Kristina Brunsgaard

Matthew Menten

Network Systems

Jose Santos

March 1, 2020

Project Proposal

Problem Description

For our final project, we plan to implement a network system using the programming language P4 to gain a better understanding of SDN and protocol independent solutions. Specifically, we plan to use mininet, a network emulation environment, to practice forwarding and receiving packets using the P4 routing protocol we develop. We'd like to develop a couple different routing solutions that look at application-specific optimizations and congestion control, and then compare the performance of each one.

Our main areas of focus are to modify the way traffic is routed for different applications (video streaming, FTP, HTTP, VoIP, etc.) and to create routing "tiers" for hosts. For example, we want to see if selecting different paths for different application types or user tiers can increase throughput and decrease latency for certain applications/hosts. Since video traffic is more sensitive to both throughput and latency as opposed to HTTP traffic, we wish to speed up its performance without severely impacting the performance of other application types. Similarly, high tier hosts will be given shorter routes and more available bandwidth to increase their performance.

To implement the above ideas, we plan to use an SDN controller and P4 to maintain multiple different topologies for our network. One of these topologies will be true to the network testbed itself, while the others will be slightly modified. These modified topologies will be similar to the true network, but the differences will allow us to use the same SPF path selection algorithm to generate different routing decisions for different application types and user tiers. For instance, traffic related to a low tier host, or low priority application, would be handled as if suboptimal paths are actually optimal.

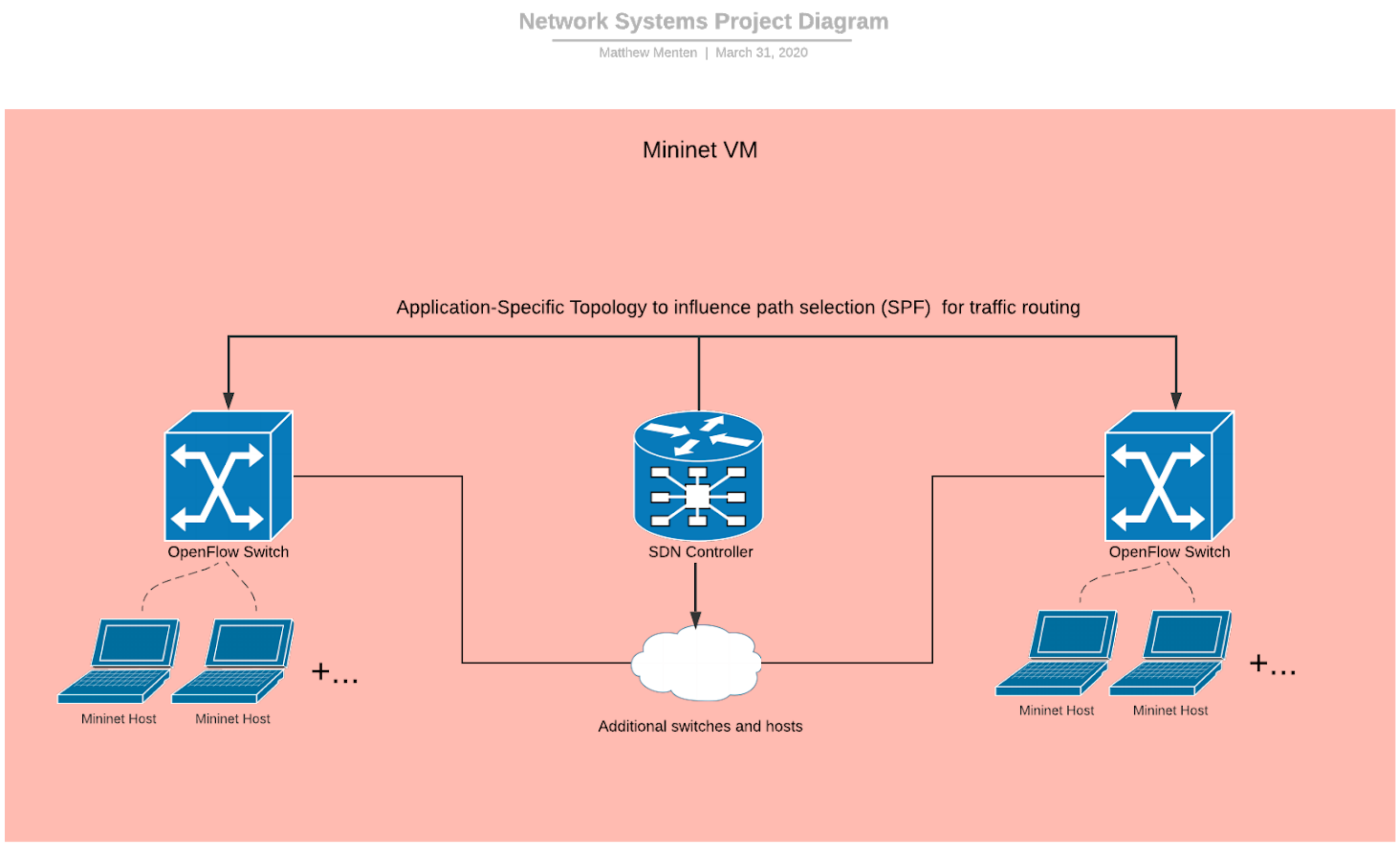
Depending on timing and how difficult P4 is to pick up, we plan to explore issues related to preemptive congestion control:

Similar work already exists, which we may expand on.

https://github.com/CS344-Stanford-18/p4-mininet-tutorials/tree/master/P4D2_2018_East/exercises/ecn

We think it would be interesting to design a better bandwidth use and congestion control system for networks with certain known parameters. For example, could routers or switches inform connected computers, or applications running on them, about current bandwidth usage, speed caps, etc. to preemptively warn systems before packets are dropped? This could apply to home routers as well as datacenter infrastructure. Specifically, for home networks, a user's home router could inform clients of speed and bandwidth caps to reduce the reliance on TCP window size adjustments.

Solution Architecture


Network Systems Project Diagram
Matthew Menten | March 31, 2020
Mininet VM
Application-Specific Topology to influence path selection (SPF) for traffic routing
OpenFlow Switch
SDN Controller
OpenFlow Switch
Mininet Host
Mininet Host
+...
Additional switches and hosts
Mininet Host
Mininet Host
+...

Tools

Virtual Box: Open Source hypervisor that supports creating and running virtual machines.

- Specifically, we will use LUbuntu (64-bit) running on both a Windows and Linux system.
- The Mininet environment is set up in virtual box using either Vagrant or a disk image file. We will use this as an introduction to P4, as it provides code examples and contains many pre-installed tools to compile and run them.

BMv2 (behavioral model): A P4 software switch written in C++11 that takes a JSON file created from a P4 compiler and implements packet processing behavior.
P4c: The reference compiler for the P4 programming language.
Mininet: A lightweight network emulation environment that allows you to launch a virtual network with switches, hosts, and an SDN controller.
Wireshark: Network packet analyzer to troubleshoot communication protocol development.
Iperf: Traffic generation and network performance measurements.
D-ITG: Wider range of traffic generation to simulate realistic network workloads. Also measures performance within the network.

Datasets

For this project there are no concrete datasets that we plan to test with. That being said, we plan to test the work within our Mininet environment in a few different ways. We plan to use tools such as iperf and D-ITG (Distributed Internet Traffic Generator) to simulate different types of traffic. For example, we will test the routing of UDP video streams, TCP HTTP traffic, VoIP, FTP, DNS, etc. using these tools within Mininet. These tools also allow us to measure conditions such as delay and throughput between each host. Along with basic tools such as ping and netcat, and the built-in Mininet functionality, we will assess the routing differences between the augmented network topologies that our SDN controller provides to each switch.

Challenges

Challenges for this project stem from the fact that we are essentially overlaying new ideas on top of already implemented networking protocols. Because of this, we don't necessarily know if our ideas are actually an improvement or if it will work.

Another challenge we face is that P4 is a brand new programming language that we have to learn. Furthermore, we are new to the networking emulation environment and how all the pieces fit together. Therefore, not only are we trying to solve a networking problem, but we have to understand foundationally how everything works together before we can even begin to improve on what's already out there.

Timeline

Checkpoint 1 - 3/29

Kristina:

https://github.com/CS344-Stanford-18/p4-mininet-tutorials/tree/master/P4D2_2018_East/exercises

- From P4 mininet tutorial complete:
    - Basic Forwarding
    - Basic Tunneling
    - P4 Runtime
    - Explicit Congestion Notification
- Create p4 file skeleton for basic routing in P4

Matt:

- Expand problem definition and proposed solutions
- Architecture diagram
- Setup Mininet environment based on Kristina's work
- Explore tools for simulating network traffic and testing our future work.

Checkpoint 2 - 4/16

Kristina:

- Write P4 parser and develop match/action tables that route traffic based on application type

Matt:

- Explore modifying the network topology stored in the SDN controller to influence path selection made by SPF. I.e. Can we modify the controller to maintain different topologies, and inform switches to use certain ones for certain applications. In this way, we avoid modifying path selection algorithms.
- Begin testing traffic using iperf and D-ITG

Final Project - 4/30

Both:

- Write final report
- Analysis of P4 protocols

Concerns

Of course, with COVID-19 everything is pretty up in the air right now. One big concern from this is that it's going to be difficult to get hands-on help. Also, if either partner gets sick during this project, that will dramatically change our timeline. With only a month left in the semester, any timeline changes could impact the scope of our project significantly.

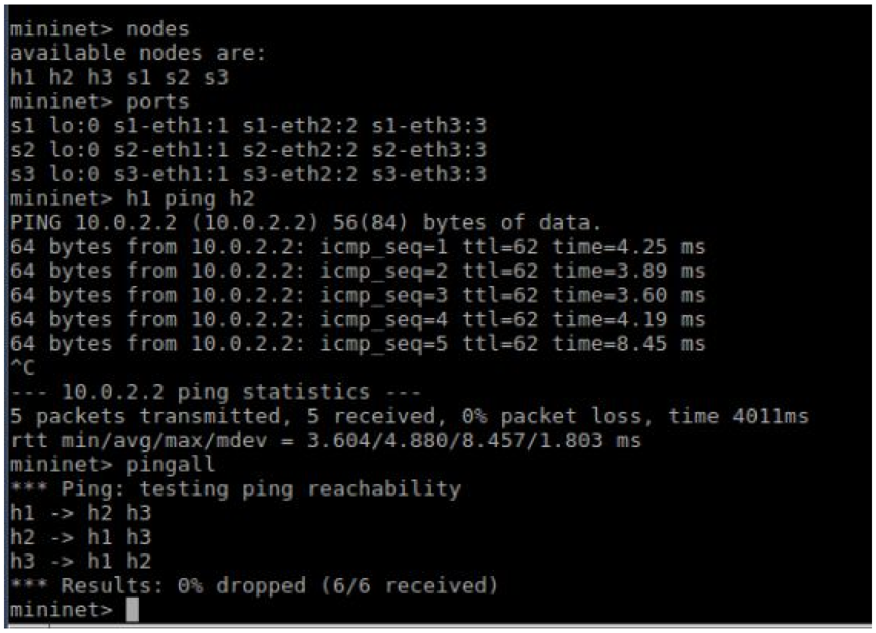
Checkpoint 1

Changes have not been highlighted as our project proposal was turned in prior to the release of the guidelines. Thus, additional sections have been added as necessary and there have been no changes to the original project proposal. The timeline has been created since the project proposal, and therefore see above for the "new" timeline.

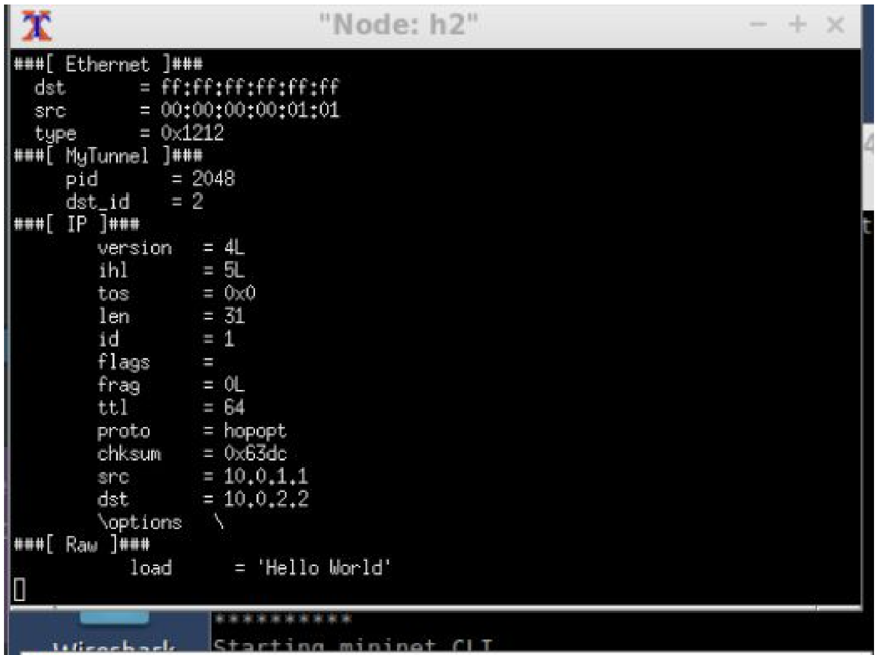
Work Completed

Following the P4 Tutorial, we created a basic forwarding implementation with three switches, each with a host. Below we use *pingall* to demonstrate communication between all hosts.

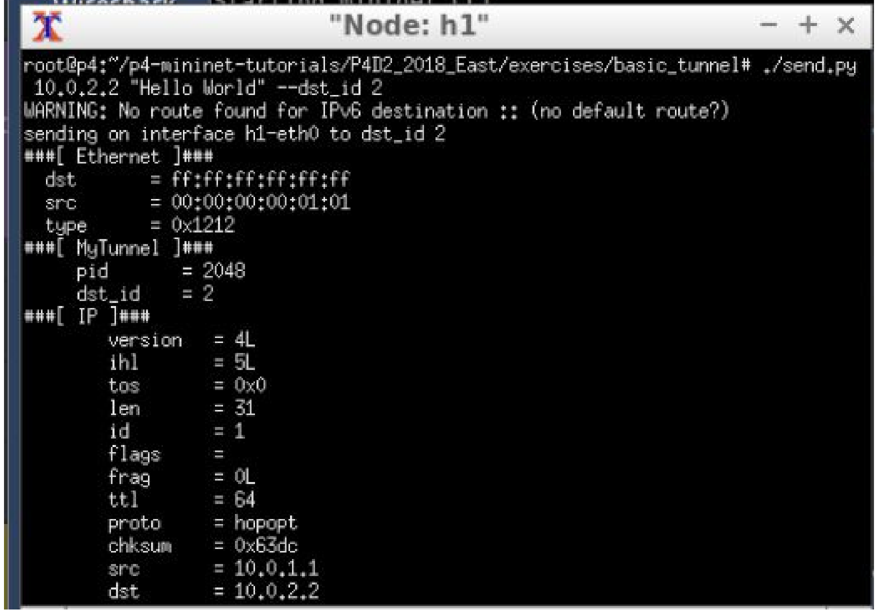
Now, rather than use the IP address to forward the packet, we can forward based on a custom header. We forward based on the custom header dst_id of 2, and using a match and action table that matches on this dst_id we send "Hello World".



```
mininet> nodes
available nodes are:
h1 h2 h3 s1 s2 s3
mininet> ports
s1 lo:0 s1-eth1:1 s1-eth2:2 s1-eth3:3
s2 lo:0 s2-eth1:1 s2-eth2:2 s2-eth3:3
s3 lo:0 s3-eth1:1 s3-eth2:2 s3-eth3:3
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=62 time=4.25 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=62 time=3.89 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=62 time=3.60 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=62 time=4.19 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=62 time=8.45 ms
^C
--- 10.0.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 3.604/4.880/8.457/1.803 ms
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> 
```



```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:01:01
  type      = 0x1212
###[ MyTunnel ]###
     pid       = 2048
     dst_id    = 2
###[ IP ]###
        version   = 4L
        ihl       = 5L
        tos       = 0x0
        len       = 31
        id        = 1
        flags     = 
        frag      = 0L
        ttl       = 64
        proto     = hopopt
        chksum    = 0x63dc
        src       = 10.0.1.1
        dst       = 10.0.2.2
        \options   \
###[ Raw ]###
           load      = 'Hello World'
```



```
root@p4:~/p4-mininet-tutorials/P4D2_2018_East/exercises/basic_tunnel# ./send.py
 10.0.2.2 "Hello World" --dst_id 2
WARNING: No route found for IPv6 destination :: (no default route?)
sending on interface h1-eth0 to dst_id 2
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:01:01
  type      = 0x1212
###[ MyTunnel ]###
     pid       = 2048
     dst_id    = 2
###[ IP ]###
        version   = 4L
        ihl       = 5L
        tos       = 0x0
        len       = 31
        id        = 1
        flags     = 
        frag      = 0L
        ttl       = 64
        proto     = hopopt
        chksum    = 0x63dc
        src       = 10.0.1.1
        dst       = 10.0.2.2
```

Validation

Our project proposal adds additional features onto a traditional routing protocol, and thus performance can't really be compared to something that has already been implemented. Instead, we plan to validate whether our system works using tools provided by Mininet and Wireshark.

For one, we can test the transmission time and overall throughput for different types of traffic within our network using a basic routing setup and the true network topology. This will give us a solid baseline as to how the network performs normally. Then we can use our augmented topologies and see how these parameters change for different hosts and applications.

If we finish this and move on to congestion control, we will test our new protocols using Wireshark. Using Mininet we will send a different amount of packets in batches, capture these packets in Wireshark, and confirm the packet headers have been modified to indicate bandwidth usage, speed caps. etc.

Below, we list some of the commands/tools we will use to validate our system.

Mininet

- Pingall: verify hosts and switches are properly communicating
- Send variety of packets
    - iperf: Send TCP packets between hosts and determine bandwidth
    - iperfudp: Send UDP packets
    - wget/curl: Send HTTP packets
    - We may need to use user-defined packets, which can be done using python scripts (see send.py and receive.py in the tutorial)
    - D-ITG: Send video streams, VoIP, FTP traffic between hosts
- *sudo mn --link tc,bw=10, delay=10ms*
    - Assign packet latency and bandwidth. This is in response to Professor Santos's concern that the intelligence might execute slower than TCP, and thus introduces an artificial delay.

Wireshark

- Visually inspect packets and confirm they contain the expected contents using capture

Checkpoint 2

In terms of github, we didn't have access to the github classroom for checkpoint 1 so we worked on our code locally. Having read the guidelines for checkpoint 2, we can see you'd like to see commits and have since committed our code. Because of this, however, we don't have code commits dating prior to the creation of our repository.

Project Changes

We have decided not to work on the following idea which was outlined in our initial proposal:

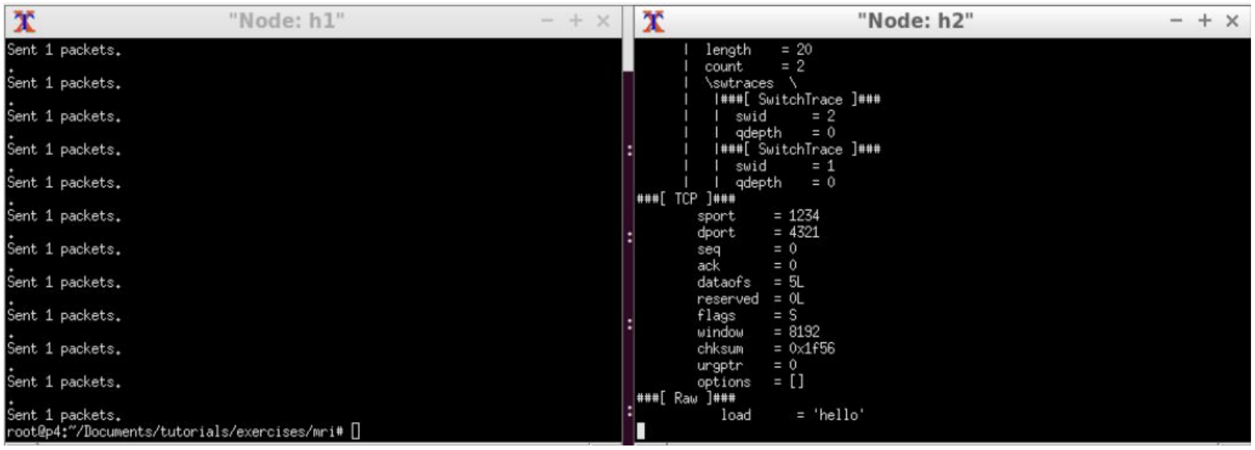
> *We think it would be interesting to design a better bandwidth use and congestion control system for networks with certain known parameters. For example, could routers or switches inform connected computers, or applications running on them, about current bandwidth usage, speed caps, etc. to preemptively warn systems before packets are dropped?*

Validation

In addition to the validation described above, we are writing a client and server python script to send and receive packets. The send script will take a destination address and message as arguments whereas the receive script will simply wait to receive a packet. Then, within our p4 program we will create custom headers that can trace the path of the packet and print out the queue depth of packets, as well as the switches it went through. Thus, we can send a variety of different packets and verify the tiers of traffic are sent correctly.

Work Completed

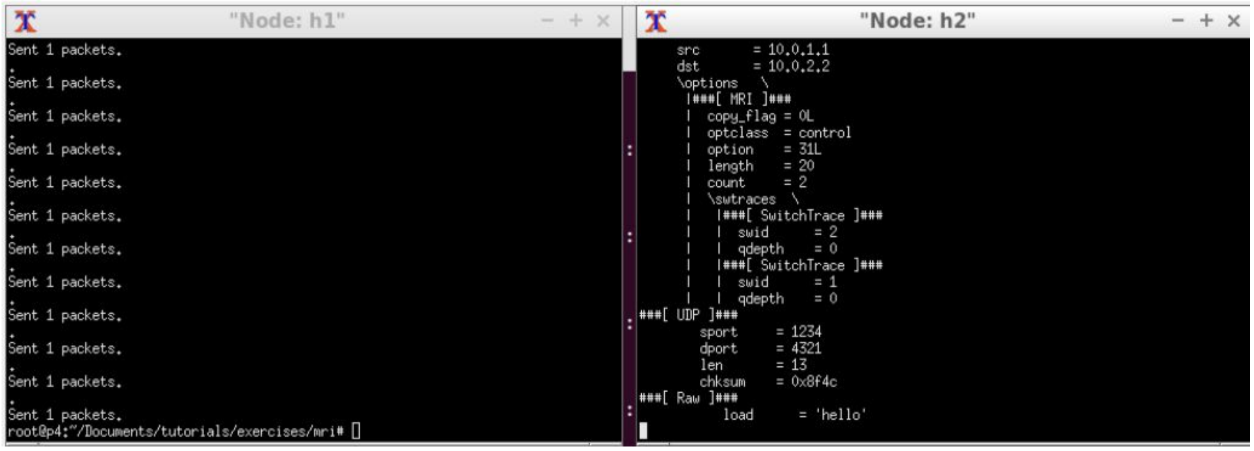
Purpose: This shows that the packets are successfully being sent and received using the P4 framework to forward packets. It also demonstrates multiple kinds of packets make it through. This specific implementation is only set up with 3 switches and 4 hosts, so looking at switch trace we can see the packet went through switch 1 and 2 as expected. This network will be increased for final analysis.
Here's send.py and receive.py with TCP packets:



```
"Node: h1"
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
root@p4:~/Documents/tutorials/exercises/mri# []
```

```
"Node: h2"
     |  length    = 20
     |  count     = 2
     |  \swtraces  \
     |   |###[ SwitchTrace ]###
     |   |  swid      = 2
     |   |  qdepth    = 0
     |   |###[ SwitchTrace ]###
     |   |  swid      = 1
     |   |  qdepth    = 0
###[ TCP ]###
        sport     = 1234
        dport     = 4321
        seq       = 0
        ack       = 0
        dataofs   = 5L
        reserved  = 0L
        flags     = S
        window    = 8192
        chksum    = 0x1f56
        urgptr    = 0
        options   = []
###[ Raw ]###
           load      = 'hello'
```

Here’s send.py and receive.py with UDP packets:



```
"Node: h1"
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
root@p4:~/Documents/tutorials/exercises/mri# []
```

```
"Node: h2"
     src       = 10.0.1.1
     dst       = 10.0.2.2
     \options   \
      |###[ MRI ]###
      |  copy_flag = 0L
      |  optclass  = control
      |  option    = 31L
      |  length    = 20
      |  count     = 2
      |  \swtraces  \
      |   |###[ SwitchTrace ]###
      |   |  swid      = 2
      |   |  qdepth    = 0
      |   |###[ SwitchTrace ]###
      |   |  swid      = 1
      |   |  qdepth    = 0
###[ UDP ]###
        sport     = 1234
        dport     = 4321
        len       = 13
        chksum    = 0x8f4c
###[ Raw ]###
           load      = 'hello'
```

Aside from sending/receiving different types of packets to/from hosts (via the switches in our topology), we have done a deeper dive into the exact processes we will use to modify the path selection based on L4 information in each packet. We also propose a method to support multiple host “tiers” as mentioned previously in our proposal. Below, #1 and #2 reagard application-specific optimizations (on switches) and preferred path selection/bandwidth allocation (at the topology level) respectively. #3 details our process for determining routes for different tiers of hosts at the control plane level.

1. Reserving certain ports/connections on each switch for certain application types. Within the network topology we define, we increase/decrease the bandwidth on these reserved switch ports based on the application type. Our P4 program can match on the application header in a packet and choose the outbound port accordingly. Each application type has its own set of reserved ports so that effective routing is maintained. For instance, video traffic will be sent along high-bandwidth links.
2. Application-preferred routing using dedicated switches for certain application types. This involves carefully creating a network topology that defines initial gateways for hosts and roles for different switches beyond the gateway. There is a dedicated switch for video traffic (RTSP), HTTP traffic, FTP traffic, and generic switches which handle all other traffic. Hosts communicate with a gateway switch, which then matches on the application header and forwards it to the correct application-specific switch. To give preference to certain application types, the application-specific switches will be more/less densely connected to other switches in the network, contain more/less direct paths to hosts, and have more/less bandwidth on each link. Here we use ECMP load balancing to distribute traffic along the paths. Densely connected switches will therefore achieve higher bandwidth and potentially shorter routes for their respective application traffic.
3. Formula for determining host routes based on “tiers” using the control plane. The SDN controller in our system architecture has a complete view of the network topology (hosts, how they are connected and what switch ports they are connected to), as well as a shortest path algorithm for determining routes between hosts and switches. In the AppController file we include additional topologies besides the “true” topology. There is a unique topology for each tier of hosts. The highest tier host gets route information from the controller based on the “true” topology. Lower tier hosts are served routes based on incomplete topologies. These incomplete topologies are a subset of the true topology with certain switches/paths omitted. In this way we can effectively hide “express” paths through the network from certain hosts. While this may seem counter intuitive (you could just disconnect that host from the express paths), it allows for dynamic reconfiguration in certain circumstances. Hosts can easily be promoted without rewiring the underlying network.

The majority of the work done for this checkpoint involved experimenting with our mininet system and figuring out what is actually possible. We obtained P4 tutorial code from a Stanford repository and went through examples like source routing, route inspection and load balancing to understand how they work. This has given us a better understanding of P4 and what we can do with it. For example, we found out that it's quite difficult to have each switch maintain multiple topologies and route packets based on different ones. That has influenced our choice to move topology/route selection entirely to the controller, which in turn informs hosts and switches of available routes (for our host tiers approach). Our initial thinking was that we could have each switch calculate the next hop on-the-fly using multiple stored topologies for different hosts (similar to ECMP in a way), but we discovered that it isn't feasible with our system.

After our in-depth testing with mininet and P4, and formulating our specific plans for each of our goals (host tiers and application specific optimization), we have begun work on implementing #2 and #3 above. This has primarily involved determining what our different topologies will look like, and how to match against the necessary fields in the P4 programs running on our switches. We believe #2 and #3 are the most interesting of our in-depth approaches and we have been allocating work to them first. If there is left over time we will implement the design proposed in #1 and compare its results to #2.

Timeline

Checkpoint 2 - 4/16

Work by Team Member

Kristina:

- In-depth testing with P4 and Mininet to better understand what is possible. E.g. Implement switch trace routing based off: https://github.com/p4lang/tutorials/tree/master/exercises/mri
- Writing custom send/receive scripts which run on our hosts. This allows us to send UDP/TCP traffic to a specific destination. It can easily be extended to support sending specific L4 packets.
- Helping to come up with the design and proposed implementation for each of our goals (#1, #2, #3 above).
- Initial topology design and matching logic for #2
- Started final paper

Matt:

- Research into P4 solutions. E.g. How to do ECMP (equal cost multi-path) load balancing on each switch.
- Research on control plane operations and how to define topologies for our mininet system. We can use json files to define multiple topologies and modify the controller to accept more than one. The controller iterates over hosts to determine routes, which is where we add logic to support host tiers and their corresponding topologies.
- Helping to come up with the design and proposed implementation for each of our goals (#1, #2, #3 above).
- Initial topology designs and control plane logic for #3.
- Basic testing of Iperf functions on our mininet system.

Work Left for Completion

Kristina:

- Finish P4 parser (ingress, egress, etc.) for #2.
- Augment send.py and receive.py for different application layer protocols.
- Test #2 using send/receive scripts along with Iperf and D-ITG.
- Collect test results and generate graphs for #2.

Matt:

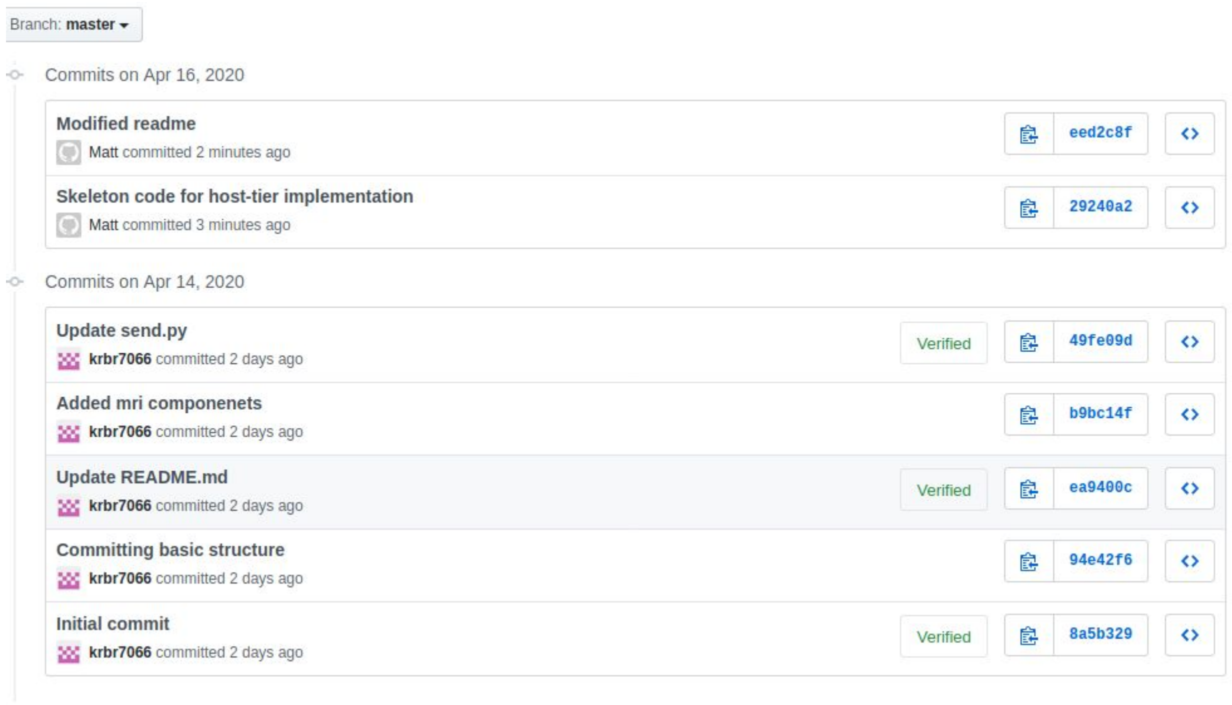
- Finish control plane logic for assigning host tiers using the created topologies (#3).
- Create additional host-tier-specific topologies for further testing and comparison on #3.
- Test #3 using Iperf and D-ITG.
- Collect test results and generate graphs for #3.

Both:

- Help with implementation on #1 if time allows.
- Finish final paper and presentation.

Github Check-Ins

We’re not exactly sure if check-ins are the same as commits, but here is our commit history. While it may not seem like a ton, we would like to highlight that we did not have a repository when we finished checkpoint 1 and a lot of our initial work for checkpoint 2 was doing local testing and in-depth research, as we discussed in the work completed section. We appreciate your understanding!


Branch: **master**

Commits on Apr 16, 2020

Modified readme Matt committed 2 minutes ago		eed2c8f
Skeleton code for host-tier implementation Matt committed 3 minutes ago		29240a2

Commits on Apr 14, 2020

Update send.py krbr7066 committed 2 days ago	Verified	49fe09d
Added mri componenets krbr7066 committed 2 days ago		b9bc14f
Update README.md krbr7066 committed 2 days ago	Verified	ea9400c
Committing basic structure krbr7066 committed 2 days ago		94e42f6
Initial commit krbr7066 committed 2 days ago	Verified	8a5b329

- Commits on 04/14: basic mininet structure, readme describing how to install the necessary software to run the code, send/receive.py scripts, initial topologies for our network
- Commits on 04/16: Structure of host-tier route selection logic on the AppController, more readme modifications

Challenges

Aside from adjustments corresponding with the chaotic and unfamiliar state of the country and world, our project has faced no large barriers so far. One minor challenge we have faced is that the syntax of P4 is pretty unfamiliar. It's hard to tell how data is passed from function to function and the general control flow was confusing at first. There is very little code, but each line is quite important and powerful. It was definitely a steep learning curve. Additionally, finding code examples and solutions for problems is more challenging than other languages. Other than that, we've revised some of our initial approaches as we have begun implementing solutions. While this hasn't set us back that much, it's always frustrating to have to go back to the drawing board and come up with a different solution when you find out your initial design won't work.

Sources:

http://mininet.org/walkthrough/

https://github.com/CS344-Stanford-18/p4-mininet-tutorials/blob/master/P4D2_2018_East/P4_tutorial_labs.pdf

https://mailman.stanford.edu/pipermail/mininet-discuss/2013-August/002807.html

http://traffic.comics.unina.it/software/ITG/download.php

https://sdnopenflow.blogspot.com/2015/05/using-of-d-itg-traffic-generator-in.html