# Viable Protection of High-Performance Networks through Hardware/Software Co-Design

Johanna Amann
ICSI / Corelight
johanna@icir.org

Robin Sommer
ICSI / Corelight / LBNL
robin@icir.org

## ABSTRACT

Commercial as well as scientific networks have increased tremendously in speed over the last years. However, in today's world, such fast networks also expose sites to incessant network attacks like theft of information, parasitic resource consumption, or suffering from (or participating in) denial-of-service (DOS) attacks. Some of the most powerful networks today remain particularly hard to defend: the 100G environments and backbones that facilitate modern data-intensive sciences (physics, medicine, climate research, etc.) prove very sensitive to the slightest disturbances. We offer a security solution for these environments, which cannot use turn-key classical solutions like firewalls or intrusion detection systems, as they cannot operate reliably at the necessary speeds. We present the design of a novel, comprehensive framework that aims to integrate software and hardware to enable economical protection of critical high-performance networks. We also argue for taking a more holistic viewpoint of the capabilities and problems that can be solved by software defined networks.

## 1. INTRODUCTION

Network intrusion detection systems like Bro [7], Suricata [34], and Snort [28] are essential tools for securing networks against an incessant flood of attacks from the Internet. However, with the enormous bandwidths available today in commercial and, especially, scientific networks, network monitoring is struggling to keep up. This is particularly true for the 100G environments and backbones that facilitate modern data-intensive sciences—physics, astronomy, medicine, climate research, etc. For such high-speed networks, classic inline security solutions, such as firewalls and intrusion prevention systems, remain infeasible options, as they cannot operate reliably at the necessary speeds [14]. Advanced network architectures, like ESnet's Science DMZ [10], thus advocate passive monitoring solutions.

With increasing network loads, today such passive systems traditionally have to resort to load distribution across more and more CPUs, and potentially machines [35]; sometimes in combination with instructing network hardware to stop forwarding high-traffic flows altogether (*shunting*; [15, 37, 3, 33]). Recently, however, the rise of programmable networking hardware has opened up the opportunity to develop novel protection mechanisms that offer increased capacity, with much less resources, by tailoring hardware support specifically to the security domain. While in the past such an approach would have been prohibitively expensive, switches and NICs are now available at reasonable price points that are either directly programmable—for example in P4 [4], or POF [32]—or for which at least the underlying platform offers the vendor the flexibility to add new capabilities without much effort (e.g., RMT [5], XPliant [9], and FlexPipe [17]).

In this work, we present the design of a comprehensive framework integrating software and hardware to facilitate economical protection of high-speed networks—more effectively and at lower cost than solutions that are available today. At a more fundamental level, our effort showcases how a specific application domain can benefit from the capabilities that today's hardware offers through careful design of appropriate APIs and feature sets for devices to support. We argue that, from such a more holistic perspective, *software defined networking* does not need to remain limited to just the small set of basic use-cases that today's OpenFlow happens to support. It instead offers the potential to enable a wealth of new powerful applications, even if they exhibit quite specific needs.

For the security domain, we identify four areas in which network hardware can significantly lower the load that is placed on network monitoring systems: *(i) Statistical measurements* are essential to many security analyses and are used, for example, to detect port scans. *(ii) Connection offloading* allows network monitoring systems to offload the handling of non-established connections into hardware. *(iii)* Hardware *TCP reassembly* reduces the connection-specific state of network monitoring systems. Finally, *(iv) reliable pattern matching* in hardware facilitates, e.g., protocol detection that allows network monitoring systems to classify traffic early on in the processing pipeline. As implementing any of these requires effective communication between networking equipment and the network monitoring system, we also envision a set of open protocol extensions that piggy-back the necessary information on top of regular networking traffic.

The rest of this paper is structured as follows: In §2 we outline the hardware design necessary to realize the outlined features. §3 discusses hardware control and feedback including the software component, and §4 reviews the design of



*SDN-NFV Sec'17, March 22-24 2017, Scottsdale, AZ, USA*



ISBN 978-1-4503-4908-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3040992.3041003

our architecture. Finally, §5 presents related work before we conclude in §6.

# 2. HARDWARE SUPPORT

In this section, we present four different areas in which network hardware can be used to increase the effectivity of network security monitoring: §2.1 presents statistical measurements, §2.2 outlines connection offloading, §2.3 sketches TCP reassembly, and §2.4 discusses pattern matching & protocol detection.

## 2.1 Statistical Measurements

Statistical summaries provide a key tool for profiling and protecting operational networks. Network operators routinely measure properties like throughput, connection sizes, port distribution, and host fan-out. By applying thresholds to these measurements, they can drive statistical anomaly detection, including reporting network scans. While conceptually simple, computing these measures imposes substantial load on software implementations, particularly in load-balanced settings where counters must remain synchronized across multiple independent monitoring systems [1, 35].

Moving these statistical measurements directly into networking hardware provides key benefits. In addition to offloading the monitor, the network-level vantage point has two additional advantages. First, it provides network operators with *ground truth* on what is actually happening in the network; effects on the monitoring path, such as packet drops due to CPU congestion, will not affect the results. Second, thresholds can trigger immediately on the hardware, with much less latency than a load-balanced setting can provide.

We envision hardware measurements to be implemented by letting users specify a mask over a combination of OpenFlow fields, while allowing users to specify aggregations. Typically, fields like IP addresses, transport-layer ports, TCP flags or switch input ports are useful for counting. For each of these components, a user should be able to select to either match against it, which will let the device consider only packets with a corresponding value; or mark them as specifying a grouping defining the measurement unit. (Wildcards are possible for all elements.) Figure 1 shows a simplified example with two rules. The example port 80 packet matches the second rule, which groups by source IP and destination port.

We aim to track three pieces of information for each of the groups. The first two pieces are the packet count and the byte count. The third piece of information tracks the type of packets that were received; for TCP, this would e.g. track that a SYN was followed by a SYN-ACK, followed by data, followed by FIN. This mimics a feature of the Bro network monitor, the *connection history*, which regularly proves highly valuable to have available for incident analysis. Once a rule becomes active, the device will report back the current register values in configurable intervals by inserting a custom reporting header (see §3) into the packet stream. Optionally, a rule can also define thresholds for the two numerical registers. When crossed, the device will insert a custom header into the affected packet to immediately notify the network monitoring system.

## 2.2 Offloading Non-established connections

Network monitoring systems normally track the state of all flows on the network—a resource-intensive task that requires computation and memory updates for every single packet. Upon closer examination, however, it turns out that one could skip tracking a substantial subset without loosing much information.

In today's network traffic a surprisingly high number of TCP connections are never fully established: they consist only of SYN packets to nonexisting hosts or services. This is typically due to scanning activity. To give a current example, *more than 50%* of the 160-180 million inbound connection attempts per day at the Lawrence Berkeley National Laboratory never see any response. By handling all these connections fully in hardware, only sending summaries to the network monitor for logging purposes, network monitoring systems would not need to track *any* state for them.

This approach can be implemented in hardware by allocating a small, fixed-sized memory record per TCP flow, just enough to store the key information from a SYN packet (including timestamp, IP addresses and ports, and the initial sequence number). When a connection's originator sends the first SYN packet, the device records its information, but does not yet forward the packet to the monitor. If the device sees a responder acknowledging that SYN, it first regenerates the originator's initial packet from the stored record before forwarding the response. For connections that are never fully established, the device eventually expires the record and then forwards a specially marked, regenerated SYN packet that the network monitor can identify as such. The system can then directly log it without going through its state management logic.

Such an approach has already been implemented in software by Dreger et al. [13]. There, however, the benefit remained too small in practice to provide much value, as the packets were still reaching the system. Implementing the scheme in hardware promises a much larger pay-off—it essentially provides "shunting for network scans".

While implementing this approach on its own would make detection of port and address scans more difficult, this can be addressed by combining it with the hardware-accelerated statistics we present in §2.1.

## 2.3 TCP Stream Reassembly

As discussed above, tracking connection state remains an expensive operation for network monitoring systems. We argue that today's powerful network hardware should make it possible to perform full TCP-layer connection handling in hardware, including all TCP stream reassembly. The goal is to send a well-defined, in-order data stream to the network monitor, freeing up CPU and memory resources by skipping reassembly and buffering. In addition, TCP reassembly in hardware enables offloading further tasks to the device that depend on reassembly as a preprocessing step, such as reliable pattern matching and dynamic port-independent protocol detection; we discuss this in §2.4.

To implement TCP stream reassembly in hardware, three main components have to be managed by the hardware: *(i)* a session table indexed by addresses and ports, *(ii)* TCP state machines, and *(iii)* buffers of reordered packets. The reassembled TCP stream can be passed on to network monitoring software as a normalized flow of TCP packets, now in the right order. Flows can be augmented with a unique flow ID through a custom header (see §3) to make it easier for the software to identify packet streams.

An important challenge with hardware reassembly concerns running out of buffer space—a situation no implementation

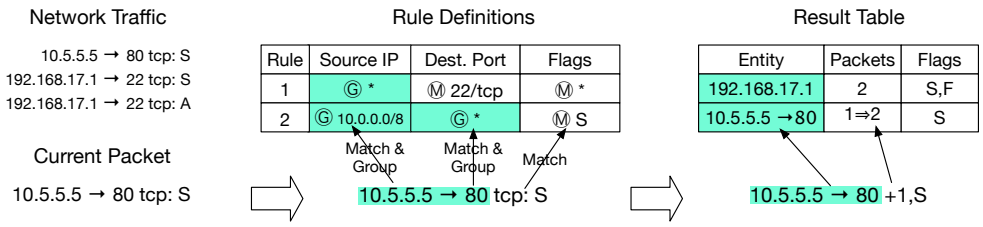


**Figure 1: Hardware based statistical summaries. Rule definitions marked with Ⓖ define the grouping. Rule definitions marked with Ⓜ apply only for matching. Rule 1 counts packets with a destination port of 22 grouped by source IP. Rule 2 counts SYN packets originating from each 10.x.x.x address, grouped by source and destination port. Here, the current packet matches Rule 2 and increases the corresponding packet counter.**

will be able to avoid given the limited size of high-speed memory in networking devices. In [11], Dharmapurikar et al. present a design for in-hardware reassembly that addresses this. First, they show that for most traffic, memory usage is not a concern due to their generally low level of reordering. For the relatively few uncommon cases where it is, the author's *inline* system can take active remedies, making it robust even against adversaries. While a passive monitor does not have the option to manipulate the traffic, we believe that Dharmapurikar et al's approach can generally work well in that setting, too, if combined with an alternative fallback mechanism: reverting to in-software reassembly if a flow exceeds the hardware's capabilities to reassemble it, first passing any already buffered data on to the monitor. This approach allows to keep most of the TCP load away from the network monitor while addressing the hardware's limitations.

## 2.4 Reliable Pattern Matching

Pattern matching is a task that hardware can perform extremely efficiently. Yet, outsourcing pattern matching remains a problem for network monitors, as correct results depend critically on matching against already reassembled payload streams—a capability that NICs and switches do not offer today. The work we discuss in §2.3 will provide that missing piece: once a device can reassemble, adding efficient pattern matching on top will be straight-forward leveraging established technology.

In turn, having pattern matching will then facilitate further advanced tasks in hardware as well. As one intriguing example, it makes it possible to move the first stage of port-independent protocol detection [12] into hardware devices. This frees up a significant load on the software side. In addition, it enables the possibility of using negative matching results: as a monitor cannot further parse a connection if it does not recognize its protocol, we can automatically stop forwarding data flows that do not match any of the prefilters—and hence would be ignored anyways. In other words, the device will now filter out directly all traffic that does not contribute anything to the monitoring results.

# 3. HARD-/SOFTWARE INTERFACES

To implement the schemes we discuss above, network monitoring systems need to interact with network hardware in two ways: the software needs to *(i)* control the device, and *(ii)* receive back results from offloaded processing. To remain independent of any specific type of device for either type of communication, we can design standardized APIs for these exchanges, per the following.

## 3.1 Control & Feedback

OpenFlow is already supported by a multitude of hardware devices; this even includes programmable NICs [25]. While the standard limits the operations available by default, additional capabilities can be expressed through a set of custom extensions that devices can implement to support our new functionality.

While exercising control has become common, reporting results back to software-land remains underappreciated in the typical SDN paradigm, leaving standard protocols and APIs ill-suited to this use case (e.g., OpenFlow limits feedback to reporting number of packets and bytes per flow, and even leaves the implementation of that feature optional).

We thus envision creating a new reporting mechanism that uses a dedicated packet header format carrying signaling information to an attached network monitor, akin to how IEEE 802.11 operates for the Radiotap [27] header. Devices can then piggy-back information on top of existing packets by inserting the new header in between their Ethernet and IP layers. If information is not directly associated with a certain packet, devices can create otherwise empty packets. In either case, networking monitoring systems can recognize and parse these headers while inspecting the network traffic.

This new header should consist of a binary tagged format, with the tags signaling different types of information, such as packet counts for a flow or the data volume seen for an IP address. As a side benefit, this also opens up the possibility of leveraging this header to standardize transmission of high-precision packet timestamps—a feature that many switches already support, yet today implement using vendor- or even device-specific formats.

## 3.2 Software design

On the software side we aim for a design that is as transparent to the operator as possible. The standardized tagged framing format can automatically be recognized by network monitoring systems without requiring any more configuration. Enabling new hardware features will require just a slight amount of configuration. We envision the user only having to provide the IP address, as well as authentication credentials, to the hardware that implements the new primitives. OpenFlow can be used for autodiscovery of the exact featureset that is supported by the hardware.

To enable a transparent user experience even if hardware may support functionality only partially in a specific setup, network monitoring systems can fall back to software as necessary. Good examples are the summary statistics framework [1] as well as the SDN framework [2] of Bro. Both of

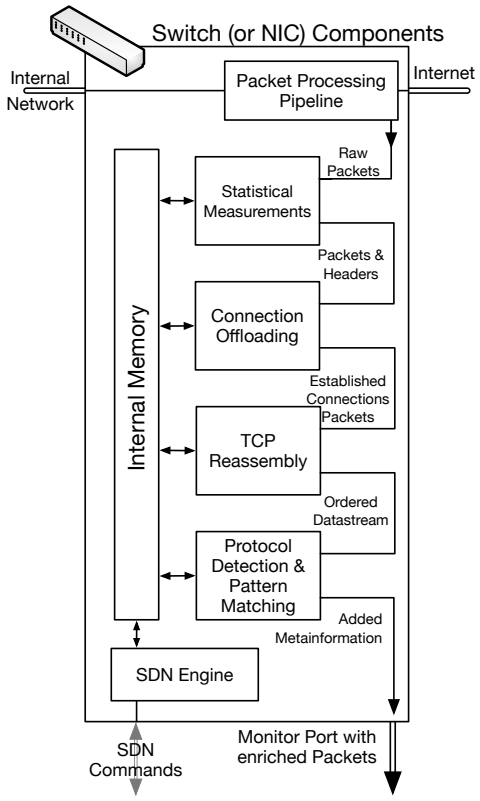


**Figure 2: Internal Switch/NIC Architecture**

them provide high-level APIs that let the user describe the outcome of what they want to achieve (e.g., count connections to port $X$ per minute grouped by IP). This kind of task-oriented design enables the software to automatically push rules that can be handled by the hardware out to it, while doing more complex manipulations in software. An example would be to do counting of port distributions in hardware; however, if a query requires protocol parsing (like, e.g., count all hosts using a certain certificate), it will be handled in software.

# 4. DESIGN

Figure 2 shows a high-level overview of a design integrating all our components. It can be implemented either inside of a network interface card, or inside of a switch. If implemented inside a switch, only traffic to specified output ports gets enriched with additional metadata; configuration should be similar to configuring a standard monitor port. Inside the hardware, packets go through a set of processing pipelines.

Statistical measurements need access to the raw, unmodified data packets (for exact packet counts), and are performed as a first step. During these measurements, custom headers (see §3) are attached to packets to show current counter values and notify the network of crossed thresholds. In the next step, TCP connection tracking is performed in the networking hardware; for TCP only packets of established connections are further processed (non-TCP packets skip this and the next stage). The TCP reassembly stage converts packets into an ordered byte stream. In the protocol pattern matching phase, the custom header is used to attach information about matches to the packet stream before the enriched packets are passed to the network monitoring system.

The individual features, statistical measurements to perform, and patterns to match, are configured via OpenFlow extensions using the SDN Engine of the switch (or NIC); configuration will typically be performed automatically by the respective parts of the network monitoring system that are able to use the feature set of the hardware.

# 5. RELATED WORK

Besides P4 [4] and POF [32], which we mention in §1, there are several other proposals that examine how to compile programs, or offload operations to switches or other networking hardware [18, 29, 16, 26]. Several past projects have proposed to use SDN or other hardware interfaces to allow an otherwise passive network monitor to actively block traffic in the network [21, 2, 31]. Other proposals use hardware to stop forwarding traffic flows considered benign to the network monitoring systems [15, 37, 3, 8].

A wealth of work evaluates the interplay of network monitoring and software defined networking, for example, using OpenFlow as a source of packet data [38, 30]. Other work uses information sourced from switches, e.g., using OpenFlow counters or flow expiry events [6, 36, 20].

The viability of efficiently implementing TCP stream reassembly in hardware has been shown in several works [11, 40]. Moshref et al. [22] discuss tracking TCP state in hardware, albeit without performing reassembly.

Most closely related to our work, but different in its goals, Narayana et al. [23] discuss how to use modern network hardware for network performance measurements. They present a SQL-like query language that allows users to perform queries over network traffic and show a hardware design based on a programmable key-value store primitive that can be used to implement this on switches. OpenSketch [39] is a proposal for hashing, filtering, and counting in the data plane combined with a measurement library in the control plane. In difference to our approach, OpenSketch aims at more complex statistical operations using probabilistic data structures. UnivMon [19] discusses how to implement certain streaming algorithms in hardware using P4. Nelson et al. [24] propose new switch features to determine the correctness of cross-packet properties.

# 6. CONCLUSION

In this work, we present the design of a comprehensive framework enabling network monitoring systems to effectively deal with large volumes of traffic. We show a new hardware design that can be implemented on current programmable network hardware. Simultaneously, we outline an OpenFlow-based command protocol and a new tagged feedback protocol. On the software side, the changes should be transparent to the user and will be handled by the respective frameworks in the network monitoring systems.

More generally, we argue that offering more specialized capabilities and APIs for applications with special demands, like network monitoring systems, opens up a wealth of new possibilities for the entire ecosystem—creating a more specialized, viable approach to software defined networking.

## Acknowledgments

We thank Aashish Sharma for providing LBNL traffic statistics.

This work was supported by the US National Science Foundation under grants ACI-1642161 and ACI-1348077. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators, and do not necessarily reflect the views of the National Science Foundation.

## 7. REFERENCES

[1] J. Amann, S. Hall, and R. Sommer. Count Me In: Viable Distributed Summary Statistics for Securing High-Speed Networks. In *RAID*, 2014.

[2] J. Amann and R. Sommer. Providing Dynamic Control to Passive Network Security Monitoring. In *RAID*, 2015.

[3] E. Balas and A. Ragusa. SciPass: a 100Gbps capable secure Science DMZ using OpenFlow and Bro. *SC*, 2014.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *CCR*, 44(3), July 2014.

[5] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *SIGCOMM*, 2013.

[6] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In *LCN*, 2010.

[7] Bro Network Monitoring System. https://www.bro.org.

[8] S. Campbell and J. Lee. Prototyping a 100g monitoring system. In *PDP*, 2012.

[9] Cavium Ethernet Switch Product Line: Cavium/XPliant CNX880xx. http://www.cavium.com/pdfFiles/CNX880XX_PB_Rev1.pdf?x=2.

[10] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski. The Science DMZ: A network design pattern for data-intensive science. In *SC*, 2013.

[11] S. Dharmapurikar and V. Paxson. Robust TCP Stream Reassembly in the Presence of Adversaries. In *USENIX Security Symposium*, 2005.

[12] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *USENIX Security Symposium*, 2006.

[13] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *CCS*, 2004.

[14] ESnet - TCP Issues Explained - Packet Loss. https://fasterdata.es.net/network-tuning/tcp-issues-explained/packet-loss/.

[15] J. M. Gonzalez, V. Paxson, and N. Weaver. Shunting: A Hardware/Software Architecture for Flexible, High-performance Network Intrusion Prevention. In *CCS*, 2007.

[16] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. Softnic: A software nic to augment hardware. Technical Report UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.

[17] Intel Ethernet Switch FM6000 Series. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf.

[18] L. Jose, L. Yan, G. Varghese, and N. McKeown. Compiling packet programs to reconfigurable switches. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 103–115, Oakland, CA, May 2015. USENIX Association.

[19] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pages 101–114, New York, NY, USA, 2016. ACM.

[20] S. A. Mehdi, J. Khalid, and S. A. Khayam. *Revisiting Traffic Anomaly Detection Using Software Defined Networking*, pages 161–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[21] S. Miteff and S. Hazelhurst. NFShunt: A Linux firewall with OpenFlow-enabled hardware bypass. In *NFV-SDN*, 2015.

[22] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 61–66, New York, NY, USA, 2014. ACM.

[23] S. Narayana, A. Sivaraman, V. Nathan, M. Alizadeh, D. Walker, J. Rexford, V. Jeyakumar, and C. Kim. Hardware-Software Co-Design for Network Performance Measurement. In *HotNets*, 2016.

[24] T. Nelson, N. DeMarinis, T. A. Hoff, R. Fonseca, and S. Krishnamurthi. Switches are monitors too!: Stateful property monitoring as a switch design criterion. In *HotNets*, 2016.

[25] Netronome agilio-ovs-hooks. https://github.com/Netronome/agilio-ovs-hooks.

[26] S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, and C. Cascone. Stateful openflow: Hardware proof of concept. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8, July 2015.

[27] Radiotap. http://www.radiotap.org.

[28] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Systems Administration Conference*, 1999.

[29] M. Shahbaz and N. Feamster. The case for an intermediate representation for programmable data planes. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 3:1–3:6, New York, NY, USA, 2015. ACM.

[30] S. Shirali-Shahreza and Y. Ganjali. FleXam: Flexible Sampling Extension for Monitoring and Security Applications in Openflow. In *HotSDN*, 2013.

[31] Snortsam – A Firewall Blocking Agent for Snort. https://www.snortsam.net.

[32] H. Song. Protocol-oblivious forwarding: Unleash the

power of sdn through a future-proof forwarding plane. In *HotSDN*, 2013.

[33] V. Stoffer, A. Sharma, and J. Krous. 100G Intrusion Detection. Technical report, LBL, 2015. https://commons.lbl.gov/display/cpp/100G+Intrusion+Detection.

[34] The Open Information Security Foundation. http://www.openinfosecfoundation.org.

[35] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware. In *RAID*, 2007.

[36] N. Van Adrichem, C. Doerr, and F. Kuipers. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In *NOMS*, 2014.

[37] N. Weaver, V. Paxson, and J. M. Gonzalez. The Shunt: An FPGA-based Accelerator for Network Intrusion Prevention. In *FGPA*, 2007.

[38] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar. SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment. In *GREE*, 2013.

[39] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 29–42, Berkeley, CA, USA, 2013. USENIX Association.

[40] R. Yuan, Y. Weibing, C. Mingyu, Z. Xiaofang, and F. Jianping. Robust tcp reassembly with a hardware-based solution for backbone traffic. In *2010 IEEE Fifth International Conference on Networking, Architecture, and Storage*, pages 439–447, July 2010.