Design of a Hybrid Modular Switch

Ashkan Aghdai, Yang Xu, H. Jonathan Chao
New York University
Tandon School of Engineering
{ashkan.aghdai, yang, chao}@nyu.edu

***Abstract*—Network Function Virtualization (NFV) shed new light for the design, deployment, and management of cloud networks. Many network functions such as firewalls, load balancers, and intrusion detection systems can be virtualized by servers. However, network operators often have to sacrifice programmability to achieve high throughput, especially at networks' edge where complex network functions are required.**

Here, we design, implement and evaluate Hybrid Modular Switch (HyMoS). The hybrid hardware/software switch aims to meet the requirements of modern-day NFV applications by providing high-throughput, highly programmable packet forwarding. HyMoS utilizes P4-compatible Network Interface Cards (NICs), PCI Express interface and CPU to act as line cards, switch fabric, and fabric controller respectively. HyMos turns PCI Express interface into a non-blocking switch fabric with a throughput of hundreds of Gigabits per second.

Compared to existing NFV infrastructure, HyMoS offers modularity in hardware and software as well as a higher degree of programmability by supporting a superset of P4 language.

***Index Terms*—software defined networks, network function virtualization, packet switching, software/hardware co-design.**

I. Introduction

Network functions play a significant role in shaping, policing, and monitoring the Internet traffic. Network Function Virtualization (NFV) lets ISP and Cloud operators utilize programmable devices to tailor the data plane behavior according to their needs. Recent advances in Software Defined Networks (SDN) provides a foundation for building programmable networks. OpenFlow [1] started a new trend in network design and operation by isolating the control plane from the data plane, and Network Operating System (NOX) [2] allows operators to build applications on top of programmable hardware or software OpenFlow-enabled switches. SDN not only provides a robust platform for virtualization of network functions, but also it enables the development of innovative network applications. Smart rule caching and placement [3]–[5], intelligent access control [6], [7], policy verification [8], [9], high-level network programming languages [10], [11], and advanced network measurements tools [12], [13] are just a few examples of SDN applications that enhance performance, manageability, and flexibility of networked systems.

Considering the large body of work in developing applications and implementing new ideas in SDN over the years, it is surprising that design and implementation of high-performance programmable switches, underlying devices that enable all SDN applications, has not received as much attention before the introduction of Protocol-independent Switch Architecture (PISA) [14]. PISA proposes Reconfigurable Match Tables (RMT), a fundamental paradigm shift in designing programmable switches. RMT introduces a generalized high-performance processing model and redefines packet forwarding as a domain problem [15]. P4 language [16], as it stands for Programmable Protocol-independent Packet Processing, adds a much needed Domain Specific Language (DSL) [15] for programmable packet forwarding. As opposed to OpenFlow-like protocols that aim at providing a reliable means for distribution and management of forwarding rules, P4 exposes the inner workings of programmable switches; it allows users to identify packet headers using a programmable parser, specify the matching fields as well as a set of available actions for each forwarding table, and lay out the flow of packets between match+action tables. PISA and PISCES [17] are examples of P4 targets, i.e., P4-configurable devices. PISCES is a compatibility layer on top of Open vSwitch (OVS) [18] that relies on software to process packets according to a P4 program.

P4 introduces a target-independent language for programmable devices, but it lacks some much-needed features required for developing NFV applications. For instance, P4 cannot program switch scheduler, manage queues, or process packets beyond pre-defined header fields. Such limitations effectively bar developers from implementing QoS protocols, or more advanced packet processing techniques such as Deep Packet Inspection (DPI) which are frequent functions at networks' edge. As a result, more flexible architectures such as NetVM [19] and Open Compute Project (OCP) Wedge [20] rely on Intel Data Plane Development Kit (DPDK) [21] and X86-powered processing to implement a highly customizable data plane. Internet Service Providers (ISP) and data center operators alike use such designs to deploy NFV applications at networks' edge which demands a higher degree of programmability than what P4 offers at $O(100G)$ or higher aggregated bandwidths [22]–[24]. However, DPDK's improved programmability comes at the cost of limited packet processing power on X86 CPUs.

To address the gap between NFV applications' requirements at networks' edge and available solutions, we take a fresh approach to design a new family of programmable switches. We aim at bringing the best of the hardware and software switches to design a highly programmable Hybrid Modular Switch (HyMoS) with the following objectives in mind:

- Aggregated throughput suited for networks' edge, i.e., dense 10/40G switches.
- Programmability beyond P4 language.
- QoS capabilities that enable network operators to define and meet Service Level Agreements (SLAs).

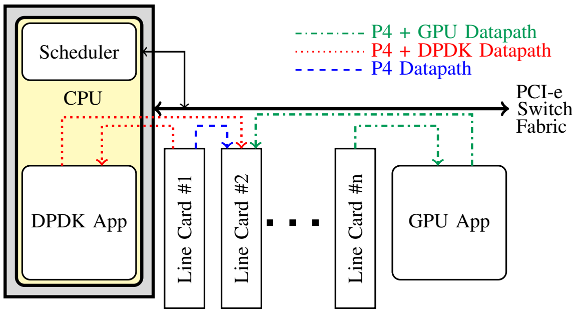


Figure 1: Proposed switch architecture

- Modularity in hardware and software.

We have designed, prototyped, and evaluated the performance of HyMoS. Our early results show that it is possible to build a cost-effective switch on top of commodity servers that utilizes P4-compatible Network Interface Cards (NIC) and PCI Express (PCI-e) [25] backplane to process and switch packets, respectively.

The rest of the paper is organized as follows. Section II introduces HyMoS' architecture. Section III evaluates HyMoS and presents early results. Section IV reviews related works in this area. Finally, Section V concludes the paper.

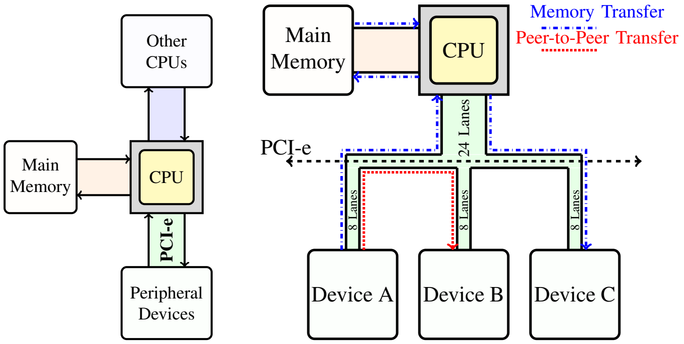
II. A Radical Switch Design Approach

Network devices perform two operations on every packet: processing and switching. Packet processing involves table lookups on specific fields in the packet header. It also decides which output ports - if any - the packet should be forwarded to. Therefore, depending on the active network protocols, packet processing could be very complex, and its software implementations' throughput is usually CPU-bound [17]. Packet switching, however, only involves copying processed packets from ingress port to specified egress port/s.

To achieve high throughput, HyMoS relies on line cards to process packets and perform destination look ups using hardware. Due to the simplicity of switching processed packets, we rely on X86 processors to orchestrate memory operations between ingress and egress ports to switch packets. We show that underlying switch fabric has enough capacity to perform the required operations.

A. HyMoS' Architecture

HyMoS utilizes Network Interface Cards (NIC) as line cards, PCI-e interface as the switch fabric, and CPU as the scheduler to build a modular switch. NICs use hardware acceleration to perform table lookups and process packets. Once processed, packets are stored in a Virtual Output Queue (VOQ) [26] structure according to their destination line cards. A pipelined CPU process polls VOQs, arbitrates between the requests, and schedules peer-to-peer PCI-e transactions for granted requests. The small size of the corresponding bipartite matching [27] problem at the arbitration phase (at most five line cards per CPU socket), enables us to cache the solution space and implement scheduling using constant-time lookups.



(a) Physical View

(b) Logical View

Figure 2: PCI Express Architecture

As shown in Figure 1, HyMoS implements more than one datapath to leverage from highly flexible software-based packet processing in addition to P4 compatibility:

- P4 datapath: P4-compatible NICs process packets and directly send them to egress NIC/s.
- P4 + DPDK datapath: NICs send packets to CPU for additional processing.
- P4 + GPU datapath: Packets are sent to Graphical Processing Unit (GPU) for more complex applications such as deep packet inspection [28], [29].

Multiple datapaths are implemented by instantiating additional virtual interfaces on each NIC to serve as virtual queues for DPDK/GPU destinations.

B. Smart NICs as Line Cards

State-of-the-art smart NICs [30]–[33] offer a reconfigurable hardware switch with multiple physical ports and tens of Single-Root I/O Virtualization (SR-IOV) [34] virtual interfaces. These devices support processing packets at stunning rates beyond 100Gbps per direction. In addition to standard P4 actions, some even support defining custom packet processing actions [30].

Utilizing smart NICs as line cards enables hardware-accelerated packet processing at line rate, P4-programmable datapath, and a modular design for end-users that allows them to choose line cards and customize switch port rate/count.

C. PCI-e Interface as Switch Fabric

PCI-e [25] is the interface that interconnects CPU and peripheral devices in the X86 architecture. Modern implementations of PCI-e, as shown in Figure 2a, directly attach peripheral devices to CPU with a PCI-e link which is a point-to-point dual simplex connection of up to 32 lanes. Table I demonstrates how PCI-e standard has evolved over the years to provide more bandwidth for peripheral devices. A third

Link Width	x1	x2	x4	x8	x16
Gen1 Bandwidth (GB/s)	0.5	1	2	4	8
Gen2 Bandwidth (GB/s)	1	2	4	8	16
Gen3 Bandwidth (GB/s)	∼2	∼4	∼8	∼16	∼32

Table I: PCI Express Evolution [35]

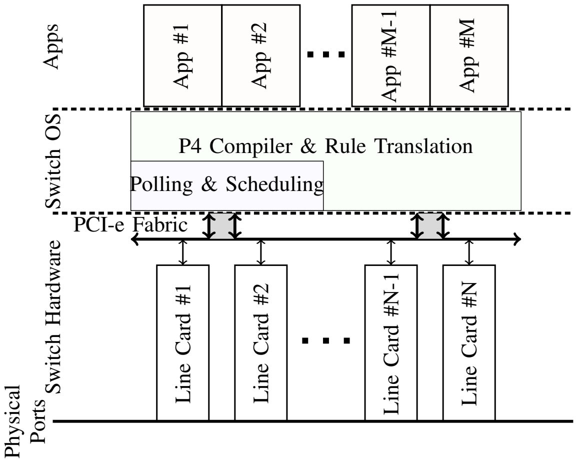


Figure 3: Modular switch design.

generation X8 link has enough bandwidth to support multiple 10/25/40G interfaces. The fourth generation of PCI express, expected to be released this year, doubles third-generation bandwidth making 100G or dense 40G switches a possibility.

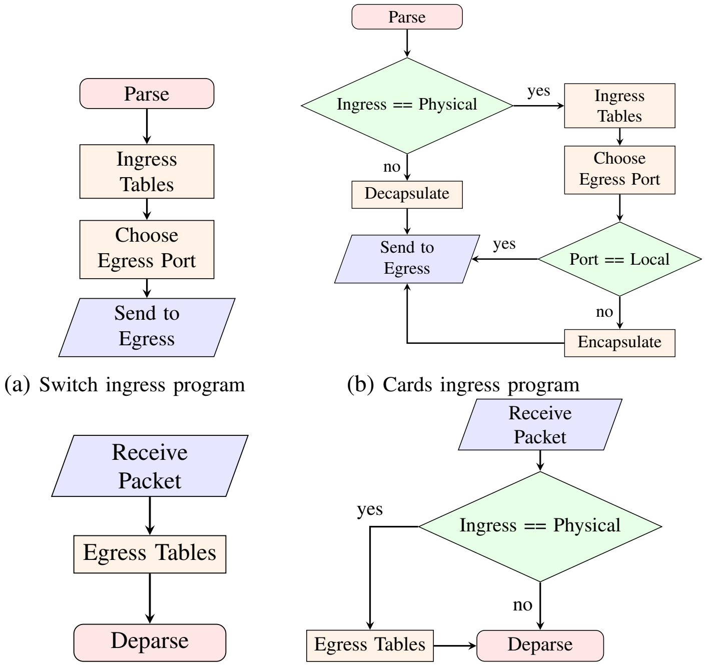
As illustrated in Figure 2b, in addition to supporting memory operations, PCI-e supports peer-to-peer device transfers. As opposed to the DPDK model in which packets traverse through the main memory for a transfer from device A to device C, a peer-to-peer PCI-e transaction transfers a packet from device A to device B directly without CPU/Memory involvement. HyMoS relies on peer-to-peer transactions to switch packets between the cards.

SR-IOV allows instantiation of virtual interfaces on NIC cards. Virtual interfaces have unique addresses and ingress/egress queues. Line cards implement a VOQ structure by instantiating virtual interfaces for each possible destination, i.e., other line cards or CPU/GPU datapath.

D. Modularity in Software and Hardware

HyMoS relies on CPU to implement a pipelined polling and scheduling process. Depending on the choice of CPU users may have a number of spare cores. As shown in Figure 3, HyMoS brings in modularity in software by making an open platform and letting users install SDN agents. For example, HyMoS users may install DIFANE [3], NetPlumber [8], and Flowvisor [36] agents for smart placement of rules on line cards, verification of policies, and network virtualization, respectively. Cloud networking services can also be offloaded to HyMoS for faster deployment and robust control of cloud services.

We put switch operators in charge of hardware selection. Number and type of CPU, the amount of memory, GPU packet processors as well as line cards can be customized. The P4 datapath requires little to no CPU intervention as PCI-e peer-to-peer transactions do not consume CPU cycles. To enable DPDK datapath or implement SDN agents, HyMoS should be configured with multiple cores at a high clock and sufficient memory. GPU packet processors offer more complex and highly parallel network functions such as deep packet inspection at the cost of occupying some of the available PCI-e slots for line cards.



(a) Switch ingress program

(b) Cards ingress program

(c) Switch egress program

(d) Cards egress program

Figure 4: P4 Translation

Switch operators also have a choice of line cards. They can mix 10/25/40/50G cards on X8 PCI-e links. A dual-port 40G card with breakout enables up to eight 10G ports enabling high port density. Using 100G line cards as uplink ports is also possible, although these cards need an X16 or two X8 PCI-e links ideally connected to different CPU sockets.

E. Challenges

1) HyMoS' Compiler: HyMoS relies on line cards to create a VOQ structure to implement internal packet switching between cards. As a result, input P4 program, P, which describes the behavior of the switch should be translated to P_i P4 programs for individual line cards to implement table lookups for internal packet switching. Therefore, we design a P4 compiler for HyMoS to add additional tables and derive P_i, line card programs, from P, the program for the switch.

The implementation of HyMoS compiler is relatively straight forward. Figures 4a and 4c show the switch program for ingress and egress pipelines respectively. As shown in Figure 4b, HyMoS compiler adds two additional tables at the beginning and end of ingress control flow to perform internal port lookups. Packets destined to other line cards should be labeled with their destination port numbers since egress port specified in the P4 code is a metadata local to line card, and it is not transferred to the egress line card. MAC-in-MAC encapsulation is utilized to transfer destination port number along with the packet. Added Ethernet header uses ingress port number as source MAC and egress port number as destination MAC. HyMoS encapsulation uses an unused Ethernet type for the parser to detect encapsulated packets.

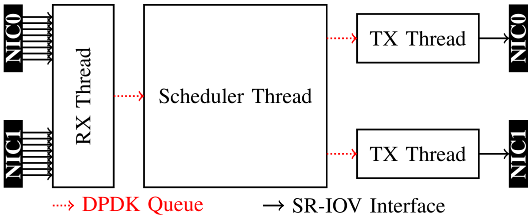


Figure 5: DPDK pipeline of HyMoS scheduler

The table at the end of the ingress pipeline decides whether the output port is local to current NIC or not. If the destination port is local, then the packet will be sent to local egress port without any additional processing. In cases where destination port is on another NIC, this table encapsulates the packet and queues it on the dedicated virtual interface for the egress line card.

As shown in Figure 4d, egress table operations are also performed at ingress line card. The table at the beginning of egress control flow matches on the input port. Packets received on physical ports are processed as specified by P, while packets received on virtual ports are merely sent to their specified egress port.

In addition to adding tables mentioned above, the parser should also be modified to support MAC-in-MAC encapsulation and extract ingress/egress ports as metadata for encapsulated packets.

2) DPDK Packet Scheduler: HyMoS DPDK scheduler implements IEEE802.1p [37], which defines 8 priority classes and provides QoS at L2. Supporting this protocol is a must for NFV targets, because it can be used as a building block to implement more advanced QoS protocols and/or guarantee SLAs. However, the current generation of P4 language and P4-compatible devices are unable to support this protocol due to language limitations.

Our implementation relies on NIC cards to parse 802.1p priority class (which is part of the VLAN header) and enqueue processed packets in different SR-IOV virtual interfaces according to their priority class. As mentioned before, HyMoS takes advantage of SR-IOV interfaces to virtualize VOQ structure. With added support for 802.1p protocol, NIC cards create eight queues per destination. In other words, on a switch with N line cards, each line card creates $8(N-1)$ virtual queues. Figure 5 illustrates a simple example of this structure with 2 NICs. In HyMoS' pipelined scheduler, a receiver thread polls SR-IOV interfaces and updates eight demand matrices (one per priority class) for the next-stage scheduler. The scheduler thread greedily solves the problem for each priority class. It starts with the highest priority and looks up the solution to a bi-partite matching problem with maximum weight objective corresponding to the current priority class, marks granted queues and iterates to the next priority level. At the last stage of the pipeline, one transmitter thread per line card transfers packets from marked queues to their destination. Results in Section III show that using single-producer multi-consumer queues from DPDK standard library and 4 CPU threads this architecture can transfer more than $100Gbps$ of traffic between HyMoS line cards.

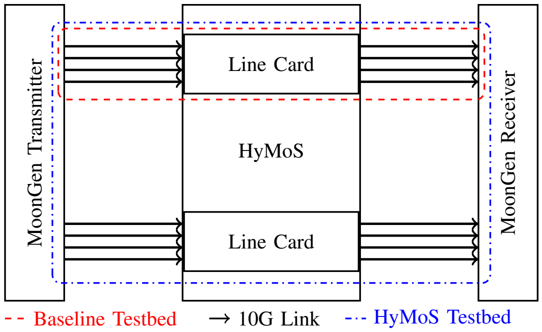


Figure 6: Testbed topology.

III. Preliminary Results

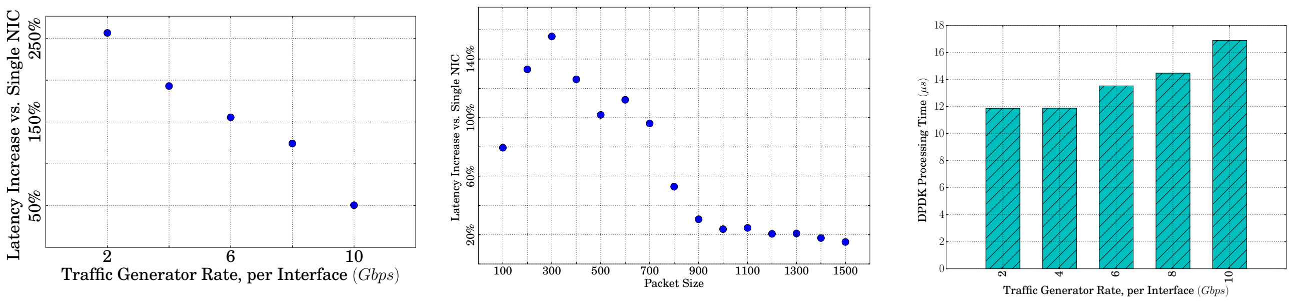
Our HyMoS prototype implements P4+DPDK datapath. Figure 6 shows the topology of HyMoS testbed. Our early implementation uses two dual-port 40G NFP4000 [30] smart NICs on a dual-socket server with Intel Xeon E5 2690 V3 twelve-core 2.6GHz CPUs. NICs are attached to the same Non-uniform Memory Access [38] (NUMA) node on the server and HyMoS' DPDK scheduler is implemented using 4 cores from the same NUMA node. The second CPU is not used in this experiment. 40G ports on the NICs are configured to work as quad 10G interfaces allowing us to implement a 16-port 10G HyMoS.

MoonGen [39] is utilized to generate multiple 10G streams with variable packet sizes. The traffic generator uses four quad-port Intel X710 NICs connected to HyMoS' line cards.

To create a baseline for performance, we have installed an L3 router P4 program on an NFP-4000 NIC. The P4 code implements IP Longest Prefix Matching (LPM), VLAN tagging, and Ethernet Forwarding Information Base (FIB) tables. As shown in Figure 6, four 10G interfaces send UDP traffic with uniformly distributed random source and destination IP addresses to 4 ports of the smart NIC. Rules are installed on the NIC card to put the remaining four interfaces on four different IP subnets which are connected to MoonGen receivers. We defined end-to-end delay - which approximates the processing time at switch - as the performance metric for this experiment. MoonGen is configured to measure end-to-end delay using hardware timestamps.

HyMoS is evaluated under similar settings. The same L3 router P4 code is installed on HyMoS using method discussed in II-E1. As shown in Figure 6, eight traffic generator interfaces are sending uniform traffic to HyMoS (four interfaces on each line card) and the remaining eight interfaces receive the traffic. Similar to the baseline, end-to-end delay is measured using hardware timestamps at traffic generator NICs.

Figure 7a presents the average increase in switch processing time of HyMoS compared to baseline NFP4000 under variable load. In this scenario, the traffic generator sends 800 Byte packets at specified rates varying between 2Gbps to 10Gbps per interface for five minutes in each measurement. Figure 7b compares normalized processing time of HyMoS to that of the baseline under variable packet sizes. In this case, packets with specified lengths are generated at line rate for 5 minutes for each measurement. Our measurements show that HyMoS'



(a) Normalized latency vs. switch load.

(b) Normalized latency vs. packet size.

(c) DPDK processing time vs. switch load.

Figure 7: HyMoS' relative performance.

performance comes very close to the baseline (less than 20% penalty in processing time) at line rate with large packets, which is very promising given that this implementation of HyMoS offers more flexible scheduling, stateful packet processing at DPDK, and twice the ports of a single NIC. At slower rates and with small packets, however, performance penalty could be larger.

As a micro-benchmark, DPDK transfer time between the two line cards is measured by installing receive and transmit callbacks. Average DPDK transfer time is an approximation of the queuing delay in addition to the processing delay incurred by HyMoS DPDK scheduler. Figure 7c presents the average transfer times under variable load with 800-Byte packets. Similar to previous results, the average values are taken from 5-minute long measurements. Based on these figures, we conclude that DPDK processing time outweighs NFP4000 processing time at slower rates and for smaller packets. However, in processing large packets at line rate processing time at NIC outweighs DPDK processing time which effectively makes HyMoS' performance penalty negligible.

IV. Related Works

HyMoS is built on top of four pillars. **P4** [16], a universal programming language to describe the behavior of packet processors. **DPDK** [21], a set of libraries that enable programmable packet processing at software. And recent advances in X86 computing, **Smart NICs**[30]–[33] that makes high-throughput packet processing at hardware possible and **PCI Express interface** [25], a high-speed interconnect for the peripheral devices. Donard [40] and Direct-GPU [41] are two recent technologies that use smart NICs and PCI-e peer-to-peer transactions to realize remote direct memory access (RDMA) for SSD/NVMe storage and GPU memory respectively.

ServerSwitch [42] is one of the earliest works that utilized PCI-e as a cost-effective alternative for Ethernet switching. Belonging to pre-SDN era, ServerSwitch does not offer much programmability atop Ethernet. As a programmable switch, HyMoS is closely related to PISA [14], PISCES [17], and OCP Wedge switch [20]. [14] processes packets using a domain-specific hardware, whereas [14], [17] rely on software to make a programmable data path. ClickNP [43] and NetVM [19] are other examples of highly programmable devices designed for NFV without supporting P4 DSL. ClickNP offers FPGA-based hardware acceleration achieving low-latency and high throughput for packet forwarding, however, it comes at the cost of complicated design and deployment of new network functions due to not supporting a DSL similar to P4. NetVM solely relies on DPDK for packet processing offering a highly programmable solution with X86 performance limitations. Unlike existing solutions, we take the middle ground and leverage both, hardware and software, to process the packets

HyMoS is a modular platform that enables the implementation of recent advances in NFV/SDN including but not limited to rule caching and management [3]–[5], network function virtualization [44], [45], control plane virtualization layers [36], [46], rule verification [8]–[10], and flexible/efficient data plane design[19], [21], [47].

HyMoS also builds on top of extensive research in switch design, most notably, scheduling in virtual output queued switches[26], [48], [49] and its relation to the classic bi-partite matching problem in graphs [27].

Whippersnapper [50] is a framework for benchmarking P4-compatible devices which we plan to use in the future for a more comprehensive comparison of our design to existing solutions.

V. Conclusion

HyMoS is a programmable modular switch designed for NFV applications at networks' edge. Its hybrid hardware/software design based on commodity servers enables modularity and a high degree of programmability. Unlike solutions that rely only on hardware or on software, HyMoS brings in the best of both worlds by utilizing P4-compatible NICs and PCI-e backplane to enable flexible packet processing and switching at line rate.

In addition to supporting a hardware-accelerated P4-compatible datapath suited for virtualizing L2/L3 functions, HyMoS offers a P4+DPDK and P4+GPU datapaths geared towards advanced L4 and up NFV applications. HyMoS improves the usability of P4 language in NFV applications by adding a programmable scheduler and enabling support for DPDK packet processing on top of P4. Using a small testbed we show that HyMoS extra features will add a performance penalty, which is negligible especially at line rate and for large packets.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[3] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010.

[4] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 175–180, ACM, 2014.

[5] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, "Cab: A reactive wildcard rule caching system for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 163–168, ACM, 2014.

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[7] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 121–126, ACM, 2012.

[8] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis.," in *NSDI*, pp. 99–111, 2013.

[9] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: Verifying network-wide invariants in real time," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

[10] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM Sigplan Notices*, vol. 46, pp. 279–291, ACM, 2011.

[11] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular sdn programming with pyretic," *Technical Reprot of USENIX*, 2013.

[12] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch.," in *NSDI*, vol. 13, pp. 29–42, 2013.

[13] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," in *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 419–430, ACM, 2014.

[14] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 99–110, ACM, 2013.

[15] A. Van Deursen, P. Klint, J. Visser, *et al.*, "Domain-specific languages: An annotated bibliography.," *Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[16] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[17] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pp. 525–538, ACM, 2016.

[18] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch.," in *NSDI*, pp. 117–130, 2015.

[19] J. Hwang, K. Ramakrishnan, and T. Wood, "Netvm: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.

[20] Facebook and O. C. Project, "Facebook wedge - 16x40gb qsfp+ - leaf/spine switch," 2015.

[21] Intel, "Data plane development kit," 2014.

[22] AT&T inc., "AT&T ecomp (enhanced control, orchestration, management, and policy) architecture white paper," 2016.

[23] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[24] AT&T. inc., "AT&T domain 2.0 vision white paper," 2013.

[25] PCI Special Interest Group, "PCI Express Base Specification Reveision 3.0," 2010.

[26] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, 1999.

[27] D. B. West *et al.*, *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[28] W. Sun and R. Ricci, "Fast and flexible: Parallel packet processing with gpus and click," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 25–36, IEEE Press, 2013.

[29] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan, "Evaluating gpus for network packet signature matching," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 175–184, IEEE, 2009.

[30] Netronome, "Nfp-6000 intelligent ethernet controller family." https://www.netronome.com/static/app/img/products/silicon-solutions/PB_NFP6000.pdf.

[31] Intel FlexPipe, "Intel ethernet switch fm6000 series-software defined networking," 2012.

[32] Cavium XPliant, "Xpliant packet architecture," 2014.

[33] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, 2014.

[34] PCI Special Interest Group, "Single Root I/O Virtualization Revision 1.1," 2010.

[35] M. Jackson and R. Budruk, *PCI Express Technology: Comprehensive Guide to Generations 1.x, 2.x, 3.0*. MindShare, 2012.

[36] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.

[37] N. Ek, "Ieee 802.1 p, q-qos on the mac level," *Apr*, vol. 24, pp. 0003–0006, 1999.

[38] C. Lameter, "Numa (non-uniform memory access): An overview," *Queue*, vol. 11, no. 7, p. 40, 2013.

[39] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: a scriptable high-speed packet generator," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pp. 275–287, ACM, 2015.

[40] Microsemi, "Project Donard: Peer-to-Peer Communication with NVM Express Devices," 2014.

[41] Mellanox, "Mellanox: NVIDIA GPU-Direct technology—accelerating GPU-based systems," 2010.

[42] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: A programmable and high performance platform for data center networks.," in *Nsdi*, vol. 11, pp. 2–2, 2011.

[43] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, and P. Cheng, "Clicknp: Highly flexible and high-performance network processing with reconfigurable hardware," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pp. 1–14, ACM, 2016.

[44] T. Wood, K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network*, vol. 29, no. 3, pp. 36–41, 2015.

[45] C. Price and S. Rivera, "Opnfv: An open platform to accelerate nfv," *White Paper. A Linux Foundation Collaborative Project*, 2012.

[46] D. Hancock and J. van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pp. 35–49, ACM, 2016.

[47] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through intel dpdk," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pp. 1–6, IEEE, 2014.

[48] N. McKeown, "The islip scheduling algorithm for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 2, pp. 188–201, 1999.

[49] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on communications*, vol. 35, no. 12, pp. 1347–1356, 1987.

[50] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A p4 language benchmark suite," in *Proceedings of the Symposium on SDN Research*, pp. 95–101, ACM, 2017.