## T5: Protocol Oblivious Forwarding

### Discussion

<table>
<thead>
<tr>
<th>Topic</th>
<th>Support points</th>
<th>Criticizing points</th>
</tr>
</thead>
</table>
| Core concept        | ● Promising to enable future developments in networking.  
● Current OpenFlow specification couples control plane tightly to the supported protocols, while POF offers flexibility in protocol support.  
● Such flexibility supports moving away from vendor-specific solutions.  
● The use of Instruction set relates to familiar design approaches in computer design and supports adoption readiness of the POF end users. | ● Serious redesign of network elements may be needed to support the POF concept.  
● Is there really need for POF or does majority of SDN applications still rely on traditional protocols? When and is such shift in protocol use expectat that POF will make sense?  
● More flexibility in design may lead to explosion of options to support which only complicates maintenance, scale and performance. |
| Instructions set    | ● In infant stage with missing/incomplete theoretical base.  
● The choice of instruction set able to capture application and network level needs is not trivial. Moreover it should be widely agreed upon.  
● It is not clear if protocol-specific instructions can be fully omitted. Perhaps there will be intermediate state before some protocols can be 'forgotten' and such instructions not needed.  
● What is the burden that the more complex packet processing puts on flow tables? |                                                                                                                                                                                                     |
| Data plane manipulations | ● FPGA or ASIC-oriented implementation may help out the more efficient realization of processing heavy manipulations.  
● Decreased dependency on the controller and more freedom to handle operations at the data plane, pushing down communication overhead with the control plane. | ● What are the pros and cons at adding more functionality to forwarding elements, e.g., where is the best place to introduce middle-box type functionality?  
● Introducing limited control plane decisions breaks the separation of planes concept. |
| Performance         | First evaluations of performance are promising with few more                                                                                                                                           | Can POF really compete in efficiency with OpenFlow, which |


<table>
<thead>
<tr>
<th>vendors/proposals backing up similar conclusions.</th>
<th>is tuned for specific protocols? Careful evaluation of implementation complexity and hardware support is needed.</th>
</tr>
</thead>
</table>

Critique

**Pappas, Christos**  
The papers make the case for packet forwarding with a protocol agnostic instruction set. The main motivation behind these proposals is a stricter separation of control plane and data plane; forwarding according to predetermined protocol semantics does not unleash the full flexibility and programmability of SDN. The idea overall makes sense, but there are certain concerns:

1. How is the FIS instruction set determined in order to allow application development and how are we ensured that this instruction set is sufficient to realize networking applications. In theoretical CS there are theoretical tools and concepts, such as the Turing Machine, that show that any computer algorithm can be implemented with them (not exactly true, TMs do not provide a complete model for computational problem solving). These theoretical tools can drive the development of actual CPUs to implement algorithms. I think that the POF proposals capture an important idea, but lack a more theoretical and convincing explanation that they can realize arbitrary applications.

2. In the first paper (POF), there are certain questionable concepts as to whether they realize the goal of a stricter separation. Specialized instructions are proposed to deal with some complex operations that are protocol specific (e.g., IP checksum and TCP checksum). The reason to have such specialized instructions that are protocol specific is due to the popularity of currently deployed protocols. Although the reasoning makes sense, is this not a step back to the OpenFlow direction where more and more protocol specific operations are added? How many such specialized instructions will we end up with? Is this not kind of the current situation, just by making a circle? The other concern are instructions that allow the data path to actively manipulate the flow tables. Is this not coupling control and data-plane more?

3. This is more of a philosophical concern: to realize a network with more functionality, today traffic is tunneled to middleboxes that perform this functionality. More capable FEs means that there is a shift and this functionality can get implemented inside the network (at least partially) by FEs. Is there a compelling reason to make all the FEs much more capable? What is wrong with today's model?

**Yu, Xinyuan**

1. If the POF can be widely used, the forwarding instruction set needs to be supported widely. So a standardization of forwarding instruction set needs to be realized. Just as in the field of PC, a platform is formed firstly and then some applications are developed. However, a standardization is very difficult to be invented by a single company.

2. POF is hard to be deployed. Because the switches need to support the forwarding instruction set. Thus, in POF deployment, many traditional network devices need to be swapped out. It may cost much. Besides, the
network devices’ expense is raised, either in software-based prototype and hardware-based prototype, since it is more complicate. As a result, the cost of network may be raised.

3. Many things needs to be implemented to realize POF. A high-level language, complier, new instruction set and so on. To realize these elements, complexity may be brought. I don’t know if high-level language is efficient in this case. It is possible that the implementation of POF makes the network slow.

Miladinovic, Djordje

Protocol-Oblivious Forwarding (POF) could be a next step towards SDN era. It brings plenty of benefits comparing to OpenFlow, but there are also few things one should be aware of before deploying POF.

1. Will there be a significant performance drop if we switch from OpenFlow forwarding to POF? Most clients use, and will continue using standard protocols we inherited from traditional networks in their systems (IP, MPLS,...). OpenFlow switches recognize these protocols and are optimized to forward packets of this kind (for almost all of them). Their hardware is built for efficient forwarding of these packets, while the hardware of POF switches is unaware of these protocols, so it’s generality leads to less efficient forwarding and implies performance drop.

2. This critique is a continuation of the previous one. Do all the clients really need specific non-standardized protocols? I reckon that most of the SDN clients (perhaps 95%) will continue using traditional protocols with extra flexibility using OpenFlow. So mostly there will be no need for user-defined protocols. Using POF in this case would be wasting performance for a generality that is not needed. Perhaps POF should be used only when OpenFlow is not enough, which means that POF is not going to be used on a large scale.

3. If POF does become widely used, very good compilers/interpreters will be needed to convert user policies into Flow Instruction Set (FIS). Not doing so could result in another performance drop. Complex policies make this task rather challenging.

4. Another very important step towards migration to POF is a standardization of southbound interface. This also a challenging problem, and in my opinion should be considered as soon as possible.

Shinde, Pravin

Criticism:
1. Too much of text explaining why FE should not be protocol specific.
2. The design of FIS building block seems to be little ad-hoc and it is not clear if it has any formal foundation.
3. An example of how FIS would look for some network function could have been useful in understanding it better. They do not talk about any new protocol implemented using POS-FIS to prove the Protocol-Obliviousness.
4. Complex functions like calculating checksum which could have tested FIS better are avoided and are implemented directly.
5. The POS-FIS provides the flexibility of programming at cost of reduced forwarding speed and increased requirements from hardware resources (to store complex POS-FIS programs), but these costs are not well quantified in the paper.
6. Typically, the size of FIB in Forwarding elements is limited, and it's not clear to me, how larger the FIB hardware resources needs to be compared to resources needed by current Openflow design.
7. After reading the paper, I do not have an idea of how big the typical code block to be inserted in forwarding table will be.

**Defense:**
1. It seems a decent attempt to develop FIS, even though the explanation was week.
2. The instructions are classified in a group, which helps in grasping them.
3. There are also arguments about completeness of the instructions.
4. The paper also gives an example of how the flow will work for a simple case with POS-FIS.
5. The paper shows how the code in languages like P4, C and Java will look which then can be converted to POS-FIS automatically.

**Stefan Schmid**
1. I really like the Protocol-Oblivious Forwarding (POF) approach and the Flow Instruction Set (FIS) implementation they describe in the paper(s). It is hard to find something to criticize. The paper itself is well written but I am missing a thorough evaluation and comparison with other OpenFlow switches and standard networking hardware. They only mention that throughput is reduced by 30 percent (compared to what?).

2. As we have already seen several times, there is a trade-off between flexibility and performance. If the forwarding device should be completely independent and does not run predefined protocols, it needs more processing power to run protocol specific microcode. The hardware implementation they provide is based on a NPU and the additional FIS layer they put on top of the NPU instruction set introduces some performance issues. But as they propose in the paper, an ASIC optimized for FIS could speed up packet processing again.

3. They introduce instructions that allow the data-path to manipulate the flow table which is normally not possible in OpenFlow where only the controller can modify flow table entries. Is this not against the idea of SDN to separate data- and control-plane? Is this not a step backwards towards original networking hardware? This might also lead to security issues: the data path could be modified in a way so that the forwarding hardware modifies its flow table, changing packet switching and introducing malicious behavior.

**Defense**

**Tae Ho, Lee**

1. One important aspect of Internet architecture is investigating how the “narrow-waist” of the Internet can be evolved. In line with such studies, protocol-oblivious forwarding can facilitate the evolution at the “narrow-waist” of the Internet (i.e., using something else than IPv4) in the data-plane.

2. In the evaluation, the authors claim that the forwarding speed is reduced by 30%. How did they get this number, under what setting? The numbers should be qualified.

3. The authors state that they “add instructions that allow the data-path to actively manipulate the flow tables.” Furthermore, the authors claim that “with the new instructions, when some conditions are met, flow tables can be created and flow entries can be added, deleted or updated automatically.” What are the
instructions? In addition, the paper would be stronger if the authors provide examples of "some conditions" to clarify the authors' statements.

van Gelder, Jasper

1. In the article about POF-FIS they offer a very flexible instruction set to define network protocols which could be very nice to talk to for example Cisco proprietary protocols. With FPGAs you can even improve the speed of processing these self defined protocols.

2. This flexibility can also have a down side when it comes to maintainability, since anything is possible which might trigger a wild growth of protocols (this might help network improvements though).

3. The statement from Protocol-Oblivious Forwarding: "Even for the protocols covered by OpenFlow, some applications are still impossible now. For example, the statement....." indicates something that comes back in all our discussions namely that we want SDN to do everything. Which is something we have to look out for because we might end up with bloatware or in reimplementation of things that have already been done.

Chothia, Zaheer

1. Already the first OpenFlow paper [1] talked about "Type 0" vs. "Type 1" switches. The former support a few barebone operations (forward, encapsulate, drop); the latter are envisioned to be more capable:
   i) "rewrite portions of the packet header (e.g., for NAT, or to obfuscate addresses on intermediate links)"
   ii) "Flow Tables will be able to match on arbitrary fields in the packet header, enabling experiments with new non-IP protocols."

POF, P4 and others are realizations of these ideas. [2] is a recent overview document (from Sept 2014) which is easy to read and covers servers as a useful introduction. (POF ~= assembly language for packet processing and P4 the high-level, target-independent language above.)

2. Each incremental revision is swelling: OpenFlow 1.5 now has 44 match fields, but the minority (only 13) are mandatory and tough luck if your desired protocol isn't supported (e.g VXLAN). The recent spec also has the notion of 'instructions' and 'action sets' (see sec 5.5, page 25) intended for more flexibility, but it delivers short. There are eleven hard-coded functions for specific tasks (e.g. TTL decrement, push VLAN / MPLS label) and only two of the instructions are actually required: Goto-Table, Write-Actions.

=> The forwarding plane is too tightly coupled to a fixed set of protocol formats and this limits innovation and extensibility (Example: NAT rewrite and TCP state / seq. no. are two tasks which cannot be effectively realized).

3. The POF proposal is backed by a vendor (Huawei), but this doesn't compromise the position being advocated. The problem and motivation are clearly stated without being boastful or overly one-sided. Particularly good is the development of a functioning prototype and accompanying numbers on instruction latency, overhead (~20%) and flow table update rates. The reader is sufficiently informed and in a position to discuss / evaluate the idea.

4. Related to the section on NPUs: there are many existence proofs that packet processing at line rate is feasible: Cisco QuantumFlow can handle ~40 Gbps, 58 Mpps (w. 64 byte packets) [3], similar with
SwitchBlade which does the same with FPGAs [4] or Click [5] providing even richer functionality directly in software.

[Further discussion points]
1. Completeness: are the suggested primitives enough to cover both current and __future__ envisioned protocols?
2. Security: what implications does programmability brings (e.g. timing leaks / side channels?) Also, how strong is the isolation between ‘threads’?
3. P4 parsers: DSL looks familiar (C bitfields); Erlang binaries have a succinct and elegant syntax + efficient processing. Overall, great idea treating infrastructure more like code. (This should also bring the culture of testing, review, etc. from software development.)

[References]

Chang, Michael

1. Makes Openflow “viable”: One of the most commonly discussed problems with Openflow is its lack of expressiveness when it comes to various applications that a forwarding elements needs to support such as NDN or CCN. Openflow v1.3 already has over 40 header fields, and these fields mostly only support Ethernet-based applications. With the advent of POF-FIS into Openflow, the future Openflow 2.0 can remove the header fields (perhaps altogether?) and more scalably handle more applications.

2. Comfort level of Network Engineers/Operators: In the first paper, the author draws parallels between SDN and the PC and makes remarks on the programmability and flexibility of the CPU. A CPU is a very intuitive concept for most engineers with a computer science background, and approaching SDN from an “older model” point of view may be interesting. The concept of a instruction set (FIS) could be more readily accepted by the industry that typically is slow to take on new concepts. The application benefits from having a GUI and CLI that are already developed; this is not a critical part of the technology, but allows the benefits to be more readily understood.

3. Less dependence on controller: POF is an ideal solution for increasing the presence of proactive flows and reducing dependence on the controller for handling flow misses. Because of the expressiveness of the FIS, it is possible to write more encompassing procedures at the data layer level to handle traffic that passes through the forwarding element.