Review: Programming Protocol-Independent Packet Processors

Summary

Software Defined Networking (SDN) provides user programmatic control over their network. There was a need for a vendor agnostic interface to program the forwarding devices so that devices can be programmed easily. Openflow was the first step in this direction, but in order to make it easy to adapt it was limited to a small number of protocols and adding a user defined header requires significant effort which is a drawback of openflow. Another drawback being that the switch does not have an easy way to express its capabilities to the control plane. These drawbacks were the motivation to define an abstract forwarding model representing how different switch implementations process packets with three goals in mind 1) Protocol Independence , 2) Target Independence, 3) Reconfigurability in the field.

In order to accommodate for new header formats the abstract forwarding model includes a programmable parser with multiple *match+action stages* making the model protocol independent.Using a new programming language P4 a programmer would be able to define how the switch is to process the packets, the compiler which knows about the underlying switch would compile the program and tell the target how to process packets. SInce the programmer does not need to know about the underlying switch configuration the model is target independent. In addition to defining how the packets are processed, switches have a function of populating flow tables by adding and deleting flow entries, accounting for reconfigurability.

Programmers start by defining a parse graph, which is a directed acyclic graph specifying the order and contents of packet headers to be processed. In chips that only support fixed parse graphs the compiler only checks if the parse graph is consistent with what the hardware supports, and in chips with programmable parsers the compiler generates a state machine described by the parse graph. Control program calls for specific actions to be performed at different stages. These actions are defined from a collection of action primitives like insert and delete. If the hardware supports then primitives corresponding to complex actions can be defined. Control program also specifies a table type for each *match-action table* and a description of how packets are to be processed. FIrst step in compiling control program is to define a control graph by detecting if two tables can be executed parallelly or if there is a dependency and must be executed sequentially.FInally the control graphs are compiled to a variety of hardware and software platforms, like if the underlying platform is a FPGA based switch then the logical tables are mapped to physical tables, for switches consisting of a pipeline of *match-action stages* each logical table can be mapped to multiple physical tables or vice versa.

This is better than OpenFlow because OpenFlow specification is targeted towards fixed function switches where P4 allows for more flexible switches whose functionality can be modified in the field.

Strengths

1. One of the major strengths is that P4 makes it easy to try and test new features like new protocols and additional header fields and things like that. If a new header needs to be handled then a programmable parser needs to be programmed to handle the header format and the code can be compiled and instantly the hardware would support the format.
2. Better visibility of the entire network, it is easy to diagnose and also gather different kinds of data like the time taken at each switch, the rules followed by the packets, path taken by the packets by using custom tags and modifying the code to handle the tags , and all of this can be done at hardware speed. This information would be useful to analyse the network and improve it or to diagnose and fix any issues.
3. Testing hardware devices is an expensive task and requires a large amount of time, P4 makes testing cheaper and faster.

Weaknesses

1. Requires a programmable chipset which supports P4. Openflow did not require that, it just required a software upgrade, but P4 needs a specific chipset in the network devices and so requires time to implement and make it available in everyday devices.
2. The decisions which need to be programmed are extremely complex as they define how the packets are processed and the language is also completely new.

What did you learn from the paper?

I was familiar with openflow as a specification used to achieve software defined networking, but did not know that it is possible to actually program switches and define how they process packets at a hardware level using software.

Avenues of future work

Consider other aspects of a switch as a network device such as queuing and congestion handling and explore how they can be handled using P4.

What would you do differently

I would include some data showing the usefulness of P4 over OpenFlow and other approaches, because there are no quantifiable values in the paper.

Comments

The paper is extremely interesting, starts with a motivation and describes the approach clearly using examples and figures when needed. Gives a detailed explanation on handling different scenarios, but it would have been better if one use case such as tracking the path a packet took, could be explored using P4 in the paper including some quantifiable results.