High Availability support for the design of stateful networking equipments

Pablo Neira Ayuso, Laurent Lefévre, Rafael Martínez Gasca

{pneira|gasca}@lsi.us.es
QUIVIR Research Group - Department of Languages and Systems
ETS Ingeniería Informatica - Avda. Reina Mercedes, s/n - 41012 SEVILLE - Spain

Laurent.Lefevre@inria.fr
INRIA RESO - LIP Laboratory (UMR CNRS, INRIA, ENS, UCB)
Ecole Normale Supérieure de Lyon - 46 allée d'Italie - 69364 LYON Cedex 07 - France

Abstract

*The availability of some critical equipments like gateways, firewalls and proxies must be guaranteed in operational networks. In early equipments, the routing and filtering decisions were maded based on the packet information, nowadays this static approach is not longer safe. Existing High Availability (HA) solutions do not cover all the aspects to ensure availability of advanced settings that are being deployed these days. Some important issues like the reduction of unavailability time and the need for failure detection in such scenarios must be studied. This paper describes the implementation of high available stateful network equipments: these systems apply policies based on the state of the connections, such information is gathered in runtime by means of packet inspection. This work specifically focus on Linux systems and firewalls because the IT industry is trusting more and more OpenSource solutions to deploy critical services because of its quality and the access to the source code. We propose the SNE library (*Stateful Network Equipment*), which is an add-on to current HA protocols, to solve the existing limitations. In this paper, we introduce describe the proposed arquitecture and we detail a set problematic scenarios supported by our library, first experiments and evaluation.*

Keywords: *High Availability, Firewall, OpenSource, Security, Stateful*

1. Introduction

Many organizations have their own servers and a private Local Area Network (LAN) which is connected to the Internet through network equipments (firewall, gateway, proxy...). This way, users can browse the web, check their emails, communicate with people from other organizations and use their internal applications in a secure way, making their life easier and keeping it increasinly difficult for any kind of intrusion.

However, there is a problem linked with this view. A Single Point Of Failure (SPOF) will inherently be introduced if a standalone network equipment is used. The availability of all the resources will depend on the availability of the SPOF. Because of that, the fraction of time the SPOF is not operational, eg. the time of unavailability, must be as low as possible. Moreover it is possible that some current network transactions could be lost. This means that a valuable amount of data is at risk.[5]

Existing solutions are based on the idea of duplicating every critical resource of our network, also known as *the primary backup approach*[4]. This way we keep a cluster of machines where one is active (Primary node) and the others (Backup nodes) are waiting for the Primary node to fail (Fig. 1).

If the Primary fails, a Backup node will take over the resources that were assigned to the Primary node as fast as possible. This way we the time of unavailability is reduced. To carry out this task one needs protocols which :

- checks the health of all nodes;

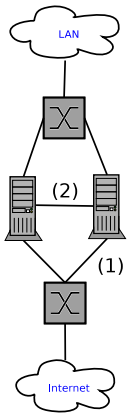

Figure 1. Primary and Backup approach

- designates as fast as possible a new Primary node if the current one has failed;
- sends as much information as possible about the current status of the primary node to the Backup nodes, this way they will be twin nodes of the active one.

On the other hand, in early firewalls handlings were made based on the packet information, eg. OSI layer 3 and 4 headers could be used in the handling decisions. The static nature of these firewalls have been exploited, resulting in Denial of Services attacks[16]. That is the reason why modern firewalls comes with so-called stateful filtering, that track and gather dinamically information about every connection[16][15]. Based on that info, the administrator can apply different policies. This results in more accurate packet handling.

The aim of this work is to describe all the possible problems and general issues related to the implementation of stateful high available network equipments, focused specifically on Linux systems[12]. Basically, this work studies the flaws of the existing HA solutions deployed in stateful equipments.

Following the *primary backup approach* previously exposed, current connections will come across problems if the Primary node fails: the Backup node becomes active but it will not have information about the state of the current network transactions. So it is likely to misbehave, resulting in the lost of connections that will need to be re-established. Such connections can be vital for critical applications.

In this paper we propose the *Stateful Network Equipment* (SNE) library. It allows network designers to easily implement and develop replicated network equipments adapted to realistic and operational scenarios. We introduce the set of scenarios supported by our library, first experiments and evaluation.

This paper is therefore organized as follows: in section 2 we quickly describe protocols and other related works about highly available network services, while section 3 will detail the framework used for our proposed solution. In section 4, we will describe the proposed architecture and section 5.2 will present the process evaluation of SNE operations. Section 6 details further works.

2. Related works

Two families of protocols, classified by vendor, have been designed and implemented to replicate network critical elements. They are :

- CISCO . VRRP (Virtual Redundancy Router Protocol[14]). This protocol was standardized by IETF. It provides a virtual IP address for a set of routers which is used by all client machines. It is deployed in the majority old CISCO routers. There are also free implementations available, however CISCO claims to have patented some aspects of this protocol. The specification is imprecise and the state machine only consists of three states[12]. On the other hand, CISCO has also developed HSRP (Hot Stand-by Redundancy Protocol[10]), a proprietary protocol implemented in the new generation of CISCO Routers.
- Open Source . CARP (Common Address Redundancy Protocol[3]). This protocol is not documented, the only reference available is the source code. It is similar to VRRP with no possible patent violations, therefore it inherits all the limitations of VRRP. Linux Heartbeat [2] is an implementation of a not described HA protocol used in *NIX systems. It is not documented either, only source code is available. Its implementation is monolithic and remains complex.

These protocols are defined to assume that at least one machine is running with all the services associated[4]. The typical scenario where these protocols are deployed is composed of two or more machines which are identical and run the same services. The protocol ensures that there is always an active machine, also known as Primary. If Primary fails it determines which idle machine, also known as Backup, will become Primary.

All the protocols have in common that one or more virtual IP addresses are always associated to the Primary, and depending on the protocol a virtual MAC or not. So clients use these virtual IP. All machines send messages to broadcast that they are alive. If Primary does

not tell others that is alive, one of the Backup machines will be selected as the new Primary.

OpenBSD release 3.5 includes a software called pfsync. It is an extension of CARP [1] [3] which replicates states in a firewall which can be linked to some aspects of this paper. Current Netfilter lead developer has also proposed a solution to implement HA firewalls in Netfilter [17].

To conclude, [12] also describes a HA protocol based on a UML state machine and a status table to decide which machine will become Primary. It claims to solve the limitations of current HA protocols and it also sligthtely defines some procedures to integrate his proposed HA protocol with firewalls.

3. Development Framework

3.1. The Netfilter Framework

In kernel branch 2.4, a framework called Netfilter[8] was introduced. It is well structured, modular, with a clean source code and well documented [13] and lets us perform several actions: inspection, mangling, filtering and routing.

Netfilter inserts five hooks (Fig. 2) into the Linux TCP/IP stack which allows you to perform packet handling at different stages.

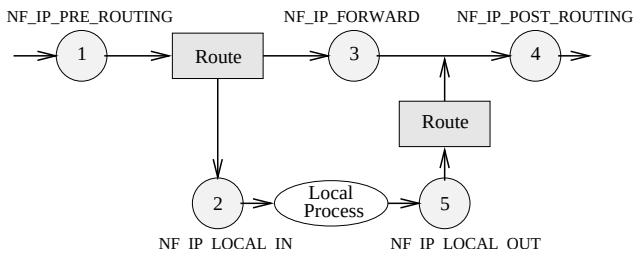

Figure 2. Netfilter Hooks

3.2. The connection tracking table system

The connection tracking system is built on top of the Netfilter Framework. It stores information related to a connection in a memory structure, all source and destination IP address and port number pairs, protocol types, state and timeouts for every connection. This is also known as stateful firewalling. It is inherently a more intelligent way to filter packets than traditional simple packet filtering.

This system allows you to perform filtering based on the state of connection in levels 3/4th of the OSI model, as well as higher layer protocols such as FTP, TFTP, IRC, PPTP. The possible states defined for a connection are NEW, ESTABLISHED, RELATED and INVALID.

The connection tracking system is a modular component that could be optionally present in the kernel, it is always required by the NAT module. Actually, it does not drop packets, except by stress and possible incoherent situations, but these cases are not considered to be the normal behaviour of the connection tracking system, that is, it always lets the packets continue to travel through the TCP/IP stack, it just tracks packets.

3.2.1. Implementation issues

Basic structure The connection tracking table is implemented with a hash table to perform efficient processing. Every position in the hash table (bucket) has a linked list to handle possible collisions. The structure *ip_conntrack* contains information about the state of a connection. Layer 3 and 4 protocol information are used to hash such structure in the table.

There is another important structure called *ip_conntrack_expect*, commonly known as *expectation*. It holds information about a connection which is expected to happen in a short period of time. There are some aspects are more difficult to track, eg. the File Transfer Protocol (FTP) in passive mode, which uses the port 21 for control issues and a port between 1024 and 65535 for data. Both connections are independent, but related. The connection tracking defines a mechanism called connection tracking helpers which let the connection tracking system identify if a connection is linked to an existing one. To do so, it defines the concept of expectation. An expectation is a connection which is expected to happen in a period of time.

This expectation has a life time. A a timeout, if it is not confirmed, it will be released. On the other hand, if it's confirmed, it will be linked to its respective *ip_conntrack* structure which contains the information about the connection and to his primary conntrack. This means that in the case of the FTP, the conntrack which represents traffic coming from and going to port 21.

Every time an *ip_conntrack* structure is defined, Netfilter tries to look for an expectation associated, if it doesn't

exist, it will try to look for a helper that matches that type of connection.

4. Proposing a Stateful Network Equipment Library

The architecture supported by our work consists of a highly available cluster of machines which can be composed of two of more machines (Fig. 1). They are connected by a switch that is considered to be the main link to the Internet(1). Another redundant link(2) is used to guarantee the communication between all the nodes, since the information to be replicated as well as the state of every node are critical. The main link could be used if this redundant link fails[4].

It is considered that there is always a primary machine in the cluster, which is active and working, and the rest of machines are considered as Backups which are ready to become active if primary fails [4]. This way we can solve possible consistency problems.

4.1. Architecture

The architecture proposed in this paper follows the microkernel operating system layout. It consists of two main parts, one in kernel space and the other one in user space. They are communicated through a messaging system. This proposed approach gives us flexibility since the main work is done in userspace, as well as the possibility of adapting our solution to the needs of a specific application like current HA protocol implementations, that live in user space.

This way, the userspace part of the Primary and Backup nodes send message each other through the network, and interact with the kernel part via Netlink message (Fig. 3).

Linux provides an interface to communicate with kernel and user process: the Netlink sockets[6], a standarized[14] and powerful[8] messaging protocol. From the programmer point of view, the API is similar to classical sockets 4), because of that it is easy to use and intuitive.

In spite of all, the Netlink Sockets protocol is unreliable. That is the reason why, we introduce the use of message sequence number to keep track of the last message received and provide retransmission mecanisms. This sequence number algorithm used is based on *Lollipop-Shaped Sequence Number Spaces* [11] which consists of a space of numbers $-k < 0 < k$, the negative numbers form the stick, and numbers from 0 to k are the circular space. Given two numbers a and b, a is more recent than b if any of the following conditions are true:

- a $<$ 0 and a $<$ b
- a $>$ 0, a $<$ b, and (b - a) $< \frac{n}{2}$
- a $>$ 0, b $>$ 0, a $>$ b, and (a - b) $> \frac{n}{2}$

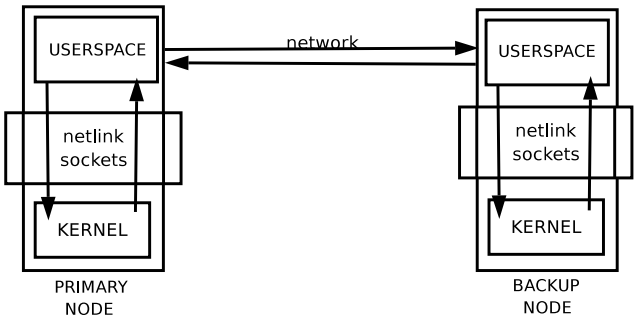

Figure 3. SNE arquitecture

The use of this sequence numbering algorithm introduces a mecanism to solve some problematic scenarios detailed in the section 4.3.

With regards to the *expectations* handling, the solution proposed will only replicate confirmed expectations, eg. those connections expected which has been finally established. So, to keep things simpler, not confirmed expectations will be kept in Primary until they get confirmation. Two reasons keep us from replicating not yet fulfilled *expectations*: they have a very short life time, normal expectations are fulfilled very soon in a normal setting. The other reason is that it would require some extra resource consumption for connections that could not ever be established.

To identify a connection in the system, an ID is asigned to the data structure that represents a connection, also known as *conntrack*. This ID together with the OSI layer 3 and 4 information uniquely identifies a connection in the systems.

As we pointed out before, this replication system is considered to be on top of a highly available system[2][9][12] which guarantees Primary's failure detection and the take over of resources.

4.2. Events

We define a set of events for our arquitecture, these events are included in the Netlink message that is send from

kernel to userspace, they are:

NEW : it contains information about a new connection that has been established. It includes OSI layer 3 and 4 information, ie. the IP address and TCP protocol ports, the state of the new connection, ie. ESTABLISHED, the status bits, eg. replied has been seen, this is a fulfilled expectation ... together other significant information like marks, timeout, ...

UPDATE : if any critical information related connection has changed, a message containing the updates information is sent to the backup machines. ie. the state of a TCP connection has passed from *SYN SENT* to *SYN RECEIVE*.

DESTROY : this event occurs if a connection is closed, it contains the ID and OSI layer 3 and 4 information.

4.3. Supported Scenarios

We support some scenarios as well as the way to solve possible problems associated with them, they are:

- Primary fails : one of the Backup nodes which has a copy of the primary's connection tracking table up to date is chosen to become primary, after the take over of resources is done, as this node keeps the state of all the connection forwarded, clients will not realise that primary fails and no active connection will be lost. To elect a Backup node candidate to become Primary, all nodes broadcast a message which has the last message ID seen. A node will discard itself if the last ID seen is negative. One of the nodes which has the greatest last ID seen will be elected as new Primary.
- Backup node S1 fails and it comes back to life again : this is the typical scenario of short-time power-cut and reboot. S1 can realise that it has just rebooted because last ID seen is negative. So, the Backup node S1 must request the whole connection tracking table to the primary. If there is more than one Backup node, to reduce the workload of the primary machine, the Backup node S1 could request the connection tracking table to another up to date Backup node S2. S1 will select the node from where it will fetch the table based on the election algorithm described in the previous scenario.
- Backup node S1 fails and remains dead : as something optional, Primary could notify the system administrator that there's a dead node. When Backup node is brought back to life, because of its last message ID seen is negative, it will request the whole connection tracking table as described in scenario B.
- An old primary node M comes back to life : if this node M is configured to become primary again, it will request the connection tracking table to the current primary node. As explained in scenario A, it could also request it to a Backup node.
- Backup nodes lost communication with Primary : commonly called as split brain. In this case, one of the Backup nodes will try to become Primary, but Primary will keep being Primary. This happens because of nodes can not know the state of the primary because they have no way of communicating with it. To solve this problem two mechanisms are used, use the main link to communicate primary with Backups instead of the dedicated link, and the knowledge of the physical status of the link which lets the machine understand the current status and, following a given policy, stay as Backup or try to take over resources.

4.4. Implementation

4.4.1. Communication Kernel/User space Since our arquitecture needs a solid method to provide an API for user space programs, such as HA protocol implementations. In this section we review all possible choices and determine which one fits our requirements.

The Netlink sockets[8] are an extension of the IP service[14] which allow messages to be exchanged with the user space. Netlink sockets provide a powerful way to notify events and a smart interface from user space. They are implemented wrapped in socket syscalls operations, to be precise they are defined as a new socket type (Fig.4). It is currently used by applications like *iproute2*, *zebra* and user space *ipsec* tools.

As a drawback, it must be considered that they are not reliable, specifically under big stress conditions. Because of that, the programmer needs to perform some kind of sequence tracking to make sure that all the packets are received. Unfortunately, there is no way to recover a possible loss of data when sending information from kernel to user space. For that reason, we have previously introduced the sequence numbering algorithm.

From user space point of view, they are similar to normal sockets. So all the specific sockets primitives, like *poll* and *select*, can be used.

There are two types of Netlink messages:

- Unicast, which lets an unique process communicate with the kernel space.
- Multicast, which lets a group of programs communicate with the kernel.

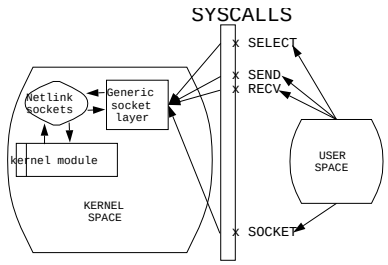

Figure 4. Netlink Sockets communications between Kernel and user space

Netlink sockets provides us granularity in the communication kernel/user space. This way there is no need to add transfer a big arrays of memory when it's something has been modified. We can just send an update which will be processed in kernel. This reduces the time necessary for updates of kernel data which is shared with user space.

4.4.2. Definition of the SNE API The SNE API provides us a method to replicate the crucial information information. To be precise, SNE notifies asynchronously to user space:

- New *conntrack* created, updated and destroyed in the connection tracking table.
- New *expectations* confirmed and destroyed.

To replicate updates, three solutions are proposed :

- Strong replication : In this case, SNE notifies every update in current connections. Specifically, the update of the *conntrack* timer implies a message for every packet which hits the connection tracking system because a timer is refreshed every time a packet arrives. This inherently consumes more resources.
- Weak replication : this mechanism only notifies crucial updates. So, updates in timers or private structures are ignored.
- Incremental backup approach, as described in [7] can be used to reduce the number of messages, this fits in well an environment which short-time connections (like web server connections).
- Pre-process replication: the Primary node flushes the changes to Backup nodes every so often. This way, the Primary pre-process connections which are open and closed in a short periods of time, not replicating them. This reduces the bandwidth consumption but consumes more CPU resources.

A set of functions is proposed in the SNE library to perform different actions from user space via Netlink sockets:

- Manipulate conntracks from user space communicate: Create, update and destroy.
- Manipulate expectations: Insert, update and destroy an expectation which is associated with a given conntrack.
- Manipulate the whole connection tracking table: Destroy and load the whole connection tracking table.
- Manipulate Netlink sockets: Create, destroy, bind sockets as well as well as sending and receiving messages.
- Communicate a primary with the Backups machines via network sockets: Create, destroy, bind normal sockets as well as send and receive messages.
- Know the status of the physical link via MII registers.

5. Experiments

5.1. Evaluation of Netlink sockets

The SNE API is built on top of Netlink sockets. For that reason, we propose a tool called *netlinkbench* which consists of two components, a kernel module and a user space tool. It allows you to communicate unicast and broadcast messages of size γ between kernel and user space. By this way throughput and timestamping can be evaluated. It must be considered that Netlink sockets are not isolated from other components of the Linux kernel like the scheduler, virtual machine, memory manager, which modify system behaviour, so some of these components could influence experiments. Because of that, the results obtained have been normalised.

To evaluate throughput, the benchmark tool starts a timer. When the last message has been sent, it stops it and evaluates the results. On the other hand, to evaluate the time needed to send a message, the time is stamped to the packet, when it is received, that timestamp is compared with the current time.

This benchmark has been done with a kernel 2.6.6 non-preemptable on a P-IV 2400 MHz CPU and 256 MB SDRAM memory.

The throughput and timestamps are asymmetric : kernel to user space performance decreases because netlink messages are held in a buffer until they are retrieved by the user space program, that delays the delivery. On the other hand, when sending information from user to kernel, messages are also enqueued, but processed ipso facto.

Another important observation can be made when sending consecutive big messages, i.e. 5 messages of 32Kbytes,

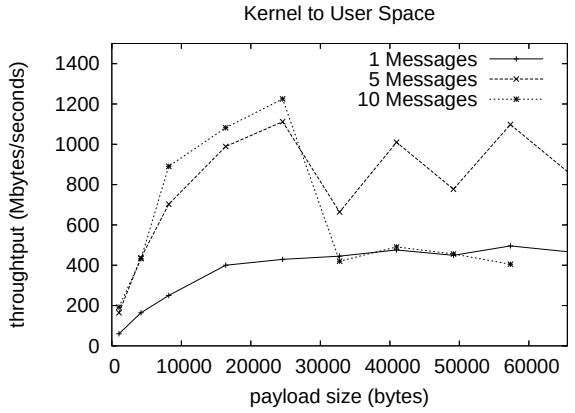

Figure 5. Kernel to user space communication

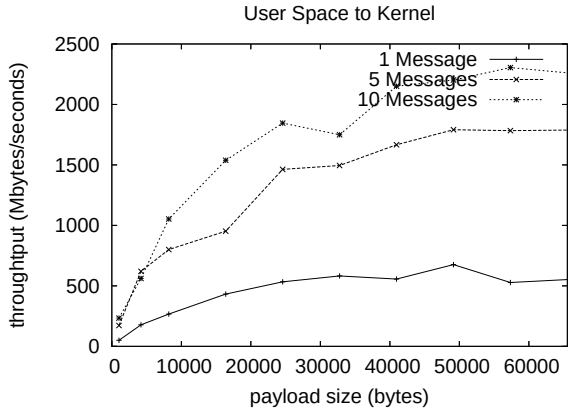

Figure 6. User space to Kernel communication

from kernel to user space (Fig. 5). The kernel internal socket buffer overruns, causing the drop of messages which cannot be enqueued because of lack of space. The size of this buffer can be increased to reduce the probabilities of suffering this problem.

We can also observe that the time needed to send big messages is proportional to the size of the message, since memory copies are $\theta(n)$ (Fig. 8 and 7). Another point is that latency when sending consecutive messages from user space gets reduced. On the other hand, from kernel, a single message obtains better results. This behaviour is also linked to the delay introduced by the socket buffer already pointed out.

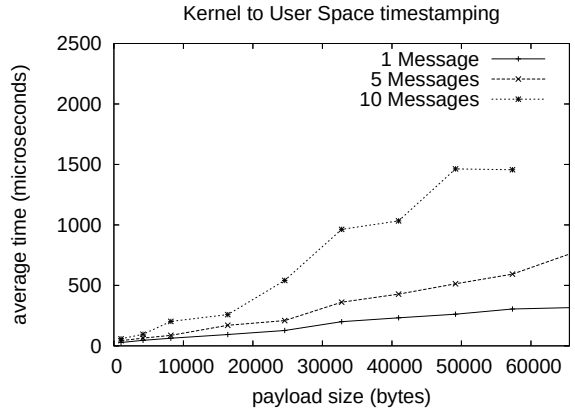

Figure 7. Timestamping kernel to user

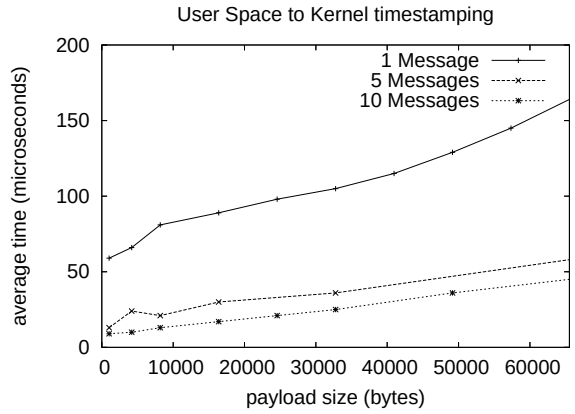

Figure 8. Timestamping user to kernel

To resume, the system performs better when sending a lot of messages from user space to kernel because of the buffer issue, and messages smaller than 32Kbytes must be used to ensure that packets are not dropped (Fig. 5).

5.2. Evalution of SNE

The limitation of messages bigger than 32 KB is difficult to exceed by SNE since the message size average is 64 bytes. The throughput obtained spamming user space with messages of 64 bytes is around 96 Mbits/s. In theory, the number of messages needed to overrun the socket buffer is:

$$Messages = \frac{Rate * 1024000}{MessageSize * 8} = 1900000$$

In conclusion, in theory, we have to send 190.000 messages per second, which is a reasonable limitation.

6. Conclusion and future works

Proposing stateful network equipments on open source systems is a challenging task. In this paper we have described and proposed the basic blocks (SNE library) for building a stateful network equipment. This library can be combined with high-availability protocols (CARP, Linux HA...). We focus on Linux system in order to provide software solutions for designing high-available solutions for NAT, firewalls, proxies or gateways equipments...This library is based on parts located in kernel and in the user space of the network equipment. First micro-benchmark of communications mechanisms with Netlink sockets have shown the effectiveness of our approach.

In addition to the technical challenges outlined in this article, the direction of our research is towards the integration of intelligent equipments (programmable switch, active network node) with stateful network components. We have also plans to study active-active settings: a more scalable solution that does not waste resources. The *primary-backup approach* wastes resources because of the fact that the Backup nodes remain latent until the Primary fails. A different approach could be share the workload between all the cluster, so all the nodes will be actives, and in case of failure the non-failing node will take over the failling node. Our plan is to integrate these technologies further, we are currently developing a complete Stateful Network Equipment adapted to firewall requirements and based on our library, joining our efforts to the Netfilter project.

References

[1] Firewall failover with pfsync and carp. http://www.countersiege.com/doc/pfsync-carp/.

[2] High-availability linux project. http://linux-ha.org/.

[3] Openbsd project. http://www.openbsd.org/index.html.

[4] N. Budhijara, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. In *Distributed Systems, ACM Press, pp.199-216*, New York, USA, 1993.

[5] T. Chou. Beyond fault tolerance. In *IEEE Computer 30(4):31-36*, 1997.

[6] G. Dhandapani and A. Sundaresan. Netlink sockets overview. http://qos.ittc.ukans.edu/netlink/netlink.pdf, 1999.

[7] T. Guang, J. Hai, and L. Pang. Layer 4 fault tolerance: Reliability techniques for cluster system in internet services. In *Advanced Environments, Tools, and Applications for Cluster Computing, NATO Advanced Research Workshop, IWCC 2001*, Mangalia, Romania, sep 2001.

[8] J. Hadi Salim, R. Olsson, and A. Kuznetsov. Beyond softnet. In *Proceedings of the 5th Annual Linux, USENIX*, Oakland, California, USA, nov 2001.

[9] R. Hinden. Rfc 3768: Virtual router redundancy protocol (vrrp), apr 2004.

[10] T. Li, B. Cole, P. Morton, and D. Li. Rfc 2281: Cisco hot standby router protocol, mar 1998.

[11] R. Perlman. Fault tolerant broadcasting of routing information. In *Computer Networks*, volume 7, dec 1983.

[12] H. Roelle. A hot-failover state machine for gateway services and its application to a linux firewall. In *Management Technologies for E-Commerce and E-Business Applications, 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002*, Montreal, Canada, oct 2002.

[13] P. Russel and H. Welte. Netfilter hacking how-to. http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.txt.

[14] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Rfc 3549 - linux netlink as an ip services protocol. http://www.faqs.org/rfcs/rfc3549.html, jul 2003.

[15] L. Senner. Anatomy of a stateful firewall. In *SANS Institute Information Security Reading Room*, may 2001.

[16] G. Van Rooij. Real stateful tcp packet filtering in ip filter. In *10th USENIX Security Symposium*, Washington, D.C, USA, aug 2001.

[17] H. Welte. How to replicate the fire - ha for netfilter based firewalls. In *Ottawa Linux Symposium*, 2002.