpd7-18-02forv10.1 .dmg	After restart, install this package. This patch corrects security issues in the Software Update utility. It supersedes earlier patches from October and November 2001.
5	Update to OS X 10.1.5	MacOSXUpdateCombo10.1.5 .dmg	Install this package. This patch supersedes all earlier point releases of 10.1.x, and incorporates Security patch from April 2002.
6	Patch Open Source (1)	SecurityUpd2002-08-02 .dmg	Install this package. This patch also incorporates general security patch from July 2002.
7	Patch Open Source (2)	SecurityUpd2002-08-20 .dmg	Install this package.
8	Patch Networking	NetworkingUpdate .dmg	Install this package. This patch improves network reliability
9	Patch IE	Internet Explorer v5.1.4	There may be a more recent version covering additional security issues !

In addition, Apple detected some security issues with the Apple Software Update process itself, so it has also released certain critical updates which you must apply before you can pull in the rest of your patches. This means that the required patch order for Puma has changed substantially during its life. The current version of Software Update is generally well-behaved, and will usually get it right if you follow its instructions to the letter.

The exact install instructions depend on where you're starting from. If you have a late-release version of the Puma Installation disks, then you are already part-patched, and so need to follow the correct patch sequence from that point. I can't describe here all possible variations. Instead I have shown in Table 7 the full patch sequence for building Escargot from scratch. I assume use of the lowest possible sources, being the two "early-adopter" install CDs: the original release of 10.0, and the original update CD for 10.1.

If you simply follow Apple's Software Update mechanism, it should get the sequence correct for you automatically. It will also combine some of these steps, to reduce the time taken. If you are unable to use Software Update for some reason (perhaps because you work from behind a strict FireWall) you can apply the patches in the correct sequence by downloading them as Disk Image (.dmg) files from Apple's support web-site, and installing them individually.

4.3.3 Escargot's Applications Software

Once you have completed the full patch sequence, you have a Mac OS X 10.1 system which is as patched as it can get for operating system, networking, and security issues. The system will currently occupy 1.4 GB of hard drive. What else you choose to install is up to you, and to how much disk space you wish to use. Apple's own additions are available from Apple's web-site, and should be installable using Software Update. Other additions are available from third-parties. Here are some suggestions:

1 Apple's language sets for Chinese, Korean, Portuguese, or Scandinavian languages;
2 Apple's Development Tools for OS X 10.1 (minus documentation to save space?);
3 The Darwin tools, available as a package from web-sites which support Darwin;
4 Xfreee86, distributed as "XonX", together with X applications;
5 The OS X Package Manager, giving control of the configuration of OS X packages;
6 The Fink Package Manager, giving access to UNIX software appropriately packaged;
7 Apple's Java update, to give improved Java 2 compatibility;
8 Your favourite Java tools or applications;
9 Additional development languages, such as Python and PHP;
10 Additional Carbon applications, or OS X native applications to suit.

As a demonstration machine, Escargot shows that all these things are possible, even on a system as humble as this one. If you try a budget system like this it may inspire you to find the funds for a real Mac OS X system, or at least to buy more RAM, more disk space, and perhaps a faster cpu card to give the budget system better performance

I emphasise that a system like Escargot can be built exactly as it is. No cheating is necessary. You do not need to configure the system with faster components or more RAM just in order to build it. It is perfectly feasible to build a machine running OS X using only a basic TNT system with a 604 cpu, 64 MB RAM, and a 2 GB HDD. Obviously you will need a CDROM you can boot from, and the means to load all the patches from Internet, CD or LAN, but otherwise it's straightforward. It is a testament to the robustness of OS X that it can be installed on this limited hardware without kernel panics or any other problems. (Just take three days off, have plenty of coffee and pizza handy and put something soothing on the hifi.)

4.3.4 Alternative OS approaches

A system like Escargot is equally suited – perhaps more so – to running another form of Open Source UNIX such as OpenBSD or YellowDog Linux. If you are not happy with OS X on such a system, try one of these alternatives.

With few exceptions, the Power Macs which can run Mac OS X are the same Macs which can run other flavours of BSD or run Linux. All need the same essentials: recognized processor, PCI architecture, Open Firmware, and the means to install the distribution. The one difference is in cpu support. The Linux kernel includes support for the PowerPC 601 processor, whereas at time of writing no version of BSD can support this processor.

However: there is good exchange of information between the BSDs and the Linux community concerning support for older Mac hardware. This helps to boost peripheral support for Mac OS X. For example, Linux has provided the basic floppy driver for Mac OS X (although – so far – it doesn't work for me !).

5 Summary and Conclusions

This paper is intended to help those who wish to run Mac OS X but have limited funds.

By building Mac OS X on an Open Source base, Apple has made it possible to install OS X in unsupported ways.

If your livelihood depends on using a supported computer, you should run Mac OS X on a budget by buying a new budget Mac from your nearest reseller, and enjoy all the benefits of warranty and manufacturer's support.

If you wish to use OS X in a more experimental capacity, there are many ways to run Mac OS X on older Mac hardware, and there is much to be learned from doing so.

In all cases, learn the hardware specs. Watch out for limitations with RAM, graphics support, disk speeds, and external interfaces.

To speed up Mac OS X, first install more RAM. Then add still more...

For portables, recent G3 PowerBooks and iceBooks give good results.

For basic desktops, most early iMacs and G3 desktop machines give adequate results.

For advanced desktops, G4 minitowers are best. Check specifications with care.

For unsupported installs, PowerSurge machines (TNT, Nitro, Tsunami) work reliably.

In general, machines which can run Mac OS X are the same as those which can run BSD or Linux. Use those projects for information to resolve driver and hardware issues.

6 References

1 In the US District Court for the District of Columbia,
Civil Action No. 98-1233 (CKK),
Direct Testimony of Bill Gates, given under oath 18 April 2002.

2 http://eshop.macsales.com/OSXCenter/XPostFacto/

3 http://www.lowendmac.com/

4 http://www.xlr8yourmac.com/

5 An Evaluation of PowerMac G4 Systems for FORTRAN-based Scientific Computing with Application to Computational Fluid Dynamics Simulation;
Craig A Hunter, NASA Langley Research Center, June 2000.

6 http://www.netbsd.org/

7 http://www.openbsd.org/

[This document was written in Microsoft Word, running native on Mac OS X]

Addresses

Le Reseau Netwerksystemen B.V.
Eilko Bos
Bieslookstraat 31a, 9731 HH Groningen, The Netherlands
Phone: +31 (0)505492701
Email: <eilko@reseau.nl>

sysfive.com GmbH / BS Web Services
Philipp Bühler & Henning Brauer
Koelhoffstr. 7, 50676 Köln, Germany
Phone: +49 2214742105
Email: <pb@sysfive.com>

X|support
Pim Buurman
Grote Beer 189, 1188 AZ Amstelveen, The Netherlands
Phone: +31 (0)237505895
Email: <pim.buurman@summix.nl>

Wasabi Systems
Alistair Crooks
17 The Conifers, Crowthorne, Berkshire, RG45 6TG London, United Kingdom
Phone: +44 1344 752021
Email: <agc@wasabisystems.com>

The NetBSD Project
Hubert Feyrer
Rotteneckstr. 31, 93053 Regensburg, Germany
Phone: +49 941 943 1333
Email: <hubertf@netbsd.org>

Inktomi Corp.
Alan Horn
1025 Shell Blvd 1, Foster City, CA, USA
Phone: +1 650 245 9351
Email: <ahorn@inktomi.com>

The FreeBSD Project
Poul-Henning Kamp
Herluf Trollesvej 3, DK-4200 Slagelse, Denmark
Phone: +45 5856 1059
Email: <phk@FreeBSD.org>

Cisco Systems
Marco Molteni
400, avenue Roumanille, 06410 Sophia Antipolis, France
Phone: +33 619 982466
Email: <mmolteni@cisco.com>

Stichting NLnet Labs
Bram Moolenaar
c/o Clematisstraat 30, 5925 BE Venlo, The Netherlands
Email: <bram@moolenaar.net>

Politecnico di Torino
Riccardo Scandariato
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Phone: +39 011 564 7048
Email: <scandariato@polito.it>

Bonn University, CS Department, Chair V
Ignatios Souvatzis
Roemerstrasse 164, 53177 Bonn, Germany
Phone: +49 228 73 4316
Email: <ignatios@cs.uni-bonn.de>

Open Source Telecom
David Sugar
218 Louis Ave, 08872 Somerset, USA
Phone: +1 732 302 1554
Email: <dyfet@ostel.com>

Valeriy Ushakov
Email: <uwe@netbsd.org>

Gerald Wilson
Email: <gww@stonehill.org.uk>