NFShunt: a Linux firewall with OpenFlow-enabled hardware bypass

Simeon Miteff
University of the Witwatersrand,
Johannesburg, South Africa.
CSIR Meraka Institute,
Pretoria, South Africa.
Email: smiteff@csir.co.za

Scott Hazelhurst
University of the Witwatersrand,
Johannesburg, South Africa.
Email: scott.hazelhurst@wits.ac.za

***Abstract*—Data-intensive research computing requires the capability to transfer files over long distances at high throughput. Stateful firewalls introduce sufficient packet loss to prevent researchers from fully exploiting high bandwidth-delay network links. To work around this challenge, the Science DMZ design trades off stateful packet filtering capability for loss-free forwarding via an ordinary Ethernet switch [1]. We propose a novel extension to the Science DMZ design, which uses an SDN-based firewall. This paper introduces NFShunt, a firewall based on Linux's Netfilter combined with OpenFlow switching. Implemented as an OpenFlow 1.0 controller coupled to Netfilter's connection tracking, NFShunt allows the bypass-switching policy to be expressed as part of an iptables firewall rule-set. Our implementation is described in detail, and latency of the control-plane mechanism is reported. TCP throughput and packet loss is shown at various round-trip latencies, with comparisons to pure switching, as well as to a high-end Cisco firewall. The results support reported observations regarding firewall introduced packet-loss, and indicate that the SDN design of NFShunt is a viable approach to enhancing a traditional firewall to meet the performance needs of data-intensive researchers.**

I. Introduction

Gordon Bell argues that data-intensive computing is the basis for a new paradigm of science [2]. The idea of eScience is that data-exploration is a new scientific method that unifies theory, experimentation and simulation.

The Large Hadron Collider [3] (LHC) serves as an example of a scientific instrument and associated experiments with data-intensive infrastructure requirements. The distributed manner in which data produced at the LHC is analysed led to new architectures for network provisioning and security, for example the LHC Optical Private Network [4]. The distributed processing strategy itself was developed to cope with unprecedented volumes of scientific data, where network transfers were measured in tens of gigabits per second [5].

Early work to understand the network needs of the Square Kilometre Array [6] radio-telescope has identified a new large-data frontier, this time in the order of hundreds of gigabits per second [7]. Just as a novel approach was applied for the needs of the LHC, so will each successive data-intensive experiment need to find efficient ways to distribute, store and analyse data.

Gorton et al. list astronomy and cyber-security as applications that are both data and computationally intensive. Since high performance computing (HPC) facilities are themselves exposed to cyber-security threats [8], the need to apply cyber-security measures to HPC compounds the total complexity of data-intensive applications.

Network use-cases for eScience require fast data transfer: copying of large (sometimes Peta-byte-sized) data sets [2] between single endpoint systems, where maximum throughput (approaching 10Gbps) is required for single TCP sessions.

Firewalls are supposed to behave transparently (for legitimate traffic), but due to scalability constraints do not always in the special case of fast network flow [1]. When high latency transfers are attempted through such firewalls, the TCP protocol stack throttles a connection's window size because packet loss is interpreted as network congestion, and as a result, data throughput suffers.

If the critical role of high speed networks in data-intensive computing is at odds with deployed network security measures, then it is important to study this apparent conflict, and to propose solutions.

A. Science DMZ

ESNet propose the *Science DMZ* network design [1] (demilitarized zone) which introduces a small, fast subnetwork at the edge of the network at each institution (e.g., universities, research labs, etc.), devoid of middle-boxes (in other words, it is connected before the firewall). High latency transfers (originating from distant endpoints) are thus not subjected to the performance-degrading effect of packet loss caused by middle-boxes. Transfers from inside the institutional network (and thus beyond the border firewall) are low latency – therefore not subject to the same effect.

Figure 1 illustrates an example of this design: network traffic represented by the green arrow traverses the "clean" network, while the orange line represents local (low-latency) traffic.

It is argued that computing resources in the Science DMZ can be adequately protected from intruders by making use of simplistic (but scalable) router interface access control lists, combined with host security measures [1].

Unfortunately, Science DMZs are not without drawbacks. A firewall is described as an insurance policy for the manager or executive at an institution accountable for cyber-security [9].

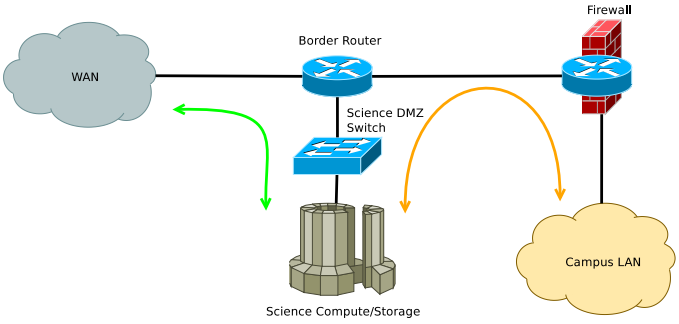

Fig. 1. Example science DMZ network diagram

It is understandable then, that some institutions are reluctant to adopt the simplified design of the science DMZ (which does not make use of a firewall in the usual sense).

Since no network security measure is absolutely effective, in the case of a breach, it could be difficult for the responsible officer to explain to the board (or relevant authority) why there was no firewall in place. Given this human bias against simplified filtering, making use of a firewall that delivers suitable performance for fast data transfer (if it exists) is perhaps the path of least (institutional) resistance.

B. Shunting

Another solution is to separate the network traffic belonging to data-intensive science applications from other flows, and only apply security measures to the remainder. This approach makes use of a custom hardware switch called a shunt [10], which is programmed to either bypass or forward traffic via an intrusion prevention system (IPS).

More recently, SciPass enhanced shunting to take advantage of OpenFlow switching [11]. While the original shunt work aimed to address the problem of IPS scalability, SciPass leverages bypass switching for the enhancement of Science DMZs. OpenFlow is a standard that implements the idea of Software Defined Networks (SDN) [12]. The control-plane functions of network elements can be centralised (to a so-called "controller") by providing a remotely programmable interface for the control-plane functions to manipulate forwarding-plane configuration.

C. Our approach

Previous work with OpenFlow in the fast data transfer context has applied it to the management of (stateless) access control lists [13], and the implementations of shunting focused on intrusion detection applications [10] [11]. Similarly, previous work to add hardware-offload acceleration for the Linux Netfilter firewall has relied on a custom kernel module communicating with specialised hardware such as an FPGA [14] or a Network Processor [15].

We research a shunting strategy for firewalls, which on the forwarding plane is similar to the Science DMZ, but the addition of a control-plane driven by a stateful firewall ruleset results in functionality very similar to a traditional firewall. The contribution of this paper is the hybrid shunting firewall design, a prototype implementation (NFShunt) and an analysis of the prototype's performance.

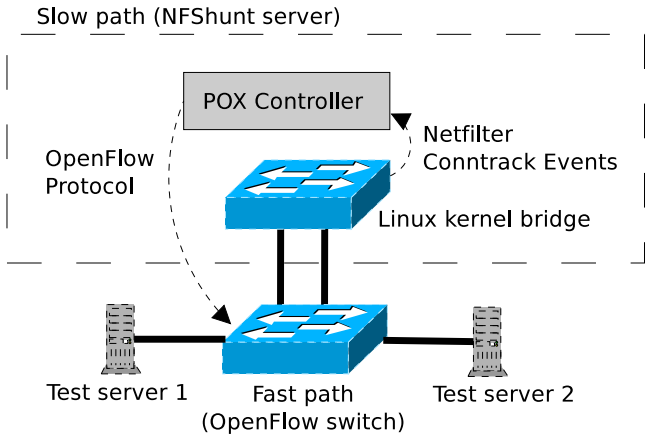

Fig. 2. NFShunt architecture

Our hybrid approach has an advantage of both the Science DMZ and stateful firewall solutions, in that it would eliminate some of the trade-off between security and performance, and be implementable in real-world applications by making use of existing, well-understood and tested off-the-shelf components. It is important to note that stateless bypass inevitably compromises on the ability to check the validity of packet headers against expected protocol state, as well as other (such as application-layer based) packet filtering capability, subsequent to shifting the traffic to the fast-path. This limitation common to our prototype hybrid firewall and similar designs, such as SciPass. We only consider a use-case involving single or small numbers of TCP connections. The research does not address a comprehensive threat-model for Science DMZs, cater for non-TCP applications, or examine connection-rate performance.

This paper is structured as follows: section 2 describes the prototype firewall's design and implementation; section 3 describes the method of evaluation we employed. Results are reported in section 4. We present our analysis in section 5, and then conclude and discuss future work in section 6.

II. Design and Implementation

NFShunt is implemented in Python based on the POX OpenFlow controller library[1]. The slow-path forwarding plane is a Linux 3.2.0 kernel bridge, while the fast-path (an OpenFlow Ethernet switch) is used for all external connections. The controller and the slow-path are co-located on the same physical server. The prototype is thus a layer 2 (transparent) firewall composed of two tightly coupled components, as illustrated in figure 2.

The prototype makes use of a Pica8 P-3290 top-of-rack Ethernet switch (based on a Broadcom switch ASIC and customised Open vSwitch as the OpenFlow agent). The switch is equipped with 48 1000Base-T and four 10GBase SFP+ ports. For the prototype firewall slow-path, we used a fit-PC3 Pro with four Intel 82574L-based 1000Base-T Ethernet NICs connected to the switch.

NFShunt's implementation is structured in five high level modules. The *controller core logic* triggers by-passing of flows

[1]Source code published at: https://github.com/simeonmiteff/NFShunt

based on input from the configuration and slow-path interface. The *slow-path interface* processes Netfilter connection tracking events and the *fast-path interface* communicates with the OpenFlow switch. The *configuration interface* adapts NFshunt to the instance-specific details of the network and firewall policy and the *logging interface* caters for troubleshooting and performance monitoring of the prototype.

The remainder this section describes manual configuration required for the prototype to function, as well as the design and implementation of the core logic, fast and slow-path interfaces. For brevity, we omit descriptions of the configuration and logging modules.

A. Fast-path configuration

Some basic configuration of the OpenFlow switch (the fast-path) is required for the shunting prototype. The first step is to create a bridge (virtual switch) with an associated controller. Since the prototype design is for all forwarding to be under the control of the slow-path, this bridge is then configured not to forward frames unless explicitly programmed to do so. Finally two slow-path ports and two fast-path ports are added into the bridge. With this configuration the switch will attempt periodically to establish a connection to the shunting controller, and when it does, it will receive flow configuration to switch frames between the fast-path ports, via the slow-path.

B. Slow-path configuration

The operation of the prototype requires specific configuration of the Linux Netfilter firewall (via the iptables tool). These configurations ensure that information about TCP flows present in the Linux kernel (due to the fact that packets associated with those TCP flows are seen by the slow-path in the kernel) are made available to the shunting controller. The controller communicates with Netfilter (the kernel-space firewall implementation) via the user-space interface (netlink) of Netfilter's connection-tracking module (conntrack).

Since the system administrator is expected to configure the default slow-path firewall policy as a set of Netfilter rules, NFShunt was designed to integrate the expression of shunting policy into the same rule-set. This is achieved through the use of Netfilter's packet mark and connection mark modules. The mark extensions define both matching extensions and rule targets. The packet mark extension permits the administrator to assign any 32-bit value to a special field in the data structure associated with each packet processed by the Linux kernel (using the mark target). This value can be read by other rules using the packet match extension, and can also be written and read using the connection mark extension, which enables marking of flows identified by the conntrack module.

The prototype uses the 16 most significant bits of the mark value to store the required shunting action, as well as flags and information about the flow's ingress and egress information. This allows the administrator to use the 16 least significant bits for other purposes. Three configurable values are defined to express the various supported shunting actions: ignore (do nothing to this flow), shunt (bypass the flow in the fast-path), and block (install a rule to drop traffic matching this flow in the fast-path).

The logical flow of matching and actions applied to packets in the slow-path is shown in figure 3

The prototype uses a dedicated chain containing shunting policy rules. The following example sets a value of 1 (defined in the default controller configuration to trigger a shunt) when a TCP flow is matched with destination port 5000:

```
iptables -t mangle -A NFSHUNT_POLICY
    -p tcp --dport 5000 -m conntrack
    --ctstate RELATED,ESTABLISHED -j MARK
    --set-xmark 0x10010000/0x100f0000
```

C. Slow-path interface

The prototype's slow-path makes use of the Linux kernel's built in Ethernet bridging function combined with Netfilter (the standard Linux firewall). Interaction with conntrack is via the "conntrack" user-space utility. The shunting controller starts an instance of "conntrack" with the "–E" parameter inside a thread, and then reads a sequence of connection tracking events from the standard output stream, in XML format.

Connection tracking events describe the creation, destruction or modification of connection tracking objects in the kernel. When an event includes the mark attribute (obtained from the iptables connection mark), it is examined in further detail.

The flags, flow mark and information about the slow-path's physical input and output interfaces is used by the controller's core logic to make shunting decisions. In addition to monitoring connection tracking events, the controller core must also be able to delete connection tracking objects for flows that have been shunted or blocked. This is achieved by executing the "conntrack" utility with the "–D" parameter and specifying the 5-tuple describing the TCP flow.

D. Fast-path interface

Interaction with the fast-path is via the POX OpenFlow controller framework. The core logic of the controller calls POX functions directly to add flow-entries for shunting and blocking.

E. Controller core logic

The core of the shunting controller receives events from the slow-path module that have been validated to contain the meta-data added by the Netfilter iptables rules configured for the functioning of the prototype. If the action indicated by the flow mark is to shunt, the controller will add two flow specifications (one for each direction of the flow) via the fast-path interface to by-pass the slow-path. These flow specifications include layer 2, 3 and 4 header match fields, specifies output via the corresponding fast-path ports, and are configured to time-out after a configured amount of idle time. For blocking actions, the only difference in the specification is that the output action is omitted (which leads to an implicit packet drop).

The state of the fast-path therefore transitions between shunting or blocking, and forwarding packets via the slow-path. Figure 4 illustrates how flow rule programming and time-out transition between the two states on a per-connection basis.

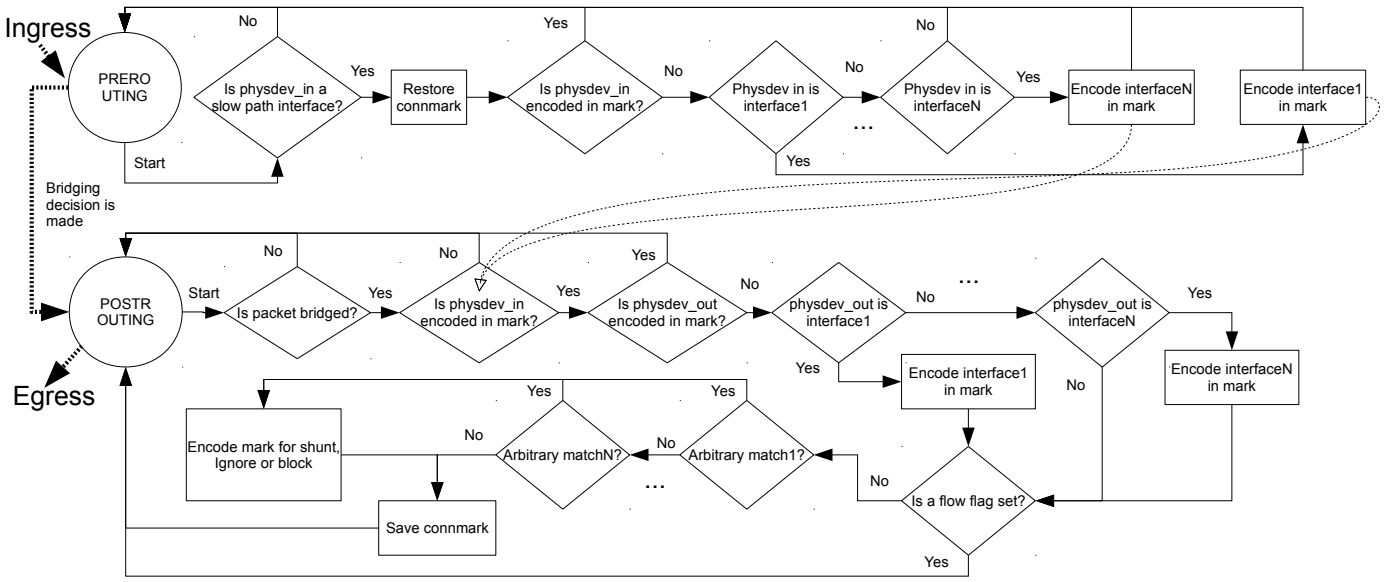

Fig. 3. NFShunt iptables flow diagram

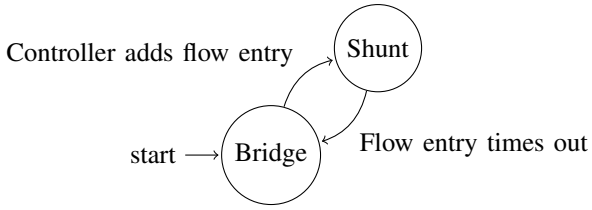

Fig. 4. Per-connection state machine of the forwarding path

If a TCP connection stalls for longer than the idle-timeout of the flow rules implementing the shunt, and then resumes, subsequent packets will re-appear on the slow-path. At this point, it is necessary for the controller to re-install the rules, but the state of the connection tracking will not necessarily change in a manner that would trigger an event visible via the slow-path interface. Deleting the entry immediately after installing a shunt or block forces the slow-path to re-create the entry, which is guaranteed to be visible to the controller. This approach also prevents stateful Netfilter tracking of TCP windows (disabled by default) from causing the kernel to terminate resumed connections.

III. Experimental methodology

We designed experiments to validate the prototype firewall by comparing the prototype's TCP throughput performance to that of a high speed firewall employing a traditional design. Since testing is focused on TCP performance, replaying of captured traffic or network layer simulation are not suitable for test traffic generation. We therefore chose application-layer load generation, which indirectly produces a network test traffic. Traffic generation (including synthetic delay) relied on two Linux servers, each equipped with Intel 82599-based dual-port 10GBase NICs.

We used a Cisco ASA 5585 firewall for comparison to the prototype firewall. Neither firewall was configured with policies intended to simulate real-world deployments. This was a deliberate choice to evaluate best-case performance on both devices. With advice from Cisco technical support engineers, some changes were made to the factory default configuration of the Cisco: ASA firewall software was upgraded to version 8.47, the firewalling mode was set to transparent, interface MTUs were increased to 9216 bytes, "jumbo-frame reservation" was enabled and TCP MSS clamping was disabled.

A. Lab setup

Four test configurations were used for experiments: the first connected the test servers directly, and served to establish the best-case throughput achievable using the experimental hardware. Our experiment required endpoints with sufficient CPU capacity as well as peripheral bus and memory bus bandwidth to support near-line rate TCP transfers. During these test runs, TCP/IP stack tuning was done to optimise TCP throughput with, or without synthetic delay. The second inserted the Cisco firewall in-line between the test servers via one set of 10Gbase-Ethernet NIC interfaces. The third configuration connected the Pica8 switch via another set of 10Gbase-Ethernet NIC interfaces, and the switch was configured for direct forwarding of frames between the test servers without the slow-path interface (no shunting mechanism). Finally, the fourth configuration allowed the Pica8 switch to be controlled by NFShunt.

The second, third and fourth configurations were used for performance evaluation as follows:

B. Factors and levels

Round trip delay is an important factor in TCP performance. Many science applications require long distance transport. For example, the planned Square Kilometre Array will require data transport from South Africa and Australia to the Northern hemisphere. For our experiments, synthetic delay values were chosen to represent typical round-trip times for intra-African (100ms), European-South African (200ms) and North American/Asian-South African (400ms) connections.

These are realistic for the distribution of data from the Square Kilometer Array MID instrument in South Africa. Factors (and their respective levels) tested during experiment runs were:

- **Selected middle-box:** three configurations exist to allow the performance comparison objective of the experiment: direct switch, the prototype firewall (NF-Shunt) and the traditional firewall (Cisco ASA 5585).
- **Synthetic delay:** since the performance reducing effect of packet loss is dramatic in the presence of delay, it is varied (no delay, long-distance network delays of 100ms, 200ms and 400ms round-trip-time) to observe this effect in the test configurations. We used the Linux netem [16] queue discipline to emulate half of the transmit delay on each of the two test servers.

C. Measurements

The following observable performance characteristics were measured in each experiment run:

- **Absolute TCP throughput**: individual as well as aggregate TCP throughput was measured using the iperf3 utility [17].
- **Dropped packets**: packets dropped between the test endpoints were measured by comparing NIC frame counters to the frame counters of the device under test. This also allowed location of the cause of the loss.
- **TCP stack behaviour**: a Web100 [18] instrumented Linux kernels were used, and a custom application logged snapshots of the Web100 variables associated with the test connection every 100ms, to allowed analysis of the TCP stack behaviour during transfers.

The above measurements were annotated with events relevant to the shunting behaviour of the prototype firewall, in particular, the point in time when a shunting instruction is sent, and the last packet switched on the slow-path.

IV. Performance evaluation

A. Validation of test procedure

Performance tuning was performed on the end-hosts with direct network switching configured via the OpenFlow switch — in order to validate the test procedure prior to the experiments. Host TCP/IP stack tuning followed conservative guidelines applicable for modern Linux kernels (optimised to allow for round-trip delays greater than 400ms at 10 $\mathrm{Gbit\,s^{-1}}$ link capacities). Parameters used are listed in Table I. No hardware or driver-specific parameters were changed — as, this would deviate from typical data transfer scenario the experiments were intended to simulate.

TABLE I. Host tuning for test servers

Tuning parameter	Value
NIC transmit queue (txqueuelen)	10,000 packets
TCP socket buffer size auto-tuning maximum (tcp_wmem/tcp_rmem)	500 MB
Network interface MTU	9,000 B

TABLE II. Shunting event performance

Event description	Time from SYN (ms)	Standard deviation (ms)
Controller detected flow mark	3.3	0.30
End flow programming	58.0	0.69
Last packet slow-switched	75.8	0.04

The netem Linux QoS module was configured on the respective network interfaces to introduce fixed transmit delays, which together amounted to the total desired synthetic round-trip delay for each experimental run. It was also necessary to adjust the default QoS buffer of 1,000 packets to 100,000 to prevent packets being dropped in the kernel transmit path.

With this configuration the testbed was capable of consistent TCP transfers at throughputs in excess of 9 $\mathrm{Gbit\,s^{-1}}$ for 0, 100 and 200ms RTT (included as the *direct* series of results in the next section).

B. Network performance results

1) Shunting mechanism: The shunting mechanism of the prototype was profiled to determine how quickly flows can be shunted. The timing of events was recorded for shunted TCP connections (measured in seconds since the SYN packet establishing the connection). Table II summarises the results of 100 tests for each combination of factors.

2) Forwarding performance: The results of the single-connection TCP performance tests are summarised in Table III. Throughput as measured by the *iperf3* utility is reported, while the packet loss rate is calculated by comparing transmit and receive Ethernet NIC MAC frame counters on the testing servers.

No packets were dropped in the direct tests, confirming that the OpenFlow switch is capable of line-rate Ethernet switching. Both firewalls were found to be receiving but not forwarding all frames. While our 100ms samples of Web100 metrics were not used for generating experimental results, they proved invaluable for troubleshooting tuning problems during validation of the experimental test setup.

TABLE III. Single flow forwarding performance

RTT	Test	Mean data rate $\mathrm{Gbit\,s^{-1}}$	Mean packet loss %
None	Direct switching	9.944	–
	Shunting	9.923	0.00
	Cisco ASA5585	9.838	0.05
100 ms	Direct switching	9.678	–
	Shunting	9.690	–
	Cisco ASA5585	5.337	0.10
200 ms	Direct switching	9.332	–
	Shunting	9.334	–
	Cisco ASA5585	4.197	0.21
400 ms	Direct switching	5.978	–
	Shunting	5.957	–
	Cisco ASA5585	3.094	0.21

Throughput tests on the Cisco firewall were repeated with two and four simultaneous connections, in order to study the effects of TCP window scaling and internal load-balancing beyond a single flow. These results are shown in Table IV.

TABLE IV. CISCO – MULTIPLE FLOW FORWARDING PERFORMANCE – MEAN DATA RATE $\mathrm{Gbit\,s^{-1}}$

Flows	**No delay**	**100 ms RTT**	**200 ms RTT**
One	9.838	5.337	4.197
Two	9.975	9.471	8.724
Four	10.011	9.866	9.430

C. Network performance comparison

We tested the hypothesis that the direct switching performance differs from the prototype's performance, and that the prototype's performance is different to the Cisco firewall's performance. The Wilcoxon rank-sum test was applied to mean throughput and packet loss measurements of the respective devices at different RTT values. We interpret that tests with $p > 0.05$ to indicate no significant difference while those with $p < 0.05$ do. Table V and Table VI summarise the results.

TABLE V. TESTS OF THE HYPOTHESIS THAT DIRECT SWITCHING AND PROTOTYPE PERFORMANCE DIFFER

RTT	Throughput	Packet loss
None	significant difference of 0.021 $\mathrm{Gbit\,s^{-1}}$	no significant difference
100 ms	no significant difference	identical (no loss)
200 ms	no significant difference	identical (no loss)
400 ms	no significant difference	identical (no loss)

TABLE VI. TESTS OF THE HYPOTHESIS THAT PROTOTYPE AND CISCO ASA 5585 PERFORMANCE DIFFER

RTT	Throughput	Packet loss
None	significant difference of 0.084 $\mathrm{Gbit\,s^{-1}}$	significant difference of 0.051%
100 ms	significant difference of 4.353 $\mathrm{Gbit\,s^{-1}}$	significant difference of 0.096%
200 ms	significant difference of 5.137 $\mathrm{Gbit\,s^{-1}}$	significant difference of 0.207%
400 ms	significant difference of 2.863 $\mathrm{Gbit\,s^{-1}}$	significant difference of 0.209%

V. ANALYSIS

A. Technical implementation analysis

Controlling a shunting mechanism using the Netfilter firewall rule-set is feasible, but we identified some shortcomings of this approach. The use of connection marking to specify actions makes for a complicated iptables rule-set, and imposes some structure that may require existing firewall policy to be re-written by the administrator. Integrating the POX controller with connection tracking via the user-space Netlink libraries proved difficult due poor Python bindings for the Netlink connection tracking protocol.

The use of OpenFlow for the fast-path interface greatly simplified implementation of the prototype. While only OpenFlow 1.0 capability is required for the trivial actions of selecting output ports based on 4-tuple matches, this would have been difficult or impossible without OpenFlow. The prototype was also tested successfully in the Open vSwitch-based Mininet environment [19], demonstrating that it compatible with multiple OpenFlow dataplanes.

The switch used for the prototype lacked support for the TCP flags match type. This prevented detection of connection tear-down in the data-plane and required the use of idle-timeouts. The disadvantage of idle-timeouts is that stalled connections will be routed via the slow-path briefly if and when they resume.

B. Network performance analysis

The delay of the slow-path interface (detecting new connections as tracked by the kernel) is small. Sending shunting flows to the switch makes up the majority of the delay between the first and the last packet being slow-switched.

The Pica8 OpenFlow switch used for our experiments was able to forward all connections without packet loss. Despite the host tuning performed, a notable TCP performance drop-off was observed at 400ms. This suggested that TCP throughput would not be the best measurement of firewall performance in our experiments, therefore we ensured that packet loss could be accurately measured and located to specific network elements.

When there was no synthetic delay, the Cisco ASA consistently dropped a small percentage of packets, but was able to firewall a single connection at nearly the maximum speed achievable with the test setup (near 10Gbps). From this we infer that the Cisco is not relying on load-balancing multiple connections over multiple cores in the forwarding plane to reach 10Gbps throughput. Transferring data over simultaneous TCP connections was effective at overcoming the high latency TCP slow-down due to loss introduced by the firewall, and thus utilising nearly the full link bandwidth at moderate RTT.

A very small amount of packet loss was observed in the tests of the prototype with no synthetic delay. While this is not a statistically significant difference from the direct switching tests, a 21Mbps difference in throughput was found to be statistically significant. At 100, 200 and 400ms, there was no difference between the measurements of the prototype and direct switching. These results are consistent with a model where the slow-path phase of the connection has little effect on performance (with TCP, this phase covers the connection setup, and at worst the slow-start phase of data transfer).

Under load, the difference in packet loss, combined with moderate RTT, results in a significant degradation of throughput performance of the tested Cisco firewall model to other approaches (such as our NFShunt prototype or that of a Science DMZ).

C. Price-performance analysis

Cost of infrastructure is important as high-throughput science expands from the preserve of the few to that of the many. The capital cost of the Cisco ASA5585-X SSP-60 firewall used for testing varies between \$ 82 000[2] and \$ 167 000, depending on configuration, licensing and Cisco partner discounts applied. Since this particular firewall's features and capabilities far exceed the the requirements of our tested scenario, a smaller configuration was chosen for the purpose of pricing comparison with the prototype. The Cisco ASA5580-20 configured with two 10G interfaces matches the prototype firewall more closely, and is estimated to cost \$ 59 000. The FitPC and Pica8 cost \$ 1000 and \$ 3000 respectively, coming to a total of \$ 4000 for the prototype.

[2]From ZAR prices at 12.15 to the USD and rounded to the nearest 1k.

We expect the additional complexity of the NFShunt prototype to increase the cost of operations and maintenance relative to traditional firewalls. It is hoped that production implementations of NFShunt will allow the operational costs to be quantified and compared, but this is not within the scope of our research.

VI. Conclusion and future work

Our test of the Cisco firewall suggests that there is merit in engineering networks for loss-free paths to serve the narrow use-cases addressed by Science DMZs (single or small numbers of high bandwidth-delay product connections between research infrastructures). We can, however, not generalise this to all traditional firewalls – indeed, with advancing technology it is possible that *some* firewalls are, or will be capable of stateful packet filtering of single flows at very high speeds, without introducing packet loss.

We conclude that, as an interim measure or an alternative to static separation of end-points into classes protected by stateful and stateless (ACL) packet filters, it is possible to build a hybrid firewall that off-loads trusted connections to stateless hardware switching. Our prototype demonstrates the feasibility of an SDN-based design making use of widely used Free/Open Source software for the stateful slow-path, and vendor-agnostic OpenFlow forwarding for the fast-path. Our experiments verify that the performance of dynamically off-loaded TCP connections benefit from Science DMZ-like loss-free forwarding.

Analysing the capital costs of implementing our prototype shows potential for substantial saving over the cost of traditional firewalls with similar performance characteristics (by our estimate, at least an order of magnitude less). The operational benefits of composing OpenFlow switching with Linux's Netfilter are difficult to quantify and may be outweighed by increased complexity. Due to network latency, the science use-case requires loss-free forwarding in countries that are geographically distant from collaborators in Europe and North America. This work is particularly important due to cost-constraints where those countries are also developing or newly industrialised nations.

Finally, the parallel-flow performance results via a non-ideal network path suggests that improving TCP, or adopting better file transfer tools, should be seriously considered as an alternative to building loss-free networks.

Our focus on an SDN-based hybrid firewall implementation makes assumptions about the information security threat model and application design. A study focused instead on data-intensive research network use-cases that examine real applications, and considers a comprehensive threat model is called for.

Future work could apply the shunting approach to routing (as opposed to transparent/bump-in-the-wire) firewalls. Early work on chaining NFShunt into the forwarding path of larger SDN-enabled systems such as Vandervecken [20] (a fork of RouteFlow [21]) shows promise. Support in open standards for tracking transport layer flow states in the forwarding plane (to take advantage of such capability in hardware) could enhance NFShunt-like designs.

References

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 85:1–85:10. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503245

[2] A. Hey, S. Tansley, and K. Tolle, *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research Redmond, WA, 2009.

[3] CERN, the European Organization for Nuclear Research, "Large hadron collider," http://lhc.web.cern.ch/lhc/, accessed: 26/02/2013.

[4] E.-J. Bos, E. Martelli, P. Moroni, and D. Foster, "LHC tier-0 to tier-1 high-level network architecture," CERN, Tech. Rep, Tech. Rep., 2005.

[5] J. Shiers, "The Worldwide LHC Computing Grid (worldwide LCG)," *Computer Physics Communications*, vol. 177, no. 1, pp. 219 – 223, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S001046550700077X

[6] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. Lazio, "The square kilometre array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1496, 2009.

[7] W. Johnston and R. McCool, "The square kilometer array a next generation scientific instrument and its implications for networks," *TERENA Networking Conference*, 2012.

[8] I. Gorton, P. Greenfield, A. Szalay, and R. Williams, "Data-intensive computing in the 21st century," *Computer*, vol. 41, no. 4, pp. 30–32, 2008.

[9] E. Dart, "Science DMZ security," in *Joint Techs, Winter 2013, Honolulu, Hawaii*, 2013.

[10] J. M. Gonzalez, V. Paxson, and N. Weaver, "Shunting: a hardware/-software architecture for flexible, high-performance network intrusion prevention," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 139–149.

[11] E. Balas and A. Ragusa, "Scipass: a 100gbps capable secure science dmz using openflow and bro," in *Supercomputing 2014 conference (SC14)*, 2014.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[13] S. Russell, "Thimble," in *Joint Techs, Winter 2013, Honolulu, Hawaii*, 2013.

[14] M.-S. Chen, M.-Y. Liao, P.-W. Tsai, M.-Y. Luo, C.-S. Yang, and C. E. Yeh, "Using netfpga to offload linux netfilter firewall," in *2nd North American NetFPGA Developers Workshop*, 2010.

[15] K. Accardi, T. Bock, F. Hady, and J. Krueger, "Network processor acceleration for a linux* netfilter firewall," in *Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*. ACM, 2005, pp. 115–123.

[16] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*. Citeseer, 2005, pp. 18–23.

[17] Energy Sciences Network, "Iperf3," http://software.es.net/iperf/, accessed: 11/03/2015.

[18] M. Mathis, J. Heffner, and R. Reddy, "Web100: extended tcp instrumentation for research, education and diagnosis," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 69–79, 2003.

[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[20] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. N. A. Corrêa, S. C. de Lucena, M. R. Salvador, and T. D. Nadeau, "When open source meets network control planes," *IEEE Computer*, vol. 47, no. 11, pp. 46–54, 2014. [Online]. Available: http://dx.doi.org/10.1109/MC.2014.340

[21] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in *Proceedings of the 6th International Conference on Future Internet Technologies*. ACM, 2011, pp. 34–37.