Redes Definidas por Software
Data Plane Programming

Version: 1

Departamento de Informática
Universidade do Minho
Feb 2025

João Fernandes Pereira

Assignment Overview

In this assignment, you will explore the practical aspects of data plane programming by implementing a custom tunneling mechanism using the P4 programming language. The focus will be on designing and implementing the **My Sequence Label Protocol** (**MSLP**), a simplified protocol for tunneling packets across a network with multiple tunnels.

The task involves creating a custom packet header format and programming the dataplane to handle, encapsulate, decapsulate, and route packets according to the MSLP protocol. This will require you to use the features of P4 to manipulate packet headers, manage metadata, and apply statically defined routing decisions.

In addition to implementing MSLP, you will also develop a **stateful firewall** capable of managing UDP traffic by selectively opening UDP ports based on defined rules. This firewall will demonstrate your ability to implement stateful processing and enforce basic security policies within the data plane.

As a final step, you will need to create a controller for your network. This controller must be capable of injecting the flows in the data plane tables and balancing the usage of the tunnels.

This assignment provides an opportunity to work on critical aspects of programmable networking: creating custom protocols, implementing stateful data plane functionalities, and adjusting your network according to traffic demand. By working on these concepts, we aim to reinforce your understanding of advanced P4 programming techniques.

This assignment can be completed in groups of **up to three students**.

Due Date: 25 May 2025, 23h59

Deliverables

You are required to submit **ONE** archive file with the following name format **RDS-24_25-GroupId.zip** (example RDS-24_25-G03.zip.). Your archive file should contain the following:

- Use the directory structure of the tasks presented in class.
- All mininet scripts needed to create the topology.
- All the developed P4 code.
- All the defined rules for each of the P4 devices.
- All the developed controller code.
- A small report, in pdf, with the following sections:
  - Introduction: Must cover, Data Plane, Control Plane, P4 and Label Switching Protocols: Explain the concepts, what they are and where they are used.
  - Implementation: Contains three subsections - My Sequence Label Protocol (MSLP) Stateful Firewall, and Controller
    - My Sequence Label Protocol (MSLP): Explain your header format, fields, and field size. Present and describe: how tunnel selection is made, the P4 actions and tables that support the implementation of MSLP.

* Stateful Firewall: use a flowchart to show the logic of your firewall. Present and describe the P4 code that allow the stateful firewall function.
* Controller: Explain your tunnel load balancing mechanism.

– Tests & Results

– Conclusion

Topology

For this assignment, you will need to implement the topology of Figure 1 in mininet. This topology presents two tunnels (#1 & #2),

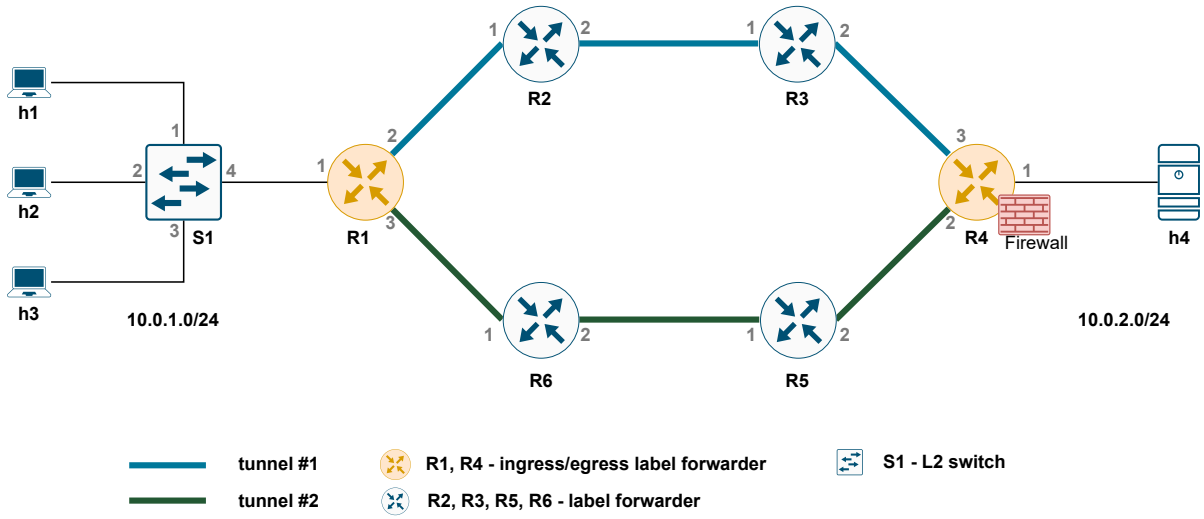

Figure 1: Network Topology.

Device	Interface/Port	MAC Address (Mininet Defined)	Label (Controller Defined)
h1	h1-eth0	aa:00:00:00:00:01	N/A
h2	h2-eth0	aa:00:00:00:00:02	N/A
h3	h3-eth0	aa:00:00:00:00:03	N/A
s1	s1-eth1 (to h1)	cc:00:00:00:01:01	N/A
s1	s1-eth2 (to h2)	cc:00:00:00:01:02	N/A
s1	s1-eth3 (to r1)	cc:00:00:00:01:03	N/A
r1	r1-eth1 (to s1)	aa:00:00:00:01:01	0x1010
r1	r1-eth2 (to r2)	aa:00:00:00:01:02	0x1020
r1	r1-eth3 (to r6)	aa:00:00:00:01:03	0x1030
r2	r2-eth1 (to r1)	aa:00:00:00:02:01	0x2010
r2	r2-eth2 (to r3)	aa:00:00:00:02:02	0x2020
r3	r3-eth1 (to r2)	aa:00:00:00:03:01	0x3010
r3	r3-eth2 (to r4)	aa:00:00:00:03:02	0x3020
r4	r4-eth1 (to h3)	aa:00:00:00:04:01	0x4010
r4	r4-eth2 (to r5)	aa:00:00:00:04:02	0x4020
r4	r4-eth3 (to r3)	aa:00:00:00:04:03	0x4030
r6	r6-eth1 (to r1)	aa:00:00:00:06:01	0x6010
r6	r6-eth2 (to r5)	aa:00:00:00:06:02	0x6020
r5	r5-eth1 (to r6)	aa:00:00:00:05:01	0x5010
r5	r5-eth2 (to r4)	aa:00:00:00:05:02	0x5020

Table 1: Network Devices and Labels. You can define your own.

Exercises

In this section, we delve deeper into the tasks for implementing the **My Sequence Label Protocol** (**MSLP**), the **Stateful Firewal** and the **Controller**.

A critical aspect of the assignment involves understanding how packets are encapsulated as they traverse the network and how specific fields in packet headers are used to identify the protocol of the encapsulated payload.

To illustrate this, Figure 2, shows the process of encapsulating packets from Layer 2 (L2) to Layer 4 (L4). Each layer adds its respective header to the packet, enabling proper handling at intermediate devices and correct interpretation by the receiver. The Ethernet **Type field** within the Layer 2 header and the IPv4 **Protocol field** within the Layer 3 header are pivotal for identifying the protocol of the next layer. These fields act as identifiers, ensuring that devices in the datapath process the packet correctly based on its encapsulated protocol.

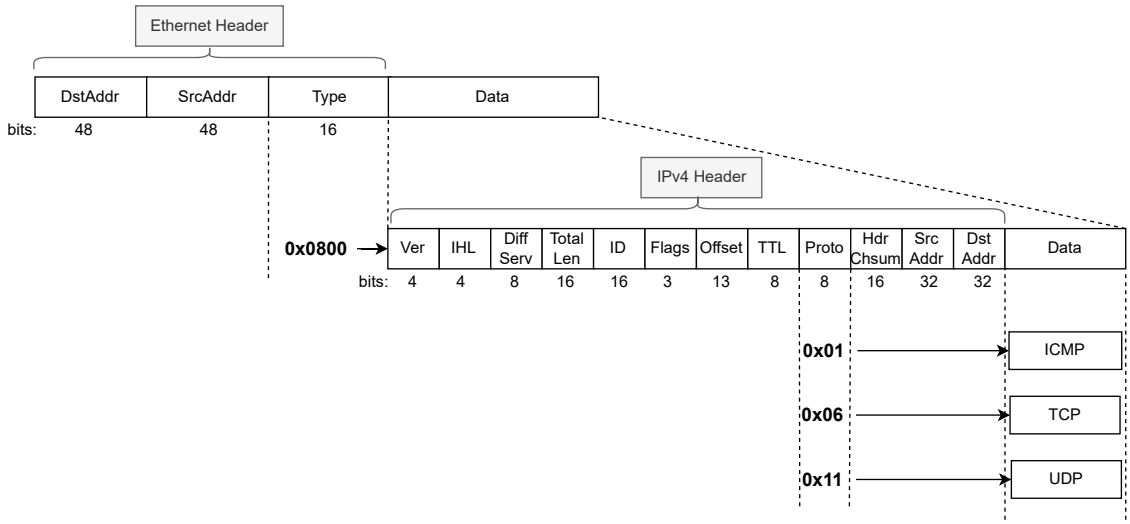

Figure 2: Network Stream.

1 - My Sequence Label Protocol - MSLP

The implementation of MSLP involves defining a new Ethernet type to encapsulate packets within the protocol and programming devices to handle the protocol's behavior, such as label pushing, forwarding, and popping. This subsection explains the necessary steps for implementing MSLP, as illustrated in the accompanying Figures 3 and 4.

A critical step in implementing MSLP is defining a new Ethernet type to distinguish MSLP-encapsulated packets from other packet types. Figure 3 illustrates the format of an MSLP-encapsulated packet:

- The EtherType field in the Ethernet header is assigned a custom value (e.g., 0x88B5) to identify the packet as an MSLP packet.
- The Payload (Data) of the MSLP packet contains the original payload of the Ethernet packet. In the example shown, Figure 3, the payload is an IPv4 packet, but the payload can be any protocol supported by Ethernet.

By defining a custom EtherType, devices in the network can recognize MSLP packets and process them accordingly.

MSLP uses a label-based mechanism to tunnel packets between endpoints. Figure 4 depicts the flow of a packet traveling through Tunnel #1, where traffic originates from the network 10.0.1.0/24 and is destined for the network 10.0.2.0/24. The encapsulation and forwarding process are as follows:

- Ingress Router (R1):
  Upon receiving the packet, R1 encapsulates it using the MSLP protocol. Labels are added to create a label stack, specifying the path the packet must follow through the tunnel. Each label corresponds to a hop in the path. The newly encapsulated packet is forwarded into the tunnel.
- Intermediate Devices (e.g., R2, R3): Each device processes the top label of the stack to determine the next hop. The device then pops the top label (removes it from the stack) and forwards the packet to the next device along the path.

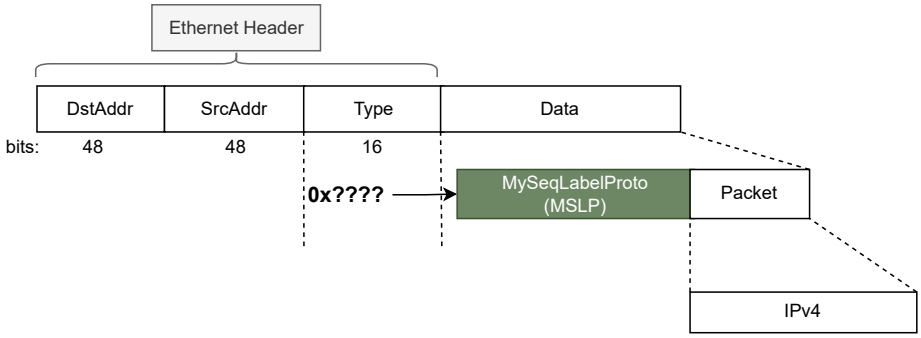

Figure 3: Tunnel Stream.

- Egress Router (R4):
  As the final device in the tunnel, R4 removes the MSLP encapsulation entirely. The original payload (e.g., the IPv4 packet) is restored and forwarded to its destination within the network 10.0.2.0/24.

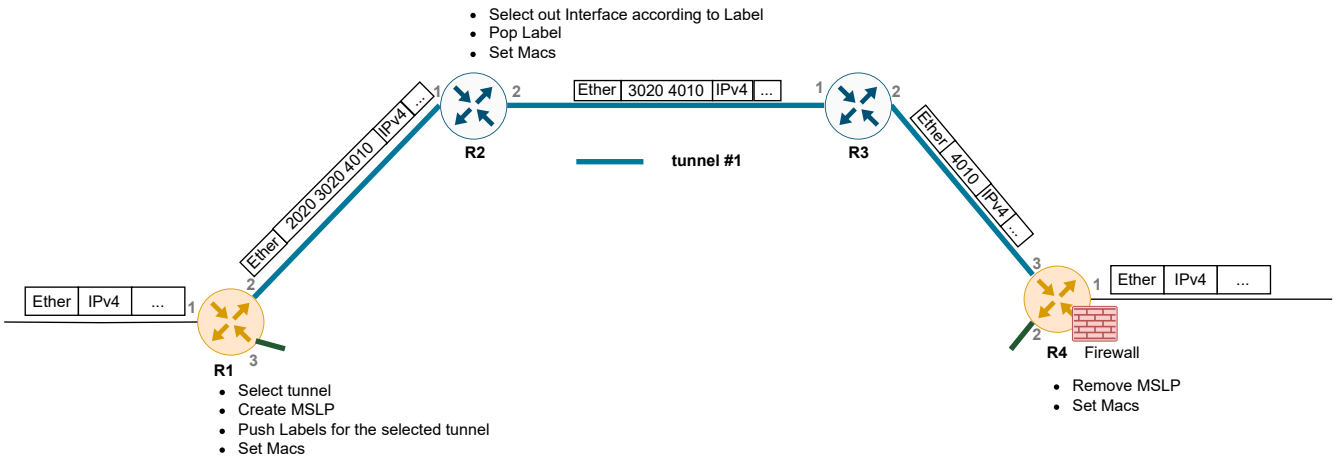

Figure 4: Tunnel #1 example.

The label stack mechanism ensures that packets traverse the tunnel efficiently while maintaining the flexibility to support multiple paths or tunnels in the network. This hierarchical design allows MSLP to encapsulate traffic for end-to-end delivery across the programmable data plane. You need to define the format of MSLP, how many fields and the size of those fields.

2 - Stateful Firewall

In this task, you will implement a stateful firewall to control UDP traffic using P4. The firewall will enforce rules to allow only specific UDP traffic, leveraging programmable dataplane features such as registers and hash functions. The behavior of the firewall is summarized in Figure 5, which provides an overview of the system.

Firewall Behavior

The stateful firewall is responsible for two main tasks:

1. **Allowing Replies for Established Connections:**
   - The firewall monitors outbound UDP traffic originating from the LAN `10.0.2.0/24`.
   - Using a **Bloom Filter**, the firewall records the flow that describes the outgoing packets. This allows the firewall to recognize and permit reply traffic from the corresponding destination.
2. **Allowing Traffic to Specific UDP Ports:**
   - Independent of the Bloom Filter, the firewall is configured to permit incoming traffic on certain predefined UDP ports (e.g., for DNS, gaming, or other specific applications).

- Traffic to these ports bypasses the Bloom Filter check.

Bloom Filter Implementation

To implement the Bloom Filter for tracking outgoing traffic, students must:

- **Use Registers:**
  - Registers are used to store hashed values of IP and UDP headers of the outgoing UDP packets.
  - These hashed values act as a compact representation of the traffic, enabling efficient lookups for reply packets.
- **Apply Hash Functions:**
  - Hash functions are applied to the tuple of `(IP header fields + UDP header Fields)` to produce one or more hash values.
  - These values are used as indices to update the bloom filter in the registers.
- **Check for Replies:**
  - For incoming traffic, the firewall computes the hash values for the defined tuple and checks the bloom filter for a match.
  - A match indicates that the incoming packet is a valid reply and should be allowed.

Overview of the Firewall

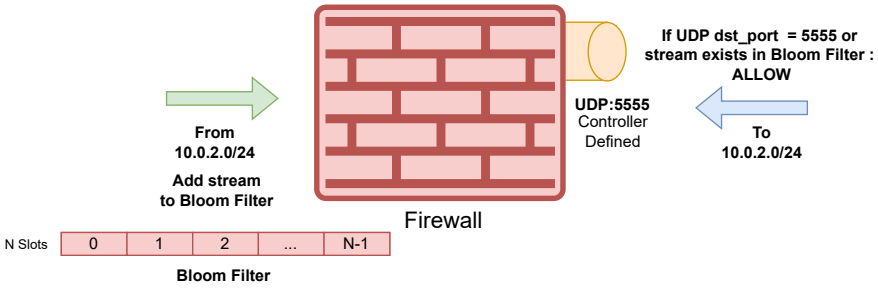

Figure 5: Firewall overview.

Figure 5 illustrates the overall flow of the firewall:

- Outgoing traffic from the LAN is recorded in the bloom filter by hashing the defined tuple and updating the register.
- Incoming traffic is checked against the bloom filter to verify if it corresponds to an established connection.
- If the incoming packet matches the bloom filter or targets an allowed UDP port, it is forwarded; otherwise, it is dropped.

This approach provides a scalable and efficient mechanism to track and manage UDP connections, ensuring that only legitimate reply traffic is allowed while maintaining control over predefined traffic rules. You will need to define which tuple describes the packet flow, and use that tuple to create hash values.

3. Controller

In this task, you will develop a Python-based controller that uses a gRPC stub to interface with the P4Runtime API. The controller will manage the flow of traffic through the network, ensuring proper functionality of the My Sequence Label Protocol (MSLP) and Stateful Firewall. It will also handle tunnel load balancing and configure rules specific to each network device (routers, switch, and firewall). Figure 6 shows an overview of the controller interactions with the data plane.

The controller's primary responsibility is to interact with the data plane, specifically managing the MSLP label forwarding, firewall rules, and the dynamic allocation of traffic across tunnels. The controller will have two main tasks:

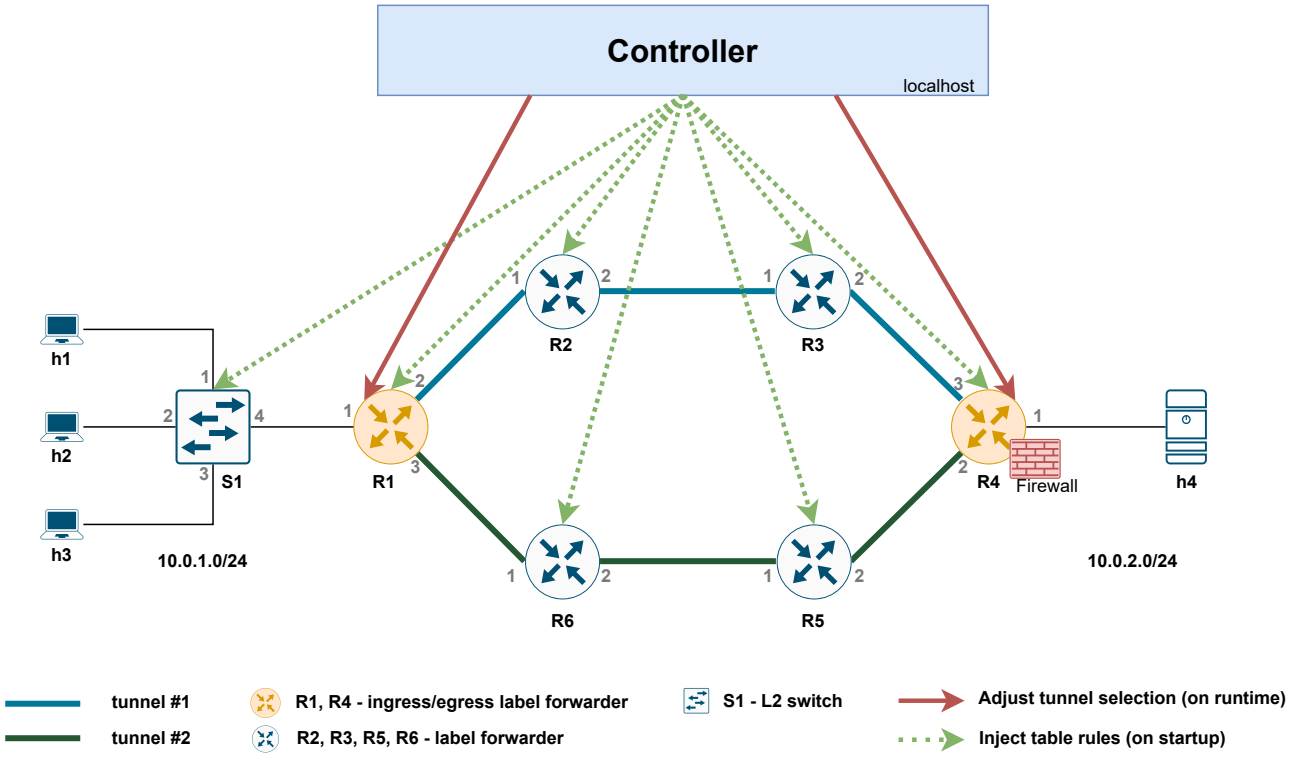

Figure 6: Controller overview.

1. **Injecting Flows into Data Plane Tables:**
   The controller will inject the necessary flow entries into the P4 tables that manage packet forwarding based on the MSLP labels. These entries will ensure proper encapsulation and decapsulation of packets, enabling tunnel-based routing. It will also configure the stateful firewall, ensuring that only valid UDP traffic is allowed through the network according to predefined rules.
2. **Tunnel Load Balancing:**
   The controller will balance the load between the two available tunnels based on real-time traffic conditions. The load balancing technique can be time-based (round-robin) or metric-based (using collected metrics such as packet count, bandwidth, or latency). The controller can gather statistics from the network devices (routers and switches) to make informed decisions and dynamically adjust which tunnel traffic should flow through.

The Python-based controller will serve as the brain of the programmable network, using gRPC and P4Runtime to configure, monitor, and dynamically adjust the network's behavior. By managing MSLP label forwarding, firewall configurations, and tunnel load balancing, the controller ensures efficient and robust packet delivery across the network while adapting to real-time traffic demands.

Tasks - Overview

1. **Create the Topology:**
   Use Mininet to create the topology shown in Figure 1. You can use the examples provided in [1]. Ensure that the connections, IP addressing, and MAC addresses align with the topology diagram.
2. **Define the MSLP Header Format:**
   Design the My Sequence Label Protocol (MSLP) header. Specify:
   - The number of fields in the header.
   - The size (in bits) of each field.
   - The purpose of each field, particularly for tunneling and label identification.
3. **Implement the P4 Program for Ingress (R1) and Egress (R4) Devices:**
   Develop the logic for handling packets entering and exiting the tunnel:

- **Tunnel Selection:**
  Identify which fields (e.g., IP addresses, protocol, or port numbers) will be used to determine the tunnel. Hash a tuple of these fields to produce a binary result:
  - '0': Select tunnel#1.
  - '1': Select tunnel#2.
- **Label Assignment:**
  Use the hash result to look up a table containing the label stack for the selected tunnel.
- **MSLP Header Construction:**
  Use the retrieved labels to construct the MSLP header by pushing the labels onto the packet.
- **Egress Logic:**
  Implement logic to remove the MSLP header and restore the original packet format as the packet exits the tunnel.

4. **Implement the P4 Program for Forwarding Label Routers (R2, R3, R5, and R6):**
   Configure these routers to handle packets within the tunnel:
   - **Pop Operation:**
     Remove the top label from the label stack.
   - **Forwarding Decision:**
     Use the popped label to determine the outgoing port for the packet.
5. **Enhance the P4 Program for R4 (Firewall Functionality):**
   Extend the R4 program to include a stateful firewall for UDP traffic:
   - **Outgoing Traffic Bloom Filter:**
     Implement a Bloom filter to record outgoing traffic originating from the '10.0.2.0/24' LAN. This filter should store a hash of the traffic to allow verification of valid reply packets. [1] & [2]
   - **UDP Port Control:**
     Implement P4 structures (e.g., tables or rules) to allow UDP traffic to specific destination ports. These ports should be defined and managed by the controller.
   - **Traffic Validation:** [3]
     Write the logic to verify if incoming traffic (destined for the '10.0.2.0/24' LAN) is valid:
     - Allow replies only if corresponding traffic was recorded in the Bloom filter.
     - Permit traffic to the explicitly allowed UDP ports.
6. **Write your Controller script:**
   It can be inspired in the controller done in class [1], it can run directly on the localhost.
   - **Inject rules:**
     - every table of every device in the network needs be populated by the controller.
   - **Implement your tunnel load balance technique:**
     - you can use a dedicated thread for your algorithm of tunnel load balancing.

References

[1] J. F. Pereira, *Rds-tasks repository*, https://github.com/jfpereira-uminho/.

[2] P4 Language Community, *Firewall exercise solution*, https://github.com/p4lang/tutorials/blob/master/exercises/firewall/solution/firewall.p4.

[3] E. Zurich, *Stateful packet processing and applications*, https://adv-net.ethz.ch/2018/pdfs/03_stateful.pdf.