Characterising the Limits of the OpenFlow Slow-Path

Richard Sanger  
University of Waikato  
rsanger@wand.net.nz

Brad Cowie  
University of Waikato  
brad@waikato.ac.nz

Matthew Luckie  
University of Waikato  
mjl@wand.net.nz

Richard Nelson  
University of Waikato  
richardn@waikato.ac.nz

Abstract—The OpenFlow standard accommodates network control traffic by providing packet in and out messages for sending packets between the network and controller. We conduct comprehensive measurements of the performance of this control architecture in different configurations across five hardware switches, each from a different vendor, representing a broad range of OpenFlow offerings, from implementations built on legacy ASIC architectures, to those implemented solely with OpenFlow in mind. The best performing switch achieved a maximum mean packet-in rate of 5,145 messages per second, representing less than 3Mbps of traffic. Additionally, all switches tested failed to maintain control traffic latency under 50ms in one or more tests. We find the slow-path performance of these hardware switches is easily overloaded and is insufficient for modern network control architectures.

I. INTRODUCTION

OpenFlow has become a popular Software-Defined Networking (SDN) standard for both vendors and application developers by balancing existing hardware capabilities against a programmable feature set [1]. Modern network architectures can leverage the flexibility provided by SDN to overcome limitations with traditional network architectures. OpenFlow accommodates control traffic through packet-in and out messages. The packet-in message allows a controller to retrieve packets from the network for additional processing including protocol operations, logging and statistic gathering. The packet-out message allows the controller to return a packet to the network, or to send any arbitrary packet in the network. Packet-in and out messages take the switch’s slow-path from the switch’s ASIC through its on-board CPU to the controller. This path is shared by all control traffic, and is susceptible to becoming overloaded. The slow-path in a switch needs to meet the latency and bandwidth of all control protocols simultaneously.

Failure to meet a protocol’s requirements, even briefly during periods of high load (possibly caused by a fault or an attack), can result in cascading failure. An operational example of such a failure occurred in Google’s B4 WAN [2]. Two switches participating in ISIS were accidentally configured with the same ID. This shared ID caused a feedback loop between these switches, resulting in a 400MB slow-path queue, which delayed all ISIS hello messages enough to declare an interface as down. Declaring an interface down caused an increase in ISIS updates throughout the network, which in turn increased latency on the slow-path and cascaded to affect the whole network. Their paper identified issues that still needed investigating, including the scalability of the slow-path and the need for additional performance profiling.

The exact requirements on control traffic will vary depending on the mix protocols used, so we use the requirements of Bidirectional Forwarding Detection (BFD) [3] to illustrate challenging requirements on both latency and bandwidth. BFD is a technique used to detect failure of the forwarding-plane quickly, BFD is usually implemented in hardware by the switch, but an SDN controller may provide a practical alternative if a switch does not support BFD. A typical failure detection time of 50ms requires 60 packets per second [3] and latency must be strictly less than 50ms. A 48 port switch running BFD on each physical link would be required to process 2,880 packet-in and out messages per second with less than 50ms of latency.

In this paper, we perform a measurement study of the performance characteristics of OpenFlow packet-in and out messages and their suitability in modern network control architectures. In industry, it is well known that the OpenFlow slow-path is slow. However, slow has not been quantitatively defined. We identify throughput and latency limits of packet-in and out messages and isolate their effect in a reactive scenario. We study five hardware switches, each from a different vendor and which are representative of a broad range of OpenFlow switch implementations. Four are legacy ASIC-based switches, while the fifth is designed for OpenFlow and uses an Network Processing Unit (NPU).

We observed significant differences in throughput, latency, buffering, and priority given to packet-in and out messages. The maximum mean packet-in rate we observed was 5,145 packets per second (pps) obtained with 64-byte packets, while a different switch rate limited packet-ins to 100pps. Mean latency increased with packet rates, with four of the five switches reaching in excess of 200ms and one more than 3s when overloaded. One switch periodically processed packet-in messages every half second, further delaying the messages.

Additionally, to promote further testing of OpenFlow devices, we have released our test suite code [4]. Our test suite builds on OFLOPS-turbo [5] by adding support to test modern OpenFlow switches. We added support for OpenFlow 1.3 and control channel encryption. To allow our study to be reproduced we have released supplementary switch configuration details and test artifacts [6].
In OpenFlow, packet-in and out control messages are used to move packets between the control-plane and data-plane. Packet-in and out messages share the resources between the ASIC and the controller with all other OpenFlow operations, including modifying flows and collecting statistics. An OpenFlow controller maintains a single TCP connection with the OpenFlow agent on the switch. In turn, the OpenFlow agent maintains a shared communication channel with the ASIC. Each component along this path adds latency and introduces a possible performance bottleneck [7].

An OpenFlow switch generates packet-in messages from the live traffic on a network. The network is an untrusted environment: hosts are typically not under the control of the network operator. As such, switch vendors should take care to ensure that packet-ins do not impact the performance of other operations. OpenFlow trusts the controller, and as such all requests the controller makes, including packet-out and flow modification messages, should be given priority by an OpenFlow switch over packet-in messages. The importance of this behavior is apparent when considering a reactive controller scenario: if packet-ins were given priority over adding flow rules, such that no flow rules were added, all traffic would remain on the slow-path rather than matching a flow in the data-plane, significantly limiting network performance.

III. PRIOR BENCHMARKS

Previous OpenFlow benchmarking focused on flow table manipulation throughput and latencies [5], [8], [9], and analysis of the latencies associated with reactively installing flows [7], [10]. All of these studies observe substantial differences in behavior and performance of OpenFlow switches, despite all following the same standard. Our study instead highlights packet-in and out performance, and the latency incurred when these channels are overloaded.

In 2012, Rotsos et al. presented OFLOPS [8], a modular OpenFlow switch benchmarking platform, and reported benchmarks of three switches. The authors found incorrect behavior in switch implementations of the OpenFlow barrier message, suggesting the control-channel cannot always be trusted. In 2014, Rotsos et al. presented OFLOPS-turbo, which added NetFPGA 10G support to OFLOPS, enabling benchmarking of high-performance 10Gbit/s switches at line-rate [5]. In 2015, Kučniar et al. provided further investigation of flow table performance [9], using a thorough approach to collect data in multiple dimensions (including the number of flows, priorities, and adding vs. modifying flows). They observed that the atomic OpenFlow flow modification request behaves as a non-atomic delete and add operation on some switches.

In 2013, Huang et al. [10] measured packet-in and out latency in a simulated situation where TCP flows trigger a reactive flow installation. The resulting measurements were used to create a realistic switch emulator which accounted for packet-in and out latencies and other performance artifacts. In 2015, He et al. [11] further investigated the latency of installing flows, measuring packet-in and rule installation latency with different rule priority insertion orders.

IV. QUANTIFYING CONTROL PERFORMANCE

In this section, we test and quantify the performance characteristics of the slow-path of five hardware OpenFlow switches, each from different vendors. We identify the limits of packet-in and out message throughput, latency, and processing priority on these switches, and identify the effects of configurable variables including encryption and OpenFlow version.

A. Testing framework

Our tests extend prior research by thoroughly identifying packet-in and out message performance in at least four dimensions (packet rate, packet size, control channel encryption, and OpenFlow version), as well as isolating their effect on the performance of adding flow rules. We apply particular focus to the switch behavior when overloaded and quantifying packet rates at which overloading occurs.

We used OFLOPS-turbo [5] as the basis of our benchmarks. OFLOPS enables developers to write modules to test an aspect of switch performance. OFLOPS manages many basic tasks for the module including the OpenFlow control channel, generation and capture of data-plane packets, timer events, and SNMP collection. Each task is assigned to a separate thread to leverage multi-core architectures. A module consists of a series of callbacks for the tasks managed by OFLOPS, including functions that are called when an OpenFlow message or data-plane packet is received. We used software-based libpcap packet generation and capture in OFLOPS, and tests were run on over-provisioned commodity hardware.

B. Features added to OFLOPS

OFLOPS lacked support for two notable features that we tested: OpenFlow 1.3 and control channel encryption using Transport Layer Security (TLS). OpenFlow 1.3 gained popularity due to its increased feature set. Many vendors’ switch offerings support OpenFlow 1.3 [12]–[17] and many OpenFlow applications now require it [18]–[20]. We benchmarked switches with encrypted control channel traffic because this improves security and is the recommended configuration for an OpenFlow deployment [21].

We added both OpenFlow 1.3 and encrypted control channel support to OFLOPS. We replaced the built-in control channel management with the lightweight libfluid_base [22] library, which we chose due to its support of all OpenFlow versions, and control channel encryption. Libfluid handles OpenFlow version negotiation and extracts individual OpenFlow messages based on the common OpenFlow message header. When libfluid receives a message, it does no further processing of the message; instead, the message is passed to the test module for specific processing. The test developer must select an appropriate OpenFlow library to interpret and construct messages. OFLOPS previously used built-in support for OpenFlow 1.0 messages. For our test modules, we chose the Revised OpenFlow Library (ROFL) [23] to construct and decode OpenFlow
TABLE I: Details of the five OpenFlow switches tested. All are from different vendors.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Type</th>
<th>CPU</th>
<th>ASIC</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allied Telesis AT-x930-28GTX</td>
<td>Top-of-Rack</td>
<td>PowerQUICC III</td>
<td>Broadcom</td>
<td>AlliedWare Plus</td>
</tr>
<tr>
<td>P3780 [12]</td>
<td></td>
<td>4x1GHz</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>Pica8</td>
<td>Top-of-Rack</td>
<td>PowerQUICC III</td>
<td>Broadcom</td>
<td>PiOS</td>
</tr>
<tr>
<td>NoviFlow NoviSwitch 1248 [16]</td>
<td></td>
<td>Intel 64</td>
<td>Mellanox NPU</td>
<td></td>
</tr>
<tr>
<td>Linux Foundation Open vSwitch [17]</td>
<td>Software</td>
<td>Intel 64</td>
<td>Ezchip NP-4</td>
<td></td>
</tr>
</tbody>
</table>

C. Tested OpenFlow switches

Table I details the switches tested, including the software version. These switches represent a variety of OpenFlow offerings from different vendors. The x930, P3780, MLX-4, and E3500 are all legacy ASIC-based based switch offerings which have had OpenFlow support added. The NS1248 is an NPU-based offering designed for OpenFlow. This research focuses primarily on the hardware switches. However, we also include the Open vSwitch (OvS) software switch as a point of comparison, and to prove that both OvS and our testing suite can match each others’ performance.

We used recent stable release versions of the P3780, E3500, NS1248 and OvS network operating systems, and recent beta releases of the x930 and MLX-4 operating systems. Both the x930 and P3780 use an OvS-based OpenFlow implementation, whereas the MLX-4, NS1248, and E3500 use their own OpenFlow implementations. Both OpenFlow 1.0 and 1.3 were supported and successfully tested on all switches except the NS1248 which only supports 1.3 and newer. Our attempts to test encrypted control channel performance met challenges: the x930 did not support encryption, the MLX-4 was unstable with encryption enabled, and the E3500 did not accept our certificate. We successfully tested the P3780, NS1248 and OvS with an encrypted control channel.

D. Testing methodology

We benchmarked switches in a controlled manner. We ran all tests, where possible, with both OpenFlow 1.0 and 1.3, and with and without an encrypted control channel. When testing packet-in or out messages, we varied the packet size, input rate and packet-in buffering. When a test added flows, we varied the total number of flows added. As a result, our data is multi-dimensional; therefore in our analysis, we focus on a base configuration of OpenFlow 1.3, unencrypted, 64-byte packets, without packet-in buffering. Section IV-J discusses the variation seen when changing each dimension in relation to the base configuration.

Figure 1 shows the testing configuration which comprised of an OpenFlow controller running our modified version of OFLOPS directly connected to the OpenFlow switch via two 1Gbps links for data-plane packet generation and capture. The third 1Gbps link is used as the OpenFlow control channel and was switched via a Control-Plane Network (CPN) to OFLOPS. All switches were benchmarked using over-provisioned servers, with at least four cores.

We ran four different tests to find the limits of the slow-path of the hardware switches and the effects on installing rules. We ran each test 5 times per configuration and report the mean with a 95% confidence interval.

E. Baseline packet-in performance

To measure baseline packet-in performance, the OFLOPS test module installs a rule to send traffic to the controller as a packet-in. OFLOPS then sends a fixed rate of packets to a port on the switch, matching the flow rule installed. To track latency and loss, we record when each packet-in message is sent on the data-plane and received by the slow-path. Each test ran for 60 seconds.

The first row of Table II lists the baseline packet-in performance with an input rate of 10,000pps. The hardware switches were unable to meet the input rate. Therefore the results show the maximum overloaded packet-in rate for these switches.

There is an order of magnitude in the range of mean packet-in processing rates between the hardware switches, ranging from 100pps to 5,145pps. When compared to BFD with a 50ms failure detection time on a 48 port switch, requiring 2,880pps, the switches either fail to meet the throughput...
TABLE II: Table showing mean processing latency observed on baseline packet-in and packet-out rates observed from 3 different test types. The baseline performance is that of each metric in isolation. The reactive control traffic and flow installation tests combine packet-in with either packet-out or flow installation. All tests used the configuration described in Section IV-D. Packet-in and out message rates are those processed by the switch with an input rate of 10,000pps and flow installation rates are those observed when adding 500 flows as quickly as possible. We report 95% CIs when ≥1%.

<table>
<thead>
<tr>
<th>Input Rate</th>
<th>x930</th>
<th>P3780</th>
<th>E3500</th>
<th>MLX-4</th>
<th>NS1248</th>
<th>OvS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100pps</td>
<td>0.93</td>
<td>0.68</td>
<td>0.59</td>
<td>250</td>
<td>5.6</td>
<td>0.48</td>
</tr>
<tr>
<td>1,000pps</td>
<td>26</td>
<td>4</td>
<td>0.56</td>
<td>270</td>
<td>6</td>
<td>0.44</td>
</tr>
<tr>
<td>10,000pps</td>
<td>440</td>
<td>280</td>
<td>350</td>
<td>500</td>
<td>3,300</td>
<td>0.15</td>
</tr>
</tbody>
</table>

TABLE III: Table showing mean processing latency observed on baseline packet-in messages vs. input rate. Packet-in latency increases with input rate for the hardware switches.

TABLE IV: Table showing mean processing latency observed on baseline packet-out messages with differing input rates.

TABLE V: Table showing mean processing latency observed on baseline flow installation rates observed from 3 different test types. The baseline performance is that of each metric in isolation. The reactive flow installation tests combine flow installation with either packet-out or flow installation. All tests used the configuration described in Section IV-D. Flow-installation and out message rates are those processed by the switch with an input rate of 10,000pps and flow installation rates are those observed when adding 500 flows as quickly as possible. We report 95% CIs when ≥1%.

<table>
<thead>
<tr>
<th>Input Rate</th>
<th>x930</th>
<th>P3780</th>
<th>E3500</th>
<th>MLX-4</th>
<th>NS1248</th>
<th>OvS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100pps</td>
<td>0.93</td>
<td>0.33</td>
<td>0.24</td>
<td>0.26</td>
<td>0.29</td>
<td>0.45</td>
</tr>
<tr>
<td>1,000pps</td>
<td>16</td>
<td>2.2</td>
<td>1.6</td>
<td>0.8</td>
<td>0.38</td>
<td>0.39</td>
</tr>
<tr>
<td>10,000pps</td>
<td>410</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

F. Baseline packet-out performance

To measure the baseline packet-out performance, the test module creates and sends packet-out messages over the OpenFlow channel to the switch under test. The packet-out messages output a packet to a port connected to the controller where OFLOPS records them. Each test ran for 60 seconds.

The second row of Table II lists the baseline packet-out performance with an input rate of 10,000pps. The P3780, E3500 and MLX-4 were unable to send packet-outs at the input rate; therefore these results show the maximum packet-out rate for these switches. The x930 successfully maintained a 10,000pps packet-out rate in 4 of the 5 test runs and the NS1248 in all tests. Further testing showed that the x930 had a mean maximum packet-out rate of 12,884pps (±2%) and the NS1248 29,283pps (±10%). All switches meet the targeted rate of 2,880pps required for our BFD example, albeit barely in the case of the MLX-4.

TABLE V: Table showing mean processing latency observed on baseline flow installation rates observed from 3 different test types. The baseline performance is that of each metric in isolation. The reactive flow installation tests combine flow installation with either packet-out or flow installation. All tests used the configuration described in Section IV-D. Flow-installation and out message rates are those processed by the switch with an input rate of 10,000pps and flow installation rates are those observed when adding 500 flows as quickly as possible. We report 95% CIs when ≥1%.

G. Reactive control traffic performance

Fine-grained flow-based network control is often achieved using reactive control. When a reactive controller receives a packet-in from a new network flow, it installs a flow to the switch and sends the packet to the network as a packet-out. We explore reactive control in two parts: the impact of packet-ins on packet-outs (reactive control traffic) and the impact of packet-ins on adding flow rules (reactive flow installation).

Examining packet-in and out performance together allows us to compare performance with the baseline and gain in-
sight into the priority a switch gives each. Ideally, a switch should give priority to processing packet-out messages before packet-ins, because packet-ins are from the untrusted network, whereas packet-outs are from the trusted controller as detailed in Section II. We used the same test procedure as the baseline packet-in test, except that a packet-in triggers a packet-out.

The second row-group of Table II shows reactive control traffic performance, for both packet-in and packet-out messages individually. The MLX-4, x930, NS1248, and E3500 match their packet-out message processing rate to the number of packet-ins processed, with the minor variations between the two rates a result of the additional latency packet-out messages incur rather than loss. The E3500 sees a 36% and the NS1248 a 29% decrease in packet-in message rate due to the increased work involved with processing packet-out messages which indicates these switches give priority to packet-out messages. Both the MLX-4 and x930 packet-in rates are identical to their baseline performance, due to packet-in rate-limiting.

Our results suggest that the P3780 gives packet-in messages precedence over packet-out. Packet-outs are processed at 5% of the packet-in rate, yet the packet-in processing rate remains unchanged from the baseline performance. This behaviour results in packet-out messages queueing on the OpenFlow control channel, delaying all OpenFlow messages, not only packet-out messages.

H. Reactive flow installation performance

Reactive flow installation looks at the impact of packet-ins on adding flows when compared to the baseline performance, and gives insight into their relative priority. The OFLOPS test module creates a constant rate of packet-ins as in the baseline test. Simultaneously, the module adds a fixed number of OpenFlow flow rules to the switch as quickly as possible, unrelated to the packet-in rate. We sent 64-byte packets matching each flow rule at 100Mbps (204kpps) in a round robin fashion to the switch. Each flow rule matches an IPv4 address using a hardware flow table. By default, the switch drops all packets; however, once the flow rule has installed, the packets are forwarded back to OFLOPS. OFLOPS records the arrival time of the first packet of each flow, thus verifying the flow has been installed into the data-plane [8].

Unlike the other tests, the runtime of this test varied. Each test ended once all flow rules were installed. The test module sent packet-ins for ten seconds before the module began adding flows. The results reported are for the period that both installing flows and packet-ins were running simultaneously.

The third row-group of Table II displays the results for inserting 500 flows while overloaded with 10,000 packet-in messages per second. All switches tested installed 500 flows to hardware. We observed no difference in the x930 and E3500 flow installation performance compared to the baseline, but there was a reduction in packet-in message rates, which indicates the switches are correctly prioritising the trusted add flow messages. The P3780’s rate of flow installation decreased by 73% when overloaded, and the packet-in rate decreased by 25% which indicates a preference towards processing the packet-in messages over the trusted messages to install flows. Table II does not report the MLX-4 packet-in rate as it did not receive any packet-in messages during this test because the test only ran for 200ms, the time the MLX-4 took to install 500 flows, while the MLX-4 sends packet-ins at 500ms intervals. The MLX-4 flow installation rate decreased by 17%, indicating some initial processing of packet-in messages during the test.

Unfortunately, we could only complete one test run on the NS1248 so we cannot include a confidence interval, due to limited access. However, it appears that the flow installation rate decreased without change to the packet-in rate.

For the P3780, NS1248, and MLX-4, decreasing the rate of flow installation in favour of packet-in messages could be detrimental as a reactive controller installs flow rules to move network flows off the slow-path. Delaying the installation of these flows will cause additional packets to be processed in the slow-path, further degrading switch performance.

The MLX-4 is the only switch where the flow installation rate exceeds its packet-in processing rate, thereby causing the packet-in rate to be a bottleneck in a reactive scenario.

I. Individual switch traits

Figure 2 shows the first 10 seconds of the baseline packet-in test, contrasting the heterogeneous traits of the switches tested. These traits include buffers, timers, and rate limits which increase latency or decrease throughput, both of which limit modern control architectures.

MLX packet-in: We found the MLX-4 to be the most intriguing switch, as both packet-in and out messages appear to be rate-limited by internal processing timers. Packet-in messages are governed by a timer that fires every 500ms, allowing a burst of processing, as shown in Figure 2a, and is independent of all other variables that we measured. Packet-ins are buffered between timer intervals and newer packets are dropped once the buffer is full. This buffering resulted in unavoidable latency on all packet-ins, which increases with high packet rates as the oldest packets fill the buffer first.

MLX packet-out: In the baseline 10,000 packet-outs per second test, we observed small bursts of packet-outs every 200ms. At lower packet rates the MLX-4 processes packet-outs immediately. We found the MLX-4’s OpenFlow control channel TCP implementation to be the cause. When overloaded, the MLX-4 sends a TCP zero-window probe, causing the Linux kernel (OFLOPS) to suspend sending packet-out messages. To resume, the MLX-4 sends a non-zero window size, but because the window size advertised is smaller than the MSS the Linux kernel does not immediately resume sending. Instead, the kernel waits for a 200ms timeout to complete. Other switches also advertised zero-windows but resumed immediately by sending a window size larger than the MSS. The characterisation of MLX-4 packet-out performance is complex, as it can reach higher rates than Table II indicates, but only until the MLX-4 becomes overloaded. We saw an example of this performance drop with 1500-byte packets; the MLX-4 can maintain all rates up to 1,000pps, but with an input rate of 10,000pps the MLX-4 only sends 209pps.
Fig. 2: The first 10 seconds of a baseline packet-in test, with a 10,000pps input rate. The MLX-4’s packet-in processing shows distinctive half-second intervals. The MLX-4, x930, P3780 and NS1248 all have extremely high latencies exceeding 100ms, exceeding 4s on the NS1248 (off the graph). The E3500 maintains a low, stable latency between 30 and 40ms.

Table V: Percentage change in mean packet-in rate compared to the baseline 64B 10,000pps test (Table II). Missing results are due to unsupported features.

<table>
<thead>
<tr>
<th>Switch</th>
<th>E3500</th>
<th>MLX-4</th>
<th>NS1248</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>-37%</td>
<td>-67%</td>
<td>-83%</td>
</tr>
<tr>
<td>1500B Packets</td>
<td>0%</td>
<td>-36%</td>
<td>-28%</td>
</tr>
<tr>
<td>OpenFlow 1.0</td>
<td>+13%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Pkt In Buffering</td>
<td>-8%</td>
<td>-52%</td>
<td>0%</td>
</tr>
</tbody>
</table>

**x930:** The x930 implements a rate limit of 100pps on all packet-ins in all of our testing. We saw no such limit for packet-outs. Higher rates of packet-ins show increased latency even once the rate limit is reached, as shown in Table III, and ultimately reaching 440ms with a 10,000pps input rate. Figure 2b shows the latency observed over the first 10 seconds of such a packet-in baseline test. It appears initial processing occurs on discarded packet-ins, as the latency between an input rate of 1,000pps and 10,000pps increases (indicating a buffer filling), despite only receiving 100pps in both cases.

**E3500 & P3780:** We examined CPU usage of the E3500 and P3780 during the baseline packet-in and out tests at 10,000pps and saw full utilisation, indicating that they were CPU bound. Although, at low packet rates the E3500 releases packet-out messages in 200ms bursts, as discussed in Section IV-F. The E3500 gives the most consistent results, with minimal latency even at high packet rates. When the E3500 is overloaded, a typical packet-in is processed within 30-40ms as shown in Figure 2d. The P3780 exhibits more variance of packet-in latencies when overloaded, typically fluctuating between 150-350ms (Figure 2c), with an additional 150ms increase seen when the control channel is encrypted.

**NS1248:** Packet-in messages incur a large latency on the NS1248 when overloaded, indicating a large buffer. For 64-byte packets a mean latency of 3.3s is incurred and 0.86s for 1500-byte packets. Because the processing rate fluctuates, latency can be higher at times. Figure 2e shows in excess of 4s latency, truncated by the graph’s Y-axis. The NS1248 has the best performance across the board, with significantly better packet-out performance and flow installation performance, but has similar packet-in performance to the next best switch.

**J. Effects of buffering, packet size, encryption, and OF version**

Thus far the results reported are focused on a limited set of varying dimensions: (switch, input rate, baseline vs reactive control traffic and reactive flow addition) while using fixed values in other dimensions, i.e. OpenFlow 1.3, 64-byte packets, unencrypted control traffic, and without packet-in buffering. In this section, we report the impact of changing each of those previously fixed dimensions one at a time against the base configuration. Table V compares results to the packet-in baseline from row 1 of Table II.

**Encryption:** The NS1248 saw no difference in performance with encryption enabled. However, the extra work involved with the encryption process decreased performance in most scenarios on the P3780. The P3780 saw a 37% decrease with 64B packet-in messages, and 69% for 1500B packets.

**Packet Size:** Larger 1500-byte packets obtain lower maximum packet-in and out message rates compared with 64-byte packets (excluding the rate limited packet-in messages on the x930). Packet rate is a more dominant factor than packet size; the overloaded packet-in baseline performance decreases by 28-83% (depending on the switch) with 1500-byte packets, rather than 96% expected from the size difference.

**OpenFlow Version:** The effect of OpenFlow version differs depending on the switch and test but can be large. There is no general trend between switch performance and the OpenFlow version. Such effects are likely a reflection of where vendor optimisation efforts were focused.

**Packet-in Buffering:** The P3780, x930, and NS1248 support packet-in buffering. The switch sends a truncated version of the original packet to the controller and stores the full packet against a buffer ID. Later the controller can include this ID in a packet-out. Packet-in buffering reduces the amount of OpenFlow TCP traffic and, in theory, the load on a switch.

Table V shows that packet-in buffering has equal or worse performance for 64B packets. The x930 and NS1248 observe no impact from packet-in buffering indicating a bottleneck elsewhere, not in OpenFlow transmission. The P3780 sees a small improvement of 2% when buffering 1500B packet-in messages over an encrypted channel, a best case for packet-in buffering, but overall sees worse performance. When overloaded the P3780 buffers only 1-2% of packet-in messages.
in both reactive and baseline tests, with the remaining sent in full. The NS1248 buffered all packets until it abruptly stopped after having sent nearly 2 million buffered packet-ins.

V. ALTERNATIVES TO THE SLOW-PATH

We have found the performance of the slow-path to be unsatisfactory and the variance between switch implementations pose interoperability issues. As a result, OpenFlow application developers may want to consider alternatives to the slow-path. OpenFlow 1.3 [21] provides auxiliary channels as a solution, in which a switch sends packet-in messages over a separate TCP, UDP or DTLS connection. Auxiliary channels simplify the protocol enough to make a data-plane implementation feasible, thus avoiding the switch’s slow-path. Of the switches we tested, only the E3500 supports an auxiliary channel but its implementation is in the software slow-path. Today, the lack of auxiliary channel support makes it impractical to use.

Other projects avoid the slow-path by installing flow rules to forward control packets to a controller. The difficulty lies in tagging pipeline information into a packet, such as the original ingress port. He et al. [24] propose tagging the ingress port in the IPv4 IPID field and forwarding packets to a proxy application which establishes an OpenFlow connection to the controller. Atrium [18], an SDN router built to illustrate interoperability, offloads control traffic over a dedicated directly connected link. The offloaded traffic is routed to the controller and does not maintain ingress port information, so only supports control traffic destined for the controller like BGP and OSPF. CacheFlow [25] offloads the least used flows to a software switch to reduce hardware tables sizes. CacheFlow uses VLAN tunnels to offload traffic to the software switch and encodes a packet’s ingress port in the VLAN tag. These techniques need to be tailored to the protocol being carried and the capabilities of the switch hardware, and can require a prohibitively large number of flow rules.

VI. CONCLUSION

OpenFlow can enable highly reactive modern network architectures. However, these architectures place performance demands on the network control-plane. In this work, we measured the performance characteristics of the existing control-plane as deployed in five OpenFlow switches. Additionally, to assist the research community, we have released our test suite [4] to allow evaluation of other switches and configuration details [6] to allow our experiments to be reproduced.

We found that the switches we tested are unsuitable for highly reactive architectures or features such as BFD. For example, the highest performing switch, which was designed for OpenFlow, can process 5,145 packet-in messages per second, representing less than 3Mbps of traffic. Four of five hardware switches tested exceeded a mean packet-in delay of 50ms when overloaded, while the E3500, which maintained <50ms latency, exceeded a 50ms packet-out delay at low packet rates. Control traffic had a negative impact on the installation rate of flow rules on three switches. Continued testing of new switches is required to reassess the slow-path performance as switches mature. However, as the current slow-path is insufficient, OpenFlow applications would benefit from further research into a deployable alternative to the slow-path.

REFERENCES