Tiny packet programs as a p4-backed DSL

Tyler Ishikawa (tyi3) & Peter Li (pl488)

November 25, 2018

1 Introduction

In lecture, we touched upon the idea of using tiny packet programs (TPPs) to implement active networking [1] as a way to leverage the network topology to learn things about the global state of the network. TPP is introduced as "a simple, programmable interface that enables end-hosts to query switch memory from packets, directly in the dataplane." This mini language, despite its limited expressivity, may be useful for protocols that require global state information such as RCP, CONGA, packet provenance, etc. since they can collect that information from the network in a small number of round trip times (providing data close to synchronously consistent). We aim to implement TPP as a DSL in the the $P4_{16}$ programming language, profile its high-level expressivity and compactness trade-offs, and explore alternative primitives to the CSTORE and CEXEC instructions, if time permits.

2 Design

This section is subject to change throughout the course of development. As of now, we plan to make use of the development environment from coursework to implement the DSL and any additional statistics tracking required to breathe life into our demo programs (e.g. RCP, NetSight). This development environment will consist of a bare-bones $P4_{16}$ switch running on bmv2 with some custom mininet json-defined topologies, and additional scapy scripts used for testing (we will try to leverage the set of send.py and receive.py from the most recent telemetry assignment). For simplicity, we will be using a predetermined routing scheme like source routing and assume the following about network topologies for testing: 1) they are connected, and 2) we will not observe any node failures. Our goal here is to focus solely on the TPP implementation, and minimize time spent on details of how to collect statistics, as those will be punted to our demo programs.

We originally explored the idea of using an existing `openflow.p4` switch, since a completed-to-spec openflow switch should provide the needed state information to implement RCP. However, the open-source code for openflow.p4 from the

p4lang/switch repository on GitHub [2] seems to be written in $P4_{14}$ and may require a disproportionate amount of engineering effort to port over to $P4_{16}$.

We intend to build a TPP "interpreter" as a standalone module whose parsing rules, ingress, and egress logic are called to by the baseline switch mentioned above (whichever one it may be). The finalized list of switch states that TPPs have access to will be further specified in the final report of this project, since we still have yet to decide on demo instances, but a tentative list of states are listed in the "Register" table below. We are especially prioritizing the data needed for RCP: link queue size, link rx-utilization, version number for link fair rate, computed fair rate for link data.

The original paper discusses the different kinds of instructions they implemented for their TPPs, but specifies that each instruction is at most 4 bytes in size. Encoding these instructions with ASCII characters would not be feasible under this constraint, so a more precise bit-wise encoding is necessary. The original paper does not specify the encodings that they used, so we have drafted the following as our own version of a plausible encoding scheme.

Instruction	Description
`PUSH, FROM_REGISTER`	Pushes the value at `FROM_REGISTER` onto the packet memory's stack
`LOAD, FROM_REGISTER, TO_LOCATION`	Copies the value at `FROM_REGISTER` to the packet memory at `TO_LOCATION`
`POP, TO_REGISTER`	Copies the value from the top of the packet memory stack to the `TO_REGISTER`
`STORE, TO_REGISTER, FROM_LOCATION`	Copies the value from the packet memory at `FROM_LOCATION` to the `TO_REGISTER`
`CEXEC, REGISTER, LOCATION`	Move on to the next instruction if the value at `REGISTER` is equal to the value at `LOCATION`, else stop
`CSTORE, REGISTER, OLD_LOC, NEW_LOC`	If the value at `REGISTER` is equal to the value at `OLD_LOC`, copy the value at `NEW_LOC` to `REGISTER`, else do nothing

Command	Bit Representation
PUSH	000
LOAD	001
POP	010
STORE	011
CEXEC	100
CSTORE	101

Register	Bit Representation
Switch:SwitchID	00 000
Switch:L2Counter	00 001
Switch:L3Counter	00 010
Switch:FlowTableVerNum	00 011
Switch:Timestamp	00 100
Port:LinkUtilization	01 000
Port:BytesReceived	01 001
Port:BytesDropped	01 010
Port:BytesEnqueued	01 011
Queue:BytesEnqueued	10 000
Queue:BytesDropped	10 001
Packet:InputPort	11 001
Packet:OutputPort	11 010
Packet:Queue	11 011
Packet:MatchedFlowEntry	11 100
Packet:AltRoutes	11 101

Location	Bit Representation
Packet:Hop[N]	binary representation of N

These encodings will be used to represent the TPP instructions in the packets.

3 Progress

We are currently still in the design and feasibility exploration phase of this project. In terms of timeline moving forward, we expect to begin working on `tpp.p4` and the bare-bones `switch.p4` over Thanksgiving break and hope to be progressed far enough to work on a demo instance (likely RCP) by December 2nd or 3rd. Following that, we hope to be done with our demo instance and report within the week, shooting for a December 9th completion date with some time left over to accommodate for unexpected difficulties.

References

[1] Jeyakumar, Vimalkumar & Alizadeh, Mohammad & Geng, Yilong & Kim, Changhoon & Mazières, David. (2014). Millions of Little Minions: Using

Packets for Low Latency Network Programming and Visibility. ACM SIGCOMM Computer Communication Review. 44. 10.1145/2619239.2626292.

[2] https://github.com/p4lang/switch/blob/master/p4src/openflow.p4