Macroflows and Microflows: Enabling Rapid Network Innovation through a Split SDN Data Plane

Rajesh Narayanan, Saikrishna Kotha, Geng Lin
Dell Inc

Aimal Khan, Sajjad Rizvi, Wajeeha Javed, Hassan Khan, Syed Ali Khayam
xFlow Research, Inc.

ABSTRACT

In this paper, we empirically quantify inherent limitations of merchant switching silicon, which constrain SDNs' innovation potential. To overcome these limitations, we propose a *Split SDN Data Plane (SSDP) architecture*—a new switch architecture which allows rapid network innovation by complementing a cost-effective, yet inflexible, frontend merchant silicon switch chip with a deeply programmable co-processor subsystem. We implemented SSDP on a prototype Dell PowerConnect platform with a programmable multi-core data plane subsystem. To demonstrate SSDP's potential, we developed diverse real-world cases on the prototype platform. Benchmarking results show that, while delivering on its rapid innovation promise (with significantly shorter turn-around time), a SSDP architecture also provides reasonably high switching rates on deep flow tables.

1. INTRODUCTION

Merchant silicon is designed to provide common switching functions at fast data rates and at commodity prices. Contemporary merchant switch chips have definitely served this purpose and most switch vendors are now locked into merchant silicon, at least for the lower end of their switch product lines. This commoditization, however, comes at the cost of feature flexibility and richness. As merchant silicon cannot support new traffic processing functions (e.g., security, DPI, sophisticated sampling, etc.), a range of middle-boxes and Network Processing Units (NPU) based solutions have emerged to fill these gaps. However, these point solutions are expensive to procure, deploy and manage.

The recent advent of software-defined networking has led to an active debate on the role of merchant silicon in the SDN ecosystem. SDNs offer the promise to consolidate evolving traffic processing functions in the control plane, which subsequently programs the data plane accordingly. However, current SDN definitions (e.g., OpenFlow) lie in an awkward middle-ground with merchant silicon on one end and NPUs/middle-boxes at the other end.

Based on the previous merchant hardware benchmarking [10][11][12], we argue that SDN's rapid innovation promise is constrained by current hardware's scale and flexibility limitations. In this work, we first establish a baseline by benchmarking prominent merchant silicon switch ASICs. To this end, we port the widely-used Open vSwitch (OVS) [5] software stack to merchant platforms and benchmark different performance metrics with an aim to establish a baseline and to corroborate the existing findings that the existing merchant hardware cannot implement SDN data plane functions

Our benchmarking results identify four fundamental limitations of merchant hardware which have the most adverse effect on SDN's rapid innovation promise: a) Small sizes of useful tables which can implement SDN data planes; b) Lack of flexible actions support; c) Slow bus speeds between the hardware pipeline and the on-chip CPU; and d) Slow on-chip CPUs which cannot complement the hardware tables with deep software-based tables.

This motivates us to investigate an alternative SDN switch architecture in which the forwarding fast path is split into two distinct data paths: a TCAM-based path in merchant silicon complemented by an NPU-driven software subsystem connected to merchant hardware with a standard high-speed bus. This architecture henceforth referred to as the *Split SDN Data Plane* (SSDP), stores elephant or macroflows[1] in the TCAM, while microflows (that need L4 and higher layer processing) are encoded into the NPU-based subsystem. Since SSDP does not require any advancement in switch silicon, it leverages the low cost of merchant hardware in combination with the scale, flexibility and programmability of multi-core programmable ASICs.

We implemented SSDP on a prototype Dell PowerConnect platform with a merchant-silicon switch chip and a programmable subsystem. Benchmarking results show that for MTU-sized packets the SSDP prototype can improve the switch-to-CPU bandwidth from 24 Mbps (maximum for merchant silicon evaluated in this paper) to 8.5 Gbps, and OpenFlow control channel bandwidth from 610 Kbps to 276 Mbps. Also, the SSDP prototype can provide switching rates of 4.5 Gbps on an OpenFlow1.0 table with up to 60K entries.

[1] *Macroflows'* represent an aggregated flow while a *'microflow'* is a more granular flow. For instance, a `<vlan, dst-ip>` flow entry represents a *macroflow*, while a `<vlan, src-mac, dst-mac, dst-ip, ip-proto, src-port, dst-port>` represents a *microflow*. The terms *macroflows* and *microflows* should be read as, the number of macroflow-entries and number of microflow-entries in the system

The SSDP architecture offers an efficient, extensible and economically-viable data plane architecture which can be used to enable many interesting SDN use cases. As proof-of-concept, we discuss and implement security, DPI and encryption use cases on the SSDP prototype. Based on our findings, we argue that the SSDP architecture opens up new dimensions in the ecosystem that can be leveraged to realize SDNs' promise of rapid network innovation.

2. BENCHMARKING MERCHANT SILICON SWITCH CHIPS

Recent studies have discussed bottlenecks of merchant silicon in implementing SDNs, and provided some benchmarks of existing switching chips [10][11][12]. However, we generated benchmarks that are used to establish a baseline so we can quantify the performance gains of the SSDP architecture. Therefore, we first evaluate the viability of implementing an OpenFlow 1.0 data plane on merchant silicon. We then benchmarked two Reference Design switch chips having 24 ports each of 1Gbps (S_1) and 10Gbps (S_2), respectively. These chips are offered by leading semiconductor vendors and are being shipped in many commercial products. To maintain IP confidentiality, throughout the paper we do not disclose the vendor names or any critical architectural details.

2.1 System Implementation

We needed consistent fast path OpenFlow implementations on the merchant silicon switches for judicious evaluation of the technical benchmarks mentioned above. To this end, we ported the Open vSwitch (OVS) software stack (version: 1.1.2) [5] to the on-chip switch CPUs. OVS is a multilayer virtual switching platform designed to operate as a soft switch running within a hypervisor. OVS provides a fully-compliant OpenFlow 1.0 software implementation.

OVS' architecture and high-level porting information is shown in Figure 1. We ported the `netdev` and `dpif` layers on the switch CPUs using Linux running on merchant silicon device drivers. The `netdev` layer managed the hardware interfaces, while the `dpif` layer communicated with the policy TCAM table. Packets that did not find a hit in the TCAM table were forwarded directly to OVS stack running on the switch CPU. The `ofproto` layer in OVS communicated with an OpenFlow 1.0 controller and, on receiving a `flow_mod` request, it writes the rule to the switch's TCAM.

2.2 Benchmarking Results

Table 1 shows that, while the switch chips implement a full bisection bandwidth fast path, packets that do not match the TCAM are forwarded to the slow path CPU for S_1 at a maximum rate of 19.2 Mbps. This bandwidth drops to as low as 324 Kbps when the packet is propagated through the OVS software stack to the controller. For unbiased controller bandwidth benchmarking, we assume a zero-latency controller and only benchmark the data rate for the packet path '`netdev-receive`→ `ofproto-layer`→ `netdev-send`'.

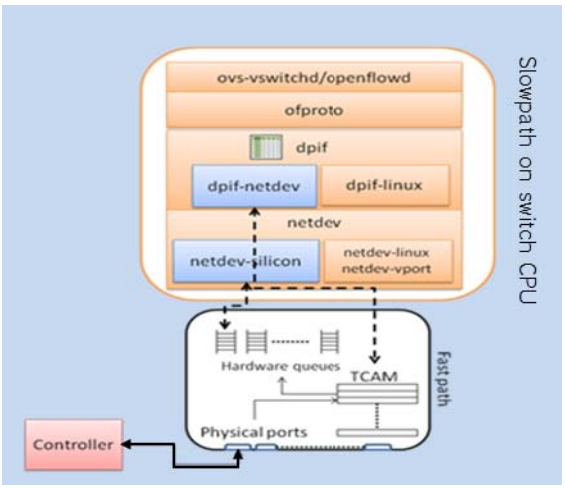


Figure 1. Open vSwitch on merchant silicon.

Increase in CPU speed increases the bandwidth to the controller (610 Kbps), but this bandwidth is still quite low for scalable SDN use cases. Nevertheless, this observation provides us an insight that a faster CPU should be able to process controller communication at a faster rate and consequently increase switch to controller bandwidth.

From the flow table aspect, the only chip resource that can effectively satisfy OpenFlow 1.0's 'flow-match' requirements is the TCAM-based policy table[2]. Since TCAMs are expensive, and power-hungry, they cannot be used to store large forwarding tables. This is substantiated by Table 1 which shows that a maximum of 1500 OpenFlow 1.0 entries can be stored in the TCAM.

Table 1: Chip Parameters and Performance Benchmarking on Merchant Silicon (bandwidth metrics on MTU-sized packets)

	S_1	S_2
CPU Speed	800MHz	1000MHz
Max bandwidth to CPU	19.2 Mbps	24 Mbps
Max bandwidth to controller	324 Kbps	610 Kbps
Policy TCAM size (# of entries)	1500	1500

Switch chips evaluated in this study allowed dual TCAM lookups on each packet to resolve the Cartesian product problem [4]. However, only one header modification was allowed in the packet thereby making the chips unsuitable for flexible SDN implementations. Moreover, OpenFlow actions—especially those requiring packet modifications—are not supported in merchant hardware today [4]. Since prior work [10][11][12] reported similar numbers, we can safely assume that similar results will hold for low-end merchant silicon switching ASICs.

2.3 Discussion

Our benchmarking results confirm the findings of recent studies that merchant silicon has four fundamental hardware limitations: ***a) slow CPUs, b) small flow tables, c) slow bus between hardware pipelines and on-chip CPU, and d) limited actions support***. With these limitations, it is

[2]While some optimizations are possible—e.g., using tunnel tables or Forwarding Database (FDB) tables,—these tables have limited sizes and cannot provide fine-granular traffic control.

simply not possible to implement a flexible and programmable data plane on merchant silicon. It can be argued that many of these problems can be mitigated by introducing higher-end CPUs, buses and memories in merchant silicon. Such solutions, however, compromise the commoditization advantage of merchant silicon.

Existing solutions either solve the programmability problem (deep flow tables and flexible actions support) in software while compromising throughput and scalability, or attempt new hardware enhancements to merchant silicon.—see [4] and references therein. This leaves us with two obvious choices: a) Wait for merchant hardware to evolve and meet the scale, flexibility and programmability requirements of SDNs; or b) Revert to software-based solutions.

We argue that a middle-ground is possible where we complement the cost and line-rate forwarding advantages of merchant silicon with the programmability offered by NPU-based software implementations. In the next section, we propose a new switch architecture which enhances merchant silicon with NPUs to leverage these advantages.

3. SPLIT SDN DATA PLANE FOR RAPID NETWORK INNOVATION

In this section, we propose a new Split SDN Data Plane (SSDP) architecture which can facilitate realization of the rapid innovation potential of SDNs. We also show that this architecture also improves the Switch←→Controller traffic by at least two orders of magnitude. The SSDP architecture leverages the observation that in many deployment scenarios all flows in a network do not require line rate processing. Hence, macroflow-level traffic can be managed by merchant silicon switches, while microflows can be handled in a software programmable backplane or subsystem as shown in Figure 2.

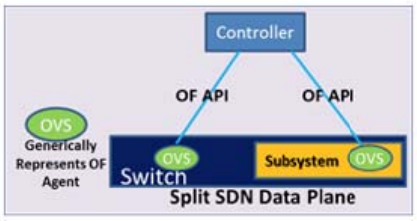


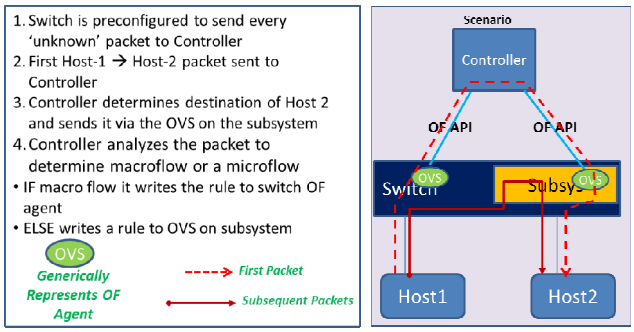
Figure 2.Block diagram of the SSDP architecture.

The SSDP architecture uses a merchant silicon switch chip at the frontplane which, similar to traditional switch deployments, makes extensive use of wildcarded macroflows in the TCAM table. The frontplane is also connected to a series of NPU coprocessors in the backplane subsystem to manage microflows. Thus, instead of the two extreme choices (all software versus all hardware), SSDP chooses a middle-ground to simultaneously leverage the benefits of low-cost merchant hardware and the deep programmability support of software.

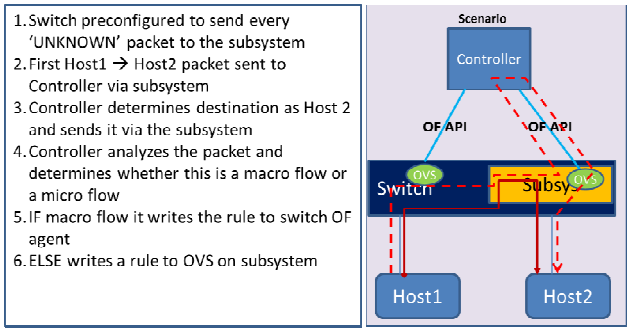
3.1 Packet Walks

In this section, we explain two different possibilities for packet walk in an SSDP system. **Figure 3**(a) and (b) show different implementations of the SSDP architecture with (b) being progressively more optimized.

- **Figure 3**(a) shows a simple SSDP implementation which overcomes the limitations of merchant switch by forwarding all unknown packets (i.e., packets without a macroflow rule in hardware tables) to the subsystem.
- On further benchmarking, we found that this method also significantly improves the switch-controller latency for unknown packets as shown in **Figure 3** (b).



(a) Unknown packet through switch slow path



(b) Unknown packet through subsystem

Figure 3. Packet walks on an SSDP switch.

We find that **Figure 3**(b) is the best architecture for reducing latency while leveraging the programmable NPU module to develop new dataplane applications. At the same time it is fully compliant with OpenFlow 1.0 standard specifications and is future-proof to the newer versions of OpenFlow standard. The packet-walk also exposed a problem where port-information is lost when the packet hits the subsystem. There are different methods to solve this effectively in a deployment scenario. The focus of this paper however is to validate the architecture, understand the dependencies, and report associated performance/scalability benchmarks.

Additionally, as a highly optimized architecture we suggest that in long term it may be best to have a single OpenFlow (in this case OVS) agent on the subsystem communicating with the controller. Thus the agent could directly modify the switch forwarding plane via an RPC module as shown in Figure 5.

3.2 SSDP Implementation

As a PoC, we implemented SSDP on a Dell PowerConnect 7024 platform. This platform uses a 24x1Gbps switch chip. The switch ASIC comes equipped with a XAUI interface

which was used to connect a programmable subsystem (based on a 4-core MIPS, with encryption and TCP-acceleration blocks) to the PowerConnect switch. Packets that did not hit the switch TCAM were forwarded to the XAUI interface while bypassing the switch CPU.

We ported the OVS code base to the programmable-card to create an OpenFlow 1.0 subsystem, henceforth referred to as the Programmable Subsystem (PS). In addition to the `netdev` interfaces, many architectural modifications were implemented in the OVS code base to allow it to scale its performance with an increasing number of cores. Furthermore, the hardware accelerators on the PS were used to support encryption, tunneling and QoS in hardware.

We implemented the architecture shown in **Figure 3**(b), where OVS on the PS supports a software flow table, and a separate OF-agent programs the frontend merchant silicon ASIC. The information about whether a particular flow should be encoded in the TCAM or software tables was communicated by the controller to the OVS on the PS. Flow modification messages from the OpenFlow controller were evaluated by the PS and macroflows (i.e., flows with a number of wildcarded entries) were encoded on the 24x1 Gbps chip's TCAM, while microflows were added to the PS' flow table.

3.3 SSDP Benchmarking

To quantify the value offered by the SSDP architecture, we benchmark the PowerConnect prototype on the following metrics: bandwidth from merchant fabric to back-plane CPU, bandwidth from back-plane CPU to controller, and packet switching performance of the back-plane.

Ten Linux machines were connected with the PowerConnect ports for traffic generation. These machines generated traffic at different rates. As mentioned earlier, the switch was programmed to send all incoming traffic to the PS through the XAUI interface. To measure the bandwidth to the controller, we kept the flow table of the subsystem empty, resulting in each packet arriving at the PS to be forwarded to the controller after a flowtable miss. The packets per second sent to the controller were measured in OVS just before forwarding the packets to the controller. Considerable improvements are also observed on the control channel bandwidth (Table 2, column 3). For MTU-sized packets, the SSDP prototype can yield a data rate of 276 Mbps as opposed to a few hundred Kbps that were achieved on the switch CPU. Again, there is degradation in the control channel throughput with a decrease in packet size. However, SSDP's control throughput stays at least an order of magnitude higher than the control bandwidth of merchant silicon.

Finally, we preconfigured the OVS agent on the PS with a fixed number of flows, with OpenFlow actions of forward to a given port. To measure the overhead of flow matching and performing the associated action, the packets were counted then silently dropped instead of actually being sent back out the XAUI interface. This allows us to measure the total packet processing capacity of the backplane without hitting the limit imposed by the 10Gbps link connecting the front and back planes.

The results for 10K and 60K flows are shown in Table 2. For MTU-sized packets matched on a 10K table, SSDP is able to achieve a switching data rate of 4.49 Gbps. The switching rate drops to 2.93 Gbps for a table with 60K entries. There is significant degradation in switching rates with a decrease in packet size. However, the PS can offer switching rates close to 1Gbps for 300 byte packets. In summary, the SSDP prototype can mitigate the bus speed, CPU, table size, and action limitations of merchant silicon.

The switching rates are low for very small-sized packets. However, for larger packets (>300 bytes), reasonable switching rates can be achieved on deep flow tables. Three observations make SSDP's switching rates reasonable in practical deployments: 1) the entire switch traffic does not need to pass through the slow path; 2) macroflows are devolved in the merchant TCAM tables; and 3) while meeting the switch's power and cost-per-port constraints, the SSDP architecture allows the programmable subsystem to be enhanced with more and more cores to increase its packet switching rates.

Table 2: SSDP Benchmarks

Packet Size (bytes)	Bandwidth to subsystem CPU	Bandwidth to Controller	Flow Table Size vs. Pkt. Processing Throughput	
			10K flows	60K flows
1500	8.5 Gbps	276 Mbps	4.49 Gbps	2.93 Gbps
700	2.8 Gbps	198 Mbps	2.65 Gbps	2.04 Gbps
300	995 Mbps	72 Mbps	986 Mbps	809 Mbps
64	227 Mbps	33 Mbps	217 Mbps	164 Mbps

4. SSDP USE CASES

In this section, we describe three representative use cases that we have implemented on the SSDP-PowerConnect prototype. The first two use cases have been designed to stress-test the flow-scalability of the PS-based OF-subsystem. We also implement one use case that leverages the subsystem's encryption block. This is designed to show the flexibility of actions that can be achieved by leveraging OF-extensions that call ASIC-specific functional blocks.

4.1 P2P Detection and Prevention

Campus networks face a significant challenge in detection and blocking of p2p file sharing traffic, which is typically achieved using expensive and centralized DPI boxes. P2p applications sporadically generate a large number of microflows from a single host. Blocking such large number of flows requires deep flow tables that need to be dynamically available close to the switch (a concept we call ***flow-bursting***), and makes it an ideal candidate to demonstrate and benchmark the SSDP.

To make a demonstrable use case, we ported the OpenDPI [www.opendpi.org] code base to the Beacon[1] controller. We then invoked OpenDPI's BitTorrent traffic classification detection algorithm in Beacon and populated the SSDP with drop and forward rules for flows which were detected as p2p and non-p2p, respectively (Figure 4).

Table 3 shows some sample flow entries (output of OVS' `OFCTL` command) from the microflow table for the p2p traffic blocking use case. The first entry (with higher L4 port numbers) belongs to p2p traffic. Initially this flow hits the default entry (`Action= Output: Port 26`, which is the XAUI port in the PowerConnect) in the macroflow table which directs it to SSDP. Since no entry exists against this flow, it is sent to the controller. OpenDPI at the controller processes packets from this flow and continues forwarding until it has classified this flow. Once it discovers the flow as p2p, the drop rule is written in the microflow table. Similarly, the second flow is classified as HTTP flow and a microflow entry is installed to forward this flow. Similarly, the second flow is classified as HTTP flow and a microflow entry is installed to forward this flow. Additionally a wildcarded rule can be installed in the macroflow table to handle all the HTTP traffic in the fast path to reduce traffic to SSDP.

Table 3: Sample Flows from Microflow Table.

Flow ID				Actions
L3_src	**L3_dst**	**L4_src**	**L4_dst**	
58.65.18.20	192.168.1.126	51413	44283	drop
173.193.6.3	192.168.1.126	80	33133	output:1

4.2 Intrusion Detection

Compromised user-owned devices are an increasing threat in the campus/enterprise network due to their ability to launch network intrusion. The SSDP architecture is suitable for traffic intrusion detection because: a) it requires microflow-level control over traffic; b) intrusion detection algorithms are continuously evolving; and c) not all packets of a flow require intrusion detection system (IDS) processing.

The Threshold Random Walk (TRW) anomaly detection algorithm [3] was ported to the Beacon controller. We collected and used a labeled traffic dataset to evaluate and forward the traffic. The TRW algorithm maintains state about success rates of every host's new connection requests. These success and failure rates are leveraged in a likelihood-ratio-testing framework to flag a host as malicious or benign. Once this determination is made, we make the SSDP flow tables are populated for subsequent packets from the host. The controller first encodes microflows to block individual flows from a host. Once the anomaly score for a host exceeds a certain threshold, a macroflow for the host is installed in the switch TCAM.

4.3 On Demand Flow Encryption

This use-case leverages ASIC-based hardware accelerators, like encryption blocks. Figure 4 shows two hosts A and B reaching the same storage target. Host-A requires a secure backup versus a non-secure backup for Host-B.

To implement this we extended the OpenFlow rules with per-flow encryption-specific flag. These per-rule flags can be communicated from the controller to the switch using unused header bits, controller cookie field, or through OpenFlow vendor-specific messages. Similarly, per-rule encryption keys can be done using a pre-configured key table or through a vendor-specific key exchange protocol. We communicate the encryption flags using the unused header fields and a pre-configured key table. This use case allows per-flow encryption using different algorithms and keys. Figure 4 shows the use-cases topology diagram.

Focus of this paper has been towards understanding and implementing SSDP. Initial results are promising and thorough benchmarks are part of ongoing work.

5. RELATED WORK

Data plane solutions to address the flexibility and speed constraints of merchant silicon fall in three categories (see [4] and references therein): a) Software or NIC switching on the edge; b) Enhanced merchant data planes; or c) Cleverly using merchant data planes. However, all of these solutions are limited in one or more of the following aspects: scale, flexibility, cost and/or semiconductor adoption. For the sake of brevity, we do not go into the details of each proposed solution. Interested readers are referred to [4] for a technical report providing further details of existing work in this domain. As a part of the ALIEN-HW project under OFELIA [http://www.fp7-ofelia.eu/], Dell also proposed to create a similar module using 10G NetFPGA. This proposal was accepted and the project is being pursued by Dell-France.

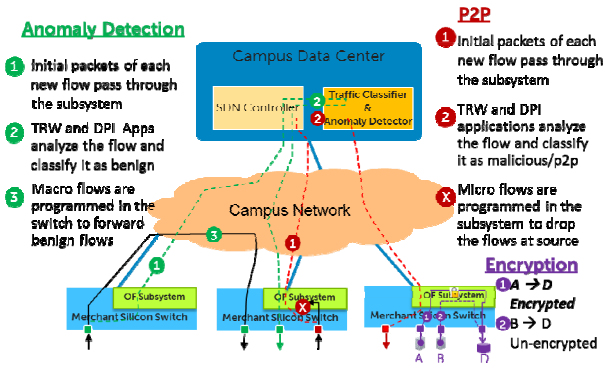


Figure 4. SSDP use cases.

6. CLOSING REMARKS

Despite its enormous promise of rapid network innovation, the SDN ecosystem is encountering serious challenges on technology, market and execution fronts. From a market readiness standpoint, SDN is in a proverbial 'catch-22' situation: A large number of applications need to be developed on/for SDNs for its demand to flourish, but neither the silicon nor the switch vendors are incentivized to provide SDN support without large customer demand. While early adoption in academic and research organizations [2] and niche networks [6] is certainly valuable, SDN is yet to break through in enterprise networks.

Technology and business challenges of hardware switches are compounded by product execution lifecycles, as supporting new features on silicon is an expensive and time consuming proposition. Even if SDN features are prioritized in the short-term, as the specifications evolve, it

is extremely difficult for an engineering organization to keep updating the feature support while maintaining product quality. Hence, creating a consistent SDN architecture that is independent of legacy hardware is simply impractical until merchant silicon vendors start to support it in an extensible manner.

A flexible data plane is a basic requirement to realize such network innovation. In this paper, we proposed SSDP—an architecture that offers a middle-ground between rigid and commoditized switching hardware and software implementations which offer the right flexibility, but at a higher price point and with limited throughput.

6.1 SSDP Advantages

Disruptive without Disruption: This architecture imposes minimal disruption to the existing product lines. It potentially requires zero-software changes for SDN features and very few changes to the hardware. While a hardware re-spin may be required in some cases, switches that already have slots to take in personality-modules or daughter-cards are ideal candidates for initial prototyping.

Fast Feature Velocity: With this architecture, vendors can rapidly develop new features and use-cases that are directly relevant and customized for their end customers. The programmability could allow field engineers to potentially develop new innovative solutions without needing to revisit a full engineering life-cycle to develop the features.

Data Plane Innovation: The SSDP can be used to flexibly support new actions or to implement new functional blocks with a short turnaround time. For instance, an NPU-based backplane can be used to configure and process different tunneling formats, encryption mechanisms and offloading supports.

Low price per port: With recent advances in fabrication technologies, a 4-core NPU with limited offloads capacity costs close to $100. This amounts to an increase of $4 per port for a 10Gbps Dell switch (currently shipping for approximately $8K).

Simplifying Third Party Application Support: The SSDP architecture treats the subsystem as an independent entity whose flow-entries are sanitized by the controller. Furthermore, any flow on the base switch needs to be explicitly forwarded to the subsystem for additional microflow-processing. This allows the subsystem to act as a natural sandbox that can support flow-entries managed by third-party applications via the controller. Thus resulting in a simple, but robust switch architecture for such an ecosystem to flourish.

6.2 Limitations

An obvious limitation of SSDP is its inability to support line rate traffic processing of all the traffic in a switch. The theoretical maximum is limited by the interface between the switch and subsystem. Any network with stringent delay requirements (e.g., HFT) cannot be efficiently supported on the SSDP architecture.

6.3 Future Work

Lastly, unlike current SDN implementations, in SSDP it is important to maintain some notion of state consistency between the two data planes. For instance, a conflicting policy being encoded in any of the two data planes should be detected and resolved by the backend dataplane. This will require out-of-band communication between the controller and the switch. A solution to this problem is a common 'openflow-agent' for the switch and the programmable subsystems (Figure 5). Though this may increase the complexity of the OF-agent it will allow end users to have OVS revisions that are independent of the main switch OS. This is a potential topic for our ongoing work.

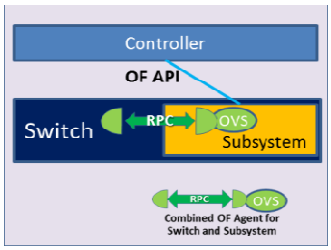


Figure 5. Common OF-Agent for both Switch and Subsystem

7. REFERENCES

[1] Beacon Controller homepage, https://openflow.stanford.edu/display/Beacon/Home.

[2] GENI homepage, http://www.geni.net/.

[3] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection using Sequential Hypothesis Testing. *IEEE Security & Privacy*, 2004.

[4] R. Narayanan, S. Rizvi, and A. Khan. An Overview of SDN Data Planes. Tech. Rep., March 2012 http://xflowresearch.com/dell-ssdp/technical-report-ssdp.pdf.

[5] Open vSwitch homepage. . http://openvswitch.org/.

[6] NTT, in collaboration with Nicira Networks, Succeeds in Remote Datacenter Live-Migration, http://www.ntt.co.jp/news2011/1108e/110802a.html.

[7] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. FlowVisor: A Network Virtualization Layer. OPENFLOW-TR-2009-01.

[8] A. S.-W. Tam, K. Xi, and H. J. Chao. Use of Devolved Controllers in Data Center Networks. In *IEEE INFOCOM Workshop on Cloud Computing*, 2011.

[9] K-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, N. McKeown. Stanford OpenRoads Deployment, *ACM WiNTECH*, 2009.

[10] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking Packet Forwarding Hardware. In *ACMHotNets*, 2008.

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *ACM SIGCOMM*, 2011.

[12] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood and A. W. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *PAM* 2012.