Stochastic Pre-Classification for SDN Data Plane Matching

Luke McHale, *Student Member, IEEE,* Jasson Casey, *Student Member, IEEE,*
Paul V. Gratz, *Member, IEEE,* and Alex Sprintson *Member, IEEE*

Abstract—The Software Defined Networking (SDN) approach has numerous advantages, including the ability to program the network through simple abstractions, provide a centralized view of network state, and respond to changing network conditions. One of the main challenges in designing SDN enabled switches is efficient packet classification in the data plane. As the complexity of SDN applications increases, the data plane becomes more susceptible to Denial of Service (DoS) attacks, which can result in increased delays and packet loss. Accordingly, there is a strong need for network architectures that operate efficiently in the presence of malicious traffic. In particular, there is a need to protect authorized flows from DoS attacks.

In this work we utilize a probabilistic data structure to pre-classify traffic with the aim of decoupling likely legitimate traffic from malicious traffic by leveraging the locality of packet flows. We validate our approach by examining a fundamental SDN application: software defined network firewall. For this application, our architecture dramatically reduces the impact of unknown/malicious flows on established/legitimate flows. We explore the effect of stochastic pre-classification in prioritizing data plane classification. We show how pre-classification can be used to increase the effective Quality of Service (QoS) for established flows and reduce the impact of adversarial traffic.

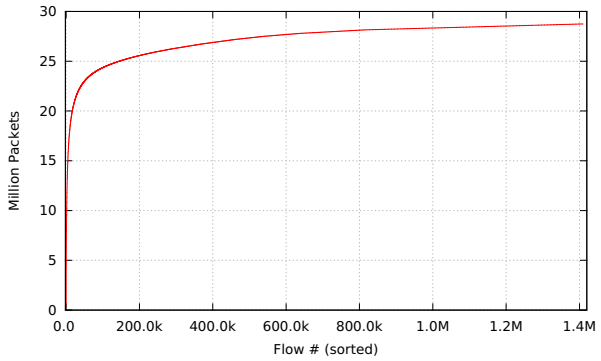

Fig. 1. CDF of unique flows in CAIDA trace: equinix-sanjose.dirA.20120119-125903

1 Introduction

Packet classification is a fundamental component of network hardware. Generally, the problem of packet classification expands in complexity as the number of rules grow. For example, typical firewalls have access control lists (ACLs) with thousands of matching rules. It is well known that packet classification is expensive and this is a well studied problem in the traditional network hardware domain [8], [9], [12]. With the move to Software Defined Networking (SDN), the complexity of packet classification is expected to grow dramatically due to the increased number of matching fields, the push to support a large number of features, and the larger degree of flexibility that SDNs encompass. While brute force hardware approaches to improve matching throughput, such as techniques that leverage ternary content addressable memories (TCAMs), can be used to improve the throughput [4], these approaches incur significant costs and result in increased power consumption.

Increasing packet classification complexity increases SDN data planes' vulnerability to Denial of Service (DoS) and, in particular, Distributed Denial of Service (DDoS) attacks. Malicious packet flows, such as those seen during a DDoS attack, interfere with authorized flows by consuming valuable data plane resources. Accordingly there is a critical need to explore architectures that can decouple the impact of potentially low throughput malicious traffic from high throughput authorized traffic in SDN data planes. This malicious traffic may consist of multiple well coordinated flows that come from a potentially large number of sources and can cause significant disruption (in terms of latency and packet loss) of the authorized traffic.

The goal of this paper is to improve the throughput and latency of packet classification for known/authorized traffic within the SDN data plane. The main idea

L. McHale, J. Casey, P. Gratz and A. Sprintson with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, 77843 USA e-mail: luke.mchale@tamu.edu, jasson.casey@tamu.edu, pgratz@gratz1.com, spalex@tamu.edu.
J. Casey is also with Flowgrammable e-mail: jasson@flowgrammable.com.
This material is based upon work partially supported by the AFOSR under contract No. FA9550-13-1-0008.

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.95

is to pre-classify known authorized traffic in SDN data planes, separating them from unknown or malicious traffic, thereby reducing the impact of malicious traffic on known flows.

As a means to accelerate packet classification, we propose to leverage the locality of known, authorized flows to enable a pre-classification stage within the SDN data plane. For example, Figure 1 is a Cumulative Distribution Function (CDF) of packets per flow from a one minute CAIDA trace [2]. As the figure shows, approximately 80% of all packets come from flows with greater than $5x10^6$ packets. There is significant locality to be leveraged in packet classification. Some traditional approaches to accelerating packet classification, such as ACL caching [6], [9], [10] leverage this locality, however these approaches neither provide the matching speed, nor the flow match capacity to truly decouple actions on known authorized flows from the effects of highly defuse malicious traffic. To address these challenges, we propose to use a stochastic data structure, a Bloom filter [5], as a pre-classification stage, decoupling known authorized traffic from unknown and/or malicious traffic.

Our goal in this work to enable a decoupling of known authorized traffic from unknown and/or malicious traffic within the SDN data plane. The contributions of this paper are as follows:

- First work to examine accelerating packet classification within the SDN data plane.
- Novel use of stochastic data structures to decoupling the impact of unknown/malicious traffic on known authorized traffic within the SDN data plane.
- A detailed hardware architecture that protects the existing flows from DoS/DDoS attacks.
- A comprehensive simulation study to evaluate the performance of an SDN forwarding data plane under attack.

The remainder of the paper is organized as follows: Section 2 discusses the background on hardware data planes for use in SDNs. It also examines prior work in packet classification. Section 3 introduces our proposed hardware design. Section 4 evaluates our design for its ability to achieve our goal of decoupling malicious traffic's effect on known good traffic. Finally, in Section 5 we present conclusions and directions for future research.

2 Background

In this work we start with a baseline OpenFlow data plane architecture and explore design permutations focused on improving overall data plane performance while under heavy and potentially malicious traffic. The baseline OpenFlow architecture and protocols are maintained by the Open Networking Foundation (ONF) [3]. These specifications outline semantics for an abstract packet processing machine. While these documents are not precise, they have been successful in outlining a basic packet processing pipeline along with a control interface for manipulating the pipeline's state.

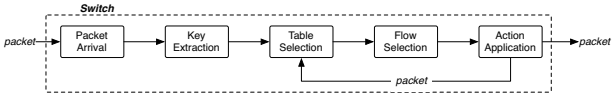

Fig. 2. OpenFlow Data Plane

OpenFlow: OpenFlow describes a simplified data plane for packet processing, this data plane pipeline is pictured in Figure 2. Packet processing happens in five stages. The first stage, packet arrival, is concerned with writing a packet to memory and sending certain bits of metadata about the packet into the pipeline such as: address in memory, packet size in bytes, arrival port id, etc. The second stage involves decoding enough of the packet's header to construct a key. The packet key is a tuple formed from a subset of packet header fields. The third stage, table selection, selects the appropriate flow table for indexing the packet's key. The first time a packet transits the pipeline, it always selects the first flow table; however, subsequent traversals can choose different tables. The fourth stage, flow selection, will use the packet's key to choose a specific entry in the flow table. The entry will contain a policy, or set of actions, to be applied to all matching packets. Finally, the fifth stage applies the selected policy to the packet. This could result in the modification of data within the packet, copying the packet, traversing the pipeline again, directing the packet towards the controller, or egressing the system.

Packet Classification: Packet classification is a fundamental activity in the core of any packet processing data plane. The basic process of classification involves forming a key that represents a packet and finding an entry in a classifier table that matches. The key is formed by extracting a set of values from a packet's protocol header fields. These values are concatenated to form a bit string that represents the packet. Classifier tables usually match the key bit string against a pair of bit strings, where the first element of the pair is a value to be matched and the second element masks bits that are important to the table entry.

There are several techniques for addressing matching problems: hash tables, tries, hierarchical tries, etc. [9], [13], but once you move to two or more dimension prefix bit string matching solutions become expensive in terms

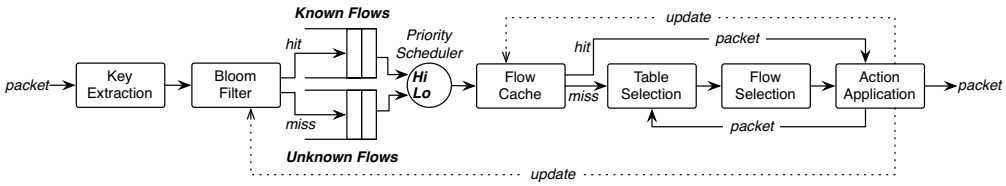

Fig. 3. Resilient OpenFlow Data Plane

of time or memory. Varghese [13] provides a comprehensive survey of the classification problems as well as approaches for their solution. For our discussion we are interested in software and low cost hardware solutions, which are likely constraints on low cost pervasive networking devices supporting SDN protocols. To further complicate the problem certain types of attack traffic can drastically reduce the performance of the classification stage in a data plane pipeline. The simple reason is that DoS/DDoS traffic cannot be classified as malicious until the classification activities have completed. Attackers have shown their ability to generate large volumes of attack traffic, in excess of 300 Gbps, congesting major Internet links [1]. Performing multidimensional classification in software is expensive, therefore it would be highly desirable to prevent malicious traffic from consuming classification resources.

3 Design

In this section, we describe a phenomenon inherent in network traffic. We further suggest two techniques that provide unique advantages to classification.

3.1 Motivation

Network packet classification is difficult, in part, due to the inherit randomness in packet arrival. In order to ensure no packets are discarded during classification, the network node must be able to process all packets at line rate. However, there are many circumstances where this is not possible due to classification complexity, throughput requirements, and available computational capability.

Pre-Classification: Our goal is to treat classification as a finite resource and prioritize packets for classification based on whether or not the packet belongs to a known, trusted flow. By partitioning incoming packets thus, we isolate the consumption of classification resources from malicious traffic. We propose to place a pre-classification stage before flow selection in the SDN data plane. The aim of this pre-classification stage is to leverage the locality in known, authorized flows in order to isolate performance of known flows from adverse effects caused by large bursts of unknown traffic.

Flow Locality: Within a period of time, old flows retire and new flows are established. The number of active flows, while highly variable, is finite and at worst equal to the number of packets. However, the number of flows is often much less than the total number of packets. This trend can be observed in Figure 1. Over this particular trace, there exists 1.4 million unique flows contained within 27.3 million packets – an order of magnitude difference in flow count versus packet count. When looking at received bytes, 95% of the aggregate throughput occurs from just 35% of the 1.4 million flows. We refer to this phenomenon as *flow locality*. We further define the *active-flow window* as a measure of flow locality over a given period of time. In this work we leverage flow locality within the given active-flow window to improve classification throughput via active flow action caching.

We propose to leverage both pre-classification and flow locality in our proposed design. As we will show in our results, techniques that take advantage of these properties provide orthogonal benefit and their effect is additive.

3.2 Architecture

Figure 3 depicts the block diagram of our general architecture for an OpenFlow data plane which leverages both pre-classification and flow locality. As shown, the OpenFlow abstractions from Figure 2 exist in this design as well. Differing from Figure 2, we first note that packets are partitioned by the *Bloom Filter* into two queues: *Known Flows* and *Unknown Flows* – corresponding to the desired pre-classification traffic classes. These flows are scheduled for processing in the remainder of the data plane by the *Priority Scheduler*, with *Known Flows* always processed ahead of *Unknown Flows* (*i.e.*, packets in the *Unknown Flows* queue will wait for processing until no packets remain in the *Known Flow* queue). The final addition to this data plane is the *Flow Cache*, which exploits flow locality within the active-flow window to cache the desired actions for a given flow. The remainder of the data plane correspond to the baseline architecture

of Figure 2. We now discuss each of the proposed components in detail.

Flow-Identifiers and the Bloom Filter: The Bloom Filter [5], a constant time stochastic containment data structure, is the critical component enabling quick and efficient pre-classification. Bloom Filters are a well-known, space-efficient, data structure which approximates the behavior of a conventional hash table for testing whether an element is within a set. In hardware, Bloom Filters store up to 10X more elements in the same space, and/or are many times faster to access than a traditional hash table. In our proposed design, the Bloom Filter serves as a space-efficient container to track flow identifiers. The Bloom Filter provides the ability to quickly and arbitrarily segregate packets, effectively decoupling flows into two logical partitions – *Known Flows* and *Unknown Flows*. Once incoming flows have been partitioned, they are fed into two different queues prior to proceeding through the rest of the data plane. The *Known Flows* queue has a higher priority than the *Unknown Flows* queue, decoupling classification performance of known flows from the effect of potentially large numbers of packets from previously unclassified, unknown flows.

In a standard SDN data plane, a unique key is extracted from the incoming packet to be used in packet classification. Typically, this key is then directly used by the *Table Selection* and *Flow Selection* stages, shown in Figure 2 to classify the packet. Here, we use this key first as input to the *Bloom Filter* stage, as shown in Figure 4.

To map an arbitrary k-bit tuple (*Key*) to a w-bit flow identifier used by both the *Bloom Filter* and *Flow Cache* stages, the Key is reduced using an *XOR tree* as the hash function ($Hash_n$). Each of the n XOR trees fold the k-bit Key into a w-bit flow identifier, where w is $\log_2$ of the Bloom Filter table size. Each w-bit flow identifier is then XOR'ed with a w-bit random number ($seed_n$) to obfuscate the hash, resulting in the Bloom Filter table index (idx_n). On every clearing interval, each seed is refreshed with a new random number.

In order to avoid n read/write ports on a single table, the Bloom Filter's table is split into n sets. Each set is managed by a single hash function. The resulting membership is simply the logical AND of the memberships determined by each set. Even though each hash performs essentially the same operation (reducing the k-bit Key to a w-bit flow identifier), the XOR combination must be unique to reduce collisions caused by compression.

In order to design a sufficient XOR tree, the binary entropy was analyzed for each bit in the Key over the length of the trace. In general, binary entropy decreases from LSB to MSB of IP and Port fields. The payload type field offers less entropy to the Key since the bits are highly correlated due to the popularity of IPv4 and UDP protocols.. The relative entropy of each bit in the Key may vary depending on the type of network traffic. In order to increase the entropy of the resulting flow identifier, the XOR tree was constructed to avoid the chance of combining bits that are highly correlated.

We segregated the bits from the Key (sorted by measured entropy) into w-bit levels. The order of the w bits within each level was randomized for each of the n hashes at design time. Finally, each of the w flow identifier bits is the column compression (XOR) through each level. This effectively randomizes the stride and reduces the chance of combining bits that are highly correlated.

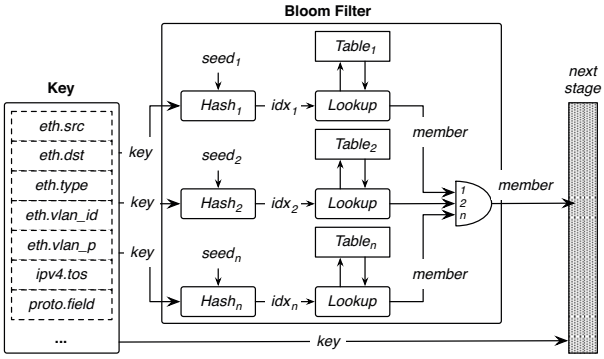

Fig. 4. Bloom Filter stage

Bloom Filters are stochastic data structures which have a low, but non-zero probability of false positive matches. In our design, false positives indicate the associated packet is falsely classified as known. Too many false positives will diminish the benefits of decoupling known flows and thus can lead to some performance degradation should the occurrence be too high. The probability of false positive classification increases as more items are added to the Bloom Filter. To ensure that false positive probability remains low, our design clears the Bloom Filter after I_{CLR} insertions. After clearing, the Bloom Filter must effectively relearn the locality of flows, thus the clearing interval I_{CLR} must be infrequent enough to reduce the likelihood of cold misses, yet frequent enough to reduce the likelihood of false positives. Real-time monitoring of false positives within the data plane is possible, however we simply use a pre-determined constant clearing interval. In our prior work involving exploration of a software-implemented Bloom Filter in a Linux system [7] showed that simply cold clearing is effective, but more intelligent clearing mechanisms are possible.

Flow Locality and the Flow Cache: As discussed in Section 3.1 and demonstrated in Figure 1, within typical packet traces there exists a high degree of flow locality.

Our design leverages this flow locality with the *Flow Cache*, shown in Figure 3. Here, the purpose of the *Flow Cache* is to cache the actions to be performed upon a given flow, reducing the burden upon the *Table Selection* and *Flow Selection* stages of the SDN data plane. In our design, the *Flow Cache* is indexed with the same flow identifier used to index the *Bloom Filter* stage (*i.e.*, *w*-bit identifier, XOR'ed down from the k-bit flow key). To ensure a deterministic match, each entry in the cache contains a full k-bit key for definitive tag match against the flow in question, together with a 32-bit "action" field for storing a pointer to the action-set associated with that flow entry.

Flow Learning: Here we proceed to describe the learning process for unknown flows. After key extraction, the packet is first checked against the Bloom Filter. When a miss occurs from the Bloom Filter, the packet enters the *Unknown Flows* queue. When no packets are currently in the *Known Flows* queue, the packet is then searched for in the *Flow Cache* (note, while it is unlikely that a flow would match in the *Flow Cache* after missing in the *Bloom Filter*, it is not impossible given the clearing interval I_{CLR} of the *Bloom Filter* and the *Flow Cache* replacement policy). Assuming no match in the *Flow Cache*, the packet then proceeds through the *Table Selection*, *Flow Selection*, and *Action Application* stages for packet classification and action-set application.

The update path to the Bloom Filter and Flow Cache can potentially be definable by the application. Applications could decide that certain flows should not be pre-classified using a *pre-classification* bit and/or cached using a *cacheable* bit.

Additionally, a cacheable or priority instruction could be integrated to offer more fine-grained control. Upon classification, the *Action Application* stage may prioritize the flow by inserting it into the *Bloom Filter* and/or improve classification performance by inserting it into the *Flow Cache*. Once the flow has been prioritized, all future packets matching the flow-identifier are pre-classified are then forwarded to the *Known Flow* queue. The action execution stage may also choose to process the packet without updating the *Bloom Filter*/*Flow Cache* selectively or only after a threshold is reached. While we always update the *Bloom Filter* and *Flow Cache* for every flow in our test application, configurable behavior may be desirable for low-throughput or low-priority flows.

4 Evaluation

In this section we first describe our experimental methodology and design implementation details. We then examine the performance of our design for varying combinations of malicious traffic and interface speeds.

Data Plane Frequency	2 GHz
Data Plane Queue Depth	2 high, 2 low
Bloom Filter Size	320Kb (5 arrays, each 64Kb)
Bloom Filter Clearing Interval (I_{CLR})	60K insertions
Flow Cache Size	69Kb (512 138-bit entries)
Flow Cache Organization	2-way set associative, LRU
Flow Selection	8,000 entries

TABLE 1
Summary of Architecture Details

4.1 Methodology

All experiments presented were performed using a cycle accurate SDN data plane simulator developed in-house using C++. SDN data plane models were developed for the following architectures:

- **Baseline:** Basic data plane architecture shown in Figure 2.
- **Partition+Caching:** Proposed architecture shown in Figure 3.
- **Partition:** Data Plane architecture with *Bloom Filter* and *Priority Scheduler* stages, but no *Flow Cache*.
- **Caching:** Data Plane architecture with a *Flow Cache* stage, but no *Bloom Filter* and *Priority Scheduler*.

Table 1 shows the microarchitectural implementation details for the designs under test (except where noted elsewhere). The Bloom Filter and Flow Cache sizes and clearing interval were set at the size empirically determined to be the point of diminishing returns in performance benefit. The data plane frequency was set to be equal to the access time of the slowest memory array defined within the system as determined by the SRAM array modeling tool, Cacti [11].

The workload examined here consists of captured traffic through an internet core switch provided by CAIDA [2]. These traces were captured on a 10 Gbps line card with a median 3 Gbps throughput during a 60 second time window. For privacy reasons, the trace was anonymized by CAIDA. The the associated ports, protocols, and relative flows were left intact. In order to observe how each architecture scales with throughput requirements, the packet-arrival time was expanded or compressed linearly to emulate 1 Gbps, 40 Gpbs, and 100 Gbps line cards.

For flow classification, using the CAIDA trace, we synthesized a set of eight-thousand classification rules utilizing protocol, IP, and port fields for both source and destination. In these experiments, the only action was either accept or drop, emulating a basic firewall application. To generate the rules we implemented a heuristic rule generator to synthesize a set of rules (8,000 in this case) for an arbitrary PCAP trace with a target rate

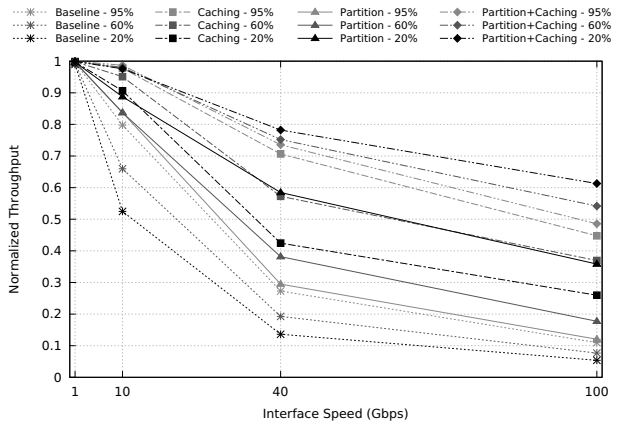

Fig. 5. Throughput of $\text{Accepted}_{\text{auth}}$ normalized to $\text{Total}_{\text{auth}}$ for each classification architecture.

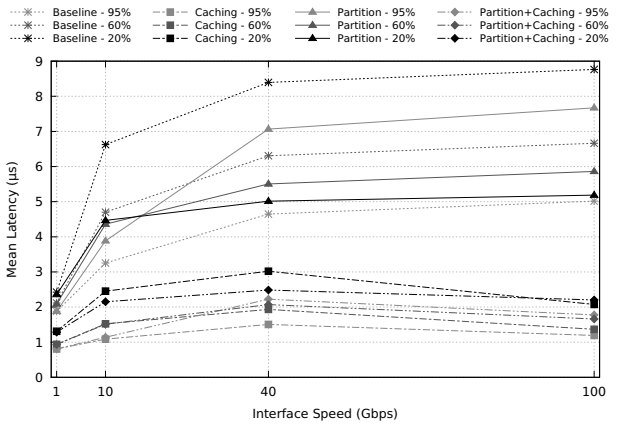

Fig. 6. Mean Latency of $\text{Accepted}_{\text{auth}}$ for each classification architecture compared to baseline.

of authorized ($\text{Total}_{\text{auth}}$) versus unauthorized ($\text{Total}_{\text{reject}}$) traffic. The generator distributes matches evenly across all 8,000 rules (as much as possible). Rule sets were generated for the following three scenarios (labeled accordingly in the experimental figures):

- 95% 95% authorized traffic, representing a network under nominal conditions.
- 60% 60% authorized traffic, representing a network with a moderately high load of unauthorized traffic.
- 20% 20% authorized traffic, representing a network under a DDoS attack.

4.2 Experimental Results

In this section, we compare the three classification architectures, *Partition*, *Caching*, and combined *Partition+Caching*, against a baseline SDN data plane configuration. We evaluate the effect of each approach on data plane performance by analyzing throughput, mean latency, and jitter (mean standard deviation of latency) for a range of interface speeds and authorized to unauthorized traffic ratios.

Throughput: Figure 5 shows classification throughput for each architecture. Here, throughput is measured as authorized packets accepted ($\text{Accepted}_{\text{auth}}$) normalized against the total number of authorized packets ($\text{Total}_{\text{auth}}$).

Generally for interface speeds above 10 Gbps, the baseline classifier is no longer able to keep up with the packet arrival rate. Since baseline is indiscriminately dropping packets, the throughput is further reduced proportional to the ratio of adversarial traffic. The effect of each architecture is revealed as the classifier is further stressed with 40 Gbps.

Interestingly, we find that both architectures which contain pre-classification (*Partition* and *Partition+Cache*) actually achieve higher relative throughput when the volume of unauthorized traffic is high as opposed to nominal (*i.e.*, 95% authorized). This is as opposed to the *Baseline* and *Cache* architectures where a higher ratio of unauthorized traffic leads to worse throughput. This highlights the pre-classifier's effective quality of service toward known flows.

Generally, in the figure we see that the *Caching* architecture provides a significant boost to classification throughput over baseline. Similar to the baseline architecture, throughput is reduced proportional to adversarial traffic. By combining both components, the *Partition+Caching* architecture leverages the benefits of both pre-classification and flow locality to provide consistent classification throughput in both nominal and adversarial network conditions. In addition, the *Partition+Caching* architecture scales much better compared to the baseline as interface speed, or similarly, classification requirements increase.

Latency: Figure 6 shows the mean latency through the SDN data plane for each of the architectures evaluated. In the figure we see that the latency results roughly map to the throughput results above. Generally, the data plane becomes stressed at 10 Gbps and saturated at 40 Gbps. Notice the *Partition* architecture behaves identical to the baseline when queues are rarely full (1 Gbps); however, the Partition mechanism maintains consistent behavior at 10 Gbps even when classifier resources are stressed.

The increase in mean latency for the *Partition* architecture under nominal traffic conditions is caused by a longer delay-until-service for authorized packets in the unknown flow queue. The *Partition+Caching* architecture is able to achieve a consistent average latency, even when

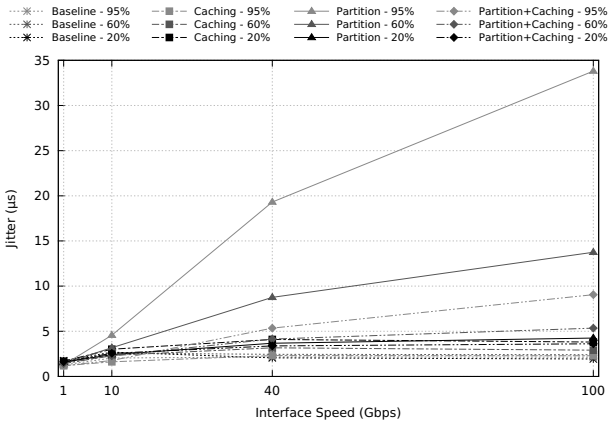

Fig. 7. Jitter - Mean Standard Deviation of Latency ($\text{Accepted}_{\text{auth}}$) for each classification architecture compared to baseline.

faced with adversarial traffic.

Latency Jitter: Figure 7 shows the latency jitter (*i.e.*, Standard Deviation of Latency) for each architecture. While the *Partition* architecture provides increased protection from DDoS attacks, it shows some increase in jitter, due to the priority mechanism. Note that jitter here is averaged across all flows and the increase is caused by the longer time-to-service of packets in the unknown flow queue. Once the flow is learned, however, the flow's jitter will be consistent with the *Caching* architecture.

The *Partition+Caching* architecture significantly reduces the observed jitter compared to the *Partition* architecture; maintaining jitter comparable to the baseline. This self-metering attribute of the *Partition+Caching* architecture allows the data plane to provide higher effective quality-of-service to known flows, avoiding overcommitment of data plane resources in addition to and improving performance overall.

We kept queue depth small to minimize the chance of packet reordering by the priority mechanism. While raw packet reordering could occur whenever a priority mechanism is implemented, we observed zero actual packet reorders within a flow.

5 Conclusion

While SDNs provide many advantages, they dramatically change the design of network hardware. As the complexity of SDN applications increase, data planes are becoming more susceptible to DoS attacks which can result in increased packet delays and loss. Thus, there is a strong need for SDN data plane architectures that operate efficiently in the presence of malicious traffic.

In this paper, we present a new approach to SDN data plane classification, utilizing a probabilistic data structure to pre-classify traffic, decoupling likely legitimate traffic from malicious traffic. We validated our approach by examining an SDN network firewall application. For this application, our architecture dramatically reduces the impact of unknown/malicious flows on established/legitimate flows.

For future work, exploring intelligent Bloom Filter clearing strategies would help reduce the impact of relearning flows immediately following a cold clear. In addition, supporting arbitrary Keys for protocol independence would further improve the generality of this architecture. The *Partition* and *Caching* techniques presented in this paper can be integrated into an OpenFlow data plane without requiring specification extensions; however, software-defined control over pre-classification and caching would require extension support. While the focus of this work is centered around OpenFlow, the approach presented in this paper is general enough for many packet processing data plane architectures.

References

[1] Biggest DDoS Attack' Did Not Hurt The Global Internet – This Time. Retrieved June 15, 2014 from http://www.techweekeurope.co.uk/news/biggest-cyber-attack-ddos-spamhaus-security-networks-111690.

[2] The CAIDA Anonymized 2012 Internet Traces - 2012, kc claffy, Dan Andersen, Paul Hick. http://www.caida.org/data/passive/passive_2012_dataset.xml.

[3] Open Networking Foundation. Retrieved June 15, 2014 from https://www.opennetworking.org.

[4] 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 *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[5] A. Broder, M. Mitzenmacher, and A. Broder, and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[6] I.L. Chvets and M.H. MacGregor. Multi-zone caches for accelerating ip routing table lookups. In *Workshop on High Performance Switching and Routing, Merging Optical and IP Technologies*, pages 121 – 126, 2002.

[7] P. Ghoshal, C Jasson Casey, Paul V Gratz, and Alex Sprintson. Stochastic pre-classification for software defined firewalls. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–8. IEEE, 2013.

[8] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. In *Micro, IEEE*, volume 20, pages 34 –41, Jan/Feb 2000.

[9] P. Gupta and N. McKeown. Algorithms for packet classification. In *IEEE Network*, volume 15, pages 24 –32, Mar/Apr 2001.

[10] K. Li, F. Chang, D. Berger, and W. Feng. Architectures for packet classification caching. In *Proceedings of the 11th IEEE International Conference on Networks (ICON)*, pages 111 – 117, Sept/Oct 2003.

[11] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: a tool to model large caches. Technical report, HP Laboratories, 2009.

[12] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *SIGCOMM Computer Communication Review*, volume 29, pages 135–146, New York, USA, August 1999.

[13] G Varghese. *Network Algorithmics: an Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann, 2005.