IEICE TRANS. COMMUN., VOL.E98–B, NO.1 JANUARY 2015

INVITED PAPER *Special Section on Quality of Diversifying Communication Networks and Services*

Software-Defined Data Plane Enhancing SDN and NFV

Akihiro NAKAO[†a)], ***Member***

SUMMARY In this paper, we posit that extension of SDN to support deeply and flexibly programmable, software-defined data plane significantly enhance SDN and NFV and their interaction in terms of (1) enhanced interaction between applications and networks, (2) optimization of network functions, and (3) rapid development of new network protocols. All of these benefits are expected to contribute to improving the quality of diversifying communication networks and services. We identify three major technical challenges for enabling software-defined data plane as (1) ease of programming, (2) reasonable and predictable performance and (3) isolation among multiple concurrent logics. We also promote application-driving thinking towards defining software defined data-plane. We briefly introduce our project FLARE and its related technologies and review four use cases of flexible and deeply programmable data plane.

key words: *Software-Defined Networking (SDN), Network Functions Virtualisation (NFV), network virtualization*

1. Introduction

Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) have recently emerged as fundamental technologies for enabling flexible, programmable networking and have been expected to form the basis of the Internet of near future for dealing with constantly arising new problems. In addition, both SDN and NFV have caught attentions from industries because they have been introduced as technologies for reducing capital expense (CAPEX) as well as operational expense (OPEX), where software-defined programmable network equipment is supposed to dispense with high maintenance cost often incurred in hardware appliances and to enable rapid deployment of functional revisions, besides the automation of operation and management (OAM) of network enabled by programmatic interface to the equipment may reduce the high cost of the OAM by manual labor.

Although the synergy between SDN and NFV has only recently been discussed, they have been proposed separately. While SDN primarily focuses on the programmability on the control of networking, NFV aims at implementing data processing functions in software on top of virtual machines (VMs) that exist today as hardware network appliances. The clear distinction of the focuses of SDN taking care of networking and NFV of computation may allow scalable construction of programmable infrastructure, since data packets can be *programmatically redirected* by SDN and can be *programmatically processed* by NFV.

However, we observe two limitations in this model of separation of SDN and NFV, leaving an interesting research area as a gap between them. First, SDN often defines predetermined interface, so called south-bound interface or SBI, mainly for the sake of standardization purpose. It is control plane software including controllers that can be programmed in software above SBI (and not to mention, above so called north-bound interface or NBI), but data plane that implements data forwarding and redirection often remains to be implemented in hardware as in, e.g., OpenFlow [1] switches. If we could arbitrarily define data plane by software, i.e., software-defined data plane, in carefully designed sandboxes such as virtual machines inside network equipment, we should be able to enhance the data plane functionalities, e.g., those related to OAM, and publish the SBI for controllers to use them. Such enhancement is only recently discussed in a few research projects [2], [3]. Second, NFV is so far limited to implementing network appliances in software, and deals neither with crafting new protocols nor with OAM functionalities, which are largely considered as SDN's responsibility. However, as mentioned above, SDN's data plane is not so much flexibly programmable. Enhancing SDN with software-defined data plane would compensate the gap in NFV.

In this paper, we posit that enhancing SDN to supporting flexibly and deeply programmable, software-defined, data plane with SBI published for controllers may bring more innovations in future networking, especially in the SDN and NFV areas. We also identify at least three benefits enabled by deeply programmable SDN data plane, (1) enhanced interaction between applications and networks, (2) optimization of network functions, and (3) rapid development of new network protocols.

First, embedding application functionalities may not only optimize their performance but also enables interesting interactions between applications and networks. We introduce such examples in Sect. 5.1. Second, we can offload NFV's virtual network functions into SDN's data plane, i.e., closer to the network to reduce latency in processing as explained in Sects. 5.2 and 5.3. And finally, we can quickly improve the existing protocol handling by tweaking the data plane of SDN, as well as defining new protocols in a clean-slate manner as discussed in Sect. 5.4.

We believe all of these benefits enabled by the enhancement of SDN and NFV would contribute to improving the quality of diversifying communication networks and ser-

Manuscript received July 7, 2014.
Manuscript revised September 5, 2014.

[†]The author is with The University of Tokyo, Tokyo, 113-0033 Japan.
a) E-mail: nakao@iii.u-tokyo.ac.jp

DOI: 10.1587/transcom.E98.B.12

vices of present day. The Internet today has changed drastically with emerging small devices such as smartphones, wearable glasses and watches, sensors and cloud data centers, which unfortunately causes constantly arising new problems. We posit that the key to solving these issues in ever-diversifying communications is to enhance flexibility of the programmable communication infrastructure, with not only programmable control plane but also deeply programmable data plane as well.

The rest of the paper is organized as follows. Section 2 discusses key challenges for deeply and flexibly programmable data plane in SDN. Section 3 compares various technologies for enabling programmable data plane in SDN. Section 4 introduces our programmable network node architecture called FLARE and Sect. 5 enumerates various application use-cases of deep programmability enabled by software-defined data plane. Finally, Sect. 6 briefly concludes this position paper.

2. Challenges

2.1 Challenges in Deeply Programmable Data Plane

In order to enable flexibly and deeply programmable, we have identified three major technical challenges, (1) ease of programming, (2) reasonable and predictable performance and (3) isolation among multiple concurrent logics.

First, we have to consider lowering the barrier to entry for programming network functions. SDN and NFV are considered as cost-effective solutions, but the premise is that we need lots of programmers for creating network functions, let alone data plane functionalities. Therefore, one of the most important challenges to resolve is how we can accommodate programmers of various levels of skills and thus increase the entire number of programmers. There could be lots of kinds of programming models for defining programmable data plane, such as FPGA, Intel Data Plane Development Kit (DPDK) [4], Network Processors with many cores [5]–[9], but we need to carefully select these platform in terms of ease of programming and debugging.

Second, performance is another challenge in programmable networking. It is often the case with programmable network equipment that there is trade-off between the programmability, i.e., how simply and flexibly we can program and the performance, i.e., how fast we can execute programs. Especially, software solutions are mostly susceptible to performance degradation, although it is highly flexible and can be quickly designed and implemented. In the light of this observation, we believe that we should select the platform with high flexibly but reasonable and at least predictable performance.

And finally, we believe the capability of programming multiple concurrent logics on top of a single physical programmable environment is significant. We can virtualize the physical hardware resources and provision necessary amount of virtual resources per logic to achieve programming of multiple logics on top of isolated virtual resources.

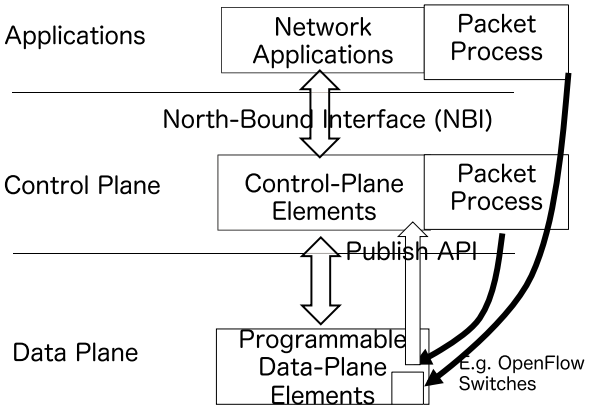

Fig. 1 Application-driven programmable networking.

Isolation of resources plays a very important role here.

It is also worthwhile to note regarding the last challenge that the isolation of resources is important also for avoiding security threats as a result of enabling flexibly creating, modifying, dismissing functions by programming, although rigorous argument of security threat is out of scope of this position paper. With rapid progress of resource isolation in operating system virtualization, various wisdoms may be applied to this challenge, at least for creating secure and isolated execution environment. However, note that there is a vast area of research left to be tackled in terms of assuring secure operation of programmable networking.

2.2 Application-Driven Programmable Networking

Another rather non-technical aspect of thought towards highly programmable data plane of SDN is to think *application-driven programmable networking*, where starting from the application that cannot be built without the help from the in-network functions, i.e., the network functions embedded inside the data plane of SDN solution.

We, network researchers, tend to develop generic infrastructure to accommodate all the applications. However, this bottom-up approach does not often capture the idea of what kind of API and the network functions behind the API are necessary. It is important to think top-down, from applications that do not exist today due to the limitation in the network, down to defining what are necessary inside the data plane of SDN. US Ignite [10] and PARADISO [11] make the most out of applications and take similar avenues as ours.

3. Related Work

3.1 Programming Environment

The most straight-forward approach to enabling flexibly and deeply programmable data plane is to use various kinds of configurable hardware such as FPGA [12] or to program software on top of general-purpose and network processors.

There are pros and cons in different programming platforms in terms of (1) ease of programming, (2) reasonable

Table 1 Pros and cons of various data plane technologies.

	Many-core Network Processors (operating system)	Many-core Network Processors (assembly-language micro-engine)	Data Plane Development Kit (DPDK) on General Purpose Processors	ASIC	FPGA
Ease of Programming	+	-	+	-	-
Performance	+	+	+	+	+
Logic Isolation	+	-	-	-	-
Power Efficiency	+	+	-	+	+

and predictable performance, (3) isolation among multiple concurrent programmed logics, and (4) power efficiency, as summarized in Table 1.

In conclusion, we believe that, *as of today*, the most appropriate platform for our purpose of providing simple, yet flexible and deep programmability while retaining reasonable performance, logic isolation and power efficiency is many-core processors with open-source operating system support, e.g., with Linux operating system with virtualization support, such as MIPS based Octeon [6] from Cavium [13], various enterprise processors such as XLR, XLP, XLS, etc. [14] from Broadcom [15] (before acquisition, RMI [7] and NetLogic [16]), and TileGx series [17] from EZChip [9] (before acquisition, TILERA [8]).

Many-core network processors are attractive in that, first, a large number of aggregated flows can be distributed to many cores and get processed concurrently, and second, when it comes to virtualization, partitioning and allocate processor cores helps preparing isolated execution environment. Although the same story may seem applicable to Intel's DPDK [4], higher power consumption and the less number of cores are yet to be improved for our purpose. However, it is sometimes useful to employ a hybrid approach, that is, allocating different kinds of resource for different processing, as in our FLARE architecture [2] that combines Intel's general purpose processors and many-core network processors (as well as GPGPU, if necessary) and uses hierarchically combined computational resources.

Micro-engine based many-core processors such as EZChip [9] and Netronome [5] and FPGA based platforms such as NetFPGA [12] are also attractive in terms of performance we can achieve, but the programming environment is harder to use than those with operating system support. However, later in near future, we expect that programming environments of these platforms will surely be improved and may fit our purpose. For example, a flexible, intelligent, and highly-optimized compiler or translator may provide high-abstraction-level, easy-to-use, and integrated programming environment even on FPGA or micro-engines.

Another approach is to use hardware, e.g., ASIC [18]. Although this approach is attractive in terms of high performance, it is less flexible to modify once programmed logic. If we could design such a logic that can be quickly modified according to specified profiles, that would be ideal.

3.2 Related Research Activities

There are several interesting research activities regarding realizing more flexible data-plane processing in SDN. ONF's L4-L7 Working Group discusses extending flow matching from currently limited header fields only to L4 to L7 fields. Another extension discussed in ONF is to enable Protocol Oblivious Forwarding (POF) to handle new protocols, not limited to the current TCP/IP architecture. Huawei has demonstrated hardware design that can handle POF in their product [3].

ETSI NFV discusses enhancement of data processing into service composition, so called, Service Chaining. Although the current focus is not really on new protocol handling and offloading of virtual functions into SDN data plane elements, similar ideas could be employed in the extension to SDN.

OpenDataPlane [19] is the closest approach to what this paper posits. It is an open-source, cross-platform set of application programming interfaces (APIs) for the networking data plane. It aims at extending vendor-specific hardware blocks and software libraries to provide data plane API. Although it is still at the baby stage, we expect this activity can be supported by many network processor vendors.

OpenFlow HAL [20] is similar to OpenDataPlane but limited to OpenFlow specification and still hardware approach. It seems useful for hardware OpenFlow vendors to identify primitive hardware abstraction to support to develop hardware OpenFlow switches, but up to date, it is unclear how it handles extension of data plane beyond OpenFlow specification.

4. FLARE : A Platform for Deeply Programmable Network Node

Our FLARE project [2] utilizes a hybrid of computational resources, such as network processors, general purpose processors, (and optionally GPGPU) in a hierarchical manner so that we can extend data plane processing functions easily by software program.

In FLARE architecture, we attempt to resolve all three major technical challenges enumerated in Sect. 2, namely (1) ease of programming, (2) reasonable and predictable performance and (3) isolation among multiple concurrent logics, in realizing software-defined data plane programmability. We introduce Toy-Block networking programming model [21] to enable drag and drop programming in FLARE to resolve (1). Also, in order to achieve (2), we combine a hybrid of computation resources especially design a hierarchical structure of high-frequency small-number-core processors and low-frequency many-core processors.

Data plane processing often require scalability in terms of the number of flows to process and power-efficient low-frequency yet many core processors are useful for massively parallel processing for a large number of flows. On the other hand, powerful, high-frequency, small-number-core general processors can be best suited for control and management functionalities. Different kinds of resources are inherently fitted for different kinds of tasks. And finally, for (3), we employ a lightweight resource virtualization technique called resource container for isolation of multiple logics. For the best isolation, we decide to partition many cores into groups and deploy a resource container per group.

Our goal is little similar to OpenDataPlane activity in that the goal is to flexibly and easily extend data plane, but our proposal is a little different in that we consider isolation of resource via virtualization as very important key factor. For example, FLARE can implement multiple concurrent logics such as OpenFlow 1.0 and OpenFlow 1.3 data plane elements in isolated environments. Since SDN allows us to slice the network into isolated execution environment, we should consider this capability as the first class feature of the data plane element.

Utilizing FLARE prototypes, we attempt to enable various use-cases of applications enabled by software-defined data plane as introduced in detail in Sect. 5.

5. Example Use Cases

In this section, we identify at least four use cases of the applications enabled by highly programmable software-defined data plane, each of which falls on to one of these benefits, (1) enhanced interaction between applications and networks, (2) optimization of network functions, and (3) rapid development of new network protocols.

5.1 Application Specific Traffic Control

When it comes to application specific traffic control, we posit that there are two shortcomings in the current SDN approaches.

First, the current SDN traffic control is mostly flow-based. For example, in OpenFlow, the switch matches the header information of the received packets against the flow entries and execute corresponding actions if matched. Thus, the abstraction for programming is three-fold, <Flow><Action><Stat>, where <Flow> stands for predefined tuples in packet header information such as IP addresses and port numbers, <Action> represents associated actions when the flow is matched, and <Stat> records packet counts. While the flow-based traffic control is a natural way from the network operator point view, application users and developers may not often find it useful, because in order for them to control their application traffic programmatically, they must find out the flows, e.g., including the source port numbers of application traffic that are not usually of their concern. For application users and developers, process-based traffic control is more natural than flow-based one, where we can use application names, the state of the application processes, as well as device IDs and status, etc., for the basis of the control. If we extend the Openflow model, the right abstraction for programming in this case may be one like <Application/Device><Action><Stat>, although we may not have to follow OpenFlow's convention for programming abstraction, and one could rather define one's own programming abstraction, as long as it is open and published as an API.

Second, even if applications keep track of their flow information, they need to let the SDN controller know the flow information out of band, that is, besides the application data traffic, they must open control channels to convey such flow information to the SDN controller so that they may be able to control their flows. This approach is prohibitive for small devices such as smartphones and sensors since that may become significant overhead for them.

In order to tackle these issues, we propose a method to modify operating systems of the end systems such as smartphones, so that we can find application process information and convey such information through an in-band communication. Our prototype system attaches the application process information at the end of each packet, i.e., trailer, on the part of the end systems, and decodes (and removes after that) the information on the part of the programmable node located at the first hop from it. In this way, we learn the mapping of the information on application processes and flows and inform the SDN controller so that subsequent nodes can just perform flow-based traffic control.

For example, our prototype system for realizing application specific traffic control for TCP applications is shown in Fig. 2. We install our simple application on smartphones for capturing the very first packet an application emits, i.e., a TCP SYN packet in case the application establishes a TCP session, and then for attaching process information such as application name and status and/or a device ID and status, etc. at the end of the packet, namely, as trailer bits. In more detail, we first capture the header information of a TCP SYN packet and examine the process table and the socket table of the operating system to look for corresponding process name and status, and attach the information as the trailer bits.

The programmable node at the first hop then detects such unusual TCP SYN (with non-zero payload size) and decodes (and removes) the information at the trailer. It observes the flow information of the TCP SYN packet at the same time and maps the information on the application processes and that of the flow, so that the SDN controller can tell subsequent SDN switches along the route to the destination to perform QoS traffic control such as bandwidth throttling for particular applications using the traditional flow-based traffic control.

Implementing our prototype system on our FLARE platform [2] explained above, we have already proved that our proposed system works quite well unless ISPs do not filter unusual TCP SYN in fear of SYN Flooding, which is not really performed in most MVNO services of today.

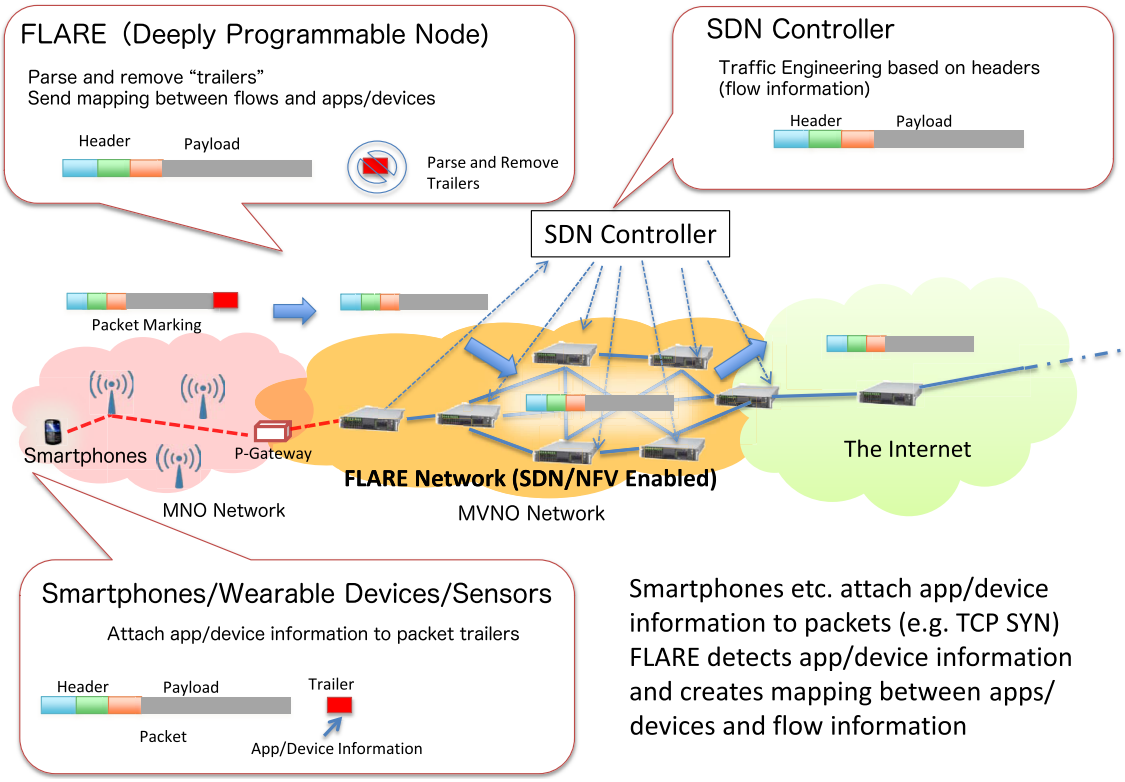

Fig. 2 Application specific traffic control.

Some may argue that piggy-backing data in TCP SYN may render incompatibility and security issues. However, such unusual piggy-backing is not uncommon today. For example, Google does this in TCP Fast Open (TFO) [22] for the different purpose than ours, where they attempt to reduce the number of packets and the delay in three-way handshaking, storing "cookies" in newly emitted TCP SYN's payloads for already authenticated end systems via the past three-way handshakes.

Others may also argue that we could perform this sort of traffic shaping right on the devices such as smartphones and wearables. However, although it might provision the bandwidth right at the devices and the very end of access network connectivity, we could not control traffic shaping beyond that, thus, this method could not achieve our purpose of application specific traffic control for end-to-end communications.

We should note that in order to achieve this type of application specific traffic control, data-plane functionality must be extended from the current SDN model where data-plane elements have limited pattern match capabilities and too few actions. Especially, the manipulation of the packet trailer at Layer 7 (L7) is largely missing from the current SDN data-plane and the extension to support such manipulation is useful to enable new applications such as application specific traffic control.

We have recently deployed the system that is essentially the same as the example shown in this subsection in our joint collaboration with an ISP in Japan. We believe that empowering MVNOs with application/device specific traffic engineering would become the norm of the next generation MVNO business.

5.2 M2M Smart Gateways

Research on M2M (Machine-to-Machine) communications has recently been acquiring much attention both in industries and academia. From the SDN and NFV research point of view, so-called M2M smart gateways may be of interest, since flexible programmability for in-network data processing is obviously a useful option for constructing such M2M smart gateways, where various intelligent functionalities are supposed to be implemented between a large number of distributed sensors and the traditional transport network. For example, such smart functions include protocol conversions between sensor-optimized protocols and the traditional Internet protocols [23], and data aggregation for enabling efficient M2M communications.

We have recently attempted to resolve the issue of the explosion in the number of flows in M2M networks. Our proposal involves the following two ideas, flow aggregation/release and route control.

The first idea of flow aggregation and release is straight-forward as follows. In M2M network lots of short packets are generated from trillions of sensors and collected at the cloud data centers. We foresee at least two network problems in such an architecture: (1) if we forward all the short packets as they are, filling queues with lots of short

packets may cause long latency, and (2) core switches and routers may have to account for lots of flows produced by a large number of sensors. In order to solve these issues, we propose to aggregate packets belonging to the same flow into larger packets at the smart M2M gateways and release them before their getting at the data centers to solve the first problem. If we observe packets belonging to different flows are headed along the same route, we aggregate those packets to reduce the number of flows and release them at the fork points of the routes that individual flows must follow, thus, solving the second problem.

The second idea is to perform route control for the sake of facilitating flow aggregation mentioned above. We consider SDN is useful for this sort of mechanism, since the logically central controller can control the route so that multiple flows can follow the same route as much as possible to optimize the flow aggregation.

We have designed and implemented a prototype system for flow aggregation/release and route control, and have learned that one of the requirements for such a system is to perform flow aggregation/release as quickly as possible (at the line speed) and without much processing delay, especially for mission critical applications such as disaster recovery and public safety. Although it would be possible to implement route controls via SDN and flow aggregation and release as virtual network functions in NFV, merely orchestrating these two functions by redirection of packets may not be optimal, since it incurs much delay in the redirection of packets back-and-forth between two components. The ideal solution is to offload such flow aggregation/release virtual network functions into the data-plane of SDN switches and expose a new set of southbound APIs to NFV control layer so that we can perform the whole process without the redirection between SDN and NFV components.

5.3 Custom Actions for OpenFlow Switches

OpenFlow is a great SDN solution for flexibly adding programmability on network operations over the control plane. In the context of coupling SDN and NFV to enable specific intelligent in-network processing, we foresee lots of example network appliance solutions currently implemented in hardware platforms are ported to virtual machines (VMs) on top of commodity hardware and switches, while the redirection of flows to specific in-network virtual functions on VMs are controlled by SDN switch solutions such as OpenFlow. However, we posit that some of simple functionalities could be offloaded to the data plane of SDN switches to reduce latency of processing, if SDN switches are implemented in software, that is, the boundary between NFV and SDN solutions become unclear.

One example we pursue in this area of research is custom actions for OpenFlow software switches. Our FLARE system adopts Toy Block Networking programming model [21], and especially supports Click software optimized for our platform. Our toy-block networking model allows construction of data plane network functionalities in software

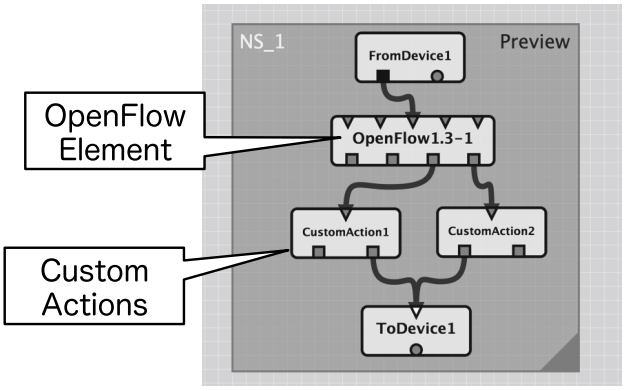

Fig. 3 Custom actions for OpenFlow switches.

blocks just like infants playing with toy-blocks to create shapes, reusing the existing blocks to quickly form desirable larger compound blocks.

Using our toy-block networking model, we implement OpenFlow Switch Version 1.3 as a Click element, so that we can attach extra functionalities to the OpenFlow switching logic, especially extending the standard actions to supporting a variety of custom actions by just connecting a virtual port of OpenFlow Click element to the elements that implement the custom actions.

In order to show the proof of concept of cross-boundary optimization between SDN and NFV, we have designed and implemented various custom actions connected to the OpenFlow switching logic. The construction of such a logic can be performed in our prototype GUI just through drag and drop of each network functions and drawing edges along data paths depicted in Fig. 3.

Usually, custom actions must be executed at the controller via Packet-In and Packet-Out, which leads to significant latency overhead since packets that OpenFlow switches do not know how to handle must be sent to the controller (Packet-In) and after necessary processing, sent back to the switches again (Packet-Out).

Our proposal of offloading packet processing being done in the controller into the switches is similar to servers offloading IP and TCP checksum calculation, ARP reply, and various TCP optimization such as TCP Offload Engines (TOE) [24], into NICs so that both server load and response time may be reduced. The other optimizations such as TCP Transmit Segmentation Offload (TSO) and Large Receive offload (LRO) for TCP (TRO) [25], etc. have been already implemented in the recent Linux Kernel [26].

As such example custom actions, we have evaluated several security related virtual functions such as p2p locator, bot miner, and port scanner [27] connected to OpenFlow switch logic. According to our preliminary evaluations, we show the effectiveness of our optimization in terms of processing time. While OpenFlow Packet-In data processing takes milliseconds of response time, our approach reduces this time down to microseconds.

As a result, we show that software-defined data plane

can easily achieve cross-boundary optimization of SDN and NFV functionalities. We strongly believe that there are lots of other functionalities offloaded from the NFV area to SDN data plane area, once software-defined data plane offers deeper programmability.

5.4 Content/Information Centric Networking

Even drastic example applications of software-defined data plane are new network architectures.

Content Centric Networking (CCN) [28], Data Oriented Network Architecture (DONA) [29], Named Data Networking (NDN) [30], and Information Centric Networking [31] are research activities attempting to depart from the Internet, namely, the existing TCP/IP networking and to define new ways of naming content objects. Instead of specifying the location of the content objects such as Universal Resource Locators (URLs) indicating the end-systems that store and provide the content objects, these network architectures define the ways of naming content objects directly. They inherently require in-network caching, because multiple end-systems may carry the cached data copies of the original content and behave exactly like the origin servers of the content objects.

Although content oriented networking architectures do not assume the underlying IP networks and could be implemented in a clean-slate manner, most research projects allow their implementations overlaid on top of IP networks. However, overlay approaches often defeat the purpose of content oriented architectures, because the overlays inherit the shortcomings of the underlying layers.

In the present research efforts regarding SDN and NFV have not fully investigated into building new network protocol stacks yet. Only a few research projects discusses the possibility of protocol oblivious forwarding (POF) in SDN switches, but the programmability (flexibility) and the ease of programming are limited yet, compared to the programmability discussed in the NFV research arena.

We posit that we could accelerate the research and development of innovative clean-slate network architectures such as content oriented ones, if we could make the best use of SDN and NFV, namely, incorporate flexibility and ease of programming into SDN software-defined data plane so that we may define capabilities for handling new protocols as a subset of southbound API and publish it to both controllers and virtual network functions.

6. Conclusion

In this paper, we posit that software-defined data plane in SDN may lift the boundary between SDN and NFV since we observe more and more functions in both areas being implemented flexibly in software, as in Linux platforms on general purpose processors and operating-system-based many core network processors, which allows one (1) to define useful data processing within data plane in SDN and (2) to publish the access method to them as a (sub)set of southbound interface (SBI) so that virtual network functions in NFV and applications may call SBI to utilize functions efficiently.

Although one may view the concept of supporting deeper programmability in software-defined data plane as a mere extension to the current SDN, we believe that its implication is significant in that we may be able to change the paradigm of the networked systems and applications. The benefits identified in this paper are summarized as follows but the applications of deeper programmability are certainly not limited to the following: (1) bringing interaction and cooperations between applications and networks such as application specific traffic control, by defining new protocols on top of the existing Internet protocols, and realizing even offload of application functions on the end-systems, e.g., smartphones, into data plane of SDN switches so that network may assist data processing of resource scarce end-systems closer than the nearby cloud data centers, (2) achieving cross-boundary optimization between SDN and NFV by offloading a part of virtual network functions under the southbound API, so that they may be executed without much delay (avoiding Packet-In/Out), and, (3) accelerating the research and development of clean slate network architectures.

We strongly believe that enabling deeper programmability in SDN data-plane with ease of programming open the door to bringing more innovations by offloading useful capabilities from NFV and applications into SDN by defining flexible boundary between them than just combining usage of two worlds as they are and that the benefits brought by the enhancement to SDN and NFV surely lead to improving the quality of diversifying communication networks and services of today.

References

[1] “Openflow.” http://archive.openflow.org
[2] “Flare: Deeply programmable network node architecture.” http://netseminar.stanford.edu/10_18_12.html
[3] “Protocol oblivious forwarding.” http://www.poforwarding.org
[4] “Intel data plane development kit (dpdk).” http://dpdk.org
[5] “Netronome, inc.” http://www.netronome.com
[6] “Octeon multi-core processor family.” http://www.cavium.com/OCTEON_MIPS64.html
[7] “Raza microelectronics, inc.”
[8] “Tilera, inc.” http://www.tilera.com
[9] “Ezchip, inc.” http://www.ezchip.com
[10] “Us ignite.” http://us-ignite.org
[11] “Paradiso.” http://paradiso-fp7.eu
[12] “Netfpga.” http://netfpga.org
[13] “Cavium, inc.” http://www.cavium.com
[14] “Xlr, xlp, xls processors.” http://www.broadcom.com/products/Processors/Enterprise
[15] “Broadcom, inc.” http://www.broadcom.com
[16] “Netlogic microsystems, inc.” http://www.netlogicmicro.com
[17] “Tilegx processor family.” http://www.tilera.com/products/processors/TILE-Gx_Family
[18] “Barefoot networks.” http://www.barefootnetworks.com
[19] “Opendataplane.” http://www.opendataplane.org
[20] “Openflow hardware abstraction.” http://archive.openflow.org/wk/index.php/HardwareAbstractions
[21] M. Fukushima, Y. Yoshida, A. Tagami, S. Yamamoto, and A. Nakao,

"Toy block networking: Easily deploying diverse network functions in programmable networks," Proc. ADMNET Workshop, COMSAC, 2014.
[22] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "Tcp fast open," Proc. 7th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT), 2011.
[23] G. Mulligan, "The 6lowpan architecture," Proc. 4th Workshop on Embedded Networked Sensors, EmNets '07, pp.78–82, New York, NY, USA, ACM, 2007.
[24] J.C. Mogul, "Tcp offload is a dumb idea whose time has come," HotOS (M.B. Jones, ed.), pp.25–30, USENIX, 2003.
[25] A. Menon and W. Zwaenepoel, "Optimizing TCP receive performance," USENIX 2008 Annual Technical Conference on Annual Technical Conference, ATC'08, pp.85–98, Berkeley, CA, USA, USENIX Association, 2008.
[26] "The linux kernel archives." http://www.kernel.org
[27] H. Farhadi, P. Du, and A. Nakao, "User-defined actions for SDN," Proc. Ninth International Conference on Future Internet Technologies, CFI '14, pp.3:1–3:6, New York, NY, USA, ACM, 2014.
[28] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, "Networking named content," Proc. 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '09, pp.1–12, New York, NY, USA, ACM, 2009.
[29] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," Proc. 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, pp.181–192, New York, NY, USA, ACM, 2007.
[30] "Named data networking." http://named-data.net
[31] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an information-centric internetworking architecture," SIGCOMM Comput. Commun. Rev., vol.40, pp.26–33, April 2010.

Akihiro Nakao received B.S. (1991) in Physics, M.E. (1994) in Information Engineering from the University of Tokyo. He was at IBM Yamato Laboratory, Tokyo Research Laboratory, and IBM Texas Austin from 1994 till 2005. He received M.S. (2001) and Ph.D. (2005) in Computer Science from Princeton University. He has been teaching as an associate professor (2005–2014) and as a professor (2014-present) in Applied Computer Science, at Interfaculty Initiative in Information Studies, Graduate School of Interdisciplinary Information Studies, the University.