Software Defined Networking Exercise

Case Study – Exercise 5

Julius Rückert

Author of selected student solution:
Daniel Gonzalez

http://www.ps.tu-darmstadt.de/teaching/ws1516/sdn/

PS - Peer-to-Peer Systems Engineering Lab
Dept. of Electrical Engineering and Information Technology
Technische Universität Darmstadt
Rundeturmstr. 12, D-64283 Darmstadt, Germany
http://www.ps.tu-darmstadt.de/

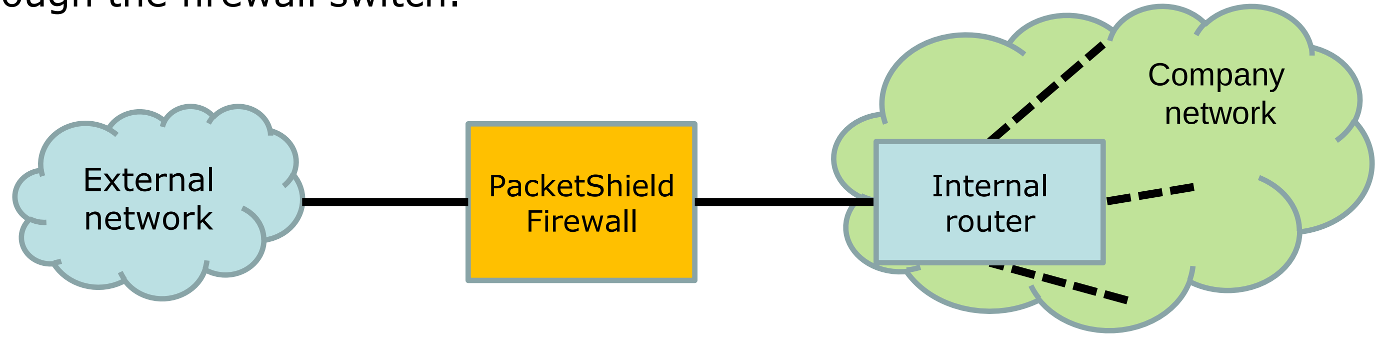
Scenario and Setting (2)

PacketShield is a new player in the network industry that sells its own **whitebox switches with OpenFlow v1.5.**0 support. Recently, they added **support for OpenState** to their devices, including the extensions to the OpenFlow protocol that are required to manipulate the new tables introduced by the approach.

As first product, PacketShield would like to implement a **stateful firewall**.

- The firewall should be able to work for both TCP and UDP and shield a company-internal network from the outside.
- Incoming packets should only be forwarded if a respective connection was previously initiated from the internal network.

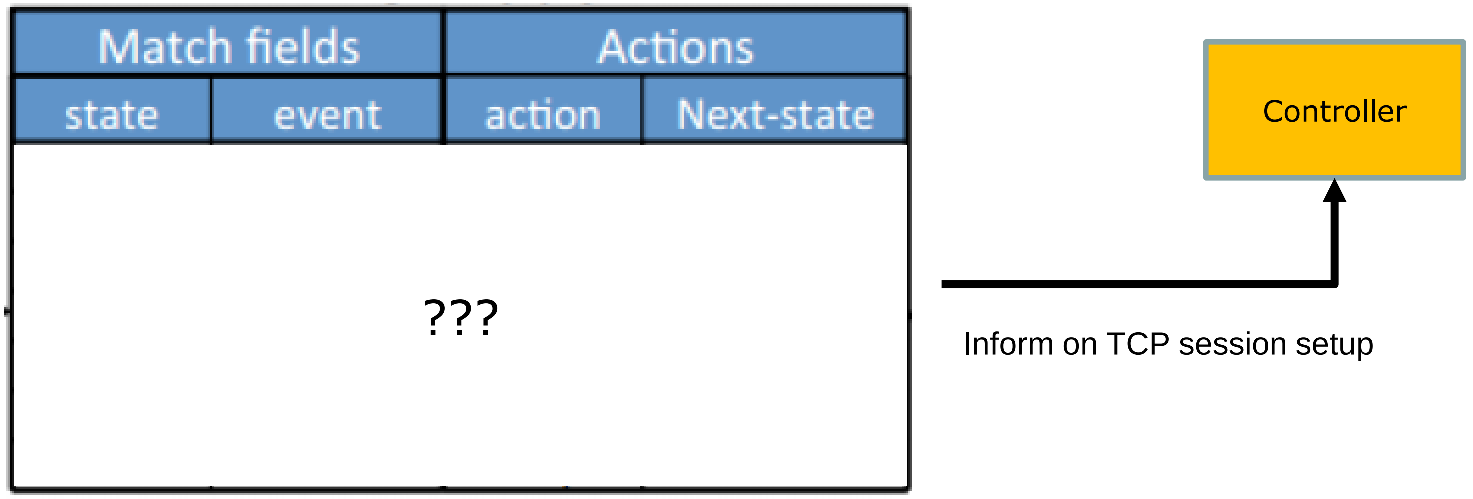
For this, assume a first simple setup, where a dedicated switches provided by PacketShield is used to connect an external network segment with an already set up router at the edge of the internal network. All external traffic from and to clients/servers inside the company network passes through the firewall switch.



Your Task

Discuss, how you would realize a stateful firewall as described above using OpenState.

- Specify the XFSM table for handling TCP sessions and discuss your approach.
- Describe how the firewall could be extended to inform a controller on TCP sessions that finished their initialization to keep a network-wide consistent view on multiple firewalls in a future setting envisioned by PacketShield.



Reminder: UDP and TCP Basics

UDP: Stateless protocol

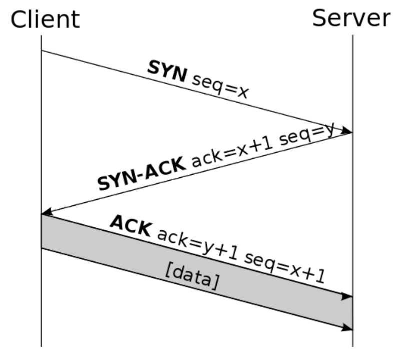
- "Incoming packets should only be forwarded if a respective connection was previously initiated from the internal network." → It is ok to assume that UDP flows are allowed to enter the network if the first packet came from the inside.

TCP Handshake example

1.	SYN-SENT	→	<SEQ=100><CTL=SYN>	→	SYN-RECEIVED
2.	SYN/ACK-RECEIVED	←	<SEQ=300><ACK=101><CTL=SYN,ACK>	←	SYN/ACK-SENT
3.	ACK-SENT	→	<SEQ=101><ACK=301><CTL=ACK>	→	ESTABLISHED

- Your solution should account for the steps of initializing a TCP connection. As we specified the use of OF 1.5, all required fields are available:

  "A new OXM field OFPXMT_OFB_TCP_FLAGS has been added to match the flag bits in the TCP header (EXT-109). This field allow to match all flags, such as SYN, ACK and FIN, and may be used to detect the start and the end of TCP connections."
  [Source: OF v1.5.0 https://goo.gl/VDxZN2]

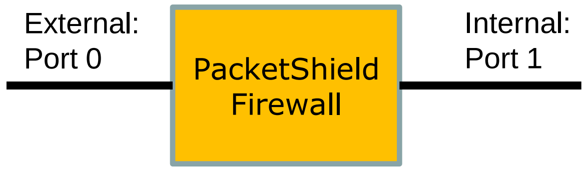


[Source for TCP: https://de.wikipedia.org/wiki/Transmission_Control_Protocol]

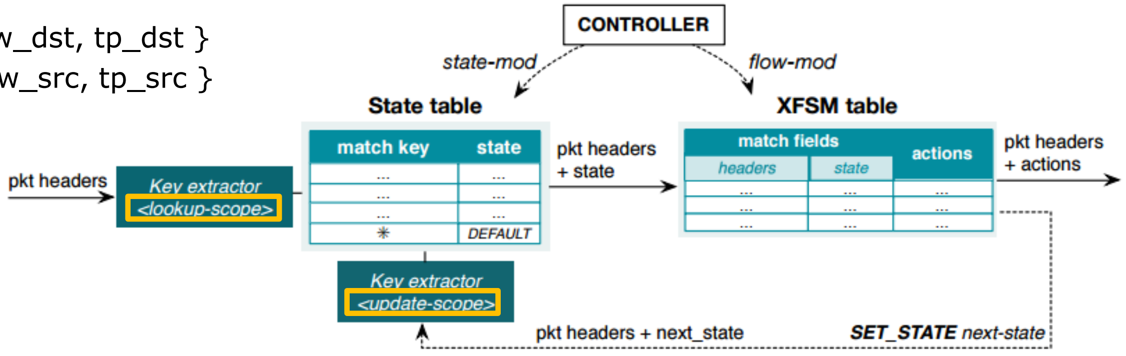
Sample Student Solution

Scope Considerations

- The switch installed by PacketShield would need two Ports: One where the external network is connected (Port 0) and one where the internal network is connected (Port 1).



- A connection can be defined as a pair of IP-Port combinations: The source and the destination of the packets of the connection. We use these combinations to identify the flow of the connection.
- Similar to the situation presented in Section 2.3 of the OpenState paper, we need to separate a lookup-scope and update-scope to ensure that we use the state table correctly and to reflect the fact that both combinations can be source or destination of a packet.
  We define:
  - lookup-scope = { nw_src, tp_src, nw_dst, tp_dst }
  - update-scope = { nw_dst, tp_dst, nw_src, tp_src }



Source: http://openstate-sdn.org/pub/openstate-ccr.pdf

Sample Student Solution

Basic Case for Illustration

- Each possible connection can have two states: NOT_CONNECTED or CONNECTED. The default entry in the state table maps to NOT_CONNECTED, to reflect the fact that unestablished connections are, naturally, unconnected.
- In the XFSM table we use in_port as event. in_port = 1 means the packet comes from the internal network, in_port = 2 means the packet comes from the external network. With this we get the following table:

Match fields		**Actions**	
state	**event**	**action**	**next-state**
NOT_CONNECTED	in_port = 1	forward out_port = 0	CONNECTED
NOT_CONNECTED	in_port = 0	drop	NOT_CONNECTED
CONNECTED	in_port = 1	forward out_port = 0	CONNECTED
CONNECTED	in_port = 0	forward out_port = 1	CONNECTED

- This means that new connection from the external network (state = NOT_CONNECTED and in_port = 0) will be dropped by the firewall. New connections from the internal network in the other side will be forwarded and the next-state will be set to CONNECTED. This means the state table will be updated with the newly established connection.

Sample Student Solution

Basic Case for Illustration: Example

- Lets assume the host on the internal network has the IP address 10.0.0.1 and that it uses the port 1234 for this connection. The destination host on the external network has the IP address 10.1.0.1 and the port is 5678.
- When the first packet from this connection, with nw_src=10.0.0.1, tp_src=1234, nw_dst=10.1.0.1, tp_dst=5678, gets processed by the switch there will be no entry for it in the state table, so it gets the state NOT_CONNECTED assigned.
- Because in_port=1 the XFSM table indicates that the action to perform is to forward the packet and to set next-state to CONNECTED. So the state table gets the new entry
  { 10.1.0.1, 5678, 10.0.0.1, 1234 } -> CONNECTED
  and the packet gets forwarded.
- When the answer to this packet reaches the switch, it will query the state table with the flow key { 10.1.0.1, 5678, 10.0.0.1, 1234 }, which is exactly the key we stored before (this is why we use different lookup- and update-scopes). The state of this entry is CONNECTED, so the XFSM table indicates that the packet should be forwarded and next-state is CONNECTED. So we save the new state
  { 10.0.0.1, 1234, 10.1.0.1, 5678 } -> CONNECTED.
- When the internal host answers this packet, it will match exactly that new entry in the state table. So after this exchange of messages we have two states in the state table, each representing one direction of the connection:
  - { 10.1.0.1, 5678, 10.0.0.1, 1234 } -> CONNECTED
  - { 10.0.0.1, 1234, 10.1.0.1, 5678 } -> CONNECTED

Sample Student Solution

Advanced Case including TCP Session Tracking

- So for this to work we need to extend our solution. As a first step we extend the events of all current rows in the XFSM table with nw_proto=17 which means that the flow uses UDP.
- With this extension we have limited our solution to UDP, and must now add new states and state transitions for TCP. The new state we add to our solution is CONNECTING, which is activate between the first SYN and the ACK. The new state transitions can be seen in this full XFSM table:

Match fields		Actions	
state	**event**	**action**	**next-state**
NOT_CONNECTED	nw_proto=17, in_port=1	forward out_port=0	CONNECTED
NOT_CONNECTED	nw_proto=6, in_port=1	forward out_port=0	CONNECTING
NOT_CONNECTED	in_port=0	drop	NOT_CONNECTED
CONNECTING	nw_proto=6, in_port=1	forward out_port=0	CONNECTED
CONNECTING	nw_proto=6, in_port=0	forward out_port=1	CONNECTING
CONNECTED	in_port=1	forward out_port=0	CONNECTED
CONNECTED	in_port=0	forward out_port=1	CONNECTED

- As one can see all state transitions which differ between TCP and UCP have nw_proto=17 (UDP) or nw_proto=6 (TCP) added to their event.
  (Remark: writing “nw_proto=UDP” or “nw_proto=TCP” is fine as well)

Sample Student Solution
Advanced Case: Example (1)

- We work with the same data as in the example before, which means we have a host in the internal network which has the IP address 10.0.0.1 and uses the port 1234 and we have a host in the external network which has the IP address 10.1.0.1 and uses the port 5678.
- The connection has to be initiated by the internal host, as all packets from the outside (in_port=0) which do not belong to connected flows (state=NOT_CONNECTED) get dropped by the firewall.
- So our internal host sends his first packet (TCP SYN) destined to the external host. The initial state of the flow is NOT_CONNECTED, nw_porto=6 and in_port=1, so the firewall will forward the packet and set the next-state to CONNECTING. So we add the following entry to the state table:
  { 10.1.0.1, 5678, 10.0.0.1, 1234 } -> CONNECTING
  (remember that we use { nw_dst, tp_dst, nw_src, tp_dst } as update-scope).
- The external host should now answer that first packet with a TCP SYN-ACK. Using our lookup-scope the firewall identifies in the state table that this flow has the state CONNECTING. Because nw_proto=6 and in_port=0 it forwards the packet and sets next-state to CONNECTING. Because of this we add a second state, representing the other direction of our connection:
  { 10.0.0.1, 1234, 10.1.0.1, 5678 } -> CONNECTING.
- So we have two states now, one for each direction of our connection, and both set to CONNECTING:
  - { 10.1.0.1, 5678, 10.0.0.1, 1234 } -> CONNECTING
  - { 10.0.0.1, 1234, 10.1.0.1, 5678 } -> CONNECTING

Sample Student Solution

Advanced Case: Example (2)

- After the packet reached the internal host, it will answer with a TCP ACK to finish the handshake. When this packet reaches the firewall it will find the state CONNECTING for the flow of this packet. Because nw_proto=6 and in_port=1, the firewall will forward the packet and set next-state to CONNECTED. This results in the following entries for the state table:
  - { 10.1.0.1, 5678, 10.0.0.1, 1234 } -> CONNECTED
  - { 10.0.0.1, 1234, 10.1.0.1, 5678 } -> CONNECTING

- So our connection is established, and the first direction of the connection is already marked as connected. The second direction will be set to CONNECTED as soon as the external host sends his first packet after receiving the TCP ACK. As soon as this packets reaches the firewall, its state will be CONNECTED, so the firewall will forward it to the internal host and set next-state to CONNECTED. This will result in an update for the second entry in the state table, which will set it to CONNECTED.

Sample Student Solution

Advanced Case including TCP Session Tracking: Remarks

- ❖ Pros: Solves problem and fits to TCP handshake.
- ❖ Cons: It does not actually check if the packets are indeed SYN, SYN-ACK, ACK packets!
- ❖ Improvements: Using OpenFlow 1.5, the flag bits in the TCP header can be matched. So the rules can be made more strict. The lookup-scope does not need to be extended as flows are still defined by the same fields as before.

Match fields		**Actions**	
state	**event**	**action**	**next-state**
NOT_CONNECTED	nw_proto=17, in_port=1	forward out_port=0	CONNECTED
NOT_CONNECTED	nw_proto=6, in_port=1, **tcp_flags=+syn-ack**	forward out_port=0	CONNECTING
NOT_CONNECTED	in_port=0	drop	NOT_CONNECTED
CONNECTING	nw_proto=6, in_port=1, **tcp_flags=-syn+ack**	forward out_port=0	CONNECTED
CONNECTING	nw_proto=6, in_port=0, **tcp_flags=+syn+ack**	forward out_port=1	CONNECTING
CONNECTED	in_port=1	forward out_port=0	CONNECTED
CONNECTED	in_port=0	forward out_port=1	CONNECTED

Sample Student Solution

Inform Controller on new TCP Sessions

- If we want to notify the SDN controller about initialised connections we have to change the fourth entry in above XFSM table:
  { CONNECTING; nw_proto=6,in_port=1; forward out_port=0; CONNECTED }.
- It is the state transition which does the initial change from state CONNECTING to CONNECTED after the TCP ACK has been sent, so this is the point the controller has to be notified about the initialised connection.
- To notify the controller, we must change the action to forward the packet to the controller. So the entry changes to
  { CONNECTING; nw_proto=6,in_port=1; forward out_port=controller; CONNECTED }.
  This will ensure the controller knows about this new connection.
- For the TCP ACK to reach its destination, the SDN controller must forward the packet after it has registered the new connection.

- Pro: Solves problem.
- Con: Adds additional delay due to active forwarding of controller in last step.
- Improvement: The entry could be extended to hold more than one action (list of actions as available also for all normal OpenFlow flow entries is possible):
  { CONNECTING; nw_proto=6, in_port=1; [forward out_port=0, forward out_port=controller]; CONNECTED }.
  This way, the above final step is obsolete and no additional delays are added.