Experience with 3 SDN controllers in an enterprise setting

Zhiyuan Teo, Ken Birman, Robbert Van Renesse
Department of Computer Science
Cornell University
Email: {zteo, ken, rvr}@cs.cornell.edu

***Abstract*—Interest in OpenFlow and software-defined networks (SDNs) has resulted in a boom in SDN hardware and controller offerings, with varying degrees of maturity, popularity and support. However, few studies have been conducted to investigate the interaction between SDN hardware and software, as well as its impact on controller design and implementation. In this paper, we chronicle our experience with deploying two commodity SDN controllers and a new system, IronStack, of our own design in a production enterprise network at Cornell University, and describe the lessons learnt. We also report on several practical limitations of SDN and controller technology, and detail important future challenges for SDN adopters and developers.**

I. Introduction

Software-defined networking (SDN) has enjoyed popularity in recent years because it makes network hardware amenable to software control methods. Among SDN control methods, OpenFlow [14] is arguably the most prevalent, well-documented and well-understood protocol. The first version of OpenFlow was published in 2009 [15], and the protocol has since been through several iterations [16]–[20] of development and refinement. Today, many hardware vendors offer switching and routing equipment with OpenFlow support. Correspondingly, SDN technology has permeated many enterprises as operators have learned to take advantage of the software control. OpenFlow networks have been deployed to service a diverse range of operating scenarios, including mission-critical commercial applications [12].

The success and excitement surrounding SDNs belies the fact that actual hardware support for OpenFlow spans a wide spectrum. Older OpenFlow-compliant devices often lack the necessary firmware to support some of the more recent versions of OpenFlow. Even among hardware that support the same version of OpenFlow, varying manufacturers, implementations and cost/performance tradeoffs result in different coverage of OpenFlow commands. Furthermore, the OpenFlow specification does not mandate the support of optional commands listed in the standard. Furthermore, some vendors provide non-standard OpenFlow adaptations or extensions [2].

Another issue is that many enterprises do not actually write their own SDN controller software, and view OpenFlow more as a unifying standard than as an opportunity to innovate by creating new specialized control paradigms. Our own research on a new SDN controller we call IronStack focuses on automating fault-tolerance and security for deployments into challenging settings [23]. But in dialog with potential users we often find that the system owner is less focused on features than on convenience and the level of effort needed to actually deploy and manage the solution. Given an easily deployed, easily operated technology, feature coverage and special properties emerge as a secondary goal. Yet in settings like Cornell, where our networks are managed by an in-house professional team, the fear that SDN might be overwhelmingly complex and suitable only for research and experimentation actually dominates the appeal of standardization. Thus until SDN learns to be a user-friendly turn-key story for the SDN manager, it is unclear how the full potential of the technology could be leveraged.

This paper first presents our experience in building and operating a small-scale production OpenFlow SDN from scratch, using market-available hardware and off-the-shelf general-purpose OpenFlow controllers. We also discuss limitations of existing commercial options. We then describe the impact of lessons learned and turn these into recommendations. Finally, we discuss some practical challenges that lay ahead for programmable network technology.

II. Background

Software-defined networking (SDN) is a modern abstraction that allows access to a network switch's routing fabric. In SDN models, the switch's control plane is accessible to an external entity known as a controller, to which all data forwarding decisions are delegated. This control plane has complete command of the data plane, where units of network packets are transferred between physical ports on the switch. There is also some limited capability to transfer packets between the data plane and the control plane.

OpenFlow [14] is the most widely deployed SDN standard today. OpenFlow is managed by the Open Networking Foundation and has seen significant evolution through multiple versions. The most recent version of OpenFlow is 1.5 [19], although many switches marketed as OpenFlow-capable support only OpenFlow 1.0. Successive versions of the standard have increased complexity and are not backward-compatible, necessitating independent firmware support for each version of the standard that a switch hardware supports.

On the software end, there are multiple efforts to develop operational OpenFlow controllers, each with varying degrees of programmability, complexity and performance. Some popular controllers include the open-source POX [8] (a generic Python-based system), Floodlight [4] (Java-based), OpenDaylight [7] and ovs-controller [6] (a C-based reference controller

written by Open vSwitch). Commercial closed-source controllers include the Ericsson SDN controller [3] (derived from OpenDaylight), NEC's ProgrammableFlow Controller PF6800 [5] and Big Switch's Big Network Controller [1].

While SDN controllers are available in many forms for different specific applications, the ones we are interested in for the purposes of this paper are generic enterprise-level controllers for general purpose computing, such as fitting for the access patterns of a campus network.

III. Overview of the Gates Hall SDN

Cornell's Gates Hall SDN comprises 15 high-capacity Dell S4810/S4820 10Gbps switches linking approximately 300 physical hosts over 3 machine rooms and multiple instructional and research labs, providing service to over 1000 students and faculty.

Administratively, the SDN is solely managed by the Information Technology Support Group (ITSG), a team that oversees and supports all engineering IT needs. ITSG does not engage in research, nor in SDN programming as an activity: their role is to ensure that the network operates in a correct, secure, and administratively controlled manner. However, uplink to the general campus network is provided and managed by a different campus-wide organization: Cornell Information Technologies (CIT). CIT requires an L3 isolation router that separates the SDN from the rest of the campus. The L3 isolation router is seen as an emergency kill switch in the event that the SDN interferes with the general campus network. This router is the sole connection to the campus network (by feeding into one of the three main campus routers), and is also responsible for assigning and managing IP addresses for all hosts on the Gates Hall SDN.

Physically, all machines on the SDN share the same switching infrastructure. In order to support Cornell's diverse mix of networking research, the SDN is fragmented into VLANs. Each VLAN is a continuous L2 segment configured with access permissions specific to the usage patterns of its member machines, so membership in a VLAN provides a coarse form of access control. For example, several racks within our datacenter supporting operating systems research require PXE boot and an internal DHCP server within their cluster, yet the cluster itself is not permitted to communicate with the external world. These machines are assigned to a VLAN distinct from the one used to service instructional lab machines which must be accessible remotely over the Internet. Although the principle of VLAN isolation could be considered archaic on an SDN compared to appropriately provisioned rules, it nonetheless provides a convenient point of control at the L3 isolation router, where all SDN VLANs converge.

IV. Hardware slicing

The SDN switches in Gates Hall are a combination of high-capacity Dell S4810 and S4820 switches. These switches are identical except for the physical ports exposed on the front panel: the S4810 switches feature copper SFP (small form-factor pluggable) ports while the S4820 use regular 8P8C (8 position 8 contact) ports. The 8P8C ports are physically more compatible with a wider range of devices, making it substantially easier to connect to commodity Ethernet devices.

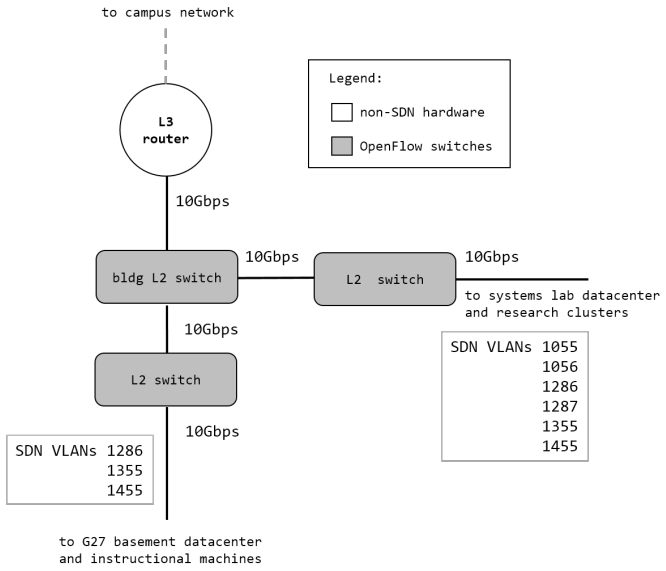

Fig. 1. Topology of Gates Hall SDN.

Both models of switches are capable of being 'sliced'[1] into instances, thereby allowing multiple controllers to operate on logically disjoint portions of the hardware. This is conceptually similar to the virtualization provided by FlowVisor [22], except that the hardware enforces the isolation. Two methods are available for this slicing.

A. Port-based instances

In port-based slicing, a Dell S4810/20 switch may be arbitrarily partitioned into as many as 8 concurrent instances of disjoint ports. Not all ports have to be assigned to an instance. Each instance can be associated with an independent OpenFlow controller, essentially partitioning the switch physically into multiple smaller switches. Using port-based partitioning, network topologies of up to 8 switches can be simulated using a single piece of hardware. This feature has proven useful in many experiments that we have conducted.

Port-based isolation has the advantage that it is easy to set up and intuitive from the physical and OpenFlow controller standpoint. We recommend using port-based instancing for developers or researchers beginning in the field.

B. VLAN-based instances

An S4810/20 switch configured to operate with VLANs in OpenFlow mode can also slice the hardware into instances through VLAN assignments. When operating under this mode, physical ports on the switch are assigned VLAN IDs and marked as tagged or untagged. The tagging status indicates whether a port emits and accepts IEEE 802.1Q frames [24], or regular untagged Ethernet frames. Ports with more than one VLAN ID assignment cannot be marked as untagged.

Up to 8 controller instances may be provisioned this way. Each OpenFlow controller is assigned to manage a set of VLAN IDs, which must be disjoint from other sets of

[1]The term 'slice' first appeared in GENI [10] literature and was used in FlowVisor [22] to describe a similar concept.

VLAN IDs managed by other controllers. From the OpenFlow controller point of view, the viewable set of physical ports comprise those that are assigned to the VLAN IDs under the instance's control. In addition, ingress traffic on VLAN-tagged physical ports are filtered to retain only packets relevant for the set of VLANs managed by that instance, so a controller for a particular instance will only see tagged VLAN traffic corresponding to its assigned set of VLAN IDs. Other VLAN traffic arriving at the switch is either sent to another relevant managing instance or dropped. The S4810/20 hardware automatically enforces VLAN isolation on a hardware level, and no OpenFlow rules are necessary for this enforcement.

VLAN-based isolation is useful in an environment with multiple VLANs and non-OpenFlow switches, when flow rules need to be conserved and/or some hardware oversight is desired to prevent controllers from making mistakes enforcing VLAN isolation. However, this mode of operation is technically non-compliant with the OpenFlow standard and has behavior that can be confusing for people new to OpenFlow. For example, an administrator wishing to create a layer 2 rule that forwards flows from a tagged to an untagged port should specify a match criteria with an Ethernet destination address and a VLAN ID. However, the action set cannot include a directive to strip the VLAN tag (an `OFPT_BAD_ACTION` error would be returned by the switch), even if it seems logical to do so before outputting the packet to an untagged physical port. Instead, the switch performs tagging and untagging automatically. Other operations in VLAN-based isolation mode, such as a flow rule that copies all packets from one physical port to another, may simply not work without any warning or error.

V. Experience with controllers

Our operational experience[2] with OpenFlow SDNs spans about 24 months, of which 4-6 months were spent on hardware familiarization and micro experiments involving isolated switches. With the SDN fully deployed in February 2014, we sliced every switch into 4 VLAN-based instances and ran different controllers on each instance. The first two instances ran production traffic using an open source controller ("Controller A") and a commercial controller ("Controller B") respectively, while the latter two were reserved for research and development purposes and ran our IronStack controller for the full period of the study. Our switch firmware only supported OpenFlow 1.0 at the time the network went into production (1.3 support arrived in spring 2015) so most of our anecdotal experience is based on the older standard. However, we believe that our insights transcend versions and remain relevant.

A. Controller A

Controller A is an open source OpenFlow controller that has enjoyed widespread popularity since its initial release a few years ago. The system is designed to operate in a centralized manner, with all OpenFlow switches directly connected to the controller. Because of this centralized mode of operation, the controller maintains an up-to-date view of the SDN topology, as well as all ancillary switch data (such as flows, port statuses and traffic counts). The web interface offered by the controller allows convenient administration of the network through an intuitive webpage accessed from the control network.

We first encountered trouble on the SDN when we grew our network to approximately 200 hosts. At that scale, we started to experience intermittent performance issues caused by discontinuous hardware flow rules on some source-destination paths. These problems would manifest as high-latency (approximately 500-1000ms), lossy flows alongside other flows that perform well. We determined that packets transiting these discontinuities caused flow-missed events to be raised in OpenFlow, which caused these packets to be encapsulated and forwarded to the controller for processing. To ensure delivery, the controller used software forwarding to copy the packets to their destinations. Our investigations also revealed no capacity problems with the switch hardware table, and we concluded that the rules were simply not being installed by the controller despite continuous flow-miss events resulting from the flow discontinuities. We were able to rectify the problem by manually installing the missing rules on the affected switches.

To find out if the missing flow problem was correlated, we restarted the controller multiple times. We found that controller restarts frequently rectify the problem of missing flows in some source-destination paths, but it did not prevent the same problem from recurring on other paths. Furthermore, the controller removes all hardware flows during a software restart, causing a long period of degraded network operation as the controller repopulates its view of the SDN and falls back on software forwarding in the interim. On our 15 switch network, it takes about 10-15 minutes for this controller to recover after a restart.

B. Controller B

Controller B is a commercially available, proprietary 1U integrated server/OpenFlow controller. It is marketed as a turnkey solution that is simple to use and fast, and the system has received many accolades over the years since it was first available several years ago. The controller is also centralized and provides multiple ways for an administrator to view and manage the network, such as through the command line and over the web. The system is robust and is able to maintain a running view of the operational data and SDN topology.

This controller also experienced scaling issues on our SDN at approximately 200 hosts. Although the controller did not create discontinuous paths, it would sometimes refuse to setup flows for a newly-introduced system. Consequently, the system does not appear on the topological view and does not receive network access. We have also encountered connectivity issues following rapid cycling of a network device's link state: the controller enforces a lockdown period of about 15 minutes before returning the device to active use.

C. IronStack

Length limitations prevent a detailed discussion of our IronStack controller. In brief summary, IronStack is an open source SDN controller intended to offer a turn-key operator experience while imposing a flexible set of security and reliability guarantees at the fabric level, for example by multiplexing traffic across redundant SDN links and encrypted

[2]The authors are not affliated with Dell, the Linux Foundation, or any organization for whose products are mentioned or featured in this paper. The views expressed herein are subjective and not indicative of any product endorsement or criticism.

for protection against intrusion. For our purposes here, the details are not important, because as it turned out, the operator experiences of the ITSG and CIT teams had a far greater role in shaping technology deployment choices than the special features IronStack was actually created to showcase.

Because ITSG and CIT were unable to successfully deploy controllers A and B in stable configurations, for a period of time ITSG actually only used IronStack in the full SDN system. Eventually, as campus network security policies evolved, a decision was made to run IronStack only within our research slices. Thus we have a total of 24 months of experience with IronStack, of which 10 months included our full production network. When IronStack was cut back to research-only use, the entire production workload was shifted to the standard (switched Ethernet) CIT network and off of SDN, highlighting the continuing concerns about SDN stability and manageability in production networking environments.

VI. Lessons learnt

A. The switch-to-controller pipe is thin

One of the first lessons we verified is that the OpenFlow control connection between the switch and the controller is a serious bottleneck. This corroborates with findings from other prior work [22] [9]. On our Dell S4810/20 hardware with TLS turned off, the control connection rarely exceeded a throughput of 2.54Mbps on a dedicated 1Gbps out-of-band network port. This is a few orders of magnitude lower than the maximum speed of the network port, and could not be explained by slow link activity. We found that the bottleneck was due to an overloaded switch processor. The embedded processor runs the Force10 Operating System, a variant of Linux that provides OpenFlow agent support through an application layer.

Because the switch processor is heavily taxed by other scheduling demands, OpenFlow functionality is prone to slowdowns at high loads. This effect is especially pronounced during times of high `PACKET_IN` throughput. `PACKET_IN` events are most commonly generated in response to flow-misses, where a switch forwards a packet to the controller following a failure to find a matching OpenFlow rule. Even on switches with light network traffic, consecutive flow-miss events can quickly overwhelm the CPU, leading to dropped `PACKET_IN` messages, slow OpenFlow throughput and high latencies processing OpenFlow commands on the switch.

`PACKET_IN` events may also be generated in response to an explicit request for flow traffic to be forwarded or copied to the controller. This is helpful in certain circumstances when a controller wishes to discover network state (for example, by snooping on all ARP and DHCP packets). However, flow-miss events will experience contention and be negatively impacted by `PACKET_IN`s received through this method. To minimize flow-miss packet losses, we advise against explicit copying of flow packets to the controller where possible.

B. Consider not flushing rules on restart

Many OpenFlow switches today have a fail-secure mode that allows installed flows to remain on a switch and provide limited operational continuity should the controller be disconnected. Our experience with controllers A and B shows that a complete rule removal on controller restarts is often unnecessary, and can be counterproductive in some situations. Apart from occasional flow discontinuities, the controllers typically regenerate the same rules across restarts. However, manually-inserted rules (such as those used to circumvent flow discontinuities) are lost when all flow rules are cleared.

Because complete rule regeneration from a scratch is a time-consuming operation and SDN controllers are unlikely to be adversarial (by installing bogus, broken, or harmful flow rules for its successor), we recommend against the practice of flushing all flow rules during a controller restart unless there is reason to suspect that correctness may be compromised on a large scale. Rules installed by a predecessor represent the product of some computation or planning and should not be wasted. Instead, we suggest that rules be inherited and verified for preservation on controller startup, and an alternate strategy be used for clearing the flows on the switch if needed.

Clearing the flow table instantly and in its entirety is rarely needed as an emergency procedure. If a genuine need to remove flows arises, we suggest that they be removed one at a time or in small quantities batchwise. On OpenFlow 1.0, the controller can do this by first initiating an `OFPT_STATS_REQUEST` with a request of `OFPST_FLOW` to retrieve a list of all flows on the switch, and then issuing staggered `OFPT_FLOW_MOD` requests to remove flows one at a time. The overall effect is to spread out flow deletes that would immediately cause flow-miss events: should the controller remove a flow that was actually active at the time, the resulting flow-miss packets would be less numerous than if multiple flows were generating flow-miss events in response to a bulk removal request. In turn, the switch is less likely to drop flow-miss events, and the controller can establish new flows more expediently.

On the other hand, if the intention of the controller is to prune unneeded rules without disrupting any flow, it could do so in an unintrusive manner. The controller could identify passive flows by sampling the list of all individual flow statistics retrieved from the switch via the `OFPT_FLOW_STATS` request. Flows that have not seen new packets in a certain amount of time can be deemed to be inactive and individually removed to free up entries in the flow table.

C. Be cognizant of hardware limitations

Although the OpenFlow specification provides a comprehensive array of matching criteria and actions that can be combined in many useful ways, the reality is that these OpenFlow capabilities are limited to what the hardware vendor chooses to support. The OpenFlow switch specification describes the full set of actions that a switch may implement, however switch vendors are only obligated to support actions that are marked as 'required'. In the 1.0 version of the specification, mandatory action support only extends to dropping packets or forwarding to certain ports; useful actions such as packet header field modification and port flooding are optional and may not be available.

Furthermore, even similar actions across different hardware could have various performance characteristics [21] [11]. This non-uniformity of OpenFlow action support and performance can be a source of surprise and frustration to the OpenFlow

developer, who may build generic software controllers and support equipment that become functionally degraded or even completely incompatible with real-world hardware. On the other hand, a developer that targets specific hardware may be exposed to vendor lock-in as it is unlikely that all other OpenFlow hardware will provide a similar level of support, let alone behave identically. This point was an especially acute lesson for us because we had been developing early versions of our OpenFlow controller using Open vSwitch [6] as a reference switch. As a result, we committed substantial time to implementing features that worked well under Open vSwitch but were not well-supported in hardware. For example, at the time of our early prototype, our Dell S4810/20 switch did not support the OpenFlow action to strip VLAN tags and we had to emulate this functionality in software, severely degrading the performance of our system.

D. Equipment-specific features can make a big difference

We attempted to understand the reasons for the scale limits experienced by the controllers we used. The Dell S4810/20 hardware feature multiple forwarding tables. On these switches, flows can be differentiated by types to fall into one of the ACL, L2 or L3 flow classifications. Ordinarily, the multiple forwarding table functionality is disabled and all flows are stored in the ACL (general-purpose) OpenFlow table, which has capacity for only 500 flows. When enabled through an out-of-band command line configuration utility, the switch transparently stores flows matching L2 or L3 classifications into dedicated separate tables. The L2 and L3 flows are particularly compelling because they feature deep tables well-suited for common switching and routing tasks, freeing up valuable ACL table space for more unusual flow rules. Table 1 shows the respective flow table capacities on the S4810/20 switches, while Tables 2 and 3 show the respective syntaxes of the L2 and L3 flows [2].

Table name	Flow capacity
ACL	500
L2	48000
L3	6000

TABLE I. FLOW TABLE CAPACITIES ON DELL S4810/20 SWITCHES.

Parameter type	Parameters
Match criteria	• `dl_vlan` (input VLAN ID). • `dl_dst` (destination Ethernet address). • all other fields must be wildcarded.
Actions	• `OFPAT_OUTPUT` output to a single physical switch port.

TABLE II. L2 FLOW CLASSIFICATION.

Parameter type	Parameters
Match criteria	• `dl_dst` must be set to the switch port's Ethernet address. • `dl_type` must be set to 0x0800. • `nw_dst` can be optionally set. • all other fields must be wildcarded.
Actions	• `set_dl_src` must be set to switch port's Ethernet address. • `set_dl_dst` (destination Ethernet address). • `OFPAT_OUTPUT` output to a single physical switch port.

TABLE III. L3 FLOW CLASSIFICATION.

Our investigation showed that both Controller A and Controller B installed rules that did not fit into either of the L2 or L3 flow classifications. Instead, those flow rules were placed into the ACL table, which prevented the network from scaling once the table was full and no new flows could be created. We discovered that the reason for using ACL entries was because the controllers were unaware of the syntax or availability of the L2 and L3 tables, which prevented them from taking advantage of the deep tables. In contrast, IronStack did not make substantial use of ACL entries at all because the policies ITSG sought to support were mostly simple enough to be expressed in L2 flows.

E. Non-standard behavior is standard

Specification-deviating behavior may come as a surprise for developers who assume that hardware marketed as OpenFlow-compliant will exhibit features and functionality exactly as written in the OpenFlow standard. This can be an issue for developers who build their controllers on reference switch implementations (eg. Open vSwitch) and later deploy them on actual OpenFlow hardware.

Non-standard hardware behavior has caught us by surprise on a number of occasions. On our Dell S4810/20 switches, flow priority is only honored within entries in the ACL table; the priority field is completely ignored for flows that fit in the L2 or L3 tables. This posed a problem for our controller, as it had to be aware of these equipment subtleties and install flows into the ACL table if it wished to override specified flows in the L2 or L3 tables.

As another example on our hardware, L2 flows are not instrumented with packet or traffic counters, which limits their utility in network analysis. This forces the developer into a dilemma between the creation of an instrumented ACL flow on a capacity-limited table, or an uninstrumented L2 flow on a large table. To complicate the situation, L2 flows cannot be configured with arbitrary idle timeouts; these flows are either permanent (if the idle timeout value is specified as 0 in the flow rule), or set to some switch preset value (if the specified dle timeout value was non-zero). Since there are no indications of warnings or errors when L2 flows are installed with arbitrary timeouts, an equipment-agnostic controller may mistakenly assume the flow to time itself out accordingly.

On the L3 table, flows are permanent and do not honor any specified idle timeouts. Similar to L2 flows, no information is given by the hardware to indicate that the idle timeout is ignored on an L3 flow.

While these non-standard behavior are generally innocuous quirks, they make it difficult for a developer to write a general-purpose OpenFlow controller that exhibits predictable behavior across different hardware. For example, an OpenFlow controller used to drive a commercial pay-per-use network might rely on L3 flow timeouts to redirect a customer to a captive portal when a lease time expires. Without equipment-specific knowledge of non-standard flow timeout behavior, general-purpose controllers may not correctly enforce customer access policies. In the case of an L2 flow, a software workaround for the idle timeout may not even be possible since it offers no packet counters that can be used to track flow activity.

F. Configuration tools are just as important as OpenFlow

Although OpenFlow presents a useful interface through which a switch may be controlled, the specifications omit a discussion of equipment configuration tools or utilities because they are often vendor and equipment-specific. Equipment configuration tools provide the means to set up operating parameters that are not necessarily controllable from or related to OpenFlow. For example, the IP address and port number of a controller must be specified to the switch before it can establish an OpenFlow connection to the controller. Other important functionality that are only accessible through equipment configuration tools may include means to power cycle the device, set up VLANs and port tags, as well as enable specific OpenFlow tables or slicing features. On most OpenFlow switches that we are aware of, the configuration tool presents itself as a command line interface physically accessed through the switch's serial (RS232) port. Because OpenFlow switches have their own IP addresses on the control network, they often also support access to the same configuration tool over telnet or SSH.

We take the view that control over these functionalities is just as important as OpenFlow itself in a comprehensive network controller, particularly if the configuration utilities also provide information or indirect control over OpenFlow capability on a switch. As an example relevant to our SDN setup, a network controller should be able to inspect the switch configuration for VLAN or port information pertaining to its slice. It should also be able to control deep table options or reconfigure the IP address and port that the switch seeks a controller at. We are presently unaware of controllers that can perform equipment configuration along with OpenFlow.

G. Switch misconfiguration can cause confusion

Even with compatible controllers designed to correctly take advantage of the various hardware tables available, it may still not be apparent to the controller that the hardware tables are indeed being used. On the Dell OpenFlow switches, an external configuration utility must be used to manually enable the switch to operate in 'multiple-flow' table mode, and then the individual L2 and L3 flow maps have to be enabled in order for flows specified in the L2/L3 formats to be stored into their high capacity hardware tables. Without the features enabled (which is the default), these flows would be stored in the ACL table.

On our SDN setup, there is no way for a controller to verify that the flows have been stored into the right hardware table, since OpenFlow flow statistics from our Dell hardware do not annotate flows with their flow table IDs. Furthermore, the OpenFlow specification does not provide a way to obtain table capacities, which precludes the ability for a controller to make an informed decision with respect to flow installation. The only way that a controller can tell if a table is full is when the error `OFPMFCTABLEFULL` is generated in response to a flow installation command. By this time, it may be too late for the controller to take remedial action. On the other hand, such table capacities are usually advertised by the vendor or otherwise accessible from the switch configuration utility. We believe that knowledge of table capacities should be propagated to the controller to aid planning purposes.

H. Isolation is not perfect

Virtualization on an OpenFlow switch provides logical isolation between one controller's traffic from another. Virtualization techniques can be software-based, as in FlowVisor [22], or hardware-based, as is provided directly on the Dell S4810/20 switches. In either case, it is important to note that virtualization does not provide perfect resource isolation between controllers. Resources such as the embedded CPU, forwarding table capacity and bandwidth are shared across all controllers working on the switch.

This is a well-known phenomenon with all virtualization techniques, and is neither a flaw nor bug in the virtualization mechanism. However, controllers should be designed with the assumption that they could be run on virtualized hardware, and thus engage in better cooperative sharing tactics. For example, controllers could minimize flow table wastage by setting an idle timeout on their flows, while also not consuming excessive switch computational power by indiscriminately copying data plane packets to the controller.

I. No controller-to-data plane communications

One feature that we would have liked to see in a SDN controller is the ability to communicate directly with networking applications that run on the data plane. At present, we are not aware of any controllers that have such a capability, except for Google's B4 [12] Routing Application Proxy, which bridges packets from the Quagga control plane and the switch's data plane. The ability for a network application to communicate directly with the controller opens up substantial development opportunities. For example, the controller can host a HTTP subsystem that serves users statistics about their network usage, or provide a webpage through which network QoS could be requested. A data plane presence on the network also allows the controller to easily implement features such as captive portals.

VII. Performance

SDN performance should be understood as having two complementary aspects. We tend to think about SDN switches and routers in terms of end-to-end flow performance, and in our experience, this aspect of performance was completely satisfactory: Controller A and Controller B both achieved their rated speed and successfully support the Gates Hall use patterns. Less well appreciated is the degree to which controller performance turns out to shape operational experience. Here, our experience has been more complicated.

When working with the vendor-supplied controller software, the many limitations and issues cited earlier combined to make it impractical to actually use them operationally for our scale of use cases. In contrast, we have been successful in operating the Gates Hall SDN configuration using our IronStack controller. Table 4 summarizes the numbers of active rules and includes some basic performance metrics gleaned from this effort.

The Gates SDN is an operational network used for Cornells research and teaching, we were not able to isolate the system and conduct stress tests on our controller or the network. However, we do hope to create an isolated research subnetwork

Metric	Value
Total rules	280 L2 rules, 4 ACL rules
Peak CPU usage	15.7%
Average CPU usage	1.3%
Reactive rules created/sec (peak)	25
Average switch echo response time (sec)	22ms
Maximum `PACKET_IN` throughput	2.54Mbps

TABLE IV. MICROMEASUREMENTS OF OUR OWN CONTROLLER ON A 8-CORE INTEL XEON E5405 CLOCKED AT 2GHZ.

OpenFlow switch model	Generic flow capacity	Other tables available
Dell S4810/20	500	L2, L3
Dell N2048	896	L2, VLAN
NEC PF5820	750	L2
Pica8 P3297	8000	L2, L3
Brocade MLX	4000	L2, L3, L23

TABLE V. OPENFLOW TABLE CAPACITIES OF SOME EQUIPMENT.

in the coming months, which would then permit us to engage in the form of more microbenchmarks that might shed deep light on the scalability of our solution and the potential for deployment of SDN in networking in larger campus configurations. The Gates Hall experience, modulo the difficulties we had with off the shelf controller software, actually encourages us to believe that larger SDN configurations should certainly be feasible, and our hope is to explore that option in future work.

VIII. CHALLENGES AHEAD FOR SDN

Going forward with SDN, several important challenges remain to be addressed:

A. Switch CPU performance

The most immediate concern facing SDN technology is the disparity in computing power between the switch processor and a controller. As our experience shows, the embedded switch processor is typically undersized and is often responsible for the bottleneck between the controller and the switching fabric. This bottleneck becomes more pronounced as successive versions of the OpenFlow standard impose additional complexity upon the embedded processor.

B. Capacity of rule tables

Another issue facing SDN hardware is its relative scarcity of general-purpose flow entries. Compared to traditional switches, general-purpose 12-tuple OpenFlow rule matching consumes more expensive ternary content address memory (TCAM) and therefore offers less entries for the same amount of TCAM space. Even with recent advances in OpenFlow TCAM storage efficiencies, the number of available general-purpose entries on most switches today is generally no more than 2000, with many switches offering less than 1000 (see Table 5). This is an order of magnitude lesser than consumers are used to with traditional hardware, and is often perceived to be a limiting factor in scaling a network. The problem is somewhat alleviated by dedicated tables that can be used to soak up commonly-installed flow types, however the shortage of cheap TCAM for general-purpose OpenFlow rules will continue to be an impediment for some time.

C. Non-standard behavior

While non-standard behavior is generally tolerated as vendor differences between hardware companies, the reality is that non-conformance to standards makes it difficult for generic OpenFlow controllers to be written without introducing substantial conditional code or a complete driver layer. The increasing number of OpenFlow hardware vendors, coupled with the growing complexity of OpenFlow standards, increases the risk of emergent vendor-specific behavior that can negatively impact controller development.

IX. RELATED WORK

Most of our unusual findings about non-standard OpenFlow switch behavior were serendipitous discoveries in the course of trying to build an operational SDN controller. Kuzniar et al [13] performed investigations on a number of OpenFlow switches to characterize the interaction between the control plane and the data plane. The authors uncovered a substantial amount of surprising behavior, including temporal locality behavior in switch updates, performance degradation caused by priority fields, non-atomic rule modifications on a switch and even incorrect OpenFlow barrier behavior. Although their discoveries were made on different switches, it supports our hypothesis that unexpected, standard-deviating behavior is a phenomenon that should be taken seriously by developers.

DevoFlow [9] examines the various causes of latency inherent to OpenFlow and describes the negative impact of flow table size and statistics collection on OpenFlow performance. They then prescribe and verify more efficient methods to install flows and gather statistics by minimizing interactions with the SDN control plane. While their choice of OpenFlow hardware was different, our experiences were largely similar.

The OFLOPS [21] framework recognizes diversity in the performance and implementation of OpenFlow switch firmware, and characterizes their behavior and performance under a variety of test cases. These characteristics can be used to model switch behavior more accurately than testing on reference implementations of OpenFlow, such as Open vSwitch [6]. Interestingly, their work also uncovered bugs in the implementation of barriers on switches.

Danny et al [11] studied the similar problem of trying to emulate specific vendor performance characteristics with respect to control path delays and flow table usage. They were able to improve the accuracy of switch emulation to a high degree of accuracy across multiple vendors, which can potentially aid developers trying to test their controllers on a variety of hardware without access to the hardware themselves.

NOSIX [25], is a proposed solution to provide better standardization across diverse OpenFlow switch hardware. The authors describe a uniform, portable abstraction layer for SDN controller development through the use of virtual flow tables and vendor-provided switch drivers. Controller developers then specify their requirements for rule processing and make promises about their usage of the virtual flow tables.

Unfortunately, NOSIX is not currently in widespread use and few switch vendors have supplied their switch drivers.

X. Conclusion

In this paper, we presented our observations and findings from deploying readily-available OpenFlow controllers on our SDN. Through operation of these controllers, we identified a number of important issues with SDN deployment and OpenFlow controller design. The paper concluded with some of the challenges that continue to hinder SDN adoption at a larger scale.

Acknowledgements

We are particularly grateful to our ITSG team, especially Scott Yoest, Dave Juers and Dora Abdullah for supporting our work.

References

[1] Big Network Controller. http://bigswitch.com/products/SDN-Controller.

[2] Dell S4810/Z9000 Software-Defined Networking Deployment Guide Version 1.0. https://www.force10networks.com/CSPortal20/KnowledgeBase/DOCUMENTATION/InstallGuidesQuickrefs/SDN/SDN_Deployment_1.0_28-Feb-2013.pdf.

[3] Ericsson SDN Controller. https://www.sdxcentral.com/products/ericsson-sdn-controller/.

[4] Floodlight OpenFlow controller. http://www.projectfloodlight.org/floodlight/.

[5] NEC ProgrammableFlow PF6800 controller. http://www.necam.com/sdn/doc.cfm?t=PFlowController.

[6] Open vSwitch. http://openvswitch.org/.

[7] OpenDaylight Project. https://www.opendaylight.org/.

[8] POX. http://www.noxrepo.org/pox/about-pox/.

[9] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review* (2011), vol. 41, ACM, pp. 254–265.

[10] Elliott, C. GENI-global environment for network innovations. In *LCN* (2008), p. 8.

[11] Huang, D. Y., Yocum, K., and Snoeren, A. C. High-fidelity switch models for software-defined network emulation. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 43–48.

[12] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 3–14.

[13] Kuzniar, M., Peresini, P., and Kostic, D. What you need to know about sdn control and data planes. Tech. rep., 2014.

[14] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[15] ONF. OpenFlow Switch Specification 1.0.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[16] ONF. OpenFlow switch specification 1.1.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf.

[17] ONF. OpenFlow switch specification 1.3.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[18] ONF. OpenFlow switch specification 1.4.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[19] ONF. OpenFlow switch specification 1.5.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[20] ONF. OpenFlow switch specification 1.2.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf, 2011.

[21] Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., and Moore, A. W. Oflops: An open framework for openflow switch evaluation. In *Passive and Active Measurement* (2012), Springer, pp. 85–95.

[22] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep* (2009).

[23] Teo, Z., Kutsenko, V., Birman, K., and van Renesse, R. Ironstack: Performance, stability and security for power grid data networks. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014* (2014), pp. 792–797.

[24] Thaler, P., Finn, N., Fedyk, D., Parsons, G., and Gray, E. Ieee 802.1 q.

[25] Yu, M., Wundsam, A., and Raju, M. Nosix: A lightweight portability layer for the sdn os. *ACM SIGCOMM Computer Communication Review 44*, 2 (2014), 28–35.