University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers | Computer Science and Engineering, Department of

1-1-2006

Router and Firewall Redundancy with OpenBSD and CARP

Garhan Attebury
University of Nebraska - Lincoln, attebury@cse.unl.edu

Byrav Ramamurthy
University of Nebraska - Lincoln, bramamurthy2@unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Attebury, Garhan and Ramamurthy, Byrav, "Router and Firewall Redundancy with OpenBSD and CARP" (2006). *CSE Conference and Workshop Papers.* Paper 68.
http://digitalcommons.unl.edu/cseconfwork/68

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

ICC '06. IEEE International Conference on Communications, 2006.
Volume: 1 Digital Object Identifier: 10.1109/ICC.2006.254719
Publication Year: 2006 , Page(s): 146 - 151

Router and Firewall Redundancy with OpenBSD and CARP

Garhan Attebury and Byrav Ramamurthy
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
{attebury, byrav}@cse.unl.edu

***Abstract*— As more reliance is placed on computing and networking systems, the need for redundancy increases. The Common Address Redundancy Protocol (CARP) protocol and OpenBSD's *pfsync* utility provide a means by which to implement redundant routers and firewalls. This paper details how CARP and *pfsync* work together to provide this redundancy and explores the performance one can expect from the open source solutions. Two experiments were run: one showing the relationship between firewall state creation and state synchronization traffic and the other showing how TCP sessions are transparently maintained in the event of a router failure. Discussion of these simulations along with background information gives an overview of how OpenBSD, CARP, and *pfsync* can provide redundant routers and firewalls for today's Internet.**

I. Introduction

The growing digital economy provides a perfect example of the need for redundant systems. When the online store of a company is not available, potential revenues are lost by the second. Quality of service applications such as streaming video are also very unforgiving towards service interruptions. Redundant systems are often used to provide increased availability, preventing such revenue loss and interruptions of service. In complex systems such as the Internet there are often many possible causes for loss of availability. One such cause is the failure of routers. In the worst case, a failed router will cause a complete outage of network communications if no other routes are available. In other cases, the failure may be temporary lasting only until new routes are discovered. However, even in this second case, a router failure may introduce instabilities into the Internet affecting both reliability and quality of service on a much larger scale [1].

The concept of redundant routing is not new. In fact, commercial solutions such as Cisco Systems' Hot Standby Routing Protocol (HSRP) [2], have existed for many years. An additional and very similar protocol is the Virtual Router Redundancy Protocol (VRRP) proposed by the IETF in the late 1990's [3]. Both of these solutions are flawed in the sense that they lack security and neither are free of patents. Specifically, HSRP is patented by Cisco Systems, which also claims the patent rights to the IETF's VRRP standard due to similarities between the protocols [4].

In August 2002 the OpenBSD community realized that Cisco Systems' claim to VRRP made it impossible to create a free implementation of the protocol [5]. Having already created the *pfsync* protocol to synchronize state between multiple firewalls, and needing a way to provide transparency of those firewalls to end hosts, the OpenBSD project [6] developed CARP. CARP, the Common Address Redundancy Protocol, was intended to solve the same problems as HSRP and VRRP while being different enough technically to not fall under Cisco Systems' patents.

Combined with the project's packet filtering (*pf*) system and *pfsync* utility, OpenBSD's CARP protocol is well suited to provide redundant routers and firewalls. In this paper we provide a background of previous redundant routing protocols, an overview of router and firewall redundancy using the above mentioned OpenBSD technologies, and an analysis of state sharing traffic and TCP session maintainability through experimentation. In these experiments, two physical routers are used to create a single virtual router. This virtual router performs basic routing and stateful firewall functions between two end host computers on separate subnets. The term router is used to refer to this router and firewall combination from here on.

The remainder of this paper is organized as follows. Section 2 presents a background on other protocols related to CARP and router redundancy. Section 3 provides the motivation for the existence and use of CARP. Section 4 explains the CARP protocol in greater detail and how it can be used for firewall redundancy and/or load balancing. Section 5 details the experiment used to evaluate CARP and Section 6 discusses the results of that experiment. Section 7 presents suggestions for future work and Section 8 concludes the paper.

II. Related Protocols

Router redundancy is not a new concept. Cisco Systems' HSRP has been available in their networking products for many years, and the IETF's VRRP is available on the products of many companies such as Juniper Networks, 3Com Corporation, and Cisco Systems, Inc. VRRP has also been implemented on many operating systems including Linux and BSD. HSRP and VRRP are the two closest relatives to CARP and are discussed in this section.

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

A. Hot Standby Router Protocol (HSRP)

HSRP is designed to provide non-disruptive fail-over routing in networks which have fixed next hop routes such as Ethernet LANs. Two or more routers are grouped together into a virtual router (also called a HSRP group or hot standby group) which presents a single host for the next hop route. Each of these virtual routers has a single well-known MAC address and IP address which are different from the addresses assigned to any of the real router's physical interfaces. While each router is then capable of receiving packets destined to the virtual router, only a single router, called the active router, actually forwards packets. The active router and a second standby router are chosen through an election process. Once the election process is over, the active and standby routers periodically pass a heartbeat message so that they can detect the failure of one another. If the active router fails, the standby takes over and another standby router is elected. If the standby router fails, the active router remains active and another standby router is elected. HSRP allows for multiple virtual routers to be created on a single LAN and for load sharing to occur by distributing physical hosts among different virtual router groups. A physical router maintains separate state and timers for each group it participates in. Communication between the routers of a group is optionally protected by an 8 character plain text password.

B. Virtual Router Redundancy Protocol (VRRP)

VRRP is almost identical in functionality to Cisco Systems' HSRP and Digital Equipment Corporation's IP Standby Protocol (IPSTB) with only minor differences in its operation [7]. Early versions of VRRP included not only the plain text password authentication mechanism from HSRP but also a HMAC authentication [8] scheme. However, experience showed both schemes offered little to no additional security and have been removed in the latest version of VRRP leaving no authentication mechanism. A second difference in VRRP is the use of ICMP redirects, a mechanism for routers to send routing information to end hosts, allowing its use in non-symmetric networks. A non-symmetric network is one where packets flowing in one direction through the router group differs from the other direction. Packets may leave a network through a router group A but return to a router group B. It is possible for a VRRP router to act as master for a group with addresses it does not own. In this case, the router would need to determine which group the packet was sent to when setting the redirect source address. In symmetric networks with load sharing between routers, this ICMP redirect ability is often disabled. HSRP explicitly forbids the use of ICMP redirects to hide the primary MAC addresses of routers in the virtual group.

VRRP also relies on an election process to determine which routers becomes master and which routers become standbys. This differs from HSRP in that HSRP only elects a single backup router, whereas in VRRP different priorities get assigned to all backup routers with the router of highest priority becoming master upon failure of the original master.

III. Motivation

Link failures in an IP network cause surrounding routers to react by updating their routing tables to reflect the change in topology. These changes often propagate through the Internet causing instabilities in the overall routing of data. Such instabilities are referred to as route flaps and are one of many pathologies affecting both the performance and the availability between end to end hosts [9]. It has been shown through empirical evidence that certain inter-domain routing protocols such as BGP suffer from delayed routing convergence after failures. These delays, which can last in the timescale of minutes, may interrupt communication between end hosts by reducing routing performance or preventing communication all together [10], [11].

Incorporating redundancy among routers reduces the probability of link failures and subsequently reduces the chance of route flaps forming and the need for routing convergence to even occur. Removing these pathologies increases the stability and performance of the Internet as a whole thereby benefiting providers and customers who rely on these traits for profit and quality of service.

The earlier router redundancy protocols such as IPSTB and HSRP are both proprietary and constrained from general use by patent law. The IETF's development of VRRP was intended to provide an open and patent-free protocol of similar design. While the VRRP standard has without a doubt enjoyed wide acceptance by commercial and open source vendors alike, Cisco Systems has claimed patent rights to it preventing it from being used in true open form. As a result of this encumbrance, the OpenBSD project designed and developed CARP to provide the functionality of the previous protocols under the original BSD license. CARP also introduces new features, the most notable being the use of cryptography to increase security. The OpenBSD project also has two more developments, namely the packet filter (*pf*) and *pfsync* utilities, which allow the creation of robust, free, and redundant combined router and firewall systems. CARP, *pf*, and *pfsync* are discussed in the next section.

IV. CARP

This section introduces the operation and features of CARP, *pf*, and *pfsync* in OpenBSD.

A. CARP standalone

CARP by itself provides redundancy between systems. These systems need not be limited to routing, and can easily serve other roles such as that of a web server. Like VRRP, CARP is a multicast protocol which groups multiple systems together into a virtual group called a CARP group. This group presents a single shared MAC and IP address combination to the hosts of a network. Just as in VRRP, a master is elected from the group with the remaining systems being assigned priorities indicating which takes over when the master fails. The inclusion of a clock skew setting allows the manual assignment of priority. This can be used to give a particular machine a greater chance of being elected master, and to cause that machine to be re-elected as master once it recovers from a failure. CARP requires each

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

system to have its own MAC and IP address in addition to the virtual addresses and also requires all IP addresses of a group be on the same subnet. Any service running on the systems can be configured to use the virtual addresses of a group transparently giving the benefit of redundancy. By itself, CARP does not provide a mechanism for replicating data among the group. This needs to be accomplished by other tools such as *rsync* for file replication or *pfsync* for firewall state replication.

B. CARP, pf, and pfsync

The OpenBSD project has included packet filtering software called *pf* in their releases since version 3.0 which can be used to create advanced stateful firewalls. When using CARP in a standalone mode to provide redundant stateful firewalls a problem arises. If the master firewall goes down, all the state information is lost and existing stateful connections will be unknown to the backup and therefore be blocked and terminated. The solution to this problem is *pfsync*.

pfsync is the OpenBSD project's protocol to synchronize firewall state tables between multiple systems. From the operating system viewpoint, *pfsync* is a pseudo-interface which can be configured in multiple ways such as to send the state update information over a physical interface or through a VPN tunnel. Like CARP, *pfsync* is a multicast protocol allowing state updates on the master to be sent to all backup firewalls. When the backups receive state updates over their *pfsync* interfaces, the updates are inserted into their own internal state tables thus synchronizing the state information between all firewalls. *pf* does allow a *no-sync* keyword to be specified on firewall rules such that state information created from that rule is not passed to the backup systems. *pfsync* also automatically tries to combine multiple state updates into a single update and to use compression where possible.

C. Load balancing with CARP

The increased amount of traffic flowing over today's networks means an increased demand in processing power for the gateways, routers, and firewalls that manipulate it. This is especially true in the cases where detailed packet inspection and modification occur such as intrusion detection systems (IDS), network address translation (NAT), and scrubbing [12], [13]. It is not unreasonable for a large network to contain more traffic than can be handled by single system. Load balancing is often used to split demand among many systems, and CARP provides a mechanism by which to accomplish this.

Load balancing with CARP is done through ARP based hashing only. This requires a CARP group to be setup for each physical host with the groups sharing a common virtual IP address but having unique virtual MAC addresses. As every virtual interface sees the traffic on its side of the network, it is simple to perform a hash on the source address of a connection to determine which group (or physical machine) acts on that connection. While primitive, this form of load balancing has the advantage of simplicity in that multiple connections from a end host always get passed through the same CARP system.

This type of load balancing with CARP need not be restricted to firewalls and routers, and can be used equally as well for other services such as a pool of web servers.

V. Implementation

To observe the usage and behavior of CARP, *pf*, and *pfsync* in operation, a small network consisting of two routers and two end hosts was created. This section describes the network layout and the configuration used to achieve router redundancy. In addition, details of the testing methods are given at the end of this section.

A. Network Layout

The two routers acted as both a gateway and a firewall between the two subnets, and for the remainder of this paper the term router shall refer to this combination. The end hosts used were standard computers running Linux with one running the Apache web server [14] and the other making HTTP GET requests using Siege [15]. The topology of this network is depicted in Fig. 1.

B. Router Configuration

The two routers, labeled Router A and Router B, were physically identical computers each with three physical network interfaces. Using physically identical computers is not necessary to achieve router redundancy with OpenBSD and CARP. On both machines, OpenBSD 3.6 was installed with all default settings except for the following change to */etc/sysctl.conf* :

```
net.inet.ip.forwarding = 1
```

This change was required to allow forwarding of packets between the interfaces. Two of the three physical interfaces, *dc0* and *dc1*, were given unique addresses on the web client subnet and the web server subnet respectively. In order for CARP to function, each participating machine in a virtual group needs to be able to receive and see the same network traffic. This was accomplished by creating a new CARP interface with a shared IP address on both routers. The two interfaces, one for each of the two subnets, were created as follows:

```
ifconfig carp0 create
ifconfig carp0 vhid 1 pass df2m1 10.1.1.1
ifconfig carp1 create
ifconfig carp1 vhid 2 pass q3c4m 10.1.2.1
```

This created two virtual interfaces, one with the IP address 10.1.1.1 and the other with the IP address 10.1.2.1. As the routers share these IP addresses, traffic sent from other computers on the subnets gets seen by both. Refer to Fig. 1 for a complete view of the layout including the IP addresses for each interface. Only the master of the group actually takes action on the traffic when not using load balancing. Note the inclusion of a simple five character password (df2m1 and q3c4m) in the CARP interface creation command. This

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

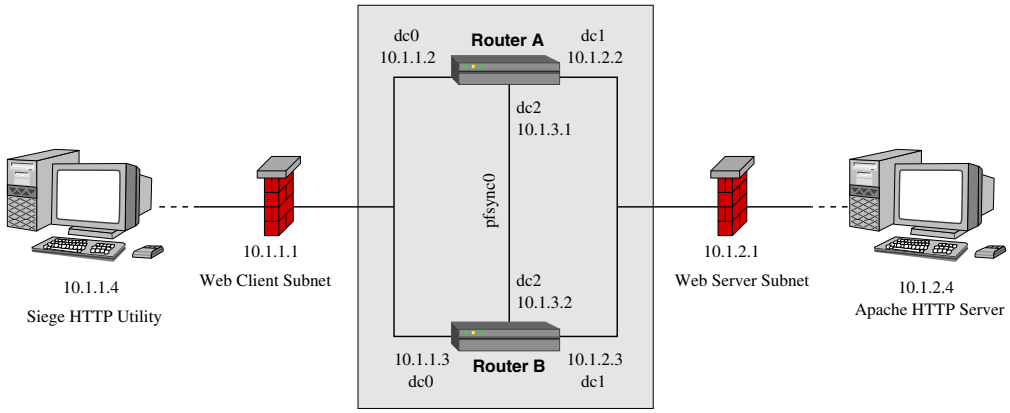

Fig. 1. Example CARP Topology

password is similar to that found in HSRP and what previously existed in VRRP. If provided, the password is used to encrypt all CARP communication packets between the routers with an SHA-1 HMAC scheme [8].

In addition to the *carp0* and *carp1* interfaces, an interface for *pfsync* to transfer state information between the routers was created with the following:

```
ifconfig pfsync0 syncif dc2
```

This *pfsync0* interface was tied to the third physical interface, *dc2*, of each router. By default the state information sent over the *pfsync0* interface is multicast and encrypted. In production environments this would most likely be a truly private and secure network, perhaps a single crossover cable in the case of two routers. The *pfsync* utility also provides a mechanism for unicasting the state traffic which could be used with other security mechanisms such as IPSec tunneling to share state information between distant routers.

The following rules which prevent the blocking of CARP and *pfsync* traffic were added to the *pf* configuration.

```
pass quick on { dc2 } proto pfsync
pass on { dc0 dc1 } proto carp keep state
```

In addition, the following rule was added to maintain the state of standard TCP connections between the subnets. No additional *pf* rules were specified.

```
pass on { dc0 dc1 } proto tcp keep state
```

C. Web server and Web client

The web server and web client computers ran Linux and each contained one physical network interface configured to a unique address on the subnet to which each computer belonged. The web server software used was Apache 2.0.52 [14] and the web client software used was Siege 2.61 [15]. Siege is a HTTP stress testing utility which was used to generate new HTTP requests many times a second. Each new HTTP request created two new TCP states, one for each direction of traffic, on the master router. This was a reliable and simple method of creating new states thereby causing *pfsync* to create state traffic on the *pfsync0* interface. The only additional configurations to the web server and web client were the manual addition of the *carp0* and *carp1* group IP addresses as routes for the corresponding subnets as follows:

(on the web client) `route add default gw 10.1.1.1`
(on the web server) `route add default gw 10.1.2.1`

D. Packet Capture and State Monitoring

Both the public interfaces with the CARP protocol traffic and the *pfsync* interfaces with the state update traffic were monitored using Ethereal [16]. During the course of the experiments, all traffic on these interfaces was captured and stored. The results of this packet capturing are discussed in Section VI.

The state tables on each router were monitored periodically during both tests using the *pfctl* utility included with OpenBSD using the following command:

```
pfctl -s info
```

This command gives, amongst other items, the current number of state table entries and the number of state table inserts, removals, and searches.

E. Testing

Two tests were performed on this network setup. The first test measured the amount and characteristics of the traffic generated on the *pfsync0* interfaces during a period where new states were generated by Siege. The creation of new states, two for each HTTP GET connection, causes the need to synchronize these state tables between the master and backup routers. Siege was stopped before packet capture ended as states will eventually expire on the master, again requiring *pfsync* updates to be sent to the backup.

The second test was to show that stateful connections such as SSH will be maintained even upon the failure of the master router providing transparent fail-over from the end user's

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

TABLE I

pfsync0 TRAFFIC STATISTICS

Parameter	Value
Total number of packets	3827
Average packets per second	81.75
Average packet size in bytes	465.68
Total bytes transfered	1782170
Average bytes per second	38067.81
Average Mbits per second	0.31

perspective. This test consisted of starting various SSH sessions between the web client and web server machines and literally "pulling the plug" on the master router causing the backup to take over while maintaining the active SSH sessions.

VI. Discussion

A. pfsync0 Traffic Analysis

The packets captured over the *pfsync0* interfaces of the two routers can be categorized into two types. The first is the state table update information generated by *pfsync* when a new state is created on the master router. These updates are sent over the *pfsync0* interface almost immediately to keep a high level of synchronization between the routers. The second of these types is the revocation updates which remove expired states from the backup routers. These updates are grouped together in batches so as to minimize the amount of traffic on the *pfsync0* interface.

During the Siege/Apache test, a total of 1269 unique HTTP transactions (HTTP GETs) were created over a period of 46 seconds, an average of 27.53 transactions per second. Each HTTP transaction created two states in the *pf* state table meaning 2538 states were generated at an average rate of 55.2 states per second. In addition to the packets required to update the state tables, *pfsync* generates periodic packets on the *pfsync0* interface which are 180 bytes in size. In the 46 second test period, 92 of these packets were sent totaling approximately 16 kilobytes of data. Fig. 2 shows a plot of the packets per second transmitted over the *pfsync0* interface during the test period. Important periods include 14-60 seconds which is when Siege was generating HTTP GET requests, 60-108 seconds when no state updates occurred, and 108-152 seconds when the batch state revocation updates occurred. The period of 0-14 seconds is where packet capture had begun but the Siege request generation had not. Table 1 displays the traffic statistics for the period of state creation only, that is during time of 14-60 seconds.

Traffic statistics show that during the creation of new states, which occurred at the rate of 55.2 states/sec, the average amount of *pfsync* traffic generated was only 0.31 Mbits/sec. As the size of the packets containing state change information does not vary greatly, it can be said that the amount of traffic generated on the *pfsync0* interface of the master router scales linearly with the amount of state changes occurring. Scaling up the 0.31 Mbits/sec to a full 100 Mbits/sec gives a rough value of over 17,000 states needing to be created per second to saturate such a link. While a 100 Mbits/sec link would be common for installations where the physical routers are located near one another, other cases exist where the *pfsync* link capacity may become an issue. Take for example the case of a large campus where border routers may be located miles apart. A 1.5 Mbits/sec T1 link may be used to share state information reducing the above number to around 250 new states/sec. Add the fact that the *pfsync* link can be easily tunneled with IPSec over the Internet and that using unicast instead of multicast scales the amount of traffic by a factor of n where n is the number of routers, it is clear that cases exist where the *pfsync* link capacity affect the synchronization of state tables among the routers.

B. SSH Sessions Maintained

The purpose of synchronizing firewall state information between the routers is to ensure no existing connections get broken when a failure occurs. Starting SSH sessions between the web client and web server computers created multiple persistent TCP sessions through the routers. Each SSH session created two state entries in the master which were replicated on the backup as expected. Failing the master and letting the backup take over did not interrupt the SSH sessions as the TCP sessions already existed on the backup router letting the traffic associated with the sessions flow freely. However, the solution is not flawless. By default the time period between advertisements of the master's operation is three seconds. During this period no packets are being routed because while the master has failed, the backup(s) have not yet realize it. Configuring the period between advertisements to a lower value will decrease this period allowing much quicker response. As TCP is a reliable transport protocol, this delay will generally not cause termination of the connection or loss of data. UDP traffic, however, may be lost during this delay period as it is an unreliable datagram protocol.

VII. Future Work

The scope of this paper is limited to OpenBSD's CARP and *pfsync* protocols with a focus on their performance in a simple two host network. More network intensive tests could be done, such as generating multiple types of traffic (multicast, video streams, etc.) which would be more indicative of real world data. Also, future tests could include other *pf* features such as queueing priorities and focus not only on data passed over the *pfsync0* interface but also network processing time. Even more complex cases remain to be explored, such as the use of load balancing among routers.

In a load balancing scenario, all routers would need to share their states while different routers operate on different subsets of connections. This technique would distribute the processing of traffic among all the routers in a virtual group, the obvious benefit being that increased demands from more traffic could be met with the addition of more routers. This would be especially true in cases where the routers are doing complex tasks such as intrusion detection or the processing of large firewall rule sets. While CARP on OpenBSD includes an ARP balancing feature, it is not well suited for this task. First, the balancing

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

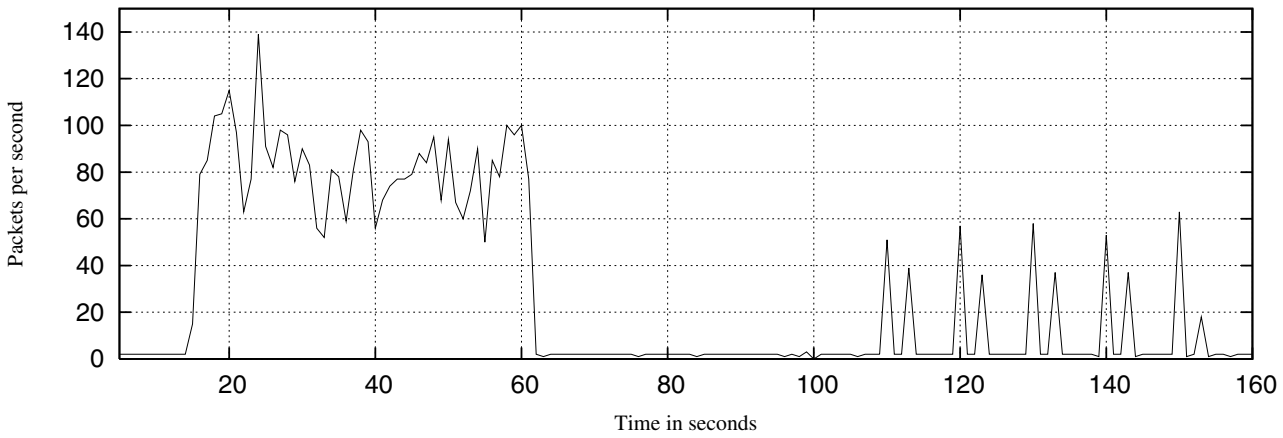

Fig. 2. pfsync0 traffic graph

is accomplished through a simple hash of IP addresses, which in practice will not give 50/50 load splits. Second, the ARP balancing feature is intended to pass traffic along to a secondary set of servers on an inside subnet protected by the routers. It is not readily apparent that load balancing among the *pf* and *pfsync* systems on the routers themselves is possible with the current implementations.

Finally, a comparison between Cisco Systems' proprietary HSRP and the IETF's VRRP standard to CARP and *pfsync* is needed to see which of the three options is best for a given application. Measuring HSRP would best be done on Cisco hardware itself. VRRP implementations are however available in both hardware and software implementations, the latter of which is free to acquire but still restricted by patents.

VIII. Conclusion

The goal of this work was to provide an overview into how the OpenBSD Project's CARP protocol can be used in conjunction with the *pf* and *pfsync* firewall utilities to create redundant stateful firewalls. Also touched on were the possibilities of using CARP by itself to provide redundant routers without firewalls, to provide load balancing, and to provide redundancy in general purpose server scenarios. Experiments were conducted and results were provided quantifying how traffic on the *pfsync* interface may limit the number of new firewall state creations per second and how reliable traffic flows such as TCP remain uninterrupted during a fail-over. Also discussed was the flexibility provided by CARP and *pfsync* such as the case where routers at distant locations can securely share firewall state information to provide transparent redundancy. The need for such redundancy has become more apparent in recent years as heavy reliance is put on the Internet to support applications demanding both quality of service and high availability. As OpenBSD and the tools discussed here are available freely under the BSD licence they provide a robust and accessible solution for router and firewall redundancy.

Acknowledgements

This work is partially supported by NSF Grant No. CCR-0311577.

References

[1] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an ip backbone," *Internet Measurement Workshop*, 2002.

[2] T. Li, B. Cole, P. Morton, and D. Li, "Cisco Hot Standby Router Protocol (HSRP)," RFC 2281 (Informational), Mar. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2281.txt

[3] R. Hinden, "Virtual Router Redundancy Protocol (VRRP)," RFC 3768 (Draft Standard), Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3768.txt

[4] "http://www.ietf.org/ietf/ipr/vrrp-cisco." [Online]. Available: http://www.ietf.org/ietf/IPR/VRRP-CISCO

[5] OpenBSD release song lyrics. [Online]. Available: http://openbsd.org/lyrics.html#35

[6] The OpenBSD Project. [Online]. Available: http://openbsd.org/

[7] A. Srikanth and A. A. Onart, *VRRP: Increasing Reliability and Failover with the Virtual Router Redundancy Protocol*. Pearson Education, September 2002.

[8] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Feb. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[9] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*. New York, NY, USA: ACM Press, 1997, pp. 115–126.

[10] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 293–306, 2001.

[11] X. Zhao, D. Massey, D. Pei, and L. Zhang, "A study on the routing convergence of latin american networks," in *LANC '03: Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research*. New York, NY, USA: ACM Press, 2003, pp. 35–43.

[12] T. Verdickt, W. V. de Meerssche, and K. Vlaeminck, "Modeling the performance of a nat/firewall network service for the ixp2400," in *WOSP '05: Proceedings of the 5th international workshop on Software and performance*. New York, NY, USA: ACM Press, 2005, pp. 137–144.

[13] D. Watson, M. Smart, G. R. Malan, and F. Jahanian, "Protocol scrubbing: network security through transparent flow modification," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 261–273, 2004.

[14] Apache HTTP Server Project. [Online]. Available: http://httpd.apache.org/

[15] J. Fulmer. Siege HTTP Utility. [Online]. Available: http://www.joedog.org/siege/

[16] Ethereal. [Online]. Available: http://ethereal.com/

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.