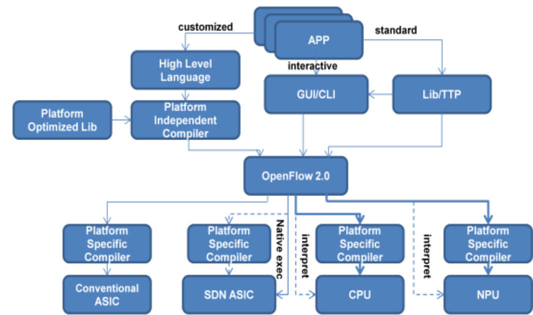
Coherent SDN Forwarding Plane Programming

Haoyu Song, Jun Gong, Hongfei Chen
Huawei Technologies

SDN needs to program heterogeneous forwarding elements (FE) with different forwarding architectures. Ideally, OpenFlow should provide a uniform interface to allow platform-independent programming and platform-specific compiling. In this paper we discuss an SDN programming framework which is suitable for flexible and protocol-oblivious forwarding plane. Specifically, we describe three different forwarding plane programming approaches and their tradeoffs. We show how an OpenFlow instruction set can support different programming styles which map to different forwarding chip architectures. We also show how applications can be compiled into NP-based FEs and compare it with the interpreter-mode implementation.

We envision the future OpenFlow 2.0 should have the following features in order to support flexible forwarding plane programming: ***(1) Protocol Oblivious***. Allow the forwarding plane to be protocol oblivious so that effectively no network forwarding behavior needs to be hardcoded in FEs; ***(2) Platform Agnostic***. Allow the controller to be agnostic to the FE architecture, which means a right level of forwarding plane abstraction can isolate the controller from the FE implementation details. ***(3) Flexible and platform optimizable***. Allow simultaneously coarse-grained programming (using packages or library functions) and fine-grained programming (using bare OpenFlow instructions). We propose a unified forwarding plane programming framework as depicted in the right Figure. The center pillar of this framework is the OpenFlow interface which provides a set of generic instructions as well as other forwarding plane provision mechanisms (e.g. resource discovery and configuration). In addition to provide a decoupling point between the control plane and the forwarding plane, other ideal properties of this interface have been discussed in [1] which include versatile, future proof, and protocol and platform agnostic.



The key for designing such a programming interface is to make it work at a right abstraction level. It cannot be tied to particular FE architecture but it should be easy to map to any platform and allow sufficient optimizations which fully exploit the FE capabilities. We propose some novel OpenFlow features which enhance the forwarding plane programmability and open the doors for data path performance optimization. One notable change is that we abstract the actions associated with each flow entry as a piece of program. This allows us to decouple the match keys and actions. The actions can be downloaded to FEs separately in the form of instruction blocks. A flow entry only needs to include a block ID to infer the associated instruction block. By doing so, different flow entries can share the same instruction block, and there is theoretically no limit on how many instructions one flow entry can execute. To facilitate the instruction block sharing and at the same time enable the differentiated flow treatment, we further enhance the flow entry with a parameter field. Developer can use this field to define any parameters for the associated instruction block. For example, in an egress table, when all the entries execute the same output action, which is only stored once in an instruction block, they may target different output ports. In this case, the output port number is stored in the parameter field of each flow entry. This mechanism is powerful to compress the code space and implementation complexity. We also abstract the globally shared memory resource as flow metadata. Flow metadata can be shared by flow entries to store statistics (i.e. counters) or any other information such as flow states. The expressivity of flow metadata enables the stateful forwarding plane programming. The core of the new OpenFlow interface is a set of instructions, which function as the intermediate vehicle between the platform-independent programming environment and each individual target FE platform.

Above the OpenFlow interface, any network forwarding applications need to be converted to the standard OpenFlow instructions first. There are several ways to do it: ***(1) Use a high level language and a platform-independent compiler***. The high level language provides yet another layer of abstraction that supports modularity and composition [2]. It frees developers from dealing with particular FE architecture and conducting tedious and error-prone flow level match-action manipulations. We are exploring the possibility of using C as our choice of high level language. Note that it is impossible to achieve 100% platform independency at current stage due to the diversified forwarding architectures. The application programs should follow some programming style upfront and may include some preprocessor directives and use some platform-optimized libraries. ***(2) Use GUI/CLI for interactive forwarding plane programming***. This is like programming in assembly language directly. Although needing to handle flow level details, it is fast and efficient. The GUI can also be used to download compiled applications to FEs. We have implemented an open-source GUI to support this kind of programming method [3]. ***(3)***

Use pre-compiled standard libraries. Many prevailing network applications, such as standard L2 switching and L3 IP forwarding, can be specified as standard forwarding processes. These applications can be developed by any third party and included in a standard library. Conceptually, this is in line with the Table Type Pattern (TTP) developed by ONF FAWG [4]. Note that the above three approaches are not mutual exclusive. One application can be realized by using more than one way. Very often an application can be customized by programming in high level language or just uses a standard library application, and then GUI/CLI is used for library application download, dynamic runtime updates, and interactive monitoring.

Each type of FE may have its own platform-dependent compiler which compiles the standard OpenFlow instructions to its local structure. We categorize FEs into four groups based on the type of main forwarding chips on them: ***(1) Conventional ASIC-based***. Conventional ASICs for FEs typically have a fixed feature set and are not considered programmable. However, since they are designed to handle classical forwarding processes at high performance, they are still usable in SDN but in a more restrictive way. In this case, the standard library applications are the most suitable way to program the FEs. Some ASICs are configurable and able to switch between different modes to support different applications. In this case, the customized programming is not impossible but need to be applied in a highly disciplined way to ensure the compatibility. ***(2) SDN ASIC-based***. Recent research has started to study SDN-optimized chips [5]. Many companies are designing new chips to support flexible SDN programming. These chips have embedded programmable capability for general packet handling but are also heavily populated with hardware-accelerated modules to handle some common network functions for high performance. For these chips, it is feasible to use any kind of programming approach. When the future OpenFlow 2.0 is standardized, it is conceivable that we can design a chip that can natively execute the OpenFlow instructions without even needing another compiler in data plane. ***(3) CPU-based***. Albeit having lower performance compared with the other platforms, general purpose processor is still the most flexible platform which can easily support any programming approach. The soft/virtual switch running in CPU can be implemented using two different modes: compiler mode and interpreter mode. The former compiles an application (in the intermediate form of OpenFlow instructions) into machine code and the latter requires the forwarding plane code to directly interpret and execute OpenFlow instructions. The open source soft switch in [3] works in interpreter mode. We are working on a compiler-mode implementation based on x86 platform. ***(4) NPU-based***. NPU is similar to CPU except that NPU has more cores and is optimized for network applications. So far existing NPUs all use proprietary programming interface. Even these chips are claimed to be C-programmable, the programming interface is not open to SDN developers, and their programming environments are not compatible with each other. We believe OpenFlow can make them support open programming and present unified interface to developers so that they can be used for SDN without any modification.

Similarly, there are two modes for NPU-based FE implementations: compiler mode and interpreter mode. We have finished the prototype implementations in both modes on an in-house designed NPU, which is used in Huawei's NE5000 and NE40 series routers. In our implementations, we show how the hardware accelerated modules can be parameterized and called through extended function instructions. We show how we can optimize the application performance by taking into consideration the specific platform architecture. Our implementation is applicable to most of the other RTC-style NPUs. We have conducted extensive evaluations on different forwarding applications to compare the code efficiency and performance. We found that the compiler-mode implementation performs consistently better than the interpreter-mode implementation. For a typical IP forwarding process in routers, the compiler-mode implementation needs 57% less microcode instructions than the interpreter-mode implementation. Compared with the conventional non-SDN microcode implementation, the compiler-mode implementation is just 11% worse. With the same number of micro cores, an compiler-mode implementation easily doubles the throughput of an interpreter-mode implementation.

Our future work includes completing the proposed SDN programming framework by developing implementations of the missing pieces in the Figure (e.g. the platform-independent compiler and some platform-dependent compilers for other FE platforms) and demonstrating real SDN applications through the full programming process. This programming framework can be considered as a proposal for the future OpenFlow 2.0 standard.

REFERENCES

[1] H. Song, "***Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane***", in ACM SIGCOMM HotSDN Workshop, 2013.

[2] N. Foster, M. Freedman, A. Guha, R. Harrison, N. P. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, and D. Walker, "***Languages for Software-Defined Networks***", IEEE Communication Magazine, Feburary 2013.

[3] Protocol Oblivious Forwarding (POF) Open Source Website (2013). http://www.poforwarding.org

[4] Open Network Foundation Forwarding Abstraction Working Group, https://www.opennetworking.org/working-groups/forwarding-abstractions

[5] 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 Proceedings of the ACM SIGCOMM, 2013.