Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane

Haoyu Song
Huawei Technologies, USA
Santa Clara, CA, 95050
haoyu.song@huawei.com

ABSTRACT

A flexible and programmable forwarding plane is essential to maximize the value of Software-Defined Networks (SDN). In this paper, we propose Protocol-Oblivious Forwarding (POF) as a key enabler for highly flexible and programmable SDN. Our goal is to remove any dependency on protocol-specific configurations on the forwarding elements and enhance the data-path with new stateful instructions to support genuine software defined networking behavior. A generic flow instruction set (FIS) is defined to fulfill this purpose. POF helps to lower network cost by using commodity forwarding elements and to create new value by enabling numerous innovative network services. We built both hardware-based and open source software-based prototypes to demonstrate the feasibility and advantages of POF. We report the preliminary evaluation results and the insights we learnt from the experiments. POF is future-proof and expressive. We believe it represents a promising direction to evolve the OpenFlow protocol and the future SDN forwarding elements.

Categories and Subject Descriptors

C.2 [**Computer Communication Networks**]: Network Architecture and Design

Keywords

SDN, OpenFlow, POF, forwarding plane

1. INTRODUCTION

SDN intends to keep the network intelligence in software. The underlying forwarding elements (FE) should be flexible and simple. OpenFlow [1] separates the control and forwarding planes to improve network application programmability and allow the two planes to evolve independently. Since its inception, OpenFlow has become the de facto standard SDN south-bound interface, supporting many mainstream protocols and forwarding actions.

HotSDN13, August 16, 2013, Hong Kong, China.

Although powerful, OpenFlow faces several challenges: (1) So far it follows a reactive rather than proactive evolving path. New protocol primitives are continuously added in new versions of the OpenFlow specification. The root cause is that the forwarding plane is assumed to be protocol aware. The controller and FEs communicate at the protocol semantics level so the forwarding plane has to understand specific packet formats in order to extract search keys and execute packet processing and forwarding actions. (2) The forwarding plane is almost stateless. It lacks the capability to actively monitor flow status and change flow behavior without the involvement of the controller. These attributes lead to several undesirable consequences:

- The control plane and the forwarding plane are not sufficiently decoupled. The control commands and packet processing instructions contain rich semantics that require the forwarding plane to have protocol-specific knowledge. As the network evolves and packet processing gets more complex, the required protocol-specific instructions increase explosively. The forwarding plane intelligence also complicates the FE design.
- There is no easy way to modify the packet format or add auxiliary data to the packets traveling in the network, let alone to apply user-defined protocols. Innovations in networks still have to use the overlay model, enforce tunneling encapsulations, or overload existing header fields. The process introduces unnecessary complexities and clean slate solutions are still impossible.
- Adding new forwarding protocol features requires an overhaul to both FE and controller, and in the worst case, a total redesign of the FE chip-set and hardware. Although OpenFlow v1.3 [2] has already covered 40 header fields, it fails to support many well-known protocols, especially non Ethernet based ones. It is also difficult to extend OpenFlow to carrier networks. As new protocols keep emerging (e.g. VXLAN and NVGRE) and some applications gain popularity (e.g. variations of L2/L3 VPNs), major revisions to the standard are expected, which makes the standard unstable and eventually unwieldy. This instability will result in a slower rate of standard adoption.
- The limited expressivity can seriously impact the forwarding plane programmability. OpenFlow lacks the capability to support stateful network processing on the data-path which covers a full range of L4 to L7 services. Even the basic L2 MAC learning function is

difficult to realize today. Relying on the controller for all state tracking gives rise to scalability and performance issues.

We argue that the requirement for the underlying forwarding infrastructure to recognize the protocol format is an artificial remnant of the old network model with closely coupled control and forwarding planes, and a fundamental flaw that will eventually hinder SDN from fulfilling its programmability potential. We also believe that OpenFlow should provide a uniform programming interface that can not only handle the basic forwarding functions but also other more sophisticated network services.

2. MAKE SDN FE A WHITE BOX

Current network devices are black boxes. The applications and services are deeply embedded in the proprietary and closed hardware/software systems. Once installed, any system upgrade and new service deployment are at the mercy of device vendors. The process can be painfully costly and slow. OpenFlow brings hope to this situation. OpenFlow and some other flavors of SDN proposed today make the network devices gray boxes with limited programmability. Third party control software can program network behavior to some extent. Network operators do not always need to go back to the device vendors to implement variations in network behavior. However, the programmability is limited to what the interface has exposed and what the forwarding plane device has been programmed or hard-wired to support. The FEs maintain a fair amount of intelligence and the controller can do nothing beyond that. What the SDN really needs is a fully programmable forwarding plane in which the FEs are all white boxes. These boxes retain only processing and forwarding capabilities but their behavior is totally under control of the SDN controller.

It is a proper analogy to compare the SDN with the PC systems as shown in Figure 1. The success of personal computing is due to the fact that PC disaggregated the mainframe computer's vertical integration model. SDN is purposely following a similar transition. However, there is a noticeable difference between the OpenFlow-based SDN and the modern PC system.

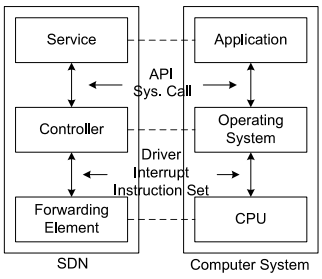

Figure 1: SDN vs. PC Analogy

With a handful of standard architectures (e.g. x86 and ARM), a CPU can do almost anything with a surprisingly small set of atomic instructions. The CPU and the other hardware resources communicate with the operating system through a standard instruction set and limited mechanisms (e.g. interrupts). The elegance is gained from the application-agnostic nature of the CPU. It just performs basic instructions without knowing or restricting what is exactly going on, while the operating system abstracts the hardware and presents a higher level function interface to the applications. We believe the FEs should have a CPU-like role in the framework of SDN to ensure their flexibility and extensibility. That is, the FEs *should not* require any priori knowledge and understand the application semantics.

To this end, a simpler forwarding plane abstraction will ensure a truly open and evolvable SDN architecture. While the current solution seeks to provide a high level function interface for the FE, we think this trend actually tightens the coupling between the control and forwarding planes. Our proposal therefore is directed toward a simpler and more generic forwarding plane model for the FE. We believe a concise set of protocol independent instructions are sufficient to compose any network services from the control plane. We name such instructions a Flow Instruction Set (FIS). FIS is carefully defined to be platform independent. Meanwhile, it is optimized for networking applications so each instruction represents a primitive which allows efficient hardware implementation. Constructing network services with FIS is just like writing application programs in assembly language. This level of complexity can be hidden by using high-level languages and function libraries that can be compiled into machine code. There are some ongoing research efforts towards this direction [3]. This approach can help to relieve some low level design frictions due to the mismatch between CPU and FE therefore make the design better mimic the modern high level languages in computer systems.

An FE chip running FIS is like a CISC or a RISC. This architecture ensures the elegance and simplicity of the underlying FEs. The control and forwarding planes are completely decoupled so they can both evolve independently. The interoperability of heterogeneous FEs is also automatically satisfied. The more important implication is that the architecture offers the ultimate programmability that the SDN desires. Users are free to use their own network protocols and packet formats, enabling an even further departure from the status quo. FIS expands the expressivity of the forwarding plane. There are cases in which multiple flow table entries map to the same set of instructions/actions. FIS can be used to create common subroutines that are shared by multiple table entries. There are also cases the packet parsing would become cumbersome and inefficient if every step is conducted through table lookup. FIS can be used to perform quick packet parsing and data analysis through logic, comparison and branching instructions.

If this vision is realized, the landscape of the network business model will be thoroughly changed. The different pieces of network is fully decoupled and bound by standard interfaces. New services and applications unimaginable today can be freely deployed down the road. As long as the performance is satisfactory, service providers will never need to deal with the vendors again after the devices are deployed. We cannot emphasize enough on how profound this can benefit the service providers. The innovation cycle will be significantly shortened and demand less investment. Meanwhile, the abundant standard-based device and software supply chain will no doubt lower the overall networking cost.

It is worth to note that the proprietary and heterogeneous hardware can still work with a highly specific interface while complying with the standard and keeping a platform-specific hardware abstraction layer (HAL).

3. POF OVERVIEW

POF makes the forwarding plane totally protocol-oblivious. The POF FE has no need to understand the packet format. All an FE needs to do, under the instruction of its controller, is to extract and assemble the search keys from the packet header, conduct the table lookups, and then execute the associated instructions (in the form of executable code written in FIS or compiled from FIS). As a result, the FE can easily support new protocols and forwarding requirements in the future.

On the surface, the processing flow appears no different from what OpenFlow has defined. However, in OpenFlow, the search key assembly is conducted by explicitly spelling out the target header fields (e.g. IPv4 destination address). Although the message conveyed by the controller is simpler, the packet parsing burden is shifted to FEs. The FEs must understand the packet format in order to extract the required header bits. In contrast, in POF, the FEs have no knowledge on any forwarding protocols. The packet parsing is directed by the controller through a sequence of generic key assembly and table lookup instructions. To achieve this, we expand the packet metadata as a generic scratch pad associated for each packet in the processing pipeline. Controller can use it freely to cache temporary data (e.g. the input port and the resolved packet header fields so far) in the life span of a packet. A search key is simply defined by one or more {*offset, length*} tuples, where *offset* indicates the skipped bits from the current cursor within the packet (or metadata) and *length* indicates the number of bits that should be included in the key starting from the *offset* position. All the tuples are concatenated to form the complete search key.

Similarly, all the forwarding instructions are purposely made protocol agnostic. There are no longer actions like *push MPLS label* or *decrement IP TTL* which requires semantic interpretation. Instead, in FIS, for the instructions that manipulate the packet or metadata (e.g. insert, delete, and modify), {*offset, length*} is used to locate the target data. The new data for insertion and rewrite can be from the immediate data carried with the instructions or the packet metadata.

The benefit of such a mechanism is obvious. For example, OpenFlow v1.3 defines three push tag actions already: *push VLAN header*, *push MPLS header*, and *push PBB header* [2]. However, some popular network service such as VPLS requires to push new Ethernet headers which cannot be done in OpenFlow. We can imagine similar actions will continue to emerge in the future. Continuing to add more and more protocol specific primitives to the specification is not a sustainable method. In POF, we just define one instruction, *AddField*, which can cover any such actions. The *AddField* instruction contains the parameters to define the location in the packet for the new data to be inserted and the source of the data.

We have instructions that can apply simple mathematical (e.g. Addition, Subtraction, and Shift) and logic (e.g. AND, OR, NOT, and XOR) operations on the data in packet or in metadata. To handle some relatively complex functions in the legacy protocols, we also add special instructions such as IP checksum and TCP checksum calculation. Keeping this kind of operations atomic is important to balance efficiency and flexibility. However, we still need to maintain the protocol-oblivious semantics for these instructions. The instruction parameters define the algorithm used, the data covered by the operation, and the location where the checksum is compared and inserted.

We enhance the forwarding plane programmability by several additional mechanisms. First, we prefer the data-path to be able to keep track of the flow status. This is achieved by introducing the concept of flow metadata. While a packet metadata has a life span of a packet, a flow metadata has a life span of a flow. Since not all the flows need flow metadata, we maintain a flow metadata resource pool and assign flow metadata to flows on an on-demand basis. Note that the fields of *counters* and *timeouts* specified in OpenFlow flow entries are indeed some sort of flow metadata. But these are not enough (and sometimes unnecessary) to address service-rich programming requirements. Like the packet metadata, generic flow metadata serves as a scratch pad attached to a flow that can be arbitrarily configured and used by network programs. With this flexible mechanism, any flow information (e.g. sequence number, anomaly, time stamp) can be recorded and communicated easily to satisfy variable application needs.

Second, we add instructions that allow the data-path to actively manipulate the flow tables. In current OpenFlow, the flow table modification can only be initiated from the controller. This not only excludes many useful network functions but also unnecessarily increases the processing burden of the controller. With the new instructions, when some conditions are met, flow tables can be created and flow entries can be added, deleted, or updated automatically. In this case, the controller is notified of these events to keep the network and the controller synchronized. The more dynamic interaction between the controller and FEs allows to weave sophisticated network services with ease.

Third, for many flow tables, each flow entry may invoke same set of instructions but with different parameters. Installing these instructions for each flow entry can be terribly inefficient in memory utilization (e.g. think about a FIB with millions of entries). Instead, we only store the unique parameters with flow entries and let the flow entries that share the same set of instructions point to a common instruction block. Within each instruction block, we can further use the conditional and unconditional jump instructions to support branching operations.

Fourth, we treat the statistic counters as a shared resource just like we treat the flow metadata. This enables us to only allocate counters to the flows that the applications are interested in monitoring. In addition to memory saving, this mechanism provides another benefit: since one counter can be assigned to multiple flows and a flow can use multiple counters, users can easily collect aggregated statistics and count based on different criteria. Similar advantage applies to flow metadata. When multiple flows share the same flow metadata, the flow metadata effectively becomes an inter-flow communication vehicle.

Fifth, we categorize the possible lookup tables into different types such as Direct Table (DT), Exact Match (EM), Longest Prefix Match (LPM), and Masked Match (MM). The table type can be extended to include other match tables such as Range Match (RM) and RegEx Match (REM) in the future. The MM table uses the most generic search keys in which arbitrary bits can be masked as "don't care" bits. While the table type is hinted by the controller, it is totally up to the FEs to choose the best way to implement

the tables: TCAMs or algorithmic search engines, off-chip or embedded memory, and dedicated or shared table resources. To further facilitate the efficient implementation, we apply the notations of physical tables and logical tables. A physical table requires dedicated hardware resource. Each physical table is assigned a unique *id* which associates with the table type, capacity, and other parameters. Multiple logical tables can be mapped to same physical table. There are two use cases: (1) The same table entries are used by multiple logical tables but the instructions may differ (e.g. IP forwarding and RPF); (2) The logically separated tables are co-located and share the resource of the same physical table (e.g. multiple EM tables in one hash table).

As in OpenFlow, the forwarding pipeline is realized by chaining the logical tables. The instructions associated with each table entry are effectively a piece of program written in FIS and downloaded from the controller. The pipeline simply repeats the parse-match-action steps and eventually forwards packets to proper ports. The detailed FIS specification can be found at [4]. We are still working on refining it. The work is by no means complete and final.

4. PROTOTYPE AND EVALUATION

We have built two POF FE prototypes: one is the hardware-based and the other is software-based. They both share the same controller. The controller is based on the open-source Floodlight SDN controller [5] with the POF extension. Floodlight is a cross platform Java-based software controller under the Apache license. We use Floodlight as the main communication module to the FEs and add POF module with GUI on top of it. The GUI is used to manually configure the forwarding pipeline step by step. A configuration file can also be directly loaded into the FEs through the GUI. Figure 2 shows the block diagram of the POF controller. Other applications can be developed using the API to the *POFManager*.

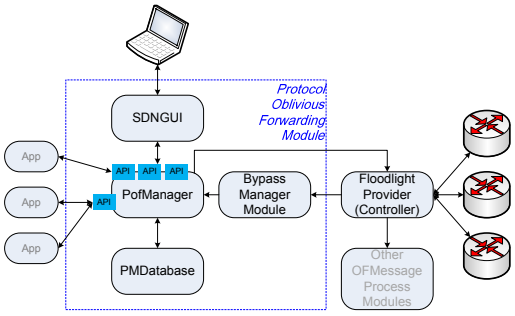

Figure 2: POF Controller

The hardware-based prototype works on Huawei's NE5000 core router platform. The line card we used has an in-house designed 40G NPU and the half slot interface card has eight GbE optical interfaces. The multi-core NPU runs in RTC mode. The NPU has large memory bandwidth which can access both on-chip and off-chip memories. It has a TCAM interface and also supports algorithmic search engines. The NPU has its own proprietary microcode instruction set. The POF instructions are implemented as functions written with the microcode.

The processor on the router MPU act as the OpenFlow client which terminates the OpenFlow channel and relays the messages to/from LPU on the line card through IPC. The processor on the router LPU interprets (ingress) and encapsulates (egress) the messages. The Hardware Abstraction Layer (HAL) and the device driver modules also run on the LPU. The HAL provides a standard and hardware-independent interface for the controller to configure and query the NPU. The HAL communicates with the NPU by calling the device driver.

The software-based prototype is written in C on a Linux environment. It can be installed and run in a Linux server with NICs. It has a similar architecture to the hardware-based prototype except there is no distinction of MPU and LPU. All the modules run in one CPU. Figure 3 shows the block diagrams of the prototypes. We make both the controller and the soft FE open source. The detailed documentations and source code can be downloaded at [4].

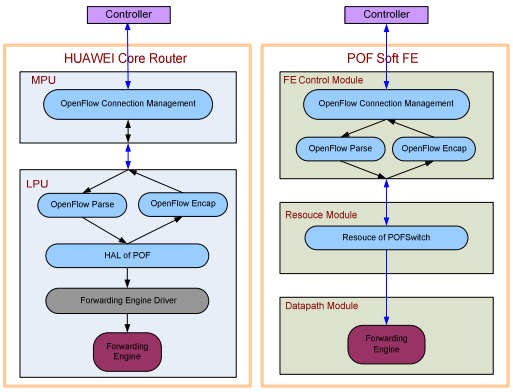

Figure 3: POF FE (a) Hardware-based (b) Software

Due to space limitations, we only report some of the evaluation results on the hardware-based prototype, which provide insight into how well existing hardware can support POF and what can be done to improve the performance. Figure 4 shows the experimental setup. The SmartBits is a tester that can generate and detect arbitrarily formatted packets. With this configuration, we can test both the functionality and the performance of the design.

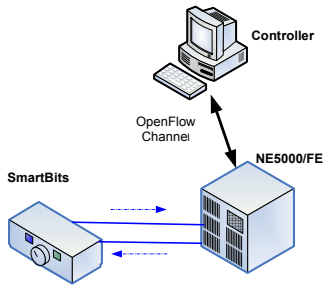

Figure 4: POF Experimental Environment

Table 1 summarizes the code performance for some instructions. In the table, *Cycle* means the number of clock cycles consumed for the instruction execution; *Jump Count* is the count of thread switches. Every thread switch consumes about 25 clock cycles.

Our general observation is that POF typically reduces the forwarding throughput by up to 30%. This is not too bad

Table 1: Instruction Performance

Instruction	Cycle	Jump Count
SetFieldFromValue	29	1
SetFieldFromMetadata	39	1
ModField	31	1
WriteMetadataFromPkt	39	1
WriteMetadataFromValue	26	1
AddField	27+3*offset	3+2*offset
DeleteField	15+3*offset	3+2*offset
GotoTable	52+27*num	6+6*num
GotoDirectTable	15	5
Count	9	2
Output	49+20*len	10+4*len
Meter	36	5
offset: field offset in unit of byte num: number of match fields len: packet length in units of 32-bytes		

considering the NPU is not POF optimized. Depending on the pipeline depth, there might be multiple *GotoTable* instructions which may be a performance bottleneck. The main reason is that key assembly cannot take advantage of the hardware packet parsing modules in the NPU. We believe a POF optimized silicon chip can help to improve the performance.

Among many other use cases, we implemented basic IPv4 forwarding plus ACL operation. Every flow table provides statistics using the *Count* instruction. The entire forwarding flow consumes 1,256 microcode instruction cycles and 167 thread switches. The overall throughput reaches 48Mpps. Since the 40G line card is designed to sustain the peak packet rate of 60Mpps, this number accounts for 20% performance loss, but it can still support line speed forwarding for 80-Byte packets.

The POF flow table entry is typically wider than the OpenFlow table entry. Table 2 shows the flow table update performance. The MM and LPM tables give the same performance because the line card uses a TCAM for both type of tables.

Table 2: Table Update Performance

	MM/LPM	EM
insertion speed	12 k/s	20 k/s
deletion speed	28 k/s	23 k/s

POF calls the NPU driver on the LPU to realize the flow entry insertion and deletion. The driver is the table update performance bottleneck. Still, the update performance is good enough to satisfy most typical network applications (e.g. The peak BGP prefix insertion rate and deletion rate is 1.6K/s and 13.5k/s correspondingly, as of 2013 [6]).

5. POF USE CASES

In addition to supporting all existing network applications that OpenFlow supports, an immediate benefit of POF is that it can support new and user-defined forwarding protocols without upgrading the FEs and the south-bound interface. Data Center protocols like VXLAN and NVGRE can be easily supported in POF controller software. While OpenFlow can only work on Ethernet-based protocols today, POF can support other line protocols such as Fibre Channel, ATM, and POS. Another example to show the extensibility of POF is that it can support Named Data Network (NDN) or Content-Centric Network (CCN) [7], in which names rather than addresses or locations are used to direct routing and forwarding. First, users can define the most efficient packet format without resorting to IP overlay. Second, the name extraction, table lookups, and forwarding instructions are well defined in POF. In fact, the name lookup in NDN is simply LPM. Third, the Content Store, as a caching mechanism, can be supported by the active data-path instructions.

Since the controller understands the packet format and can transform the packets at the network edge, it is possible to attach and detach user-defined scratch data to and from packets for in-band information transmission. This opens the door for unlimited network OAM capabilities. Its great potential is yet to be explored.

While enabling many new services, POF also offers opportunities to simplify the current protocol stack. The complex packet transformation, tunneling, and overlay are deeply rooted from the protocol-dependent nature of the forwarding plane infrastructure. Once this artifact is removed, users can pay more attention on the services delivered rather than on the forwarding mechanisms. For example, POF can realize a fabric as suggested in [8] but with a lighter protocol than MPLS.

Even for the protocols covered by OpenFlow, some applications are still impossible now. For example, the stateful firewalls only allow packets through if they fall in an acceptable range of TCP sequence numbers [9]. OpenFlow cannot match on TCP sequence numbers and record the TCP flow states. With POF, the stateful firewall can be implemented easily. NAT service is another example where the seemingly simple packet header rewrite is just beyond the coverage of current OpenFlow actions. By contrast, POF is generic enough to handle arbitrary packet editing tasks. POF also enables functions such as MAC learning. This kind of active data-path functions can significantly offload the controller, reduce response time, and improve scalability.

Network devices are often expected to do more than simply forwarding packets, such as stateful inspection, multipath load balancing, and deep packet inspection (DPI). Many other L4 to L7 network functions are of interest to the service providers to provide value-added or differentiate services to customers. These functions are not easy to implement in the current OpenFlow model. As a flexible and extensible data-plane model, POF can also bridge Network Function Virtualization (NFV) [10] and OpenFlow.

6. RELATED WORK

The ideal characteristics of the SDN forwarding hardware have been articulated in [11]. However, the proposed architecture is in line with the current OpenFlow switch model; therefore, the forwarding hardware flexibility and generality is limited. An effort to provide an API for Open Router Platforms on proprietary hardware was shown in [12]. It highlights the challenges to achieve such a daunting task, which hints, in our opinion, that a standard and simple forwarding hardware model is more proper for SDN programmability. Several issues of OpenFlow specification were discussed in [8]. It reaches the similar conclusion that the hardware

in compliance with OpenFlow specification is neither necessarily simple nor sufficiently flexible. While the authors suggest to keep the network intelligence at edge and to use a simple MPLS-based network fabric for interconnection, we aim more general FEs that can be used everywhere and for any services.

7. CONCLUSIONS AND FUTURE WORK

For SDN to support heterogeneous FEs, the common approach is to just provide higher level abstractions (WHAT) for the south-bound interface and ask a smart FE to handle all the detailed work (HOW). However, every time a new feature is introduced, the FE needs to be amended to know how to handle it. This is painful and unsustainable for both service providers and infrastructure providers. We believe SDN ultimately requires a simpler and more generic forwarding plane model for which not only WHAT but also HOW can be programmed. After all, software is good at handling complexity.

In order to ensure a smooth and successful transition path towards POF, we need to pay close attention to two issues: (1) OpenFlow Compatibility. Since many legacy OpenFlow-enabled devices already exist, one way to keep using them while introducing POF for new services is to augment the current OpenFlow specification with the POF related instructions. The POF-optimized FEs can be swapped in as needed in order to ensure a smooth system migration. (2) FE Implementation. It is ideal to support POF natively with brand new silicon chips. In reality there are many existing network chips in use today. We need to use these existing chips to support POF first. There are basically two modes to support POF on existing hardware. The first one treats each instruction as a macro or function. A piece of microcode is written and compiled for each instruction. The data path calls these macros or functions for packet processing. This approach is suitable for NPs and other multi-core processors with their own instruction sets. The second mode requires software on the FEs to translate and map the POF instructions into the hardware modules in ASICs and some NPs for packet parsing and processing. Although this approach is a little rigid and complex, it may have performance advantages.

POF is an exciting idea and a groundbreaking technology to shape the future SDN's forwarding plane infrastructure. We have just revealed the tip of the iceberg. A lot of hard work remains to be done to make this really happen. Meanwhile, the POF idea may appear hard to swallow for some incumbent device vendors because it is set to overturn the current ecosystems. But we believe it reflects the SDN's true requirement for a programmable infrastructure. It offers brand new opportunities and nurtures a healthier and evolvable SDN ecosystem. Therefore, it is better to start the research and market education sooner rather than later. It also calls for the contributions from the whole network community to polish and advance the technology. By publishing our initial work and the open-source software, we wish the whole community can improve it and build innovative services on top of it. It is our hope that our efforts can stir interest in both industry and academia so we can join forces to push the idea and realize the vision.

Many other tasks and research work lay ahead: we need to perfect the FIS, come up with optimal chip and system architecture, design high level languages and compilers, come up with appealing applications, and solve related technical issues. But the most important thing is standardization, just like what has already happened in the PC industry. Performance could still be a differentiator but everybody talks the same language so applications gain the ultimate freedom. This should happen to SDN too. OpenFlow is a good start and should keep evolving under the guidance of ONF [2]. We believe it is the most proper platform to incorporate this idea. Although we are still at an early stage for POF research, it is not too early to think about a finally future-proof and expressive version of the OpenFlow specification.

8. ACKNOWLEDGMENTS

The Network IP Research team, including Wei Cao, Zhi Chai, Hongfei Chen, Jun Gong, Wenyang Lei, Shuying Liu, Jian Song, Xiaozhong Wang, Zhen Wang, Xiaofei Xu, Jingzhou Yu, and Yuanming Zheng, helped develop the idea and the prototypes. We thank Professor Jennifer Rexford, David Walker, and Michael Freedman at Princeton University for early discussions.

9. 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, April 2008.

[2] (2012) OpenFlow Switch Specification v1.3.0. [Online]. Available: http://www.opennetworking.org/

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *ACM SIGPLAN ICFP*, 2011.

[4] (2013) Protocol Oblivious Forwarding. [Online]. Available: http://www.poforwarding.org

[5] (2011) Floodlight. [Online]. Available: http://floodlight.openflowhub.org/

[6] (2013) BGP Reports. [Online]. Available: http://bgp.potaroo.net/

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. Briggs, and R. Braynard, "Networking Named Content," *Communications of the ACM*, vol. 55, no. 1, January 2012.

[8] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A Retrospective on Evolving SDN," in *ACM SIGCOMM HotSDN Workshop*, 2012.

[9] S. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 3, July 1989.

[10] M. C. et.al., "Network Functions Virtualisation - Introductory White Paper," in *SDN and OpenFlow World Congress*, October 2012.

[11] M. Casado, T. Koponen, D. Moon, and S. Shenker, "Rethinking Packet Forwarding Hardware," in *ACM SIGCOMM HotNets Workshop*, November 2008.

[12] J. Mogul, P. Yalagandula, J. Tourrilhes, R. McGeer, S. Banerjee, T. Connors, and P. Sharma, "API Design Challenges for Open Router Platform on Proprietary Hardware," in *ACM SIGCOMM HotNets Workshop*, November 2008.