Programmable Routers: RMT and P4

Tavis Johnson and Delta Lyczak

Abstract

Internet routers deal with significant amounts of internet traffic in everyday use. This increasing amount of traffic over the internet affects the performance, and speeds, at which these routers operate. The goal of our research project is to understand how routers manage processing packets at such high speeds while remaining flexible and reconfigurable. This paper demonstrates what we have accomplished during the 2021 DIMACS REU program understanding the layers of abstraction, key networking concepts, pipelined systems, along with the history of routing software, hardware, and information on the completed P4 programming exercises. We hope to be able to design improvements for the performance of routers and the management of campus network resources.

Introduction

Routers

Data sent over the internet travels in the form of discrete packets. These packets pass through devices called routers. Routers process headers containing information about the packets then forward them to the next step in their journey. Router design has been improved over the years in an effort to maximize router speed, and programmability while minimizing power and area requirements. Early router design optimized using hardware for an increase in speed. However, with this came problems as specifically designed hardware does not allow for ease of modification. This was something modern routers needed to solve. Modern Routers would need to now allow for reconfigurability and the research and development of new network protocols.

Router Programmability

In terms of programmability modern day routers need to be able to process data in accordance with protocols designed and/or set by the user. Thanks to the achievements of modern programmable routing hardware, packets are able to be parsed and matched with increased efficiency and power. This increased flexibility allows for more unique and efficient protocols to be designed. This along with the development of programming languages allows for greater control over the hardware.

This Project

Our goal for this project was to learn and understand the connection between the hardware and software in router programming. Then to use it to implement the network protocols that we had learned. This was important as we needed to understand what we were going to program before

we started to program it. If we could understand the constraints of the hardware we could see where the connections to the software played a role. Our main goal was to achieve an understanding of how these concepts relate to router programming. In addition, to understanding the history of the routers, protocols, and how algorithms and P4 language improve the speed and efficiency of router forwarding.

History

Multi-Gigabit Router (MGR)

Released in 1998, “A 50-Gb/s IP Router” describes the MGR router-design which would become the standard for about the next decade. MGR replaced previous hardware-based designs with a software approach. The core forwarding functionality of the router was implemented as assembly instructions for a reduced instruction set processor. Packet header fields are extracted by a fixed-function parser and passed to the forwarding engine that runs the forwarding algorithm. The fields are updated and the output-port selected before being passed back to be reassembled and passed to the egress port through a crossbar switch. Any cases not handled by the forwarding engine result in the packet being sent to a general-purpose network-processor that operates much slower but handles all cases (Partridge et al., 1998).

While MGR was capable of improving the speed and reconfigurability of routers, it had a few problems that resulted in the design eventually being superseded. The consequence of implementing the forwarding algorithm in software and relying on caching to improve performance is non-deterministic behavior. Depending on the workload, cache-misses may be common or the forwarding or network processors may be overworked. Additionally, limits on preassigned cache and memory sizes made it difficult for engineers to accommodate every use-case with differently sized tables. Finally, newer developments in programmable hardware eventually made hardware more practical due to speed and parallelism.

OpenFlow

In the years following the proliferation of MGR, hardware performance improved and pipelined router designs began to be created. These designs brought improvements over the non-deterministic, slower, and less parallelizable MGR routers. The main problem with these pipelined hardware routers was their fixed-function nature. Routers often offered a narrow set of control through vendor-specific interfaces and very-limited programmability. The OpenFlow protocol was introduced to improve the reconfigurability of these devices, opening new possibilities for research on emerging networking protocols and technologies in the field (McKeown et al., 2008)
.

Reconfigurable Match-Action Tables (RMT)

While the fixed-function pipelined routers of the 2000s offered some improvements over MGR, they lacked severely in reconfigurability. A number of fixed-function tables with a predefined number of rows would match on a number of fixed fields to decide how packets should be routed. With emerging network protocols and diverse switch use-cases, these limitations were real problems.

The solution to this was RMT. This design relied on a fully-programmable pipelined architecture that could be reconfigured in the field to serve different use-cases and new network protocols.

At the start of the RMT pipeline is a programmable parser configured with all required header fields and possible header orders. The parser extracts the required fields and passes them to the match-action pipeline. The match-action pipeline consists of a series of tables of programmable dimension that match on parsed fields and perform a corresponding programmable action depending on the match. Each table is one stage of the pipeline and the resources available to the pipeline are shared across all stages and can be allocated however is needed depending on the use-case. RMT describes a set of match-action stages on the ingress and egress side separated by a switching fabric though this distinction is mainly logical rather than physical. After the final match-action stage, the modified packet headers are passed to a programmable deparser that reassembles the headers and sends them along with payload to an egress port (Bosshart et al., 2013).

Programming Protocol-Independent Packet Processors (P4)

While the OpenFlow standard proved useful for programming early pipelined routers, the introduction of RMT came with a problem: OpenFlow was designed for modifying the behavior of fixed-function pipelined routers and was not expressive enough to leverage the full reconfigurability potential of RMT routers. To solve this problem, the P4 programming language was created, extending off of the OpenFlow standard. The logical structure of P4 models the physical structure of the RMT pipeline, allowing full control over RMT hardware. Furthermore, the logical layers of abstraction provided by the P4 language allow for the same program to be compiled for a number of different architectures including fixed-function and software routers. Programs not-suited to the target architecture will simply fail to compile while valid programs can be deployed to the hardware and managed using a standardized control-interface (Bosshart et al., 2014).

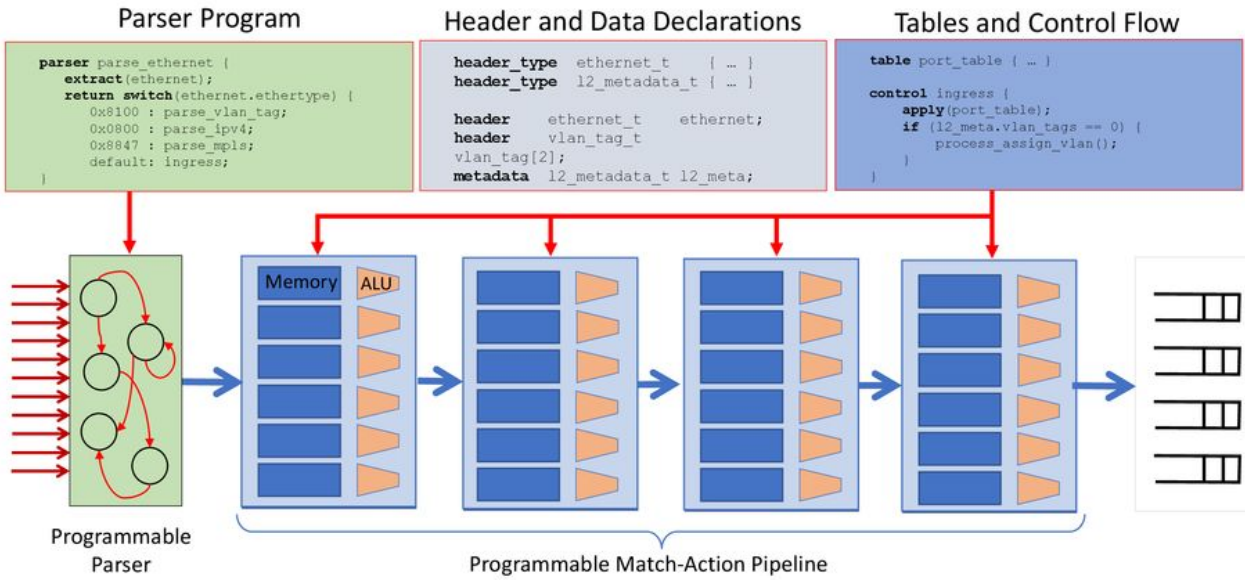


Figure 1: P4 language constructs mapped onto RMT pipeline
Credit: Rutgers Networking Lecture at https://slideplayer.com/slide/17811699/

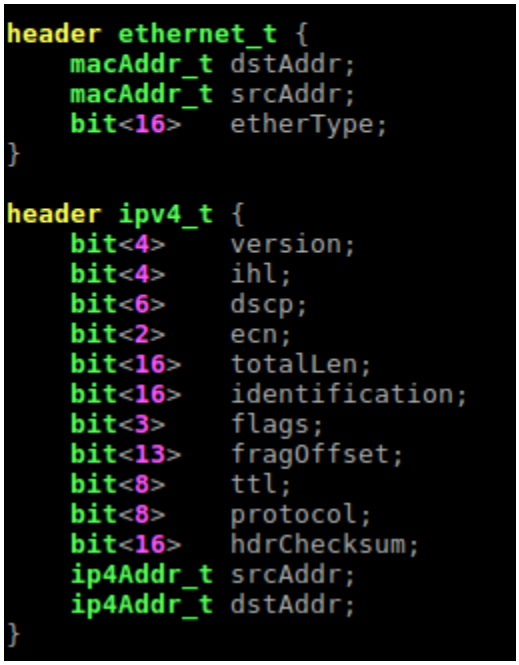
Applications

Mininet

The majority of the exercises we completed over the course of this project were done in software. There are multiple reasons that this might be advantageous. Experimenting using virtual software-defined networks decreases prototyping time and improves flexibility. Furthermore, it eliminates the need for expensive physical hardware and eases debugging. For the software-based exercises we performed, we used the virtual-network software Mininet. Mininet allows for easy provisioning and experimentation with virtual hosts and switches using a command-line interface and API. Using Mininet, we were able to quickly set up virtual network environments and experiment with each host from the command-line. Gaining an understanding of Mininet enabled us to better understand network topology concepts and Layer 2 and 3 network protocols.

P4 Exercises

The bulk of the code written for this project consists of a series of virtual software-defined networking exercises. These exercises were published in the *nsg-ethz/p4-learning* GitHub repository along with accompanying lecture-slides. Skeleton P4 programs were provided with important sections removed that we were responsible for completing and testing in a Mininet environment (*p4-learning*, 2021). Through these exercises, we investigated topics including but not limited to Layer 2 and 3 packet routing, equal-cost-multi-path routing, sketch data structures, packet time-to-live, and fast reroute. In addition to writing P4 programs for the router data-plane, some of these exercises involved writing control-plane code in Python as well as scripts to initialize router tables. Working on both the data-plane and the control-plane gave us a better understanding of proper separation of roles and the strengths and limitations of each.



```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<6>    dscp;
    bit<2>    ecn;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

Figure 2: Snippet of Header section

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {

        transition parse_ethernet;

    }

    state parse_ethernet {

        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol){
            6 : parse_tcp;
            default: accept;
        }
    }
```

Figure 3: Snippet of Parser Section

```
Your P4 program is installed into the BMV2 software switch
and your initial configuration is loaded. You can interact
with the network using the mininet CLI below.

To inspect or change the switch configuration, connect to
its CLI from your host operating system using this command:
  simple_switch_CLI --thrift-port <switch thrift port>

To view a switch log, run this command from your host OS:
  tail -f /home/p4/p4-tools/p4-learning/exercises/05-ECMP/solution/log/<switchna
me>.log

To view the switch output pcap, check the pcap files in
 /home/p4/p4-tools/p4-learning/exercises/05-ECMP/solution/pcap:
 for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Figure 4: Snippet of ECMP exercise

Tofino

Complications regarding scheduling and hardware-availability prevented us from working with the Tofino hardware-router as early as we would have liked. We were however, able to make the most of this delay by working on more advanced software-based P4 exercises. Once the Tofino router was reconfigured and ready for our exercises, we were able to get started working on a simple boilerplate exercise just prior to the end of the program. A few factors made this more difficult compared to the software-routing exercises. Working with the physical hardware required more setup and an understanding of some hardware-specific blackbox components. Additionally, the Barefoot software development kit we were using was designed around an older version of P4 ($P4_{14}$) with somewhat different syntax than the version we had used in our exercises (*tofino-boilerplate*, 2020). We were able to successfully step through the building process and are currently working on reverse engineering the boilerplate code and completing some exercises with it. We expect to continue working on the Tofino router for some time after the formal end of the 2021 DIMACS REU program.

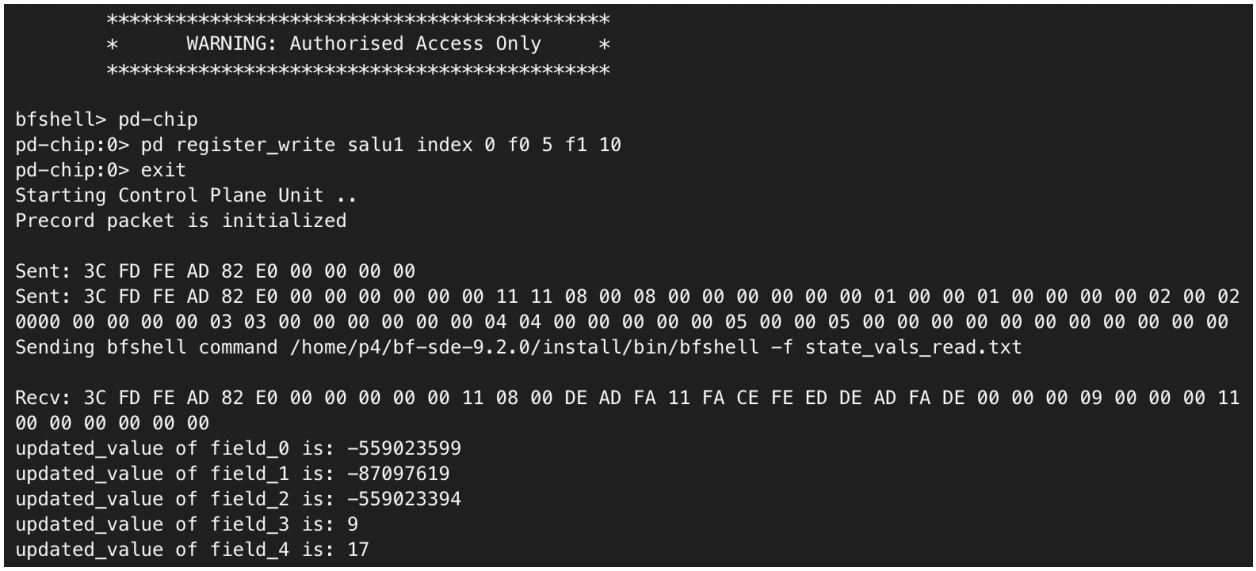


Figure 5: Tofino Boilerplate Control Plane Test

Discussion

Key Concepts

A key component of this project is the combination of application-exercises with historical research. Understanding the history of routing hardware was critical in developing a deep understanding of the logical structure of P4 and the way programmable routing is handled at scale. Additionally, studying routing technologies chronologically helped to reveal the interdependent nature of these architectures and tools. For example, because P4 mirrors the pipelined architecture of an RMT router, our research into the history, structure, and design of RMT gave us a much stronger understanding of the capabilities and limitations of the data-plane and the anatomy of a P4 program. Investigating L2 and L3 routing demonstrated the multiple levels of abstraction at which routers can effectively operate.

Each of the exercises taught valuable skills that could be carried over to different networking fields. From the equal-cost-multi-path routing, we were able to understand how to perform load balancing using hashing based on the packet flow. This allowed us to keep packet order while distributing the packets by ensuring packets with the same source and destination Ip as well as the same source and destination port hashed to the same next hop. In the Count-Min-Sketch exercise, we were able to understand how probabilistic data structures could be used to ensure packet speed at the cost of occasionally having a wrong answer. We were able to use registers and hash functions to estimate the occurrences of distinct elements. This also showed us how the estimation using the Count-Min-Sketch changes based on the flow as the accuracy sketch increased with the size of the flow. By studying these exercises not only did we receive a greater understanding of the P4 programming language but we received a greater understanding of networking concepts that we could translate to different fields.

Research and Team Meta

Before working on the project exercises, we read through papers on the history of routers. This allowed us to have a better understanding of what we were learning. We would then meet as a group and discuss the papers and our understanding along with any of the questions that we may have had. This created a strong foundation for when it came to working through the problems that occurred with each of the exercises. Working through the exercises the team worked by trying problems on their own and then moving towards the solutions any problems crossed would first be brought to the group to discuss understanding and then bringing any questions that could not be solved to our mentor Srinivas. By having group meetings we were able to break down and understand the material on a weekly basis. The academic environment of the group allowed for exchange of ideas from all areas of networking research.

Conclusion

While our work with the Tofino router was relatively limited, we were able to meet our listed forwarding goals and more in a software environment in addition to our historical background-research. The background we have gained regarding these tools and the technologies upon which they rely are applicable to network programming and beyond. While we intend to continue exploring P4 and beginning work on the Tofino router beyond the end of this program, the knowledge we have gained thus far is already widely-applicable. Pipelined hardware systems are common far beyond networking fields and stagnation in general-purpose processors has necessitated a shift toward programmable hardware solutions for problems where speed and throughput are high priorities. With this project, we have developed an intuition for the hardware trade-offs present in pipelined programmable hardware, the separation of responsibility between high and low-speed flows (data/control-plane).

Acknowledgements

We would like to thank Dr. Srinivas Narayana for his mentorship, the NSF OAC for their funding via grant OAC-1925482, the authors of our letters of recommendation, and the 2021 DIMACS REU program for this research opportunity.

References

Bosshart, P., Gibb, G., Kim, H. S., Varghese, G., McKeown, N., Izzard, M., ... & Horowitz, M. (2013). Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review*, *43*(4), 99-110

Bosshart, Pat, et al. "P4: Programming protocol-independent packet processors." *ACM SIGCOMM Computer Communication Review* 44.3 (2014): 87-95.

chipmunk-project/tofino-boilerplate: Chipmunk to Tofino. (2020). GitHub. Retrieved July 23, 2021, from https://github.com/chipmunk-project/tofino-boilerplate

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, *38*(2), 69-74.

nsg-ethz/p4-learning. (2021). [P4]. Networked Systems Group (NSG). https://github.com/nsg-ethz/p4-learning (Original work published 2019)

Partridge, C., Carvey, P. P., Burgess, E., Castineyra, I., Clarke, T., Graham, L., ... & Winterble, S. (1998). A 50-Gb/s IP router. *IEEE/ACM Transactions on networking*, *6*(3), 237-248.