Probabilistic Policy Violation Detection with IPFWD

Dr. Phil Nelson and Austin Voecks

***Abstract*—When using a firewall system like IPFW to detect threats, we can end up doing a lot of packet processing. This can negatively impact performance-sensitive systems such as storage nodes. This paper describes a practical solution to this problem using a load-weighted probabilistic mechanism that allows a trade-off between perfect visibility of incoming packets and reduced impact to system load.**

***Keywords*—FreeBSD, IPFW, firewalls, performance.**

I. Introduction

IPFWD is a daemon for FreeBSD that updates an early rule in IPFW that has a chance to accept any packet. The probability of acceptance is updated over time and dependent on the current system load. To account for this, IPFWD extends IPFW logs to include the likelihood an undetected policy violation occurred.

This approach is based on the premise that firewall performance can be improved by reducing the number of rules applied to each packet. Supposing a whitelist policy, a firewall has to apply every rule to each packet before it's denied. With IPFWD, we have a chance to accept any packet early and skip any further computation. Test results shows this reduces the system resources required to handle the same amount of traffic.

This is a shift in mindset from typical firewalls. Instead of enforcing every part of a security policy all the time, IPFWD enforces the policy some of the time and extrapolates from the violations it encounters. This provides insight into the violations that weren't detected. For this cost, you gain increased firewall performance and network throughput in resource bound systems. It's acceptable to allow a percentage of policy violations given that network traffic patterns are often repeated and the goal is detection and not immediate prevention. Preventative action may be taken later when resource requirements are lower.

As an example, under heavy load IPFWD may immediately accept 40% of packets. Supposing a port scan was initiated during this time and rules exist to block it, 60% of the port scan would still be rejected and logged. Since the probability will fluctuate over time, IPFWD provides information in the IPFW logs to show the chance undetected violations occurred for each detected violation. IPFWD allows administrators to keep extensive rule sets that fully implement their security policy. Instead of having to simplify rule sets to increase performance, IPFWD balances policy enforcement and performance automatically. Under normal or light load, IPFWD will enforce the entire security policy 100% of the time.

In collaboration with Dell EMC Isilon

II. IPFW

Firewalls restrict the types of incoming and outgoing network traffic according to a security policy. Without a security policy, administrators cannot know what to allow and disallow. Typical firewalls enforce this policy by blocking packets that do not match the rules. Whitelist firewalls default action is to block all traffic, additional rules describe exceptions. Blacklist firewalls are the opposite in that by default they accept all traffic and additional rules describe packet types that are not allowed. Since predicting and blacklisting all possible attacks is infeasible, whitelist firewall rulesets are the accepted standard and the focus of this paper.

IPFW is a stateful firewall written for FreeBSD which supports both IPv4 and IPv6. It is comprised of several components: a kernel firewall filter rule processor and its integrated packet accounting facility, the logging facility, NAT, the dummynet traffic shaper, a forward facility, a bridge facility, and an ipstealth facility.

IPFW was chosen for it's close integration with the FreeBSD operating system, in kernel packet filter, and general performance. Performance is dependent on system parameters and circumstances, but IPFW outperforms PF in stateful packet filtering. Stateful rules are more powerful and flexible than stateless rules, were no session information is maintained. Additionally, the capabilities of IPFW extend beyond packet filtering into source based routing, traffic shaping and more.

Firewall rules are treated by IPFW in a first-match-wins fashion. IPFW also contains built-in functionality and rule syntax for probabilistic packet matching. From the documentation, this feature was intended for load balancing and other traffic shaping tasks. However it's the core of IPFWD's operation.

This paper is not meant as a comparison between IPFW and PF; the functionality provided by IPFWD could be ported to PF and serves as a proof of concept for firewalls in general.

III. IPFWD

IPFWD increases firewall performance by reducing the average amount of work it takes to process a packet. Depending on the current system load, IPFWD increases or decreases the chance the IPFW will immediately accept a given packet. It does this by adding and maintaining an early rule in the IPFW rule set of the following form: `prob 0.000 allow ip from any to any`.

Since IPFW uses a first-match-wins rule system, this early rule will be encountered before the majority of the other rules in the rule set. The probability given in this rule determines the chance that we skip processing all further rules, supposing that there hasn't been a match already.

This is especially valuable for a whitelist firewall rule set dealing with large amounts of rejected traffic; normally each rejected packet has all rules applied to it while IPFW looks for a match. When no matches are encountered, the default rule is employed and the packet is rejected.

In a typical data center environment the vast majority of network traffic is valid and going to be accepted eventually. However, these rule sets are often complex and take increasing time to match a given packet depending on the length. For a rule set of length n, we can expect to a packet to encounter $n/2$ failed matches before matching the correct rule and being accepted.

IPFWD improves the average number of failed matches before successful match from

$$\frac{n}{2}$$

to

$$\Big(k \cdot \mathsf{P}(EA)\Big) + \Big(\frac{n}{2} \cdot \mathsf{P}(\neg EA)\Big)$$

where k is number of rules before the early acceptance rule and $\mathsf{P}(EA)$ is the probability that the early acceptance rule is matched.

Removing the constant k, IPFWD reduces the number of failed matches before a successful match according to the following ratio:

$$\frac{\frac{n}{2}}{\frac{n}{2} \cdot \mathsf{P}(\neg EA)} = \frac{1}{\mathsf{P}(\neg EA)}$$

However, while this method increases the speed the system processes packets, it creates a chance that packets that would have normally been rejected will get through the firewall. We'll refer to these types of packets as invald.

To account for this, IPFWD extends the IPFW system logs to include information about the early accpetance probability. From this information, administrators can extrapolate the probability that additional invalid packets made it through the firewall for each that is detected. This is discussed further in the Security Implications section.

IPFWD is not intended to be used on all types of systems. It's application instead most directly benefits the following types of systems:

1) **High Network Performance**
The gains provided by IPFWD are only apparent in systems were network throughput is matched by or out performs CPU speed. Systems that can already easily handle fully processing each packet will not benefit by reducing processing time. Given modern CPU clock speeds, systems with network cards slower than 1 Gb/sec will likely not benefit either.
2) **Resource Conscious**
In line with high performance systems, systems with strict resource restrictions can benefit by the reduced network related CPU usage.
3) **Complex Firewall Requirements**
IPFWD works by reducing the average number of rules that fail to match before a packet is finished being processed. Very simple rulesets will gain little from reducing the number of rules applied to each packet.
4) **Detection Based Security Policy**
By using IPFWD, administrators sacrifice the assurance that every invalid packet will be rejected. However, they are still provided information as to whether some invalid packets were rejected, and that information can be used to infer the existance of additional invalid packets.

IPFWD is based on the assumptions that the majority of network traffic is valid, malicious events are rare, and

IV. Security Implications

V. Benchmarking Performance

Tests were run on three platforms and five different network cards, resulting in five different systems. The most important parameters to our tests are CPU, NIC, and ENV. Descriptions of each system are given in the tables below:

Isilon OneFS 40Gb System	
OS	FreeBSD 11 Release
CPU	8 x64 Intel® Xeon® CPU @ 2.20GHz
MEM	64Gb
NIC	Ethernet 40Gbase-T
ENV	Physical

Isilon OneFS 10Gb System	
OS	FreeBSD 11 Release
CPU	8 x64 Intel® Xeon® CPU @ 2.20GHz
MEM	64Gb RAM
NIC	Ethernet 10Gbase-T
ENV	Physical

WWU 10Gb System	
OS	FreeBSD 11 Release
CPU	8 x64 Intel® Core™ i7-2600 CPU @ 3.4GHz
MEM	16Gb
NIC	???
ENV	Physical

WWU 1Gb System	
OS	FreeBSD 11 Release
CPU	8 x64 Intel® Core™ i7-2600 CPU @ 3.4GHz
MEM	16Gb
NIC	Ethernet 1000baseT
ENV	Physical

Virtualized 10G System	
OS	FreeBSD 11 Release
CPU	1 x64 Intel® Xeon® CPU @ 1.80GHz
MEM	512Mb
NIC	Ethernet 10Gbase-T
ENV	Hypervisor

The systems tested all used 64bit FreeBSD 11 Release, with varying CPU type, CPU speed, RAM and network cards. Isilon and WWU systems were both physical hardware, while the Virtualized system was in a hypervisor environment hosted by a third party service.

Virtualization makes it more difficult to have the same confidence in the results as with the physical machines. Tests on the Virtualized system were repeated additional times to account for the variability of the underlying physical hardware and contention between virtual machines.

The WWU and Virtualized tests were conducted on stock FreeBSD 11 without additional network performance configuration applied. Isilon OneFS does contain additional network performance tuning and demonstrates the additional gains IPFWD can provide with careful FreeBSD 11 kernel configuration.

A. Netperf

Multiple network performance tools were explored but the final results were compiled using Netperf. Netperf was chosen because it was readily available on all test platforms, provides built-in CPU utilization measurement, and a wide range of test types.

The test type UDP_STREAM was chosen over TCP_STREAM for simplicity and to preserve CPU cycles for IPFW. Raw network throughput and CPU performance is the target metric, which can be difficult to assess with TCP.

Netperf is single threaded, which allows easier comparison of results between systems with different numbers of cores. Additionally, all tests were run while the system was idle to minimize contention over system resources.

All tests were conducted using the following steps:

1) Machine 1
    a) ipfw
    b) netserver
    c) CPU monitor
2) Machine 2
    a) ipfw
    b) ipfwd
    c) netperf
    d) CPU monitor

All tests were run with minimal IPFW rule sets in a range of 30 to 60 rules. Intuitively, larger production rule sets would benefit even further from IPFWD.

B. CPU Measurement

CPU utilization measurement is the most difficult parameter to measure in these tests. There are many factors that can account for variability between tests and jitter in the results. The Netperf manual describes these challenges and the variety of techniques used on different systems.

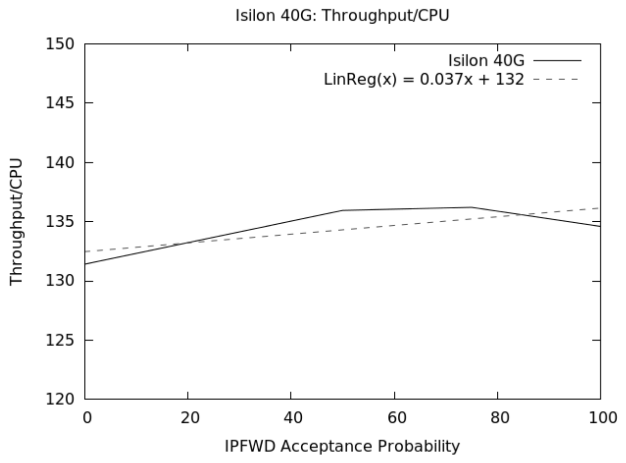
Netperf was run in single threaded mode, which explains why throughput wasn't closer to the maximum throughput for the NIC on each system. The FreeBSD network stack allows multithreaded packet processing by utilizing multiple queues. Since we're indirectly testing the performance of IPFW through Netperf, there are two different CPU measurements to account for: kernel time and user time. Since IPFW runs in the kernel, it's CPU time is classified as kernel time.

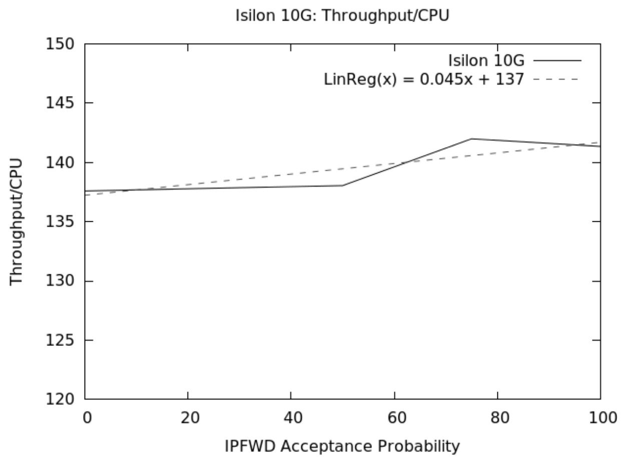
Netperf was able to measure CPU utilization on the Virtualized and WWU systems, but didn't work on the Isilon systems. Instead, ps was used on those systems. These tests are comparable since both measured total system CPU utilization

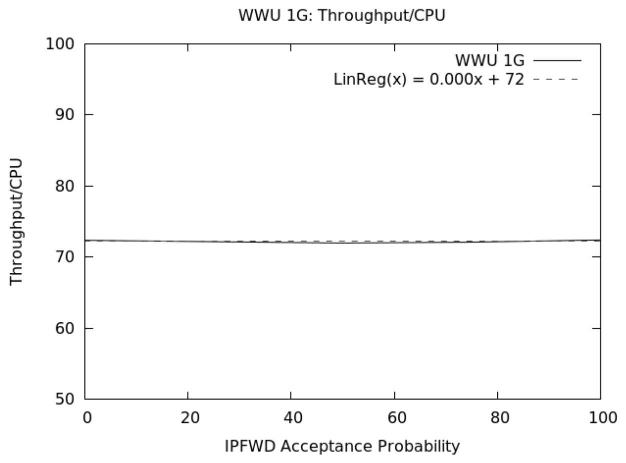
VI. Results

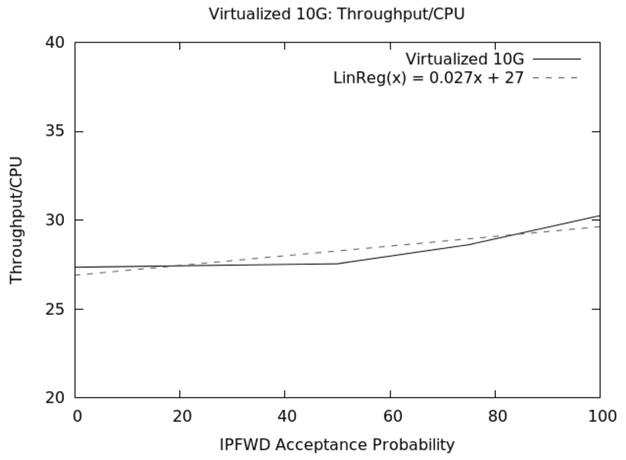
A. Throughput / CPU

These graphs show network throughput divided by CPU utilization for each system. Higher values mean that the CPU time required to process packets is lower.



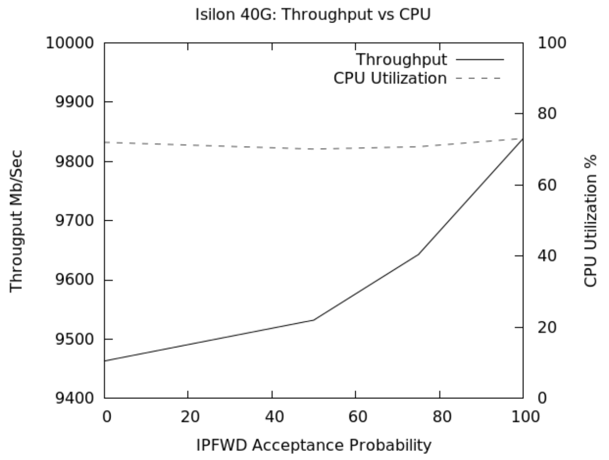


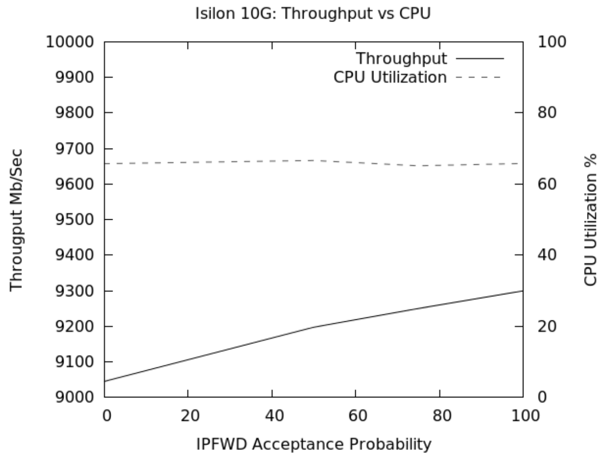


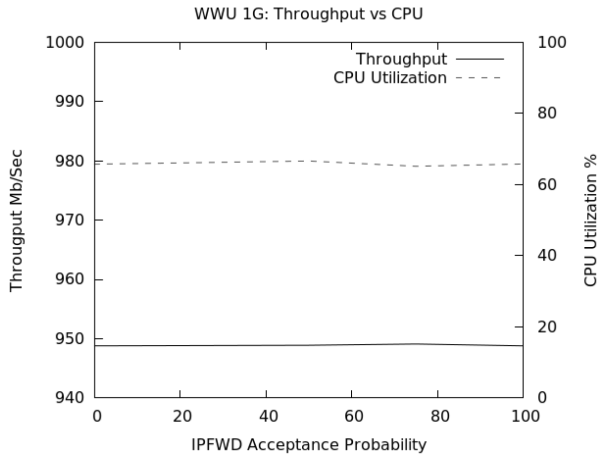


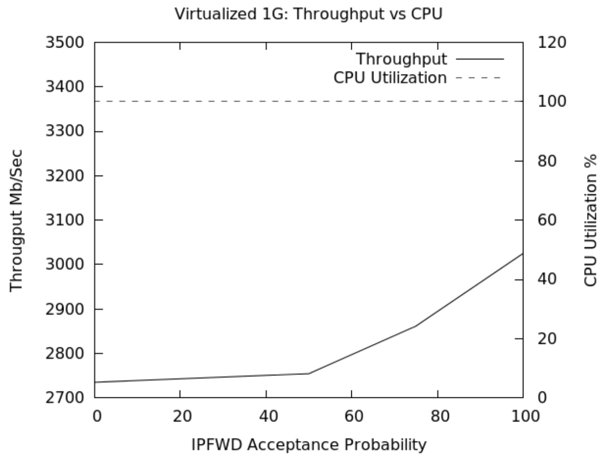
B. Throughput & CPU

These graphs show the interaction between CPU utilization and network throughput for each system with respect to IPFWD's acceptance probability. The dashed lines are CPU utilization, solid is network throughput in Mb/second. Higher acceptance probability means a greater chance that a given packet is immediately accepted without further processing.









C. Discussion

All the systems tested can be broadly categorized into two groups: CPU bound and NIC throughput bound. We can see that IPFWD affects performance positively but differently for both types.

In CPU bound systems, IPFWD reduces the number of cycles it takes to process each packet, allowing more packets to be processed in the same time and increasing throughput. This is shown most clearly in the Virtualized system tests, which have the weakest CPU but 10Gb Ethernet NICs. Regardless of test type, the receiving system is always at 100% CPU utilization. However, as the probability to accept increases in IPFWD the throughput also increases. This clearly shows that IPFWD decreases the CPU load required to process incoming traffic.

In NIC throughput bound systems, reducing the number of CPU cycles to process packets will not increase throughput. However, IPFWD still reduces CPU load which allows more cycles to be dedicated to other system processes. This is more difficult to see in the test results, particularly since the CPU measurement tools are inherently imprecise and difficult to compare between systems.

VII. Related Work

Firewall decision trees? (Complete Redundancy Detection)

VIII. Future Work

Ideally, a whitelist firewall's rule set would be constructed so that the most commonly applied matches are reached earlier rather than later. IPFWD accomplishes this task by artificial early stopping. Firewall rule reordering is a could be another solution to this problem. Given logs of common network traffic for the system, statistical analysis could show which rules are matched more often than others. With this information, the rule set could be reordered so that the most commonly applied rules are earliest.

IX. Conclusion

Appendix A
Additional Figures

Some text for the appendix.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.