EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Network Softwarization
SS 2022
Prof. Dr. M. Menth
M. Häberle, S. Lindner, D. Merling, B. Steinert

Exercise 2

May, 30 2022

Deadline: July, 3 2022, 23:59

Problem 2.1: *Introduction to Network Softwarization*

Each participant will be asked **5 questions** from the following pool of questions **on June 11, 2018 in the exercise lesson**. Giving answers to all questions is a prerequisite for getting a mark for the assignment!

Chapter 4 - P4 Basics

1. What is P4?
2. Briefly explain the P4 programming model!
3. What is a P4 architecture?
4. Name the components of the Protocol-Independent Switch Architecture (PISA)!
5. Name benefits of P4!
6. What is an extern?
7. What is the P4 Runtime?

Chapter 5 - P4 Targets + Controller

1. What is a P4 target?
2. Explain the two-layer compiler model in P4!
3. What is the difference between a FGPA-based P4 target and an ASIC-based P4 target?
4. Which architecture is implemented in the Intel Tofino?
5. What is a drawback of ASIC-based P4 targets?

Assignment 2

1. How does a P4 controller communicate with a P4 switch?
2. How are registers defined in P4? How can they be accessed?

3. Explain a usage scenario for 6in4 tunneling!
4. How can additional headers be added to a packet with P4?
5. Name two scenarios where link aggregation is usefull!
6. What are advantages/disadvantages of per packet vs. per flow aggregation?

Problem 2.2: *Link Aggregation*

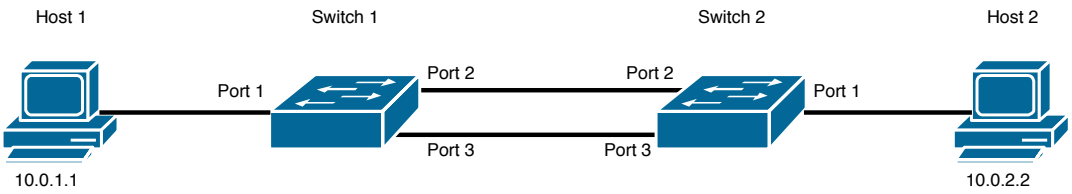

Figure 1: Network topology using link aggregation

In the following, we implement a simple link aggregation solution in P4. The network topology is shown in Figure 1. It consists of two hosts, each connected to a switch and two separate links between the switches.
Use the code from the tar archive `exercise-base.tgz` available in the Moodle course as a basis for four solutions.

25 Points

1. Download the archive `exercise-base.tgz` from the Moodle course and extract it in the folder `/home/netsoft/netsoft/p4/exercises` in the NetSoft VM. Adjust the file `topology.json` according to Figure 1. The IP addresses are configured automatically.

   Start the virtual testbed by running the `make` command.
   Start the controller by running the command
   `./controller.py -p4info ../build/basic.p4info -bmv2-json ../build/basic.json`.

   Start terminals for Hosts 1 and 2 by running the command `xterm h1 h2` in the interface of the virtual testbed.
   Make sure that both hosts can reach each other, e.g. by using `ping`.

2. Implement packet based link aggregation. Packets that are exchanged between switch 1 and switch 2 should be sent via the switch ports 2 and 3 alternatingly.

   Hint:
   Define a ”link aggregation port” that is used in the forwarding table and replace it after longest prefix matching with the actual port number that needs to be used.

   Hint:
   You can keep states in the P4 program by using registers.

3. Extend your P4 program by flow based link aggregation for UDP and TCP traffic. Packets of one flow should always be transmitted on the same link. You can identify flows by calculating a hash of the five tuple (source and destination IP address, source and destination port, protocol number).

Hint:
Flow based link aggregation can be implemented stateless.

Hint:
You can test your implementation by sending messages with netcat and looking at the traffic on the links with wireshark.

4. Monitor the number of packets and the number of bytes that are sent and received on each port of the switches by using counters. Print the content of the counters every two seconds in the controller output.

Upload a single implementation that contains packet based link aggregation, flow based link aggregation for UDP and TCP and the monitoring to Moodle.

Problem 2.3: *IPv6 and 6in4 Tunnels*

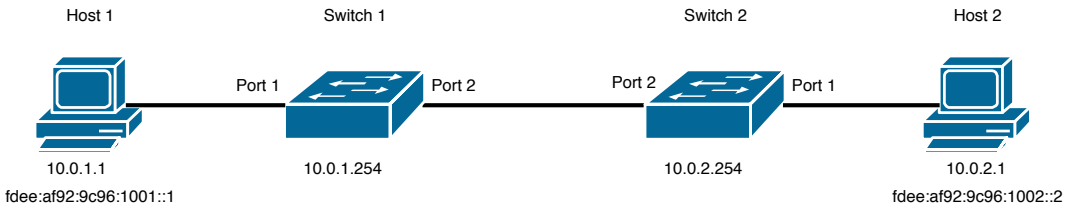

Figure 2: Network topology for exercise 2

In this exercise, we take a look at IPv6 and 6in4 tunnels. Use the code from the tar archive `exercise-base.tgz` available in the Moodle course as a basis for your solutions. 25 Points

1. Set up the topology shown in Figure 2.
   Add longest prefix matching for IPv6 to the P4 code and add forwarding rules to the controller.
   Configure the IPv6 addresses and routes using the commands shown in Figure 3 for host 1 and Figure 4 for host 2.
   Check that both hosts can reach each other using IPv6.

   Hint:
   The P4 Runtime Library is not capable of converting strings containing IPv6 addresses. You will have to convert the hexadecimal representation of the IPv6 addresses to the bytes datatype, e.g.
   `bytes(bytearray.fromhex("fdeeaf929c9610010000000000000001"))`

2. Many hosts on the Internet only have IPv4 connectivity. IPv6 connectivity can be established by using a 6in4 tunnel for tunneling IPv6 traffic via IPv4 to a gateway that has both IPv4 and IPv6 connectivity. 6in4 adds an IPv4 header with protocol number 41 between the client and the gateway.

   Extend the given P4 program by 6in4 gateway functionality.
   When the switches receive an IPv4 packet from one of their 6in4 clients with protocol number 41 in the protocol field of the IPv4 header, they need to remove the IPv4 header and do IPv6 routing.
   When the switches receive an IPv6 packet going to one of their 6in4 clients, they need to add an IPv4 header with the switche's IP as source IP, the clients IP as destination IP and 41 as protocol number.

   Test your solution. 6in4 tunneling can be configured on host 1 using the commands in Figure 5. Do not run the commands in Figure 3 on host 1 this time!.

   Hint:
   You can implement 6in4 tunnels without adding additional match-action-tables.

   Hint:
   Entries in a Match-Action-Table may have different attributes depending on the action.

Upload your final implementation to Moodle.

```
sysctl -w net.ipv6.conf.all.disable_ipv6=0
sysctl -w net.ipv6.conf.default.disable_ipv6=0
ip -6 a a fdee:af92:9c96:1001:0000:0000:0000:0001/64 dev h1-eth0
ip -6 n a fdee:af92:9c96:1001:0000:0000:0000:1001 lladdr 00:00:00:01:01:00 dev h1-eth0
ip -6 r a fdee:af92:9c96:1002:0000:0000:0000:0001/64 via fdee:af92:9c96:1001:0000:0000:0000:1001 dev h1-eth0
```

Figure 3: Commands needed to configure IPv6 on host 1

```
sysctl -w net.ipv6.conf.all.disable_ipv6=0
sysctl -w net.ipv6.conf.default.disable_ipv6=0
ip -6 a a fdee:af92:9c96:1002:0000:0000:0000:0002/64 dev h2-eth0
ip -6 n a fdee:af92:9c96:1002:0000:0000:0000:1001 lladdr 00:00:00:02:02:00 dev h2-eth0
ip -6 r a fdee:af92:9c96:1001:0000:0000:0000:0001/64 via fdee:af92:9c96:1002:0000:0000:0000:1001 dev h2-eth0
```

Figure 4: Commands needed to configure IPv6 on host 2

```
sysctl -w net.ipv6.conf.all.disable_ipv6=0
sysctl -w net.ipv6.conf.default.disable_ipv6=0
ip tunnel add 6in4 mode sit remote 10.0.1.254 local 10.0.1.1 ttl 64
ip link set 6in4 up mtu 1400
ip addr add fdee:af92:9c96:1001:0000:0000:0000:0001/64 dev 6in4
ip route add ::/0 dev 6in4
```

Figure 5: Commands needed to configure 6in4 tunneling on host 1

Problem 2.4: *1+1 Protection*

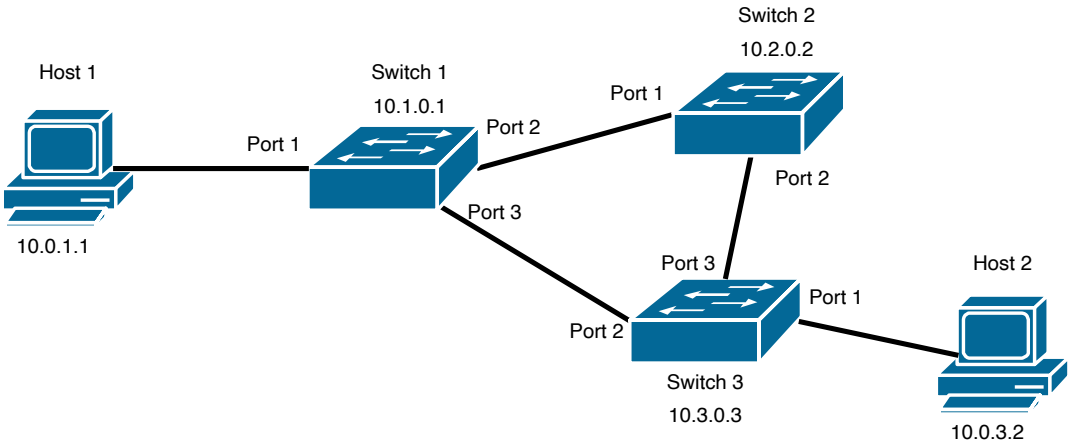

Figure 6: Network topology for exercise 3.

In this exercise, we take a look at 1+1 protection. With 1+1 protection, a sender duplicates traffic and forwards it over disjoint paths while the receiver forwards only the first copy received for every packet. In case of a failure, any packet loss can be avoided, which makes 1+1 protection attractive for highly reliable applications.

In Figure 6, Host 1 and Host 2 are connected over two paths, i.e, `H1-S1-S2-S3-H2` and `H1-S1-S3-H2`. 1+1 protection can be deployed between Switch 1 and Switch 3. That means, Switch 1 duplicates packets destined for Host 2 and sends them using the paths `S1-S2-S3` and `S1-S3` to Switch 3. This is done using tunnels, i.e., Switch 1 adds a tunnel header to the packets which addresses Switch 3. The tunnel depends on the used technology, e.g., IP or MPLS. All nodes along the path forward the packets using the outer, i.e., tunnel, header. The protection endpoint must decide which packets are further forwarded, and which packets are duplicates and therefore dropped. This is usually done with the help of sequence numbers.

You will implement a simplified form of 1+1 protection in this task. Use the code from the tar archive `protection-base.tgz` available in the Moodle course as a basis for your solutions. 25 Points

1. For 1+1 protection, you need a function called *clone*. The `simple_switch_grpc` requires the thrift-module in order to configure this function.

   Go to `/home/netsoft/p4/vm/behavioral-model/targets/simple_switch_grpc` and re-compile the target with the following commands:

   ```
   ./autogen.sh
   ./configure -with-thrift
   make -j 2
   ```

```
sudo make install
sudo ldconfig
```

2. Set up the topology shown in 6. In order to clone a packet in P4, a mirror session has to be configured for the `simple_switch_grpc`. This is already done using `s1_cli.txt` and `s3_cli.txt`.

   Start the virtual testbed and the controller as shown in Exercise 1 and verify that Host 1 and 2 can ping each other.

3. Implement 1+1 protection between Switch 1 and Switch 3. Every packet destined for Host 1 and Host 2 should be forwarded between Switch 1 and Switch 3 using the available disjoint paths, e.g., Switch 1 duplicates packets for Host 2 and sends the packets to Switch 3 and Switch 2.

   - Use IPv4-tunnels for the disjoint paths to address the de-duplication node, i.e., an additional IPv4 header is added to the packet, and the de-duplication node is addressed in the outer IPv4 header.
   - Both directions, Host 1 to Host 2 and Host 2 to Host 1 should be protected.
   - Use your own protection header to store the sequence number of the packets. Your protection header might contain further fields, e.g., a next protocol field to enable flexible parsing.
   - Packets are accepted and forwarded, if the sequence number in your protection header equals the next expected sequence number.

     Example: Switch 3 expects a packet with sequence number 5, receives a packet with sequence number 5, forwards it and adjusts its next expected sequence number to 6. Now, the duplicate packet with sequence number 5 arrives. As Switch 3 expects a packet with sequence number 6, the packet is dropped.

   Check that Host 1 and Host 2 don't receive duplicates!

   Hint:

   a) You have to remove your tunnel header at the de-duplication node.
   b) You can store the next used sequence number and the next expected sequence number in registers.

4. Verify that your protection works by disabling the link between Switch 1 and Switch 3 while you ping between Host 1 and Host 2!

   Hint:
   You can disable a link in the mininet cli with `link s1 s3 down`.

5. What are the disadvantages of this "only accept the next expected sequence number" procedure? How can you improve the acceptance test?

Upload your final implementation to Moodle.

Total: 75 Points