BSD Unix Solutions in the Australian NFP/NGO Health Sector

Jason Tubnor – ICT Senior Security Lead

Latrobe Community Health Service, Victoria, Australia
jason.tubnor@lchs.com.au, jason@tubnor.net

Abstract

Latrobe Community Health Service (LCHS) is a Not for Profit (NFP)/Non-Government Organisation (NGO) in Victoria, Australia. By 2018, the organisation had grown to 51 offices across the State of Victoria with over 1,000 employees. All LCHS infrastructure is designed and managed in-house without the use of large-scale cloud infrastructure.

Since 2015, BSD Unix has been used for various workloads within the organisation, with application instances interchanging between BSD distributions where it was deemed that one type was stronger than another at specific roles in the LCHS environment. LCHS is operating system distribution and technology-agnostic and prefer both FreeBSD and OpenBSD where either one and/or its associated base tools are most suitable.

This paper outlines the various design considerations and configurations of OpenBSD and FreeBSD in multiple roles in our organisation.

1 Introduction

In 2015, Latrobe Community Health Service (LCHS) started to look towards open source to solve problems that couldn't be easily solved with available Microsoft technology or at a reasonable price point. The Information, Communication Technology (ICT) team were tasked to think outside the box and come up with solutions to these problems.

This led to some technology decisions that when reflected upon, were good for the company's long-term goals.

While there are more features of the different BSDs in use at LCHS, this paper will focus on the high level of the following technology implementation:

- OpenBSD PF(4) and traffic queueing.
- OpenBSD ripd(4) dynamic routing, network design considerations and scaling it across sites with multi-vendor implementations of RIPv2.
- Design and deployment of a commodity, 1U appliance using FreeBSD, OpenZFS and bhyve(8) as a host.
- A virtualized OpenBSD guest to provide branch offices network infrastructure and access to head office corporate services, using simple and cheaply available layer 3 Internet services.
- Migrating the marketing USB archive drive to a 36TB raw storage FreeBSD OpenZFS server, replicating data sets across 3 sites.
- Using tools such as setfib(1) in FreeBSD or rdomain/rtable(4) in OpenBSD to connect a host to a separate storage network.

2 OpenBSD ripd(8) and pf(4)

A strategic change of direction for the organisation occurred in 2015 and from an ICT perspective, it meant a move away from being solely a Windows shop. Network management was being brought back in-house from an outsourcing provider and a local Internet Service Provider was engaged to provide the organisation a private, layer 3network between sites using whatever technology was available to provide the bandwidth required. Initially this included eight sites that needed to be interconnected with around 10/10Mb symmetrical connections in a hub and spoke configuration. Due to the size of the network and the unknown state of the topology because of outsourced management, original agreements stated that routing would be managed by the ISP using static routing that was then distributed in their core network via BGP.

Once it became apparent that this was not going to be scalable if the organisation ever grew, a plan was required to allow for management and distribution of routes across the entire network. Due to the configuration of our ISPs network, BGP was ruled out. The organisation was also swapping out aged Cisco Catalyst switches for Brocade ICX switches, limiting the use of other protocols that could be used for route distribution.

The network design contained an OpenBSD firewall for the Internet gateway. Using the excellent documentation provided by the project located in the base install, assisted in making the most appropriate decision. As most are aware, OpenBGPD is a major sub-project within the OpenBSD base system but the BGP it offered was not useable due to what was mentioned above. However, another routing protocol that was part of the project at the time and still being maintained is ripd(8). While this protocol is old and there are many more modern and preferable routing protocols, this would be enough for the organisation based on the new network topology and being interoperable between all software and hardware that will make up the network refresh.

As the connection and configuration of the network took place, work went into the design and build of the OpenBSD Internet gateway. Key components of the gateway simply consisted of network components that were found in the base operating system, such as ripd(8) using version 2 only of the protocol and pf(4). Initially, the organisation size only dictated the configuration of the pf(4) rules to be block all/explicitly permit and keep the traffic as first in, first out. As the organisations requirement for Internet connectivity grew and a limited budget prevented an increase in bandwidth, creativity was required to design and build queues to manage end users' consumption, ensuring all users had a good experience.

2.1 ripd(8)

ripd(8) configuration in OpenBSD is very simple and only requires the daemon to be enabled (`rcctl enable ripd`) after the modification of the excellent example ripd.conf(5) that is located in `/etc/examples/` and copied to `/etc`.

A simple example of ripd.conf(5) when running on a firewall/route of last resort without any authentication is (bge1 being the external interface):

```
fib-update yes
redistribute default
split-horizon poisoned
triggered-updates yes

interface bge0 {
}

interface bge1 {
        passive
}
```

The ripd(8) routing protocol implementation was successful, allowing the firewall to participate in announcements and broadcasts. During implementation, use of the ripctl(8) tool allowed for debugging and rectification of the ripd(8) configuration. It provided the ability to determine if neighbours are connected and active as well as what routes the daemon was receiving.

Stability of ripd(8) was an issue in OpenBSD 5.8 with the ripd(8) daemon consuming RAM over time and not releasing it even though the network wasn't growing in size. This would cause the host to remain up but the RIPv2 routing daemon would crash, losing routes to the rest of the network. A restart of the ripd(8) daemon would be required to have the host operational again and it was hard to pinpoint what was the problem initially. More recent versions of OpenBSD appear to have corrected these issues.

One of the newer features that was tested but not yet used in production is the ability to launch multiple ripd(8) instances to run in other rdomain(4)s without one knowing about the other. This is good for where an OpenBSD router is multi-tenanted with isolated network traffic.

One feature request for ripd(8) would be to have the ability to reload the configuration file or perform other operations without having to restart the daemon (similar to iked(8) using ikectl(8)) to avoid causing an interruption to the routed traffic.

2.2 pf(4) and traffic queueing

Out of the box, pf(4) on OpenBSD provides an excellent firewall solution for the front of a private network. A clean room approach was taken in the design of the rules that were required for internal applications to work.

By using a default block all approach in the rule set, a couple of applications that were affected by missing rules failed, however, these were fixed, and appropriate documentation was created for these applications as well as their networking requirements.

Over the next 6 months, network issues started to appear with DNS timeouts and other applications failing that were latency sensitive. An investigation uncovered that the internal userbase were consuming all the inbound bandwidth of the public Internet connection. Bandwidth is expensive in Australia and it is not as simple as turning a knob to increase bandwidth.

The pf.conf(5) rule set was reviewed to break down the different protocols and sort into time sensitive, low bandwidth to low priority, high bandwidth.

What is misunderstood is that you can control inbound traffic coming into your network, even if access to the upstream device is unavailable – typically only outbound traffic can be controlled. By setting outbound queues on the internal interface, it effectively rate limits the traffic heading to end-users. In turn, this holds back the ACK for TCP traffic or the data stream for UDP.

Both pf(4) and its queueing algorithm work exceptionally well, with only minor rule changes when a new application comes along or in the case of queueing, when bandwidth is eventually increased.

Since 2018, the organisation increased the office count that used this firewall from 8 to 24 and bandwidth to 100/100Mb symmetrical with a significant increase of devices utilising this firewall as their primary Internet gateway. While different protocols NAT to individual IP addresses, the default state limits in pf(4) became an issue. The parameter *set limit states* was raised from the default 10,000 to 30,000 without any notable impact to RAM. However, since OpenBSD 6.4, the default state limit has been increased to 100,000 states[1].

On average, an observed 25MB of RAM is consumed with 12% of CPU utilisation on a single core for pf(4) under peak user load.

[1] henning@ increased the default state table size from 10,000 to 100,000 https://cvsweb.openbsd.org/src/sys/net/pfvar.h?rev=1.480&content-type=text/x-cvsweb-markup

3 FreeBSD bhyve(8) appliance

In 2017 the organisation was awarded a second round of contracts to provide services for the National Disability Insurance Scheme (NDIS), an Australian Government initiative.

The ICT team were tasked to come up with a solution that would provide network and site server facilities to satellite offices at a cost-effective rate. After a significant amount of research to come up with suitable hardware that could withstand hostile environments (high temperatures/dust), the decision was made to use FreeBSD 11.0 with bhyve(8) and OpenZFS to build a custom hypervisor instead of using VMWare's ESXi server, which was also trialled during the proof of concept stage (PoC).

In the first design of these hosts, a reliable management tool that was easily used by team members still didn't exist and bhyve/UEFI was still relatively new, making the project look rather un-realistic from the start.

The *chyves* bhyve management project was evaluated and achieved 90% of requirements and that was enough for the V1.0 of the device, further iterations of the device could occur if better tools came along.

Installing *chyves* and modifying the management shell scripts to work for more modern version of OpenBSD was successful and a solid and reliable platform was ready for use.

In V2.0, the appliance moved to FreeBSD 11.1, *vm-bhyve* replaced *chyves* and an all UEFI guest configuration. This approach simplified the number of datasets and the knowledge required to manage guests at the console level.

The V2.0 appliance then had acceptance of the team, some team members were able to perform zvol expansion for guests where required and others could manage guest operating systems with ease via a VNC console.

3.1 Storage and replication

OpenZFS allows for easy upgrades of guest operating environments by providing simple snapshots. The Copy on Write (COW) file system provided by OpenZFS is superb in this scenario, if guests don't get shut down prior to snapshots, rollbacks can occur to a previous running state without an issue.

Additional tools that were bundled into the build to automate backups, snapshots and transfers were *zfsnap2* and *zxfer*. While simple in their nature, they work well to ensure there is a consistent copy of each guest that can be rolled out to new devices in the event of a hardware failure of the production device. These backups are replicated to build servers around the state with hot spares of the appliance ready to go so they can be remotely re-imaged by the administration team and deployed by onsite ICT technicians.

Having guests imaged in the way mentioned above allows deployment of new or replacement devices within 15 minutes after power-on. Most of the delay is due to network speed for replication.

3.2 Guest network security

To mitigate the risk of guests seeing data that they are not entitled to on the wire, all VLANs were configured at the host level to the physical interface that either plugged into the switch or Internet device. Each guest then had the required interfaces presented to it and bridged to the applicable VLAN at the point of presentation. This avoided the need for configuration of VLANs within the guest and reduced the complexity for support staff.

4 OpenBSD VPN gateway

As part of the hypervisor appliance mentioned in section 3 of this document, a key feature needed was a feature rich router and firewall. OpenBSD was already providing a solid foundation to work off and was chosen for the task.

In addition to the firewall and routing capabilities of OpenBSD, the operating system also provided OpenIKED (IKEv2) in the base system, providing a simplistic and secure VPN tool to connect the remote offices up with the rest of the network, using a cheap and cost-effective public Internet link (figure 1).

Over the previous 18 months, different design concepts and configurations of OpenIKED were worked on to connect two medical centres to the core network, this helped iron out the final design for this virtual router.

Firstly, an OpenBSD VPN terminator was commissioned in the DMZ that end sites could connect to, forming part of the core network infrastructure. Setting this into a passive state and using key based authentication with VPN end points kept the configuration simple. Remote VPN hosts would be active clients and perform the connection to the main termination device.

The rest of the configuration was the same for both the main terminator and the clients. Each site was designed to have at least 5 VLANs for various tasks. The traffic then needed to be encapsulated, encrypted and signed for transport. Once end-to-end encryption was set up, a unicast virtual network using two virtual adaptors across the encapsulated link (using vxlan(4)) could be created that then could then have ripd(8) overlayed, managing the routes seen by either end.

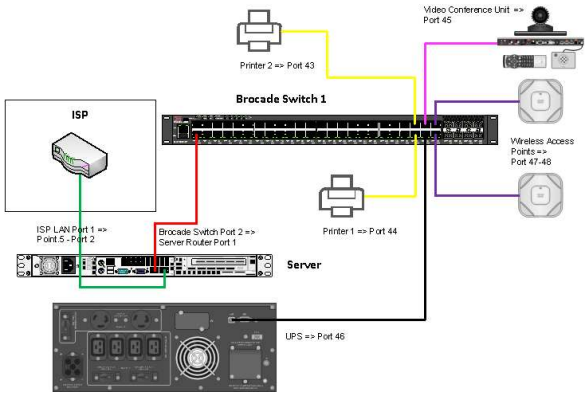

Figure 1

4.1 ipcomp(4)

The initial designed used the IP Payload Compression Protocol (ipcomp(4)) for all links to keep consistency as some remote sites have poorly performing ADSL connections providing as little as 6,000/768Kbs asymmetrical throughput. While this provided marginal improvements for these ADSL sites, more and more payload traffic are using encryption and/or transparent compression techniques and has made this no longer a benefit.

Before unwinding the use of ipcomp(4), kernel crashes were being observed in the main terminator that runs in the VMWare ESXi cluster when a significant and consistent traffic load had been passing through it. Further investigations found that this was an ESXi issue as it could not be replicated with bare metal hardware.

Another regression was also discovered between OpenBSD 6.3 and 6.4 that inadvertently forced the removal of ipcomp(4) from all links so devices could be upgraded to 6.4.

4.2 vxlan(4)

Configuring iked(8) to understand and handle traffic for a significant number of VLANs across all networks was not an option. A solution was required that would allow for the use of ripd(8) across the encrypted encapsulated link. OpenBSD comes with various encapsulation or tunnelling protocols such as gre(4), gif(4), etherip(4), however, vxlan(4) was chosen as it was a modern option and had scope for future expansion. As all traffic between the iked(8) end-points was encrypted, the vxlan(4) interface could be configured as a /30 subnet at either end of the tunnel to form a unicast network suitable for ripd(8).

By configuring ripd(8) on the vxlan(8) interface, both neighbours could see each other and pass their applicable routing tables.

4.3 pf(4) queues

While ipcomp(4) was deprecated, there was still a need to manage traffic moving between the remote and main networks.

All traffic that passes over an iked(8) connection moves through the enc(4) interface at either end. This interface does not support queue-based bandwidth control so traffic shaping cannot take place. Using pf(4) tagging allows the control of traffic flow between both end points but not the various traffic that uses this 'bridge'.

Since vxlan(4) was being used for ripd(8) and to route traffic between networks, pf(4) queuing was enabled on this interface at either end to ensure quality of service (QoS) was maintained for time sensitive, low latency packets. During the development and testing phase, it was found that no distortion occurred to Cisco Call Manager audio packets while the data links were under heavy congestion.

Two years on, queue and rule configurations produced in the initial rollout are still providing end-users with an excellent audio or video experience when they are either using handsets for regular voice calls or video conference units.

5 FreeBSD OpenZFS Storage Cluster

With the organisation growing rapidly over the last three years, ICT have had to constrain the type of data that utilised space on the flash storage array. Expanding it in the short term was not a viable option due to cost.

This saw some users within the organisation looking for options to store, low changing, static media that was significant in size. Marketing fell into this category due to the type of assets they worked on but there just wasn't the space for this type of media. When it was discovered that they were simply using a portable hard drive to store these assets on, it forced action to provide a facility for storage as well as the appropriate protection for this sort of data.

After discussions with vendors regarding the requirements and financial constraints, a hardware build was designed that was fully supported by FreeBSD 11.1 and its implementation of OpenZFS.

The design was simple, and it didn't need to be included it into the backup cluster if there was enough redundancy and protection built into the solution. By utilising the DR storage WAN links and other network links, a 3-node cluster was designed and built across three sites (figure 2).

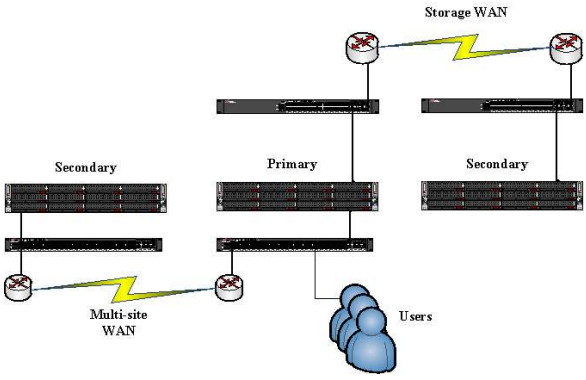

Figure 2

5.1 Storage

Deliberation occurred to determine if FreeNAS or vanilla FreeBSD 11.1 should be used for the project. As it was unknown how this storage would be utilised outside of the immediate scope, it was decided that FreeBSD would give greater flexibility moving forward as well as being able to customise it to meet the network and replication requirements.

The build consisted of Lenovo 2RU storage 3.5-inch storage chassis, 64GB RAM, a single Xeon CPU, 2 x 480GB high endurance SSD, 6 x 6TB HDD and a 2x10Gb Intel SFP+ network card.

The main storage was configured in a RAIDZ2 pool with 32GB of mirrored SLOG space dedicated from the 480GB SSDs. Only 200GB from the 480GB SSDs was given to the hosts operating system to allow for wear caused by the SLOG throughout the hosts life.

The ZFS ARC was limited to 48GB to allow for a minimal toolset to work correctly on the host and not put pressure on the available memory or swap if RAM was suddenly required.

The pool was carved up into many datasets depending on the work load and replication requirements. Initially this was minimal in the beginning, but it is now becoming more complex though is still quite easy to manage.

5.2 Replication

As the datasets are needed in the 3 locations without any special operations, *zxfer* was sufficient in managing the transfer of datasets and their associated zfs(8) properties. Data transfer was performed over *OpenSSH-portable* from the ports tree without any specific configuration except for key based authentication. While there are options and patches for tuning *OpenSSH* for large workloads, there were no perceived benefits in doing so for this use case.

At times, users can dump in excess of 100GB of files in a session. This does cause issues for us on the slower MPLS link (Multi-site WAN in figure 2) and cause a backlog of expired datasets that require removing. While not the most optimal solution, sneaker net is currently the best way to get the large dataset that is causing issues to the third site. Mounting a zpool(8) that is located on a portable hard disk, the dataset that needs to be replicated is copied to it using *zfs send/recv* and taken to the remote server by an ICT technician. The dataset in question is then copied to the third server, allowing for standard replication to resume as well as dataset clean up.

5.3 Services

Initially, only *smb* connectivity was provided for end users and *smb* used authentication from our main Active Directory (AD) infrastructure.

As time progressed, the internal mirror of application files was growing rapidly so this saw a jail serving out a mirror dataset via *nginx* be implemented. This worked so well that since it also contained the ISO files that were used for guest builds on our VMWare ESXi infrastructure, a read-only NFS share of the same dataset was created and presented as a data store to ESXi.

Eventually further storage was added to the main production environment. Scratch space was needed to move data around and to re-structure it. These storage servers were able to provide temporary iSCSI targets with zvols for storage presentation to the ESXi hosts making it easy to give redundant storage for *vmdk* moves.

6 Multiple routing tables

Without multiple routing tables, the replication discussed in section 5.2 could not occur over the DR storage WAN connection as the two hosts did not share the same layer 2 (L2) network segment. This is where the features of OpenBSD rdomain/rtable(4) and FreeBSD setfib(1) become essential in managing two isolated networks on the same host.

More than one routing table allows a system to host multiple applications on the same host that the attached networks or the applications are not aware of the other. For example, a host could run two vastly different web server instances for two different groups on the same host and neither group would know about the other instance except for the system administrator. Both groups would come in via two different network paths and their traffic would never cross.

This is what was used to connect hosts to the IP storage network that needed to see storage across multiple subnets (figure 2).

Another aspect is that it allows for more than one default router or route of last resort on a host so depending on the use case, an application can use the most optimal path.

Using this type of feature in either operating system is quite simplistic. In FreeBSD, invoking setfib <*table number*> <*command*> will run a command, launch a daemon or even adjust the applicable routing on the table defined in <table number>. OpenBSD has the feature built into route but the network stack can be modified independently through features in pf(4) and ifconfig(8). OpenBSD ping(8) and ps(1) are also both rdomain aware programs.

Out of the box, no additional configuration is required in OpenBSD, however FreeBSD requires the */boot/loader.conf* to contain *net.fibs="N"* with *N* being the number of tables your host needs.

7 Conclusion

A mix of BSD Unix variations have proven a valuable and successful asset to the LCHS organisation. The tools and technologies that are developed and maintained by FreeBSD and OpenBSD are on par with, or better than their commercial counterparts. Overall, adopting BSD has ensured that the technology managed within the organisation assists in providing better delivery of services for our clients.