Advanced Network Architectures and Wireless Systems Project Report

Lorenzo Catoni   Leonardo Giovannoni   Marco Lucio Mangiacapre

Contents

1 Project Requirements

This project aims to implement a Floodlight[1] module for a network scenario where computing nodes are connected through an SDN-based network. The module will have the following objectives:

- Periodically collect utilization statistics of Flow Tables by using OFTableStatsRequest[2].
- Provide a RESTful interface for users to set the expected duration of flows based on specific source and destination IP addresses, as well as source and destination ports.
- Process Packet-In messages, match their headers with the configured fields through the RESTful API, and generate a new flow rule with a hard-timeout value based on the expected duration. If the current utilization ratio of the switch's Flow Table exceeds a threshold of 0.7, the hard-timeout is set to a short-lived default value.
- The overall system will be tested and demonstrated using GNS3 and Floodlight, with an example scenario shown in Figure 1

To elaborate on the Flow Table utilization statistic, it must be calculated by dividing the number of installed flows at a given time by a customizable maximum number of flows. Also it should be noted that the flow reservation REST API operates out of band with respect to the devices on the network.

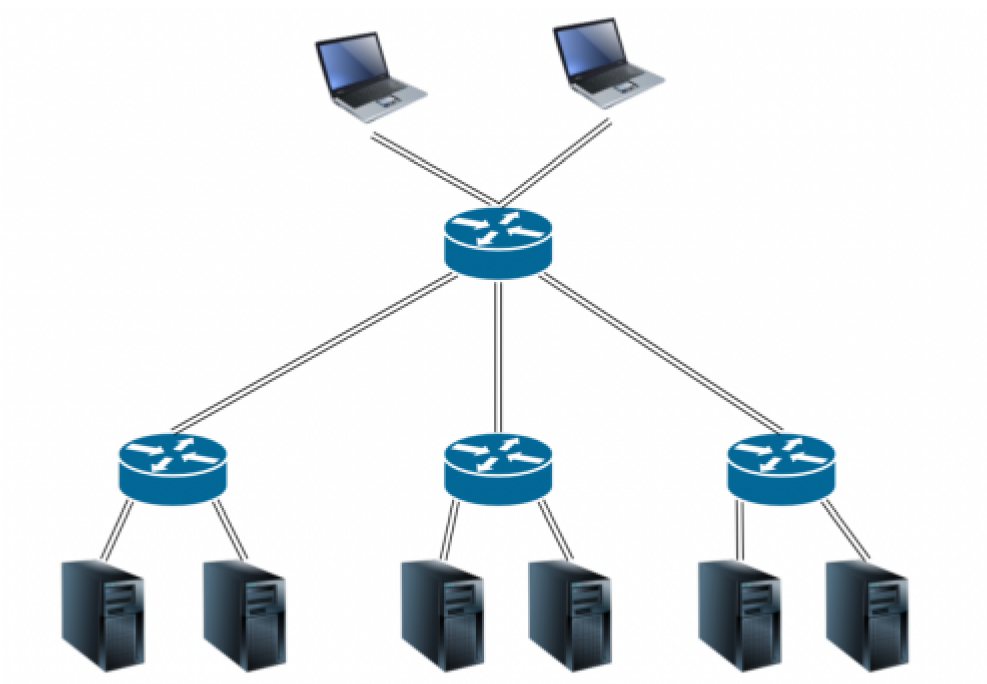


Figure 1: Required network layout

2 Our Approach

In this project, we have opted for an **out-of-band** controller approach, this decision has been made based on the context described above and the specific requirements of the project, it came naturally since the hosts do not need to be able to reach the controller. Therefore, each SDN switch has one interface connected to the controller's subnet (depicted as the red subnet in Figure 2), while all other interfaces are connected to the hosts' network (depicted as the blue subnet in Figure 2). These interfaces are part of the OVS bridge, which is controlled by the controller. Further details regarding the switch configuration will be provided in section 4.1. Both of the mentioned networks are L2 networks, meaning that only switching is involved and no routing is necessary. It is worth noting that in the context of SDN, the difference that this choice makes is minimal.

We have chosen to adopt a **reactive** approach for the controller due to the specific requirement of controlling the hard timeout of a packet. The reactive approach was the only possible solution because it ensures that the

[1]`https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview`

[2]This functionality is made available at a higher level by a Floodlight module[3]

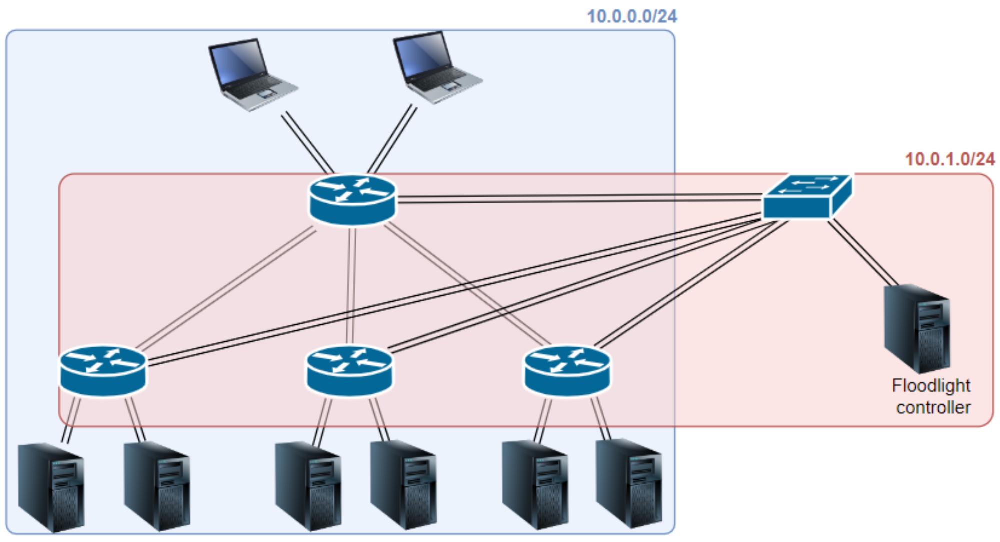


Figure 2: Network layout with emphasys on the two different subnets

necessary flow rules are in place precisely when the flow starts, allowing us to control the hard timeout of the packet effectively.

In contrast, if a proactive approach were adopted for the controller, the hard timeout counter would be triggered regardless of whether any packets had passed through the flow. This behavior is not desired in our scenario.

This sequence of bullet points together with Figure 3[4] provides a clear overview of the interactions between the client, switches, and controller, highlighting the flow of packets and the role of each component in managing the flow's behavior:

- The client sends the first packet (1).
- The switch sends the packet to the controller (2).
- The controller sends the flow rule to all switches in the path to allow the packet (3).
- The packet is forwarded to the destination device (4 and 5).
- The same process is repeated for the response packet.
- After the timeout period is over, the switches consult the controller again to determine further actions.
- The controller instructs the switches to drop the flow.

[4]The flow of packets for the response packet is not shown for the sake of image readability

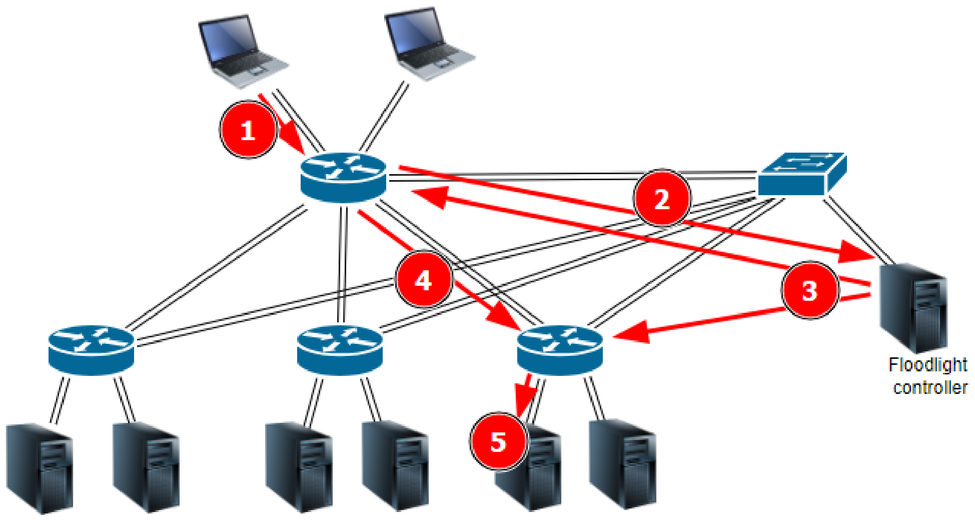


Figure 3: Flow of packets during a request

In our design, we have made the following decision: by default, all ARP traffic and ICMP traffic are allowed. However, for TCP and UDP traffic, access is granted only if a corresponding flow rule is requested, matching the parameters of the specific connection, as per requirement. Any protocols other than the ones mentioned above will be dropped. We made this decision based on the project specification, which states that flows must be reserved by specifying the source IP, destination IP, source port, and destination port. Considering this requirement, it is logical to allow ARP and ICMP traffic by default. We arrived at this decision because ARP is necessary for proper network functionality, ensuring the correct resolution of IP addresses to MAC addresses. Additionally, ICMP is valuable for network debugging purposes, providing essential diagnostic and troubleshooting capabilities. By allowing ARP and ICMP traffic by default, we ensure that the network functions correctly and facilitate efficient network debugging when needed.

3 Floodlight

By examining other Floodlight modules, such as the firewall module, we discovered an opportunity to leverage the existing forwarding module for installing flows in switches. In these modules, the communication with the forwarding module occurs through the passing of the `RoutingDecision` object. The code snipped 1 taken from the firewall module gives an example of how that interaction takes place.

```
if (eth.isBroadcast() == true) {
 if (allowBroadcast == true) {
   decision = new RoutingDecision(sw.getId(), inPort,
       IDeviceService.fcStore.get(cntx, IDeviceService.CONTEXT_SRC_DEVICE),
       IRoutingDecision.RoutingAction.MULTICAST); // <- allow the packet
   decision.setDescriptor(ALLOW_BCAST_COOKIE);
   decision.addToContext(cntx); // <- store the routing decision so subsequent molues can
   read it
 } else {
   decision = new RoutingDecision(sw.getId(), inPort,
       IDeviceService.fcStore.get(cntx, IDeviceService.CONTEXT_SRC_DEVICE),
       IRoutingDecision.RoutingAction.DROP); // <- drop the packet
   decision.setDescriptor(DENY_BCAST_COOKIE);
   decision.addToContext(cntx); // <- store the routing decision so subsequent molues can
   read it
 }
```

```
  return Command.CONTINUE;
}
```

Listing 1: Firewall module instructing the Forwarding module about what action to take with the incoming packet (taken from Firewall.java:600, simplified)

To align with the design pattern of the firewall module, we made the decision to follow a similar approach. However, to make things work, we needed to make certain modifications to the forwarding module itself. After implementing these adjustments, the exam module emerged organically and seamlessly integrated into the overall architecture. The resulting module design is both intuitive and streamlined, benefiting from the foundation laid by the modifications made to the forwarding module.

3.1 The Forwarding Module

First and foremost, we began by modifying the abstract base class `ForwardingBase` to accommodate our requirements. Specifically, we introduced the capability to specify idle and hard timeouts and incorporated them into the construction of the flow mod packet. The `pushRoute` function, responsible for distributing the flow mod command to all switches along the designated path, was adapted accordingly. These adjustments enabled us to effectively incorporate the desired timeout functionality and efficiently propagate the command throughout the network. Code snippet 2 shows the differences made to the `ForwardingBase` class.

```diff
diff --git a/floodlight/src/main/java/net/floodlightcontroller/routing/ForwardingBase.java b/floodlight/src/main/java/net/floodlightcontroller/routing/ForwardingBase.java
index 9b3bbc6..b8796ab 100644
--- a/floodlight/src/main/java/net/floodlightcontroller/routing/ForwardingBase.java
+++ b/floodlight/src/main/java/net/floodlightcontroller/routing/ForwardingBase.java
@@ -160,9 +160,10 @@ public abstract class ForwardingBase implements IOFMessageListener {
      *        OFFlowMod.OFPFC_MODIFY etc.
      * @return true if a packet out was sent on the first-hop switch of this route
      */
-    public boolean pushRoute(Path route, Match match, OFPacketIn pi,
-            DatapathId pinSwitch, U64 cookie, FloodlightContext cntx,
-            boolean requestFlowRemovedNotification, OFFlowModCommand flowModCommand, boolean packetOutSent) {
+       public boolean pushRoute(Path route, Match match, OFPacketIn pi,
+                       DatapathId pinSwitch, U64 cookie, FloodlightContext cntx,
+                       boolean requestFlowRemovedNotification, OFFlowModCommand flowModCommand, boolean packetOutSent,
+                       int idleTimeout, int hardTimeout) {

         List<NodePortTuple> switchPortList = route.getPath();

@@ -221,8 +222,8 @@ public abstract class ForwardingBase implements IOFMessageListener {
             }

             fmb.setMatch(mb.build())
-            .setIdleTimeout(FLOWMOD_DEFAULT_IDLE_TIMEOUT)
-            .setHardTimeout(FLOWMOD_DEFAULT_HARD_TIMEOUT)
+            .setIdleTimeout(idleTimeout)
+            .setHardTimeout(hardTimeout)
             .setBufferId(OFBufferId.NO_BUFFER)
             .setCookie(cookie)
             .setOutPort(outPort)
```

Listing 2: Differences made to the `ForwardingBase` class

Naturally we incorporated these additional parameters into the function call to `PushRoute` made from the forwarding module, see code snippet 3. We made sure not to assume the presence of a routing decision object, and if it is null, the default values for the idle and hard timeouts are used. This approach prevents any unintended modification of the behavior for other modules utilizing the forwarding module.

```diff
diff --git a/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java b/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
index 3d8a5e4..ec26171 100644
--- a/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
```

```
+++ b/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
@@ -478,7 +482,9 @@ public class Forwarding extends ForwardingBase implements IFloodlightModule, IOF

                 pushRoute(path, m, pi, sw.getId(), cookie,
                         cntx, requestFlowRemovedNotifn,
-                        OFFlowModCommand.ADD, false);
+                        OFFlowModCommand.ADD, false,
+                        decision != null ? decision.getIdleTimeout() : FLOWMOD_DEFAULT_IDLE_TIMEOUT,
+                        decision != null ? decision.getHardTimeout() : FLOWMOD_DEFAULT_HARD_TIMEOUT);

                 /*
                  * Register this flowset with ingress and egress ports for link down
@@ -510,7 +516,9 @@ public class Forwarding extends ForwardingBase implements IFloodlightModule, IOF
             Path newPath = getNewPath(path);
             pushRoute(newPath, match, pi, sw.getId(), cookie,
                     cntx, requestFlowRemovedNotifn,
-                    OFFlowModCommand.ADD, packetOutSent);
+                    OFFlowModCommand.ADD, packetOutSent,
+                    decision != null ? decision.getIdleTimeout() : FLOWMOD_DEFAULT_IDLE_TIMEOUT,
+                    decision != null ? decision.getHardTimeout() : FLOWMOD_DEFAULT_HARD_TIMEOUT);

             /* Register flow sets */
             for (NodePortTuple npt : path.getPath()) {
@@ -718,7 +726,9 @@ public class Forwarding extends ForwardingBase implements IFloodlightModule, IOF

             pushRoute(path, m, pi, sw.getId(), cookie,
                     cntx, requestFlowRemovedNotifn,
-                    OFFlowModCommand.ADD, false);
+                    OFFlowModCommand.ADD, false,
+                    decision != null ? decision.getIdleTimeout() : FLOWMOD_DEFAULT_IDLE_TIMEOUT,
+                    decision != null ? decision.getHardTimeout() : FLOWMOD_DEFAULT_HARD_TIMEOUT);

             /*
              * Register this flowset with ingress and egress ports for link down
```

Listing 3: Differences made to the forwarding module

The last modification we implemented worth mentioning is the introduction of the DROP_TCP routing action. This addition is necessary because, in Floodlight, dropping a flow means taking no action. Without intervention, the connection remains open until the TCP timeout is triggered, this behavior is not desired. Instead, our objective is to forcefully close the TCP connection by sending a TCP reset. This addition is shown in code snippet 4 and 5.

```
diff --git a/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java b/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
index 3d8a5e4..ec26171 100644
--- a/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
+++ b/floodlight/src/main/java/net/floodlightcontroller/forwarding/Forwarding.java
@@ -227,6 +227,10 @@ public class Forwarding extends ForwardingBase implements IFloodlightModule, IOF
                 case DROP:
                     doDropFlow(sw, pi, decision, cntx);
                     return Command.CONTINUE;
+
+                case DROP_TCP:
+                    doDropFlowTcp(sw, pi, decision, cntx);
+                    return Command.CONTINUE;

                 default:
                     log.error("Unexpected decision made for this packet-in={}", pi, decision.getRoutingAction());
```

Listing 4: Differences made to the forwarding module

```
/**
```

```
 * Write PacketOut to switch to drop the tcp packet by sending RST
 *
 * @param sw The switch on which the packet was received
 * @param pi The packet-in came from that switch
 * @param decision The decision that cause drop flow, or null
 * @param cntx The FloodlightContext associated with this OFPacketIn
 */
protected void doDropFlowTcp(IOFSwitch sw, OFPacketIn pi, IRoutingDecision decision,
    FloodlightContext cntx){

    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx, IFloodlightProviderService.
    CONTEXT_PI_PAYLOAD);
    IPv4 ipv4 = (IPv4) eth.getPayload();
    TCP tcp = (TCP) ipv4.getPayload();

    // if the packet is already RST, we don't need to send another RST
    if ((tcp.getFlags() & TCP.Flags.RST) != 0) {
     doL2ForwardFlow(sw, pi, decision, cntx, false);
     return;
    }

    OFPort inPort = OFMessageUtils.getInPort(pi);

    IPacket ethout = new Ethernet()
            .setSourceMACAddress(eth.getDestinationMACAddress())
            .setDestinationMACAddress(eth.getSourceMACAddress())
            .setEtherType(EthType.IPv4);
    IPacket ipv4out = new IPv4()
            .setSourceAddress(ipv4.getDestinationAddress())
            .setDestinationAddress(ipv4.getSourceAddress())
            .setProtocol(IpProtocol.TCP);

    // must also send the ack
    IPacket tcpout = new TCP()
            .setSourcePort(tcp.getDestinationPort())
            .setDestinationPort(tcp.getSourcePort())
            .setFlags((short) (TCP.Flags.RST | TCP.Flags.ACK))
            .setSequence(tcp.getAcknowledge())
            .setAcknowledge(
                    tcp.getSequence() + (((tcp.getFlags() & TCP.Flags.SYN) != 0) ? 1
                            : tcp.getPayload().serialize().length))
            .setWindowSize((short) 0)
            .setChecksum((short) 0);

    ipv4out.setPayload(tcpout);
    ethout.setPayload(ipv4out);

    byte[] packetdata = ethout.serialize();

    OFPacketOut po = sw.getOFFactory().buildPacketOut()
        .setData(packetdata)
        .setActions(Collections.singletonList((OFAction) sw.getOFFactory().actions().output(
    inPort, 0xffFFffFF)))
        .setInPort(OFPort.CONTROLLER)
        .build();

    messageDamper.write(sw, po);
}
```

Listing 5: Function added to correctly drop tcp packets

3.2 The Exam Module

The interface of the exam module is designed to be straightforward, requiring only two essential functionalities. Firstly, it provides a method for submitting a flow request. This allows users to specify the desired flow parameters and initiate the process of installing corresponding flow rules. Secondly, the interface includes a method for retrieving all the currently available flow requests. This enables users to access information about existing flow requests and their associated parameters.

```java
public interface IExamService extends IFloodlightService {
    public boolean submitFlowRequest(FlowRequest fr);
    public ImmutableCollection<FlowRequest> getFlowRequests();
}
```

Listing 6: Exam module interface

A `FlowRequest` is represented by a tuple of six elements. These elements collectively identify a TCP or UDP flow. The fields within the tuple include the network protocol, source IP address, destination IP address, source transport layer port, destination transport layer port, and the desired duration for the flow. This concise representation captures the essential information needed to define and manage flow requests within the exam module.

```java
/**
 * This class represents a flow request
 * it is just a glorified tuple of 6 elements:
 * - nw_proto: the network protocol (TCP or UDP)
 * - nw_src: the source IP address
 * - nw_dst: the destination IP address
 * - tp_src: the source transport port
 * - tp_dst: the destination transport port
 * - duration: the duration of the flow in seconds
 */
@JsonDeserialize(using = FlowRequestDeserializer.class)
@JsonSerialize(using = FlowRequestSerializer.class)
public class FlowRequest {
    public final IpProtocol nw_proto;
    public final IPv4Address nw_src;
    public final IPv4Address nw_dst;
    public final TransportPort tp_src;
    public final TransportPort tp_dst;
    public final int duration;

    // constructor omitted ...
}
```

Listing 7: A simplified version of the FlowRequest class

3.2.1 Packet-In handling

With this approach, the packet-in handling within the exam module becomes minimal and straightforward. The responsibility of creating and sending packet-out messages is effectively delegated to the forwarding module. By leveraging the capabilities of the forwarding module for packet management and transmission, the exam module can focus on its primary task of handling packet-in messages and processing them accordingly. This delegation ensures a clear and streamlined division of responsibilities, allowing each module to operate efficiently within its designated scope. The pseudocode in Algorithm 1 demonstrates the straightforward logic for processing a packet-in event.

Algorithm 1 Pseudocode for the `processPacketInMessage` function

```
packet ← payload of the PacketIn
if packet is ARP then
    ALLOW
else if packet is ICMP then
    ALLOW
else if packet is UDP or packet is TCP then
    if flow request matching packet is present then
        timeout ← calculate_timeout()
        ALLOW_with_timeout
    else
        DROP
    end if
else
    DROP
end if
```

3.2.2 Flow Table utilization

Gathering statistics from the switches is a straightforward process facilitated by the Floodlight statistics module[5]. We simply need to enable statistics collection and invoke the function provided by the `IStatisticsService` interface. The statistics module takes care of periodically querying the switches in a separate thread and updating its internal data store. This automation ensures that the statistics module maintains up-to-date information without requiring manual intervention. The switches are queried at regular intervals of 10 seconds, it can be configured in the `floodlight.properties` file by updating the entry with name `net.floodlightcontroller.statistics.StatisticsCollector.collectionIntervalPortStatsSeconds`

The values for the maximum number flows, default short-lived timeout, and utilization threshold are customizable by modifying the corresponding entries in the `floodlight.properties` file, as illustrated in Listing 8. This approach provides flexibility and allows for fine-tuning the behavior of the system based on individual needs.

```
net.floodligtcontroller.exam_adaptiveflowmanager.ExamModule.max-flows=32
net.floodligtcontroller.exam_adaptiveflowmanager.ExamModule.timeout-when-overloaded=10
net.floodligtcontroller.exam_adaptiveflowmanager.ExamModule.utilization-threshold=0.7
```

Listing 8: Customizable parameters for the exam module

Thanks to the aforementioned features and functionalities, the implementation of the calculate_hard_timeout function in Java becomes concise and readable, as shown in code snippet 9.

```java
/**
 * This method calculates the hard timeout for a flow request
 * if the utilization of the flow table is above a certain threshold
 * it sets the hard timeout to a fixed value, otherwise it sets it to
 * the duration specified in the flow request
 * The utilization is calculated as the number of flows divided by the
 * maximum number of flows
 */
protected short calculate_hard_timeout(IOFSwitch sw, OFPacketIn pi, IRoutingDecision
    decision, FloodlightContext cntx, FlowRequest fr) {
  Set<?> flowStats = statisticsService.getFlowStats(sw.getId());
  int num_flows = flowStats.size();
  logger.info("Number of flows: {} for switch {}", num_flows, sw.getId());

  if (num_flows > UTILIZATION_THRESHOLD * MAX_FLOWS) {
    return (short) TIMEOUT_WHEN_OVERLOADED;
  }
```

[5]https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/21856267/How+to+Collect+Switch+Statistics+and+Compute+Bandwidth+Utilization

```
  return (short) fr.duration;
}
```

Listing 9: The `calculate_hard_timeout` function

4 GNS3

We made the decision to utilize GNS3 within a Docker environment[6] to facilitate project sharing. We created two Docker images: one containing GNS3 and another containing Floodlight. These images were connected through a shared network. As depicted in Figure 4, the Floodlight controller was accessible from the GNS3 application via the cloud object. We decided to use only three switches instead of the four depicted in Figure 1. This adjustment was made because the performance of the program was significantly impacted when using four switches and two additional hosts, the system became noticeably slow and almost unusable[7].

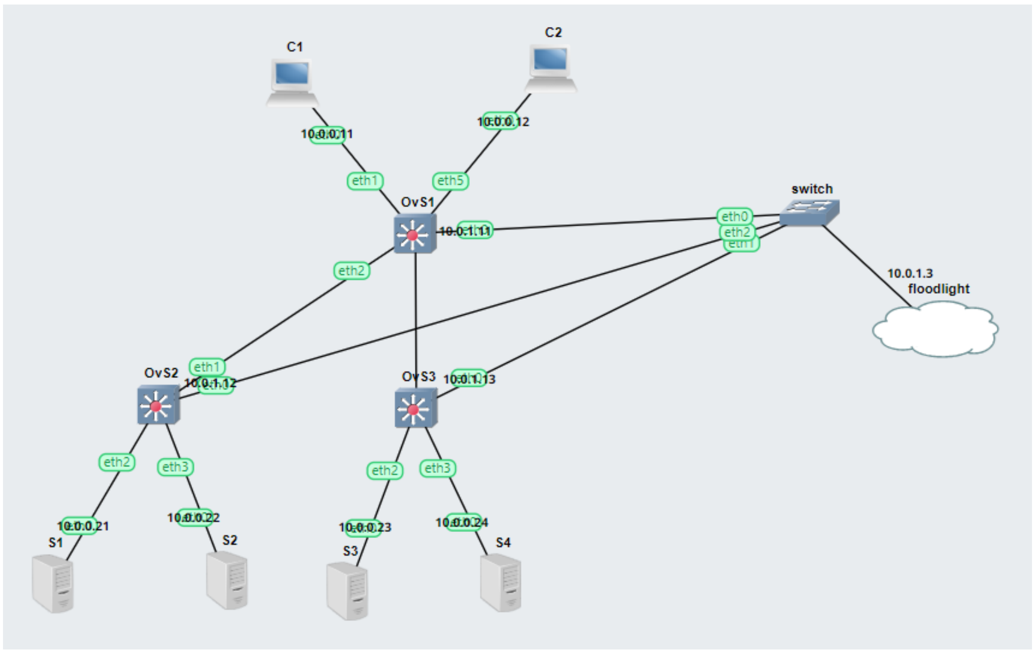


Figure 4: GNS3 network layout

4.1 Device Configuration

In this section, we will provide an overview of the configuration process for the network devices.

4.1.1 Open vSwitch Switches

The code snippet 10 shows the configuration of the three SDN switches. The official GNS3 Open vSwitch appliance was used[8] for the SDN switches.

```
id=‘hostname|tail -c 2‘

ip addr add dev eth0 "10.0.1.1${id}/24"

rm -f /etc/openvswitch/conf.db
ovsdb-tool create /etc/openvswitch/conf.db /usr/share/openvswitch/vswitch.ovsschema
ovsdb-server --detach --pidfile --remote=punix:/var/run/openvswitch/db.sock
ovs-vswitchd --detach --pidfile
ovs-vsctl --no-wait init
```

[6]https://github.com/jsimonetti/docker-gns3-server
[7]The problem is my PC that is not very performant
[8]https://gns3.com/marketplace/appliances/open-vswitch

```

ovs-vsctl add-br br0
ovs-vsctl set bridge br0 datapath_type=netdev
ovs-vsctl set bridge br0 "other-config:datapath-id=000000000000000$id"
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth2
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth4
ovs-vsctl add-port br0 eth5
ovs-vsctl set-controller br0 tcp:10.0.1.3:6653
ovs-vsctl set controller br0 connection-mode=out-of-band
ovs-vsctl set bridge br0 protocols=OpenFlow13

ip link set dev br0 up
```

Listing 10: Configuration of Open vSwitch Switches

Here is a detailed explanation of each line of the script:

- `id=‘hostname|tail -c 2‘`: get the last[9] character of the hostname and assign it to the variable ‘id‘.
- `ip addr add dev eth0 "10.0.1.1${id}/24"`: add the IP address "10.0.1.1" + ‘id‘ to the `eth0` interface.
- `rm -f /etc/openvswitch/conf.db`: remove existing Open vSwitch configuration database file.
- `ovsdb-tool create /etc/openvswitch/conf.db /usr/share/openvswitch/vswitch.ovsschema`: create a new Open vSwitch configuration database file.
- `ovsdb-server --detach --pidfile --remote=punix:/var/run/openvswitch/db.sock`: start the OVSDB server in the background.
- `ovs-vswitchd --detach --pidfile`: start the Open vSwitch daemon in the background.
- `ovs-vsctl --no-wait init`: initialize the Open vSwitch configuration.
- `ovs-vsctl add-br br0`: create a new bridge named `br0`.
- `ovs-vsctl set bridge br0 datapath_type=netdev`: This line sets the datapath type of `br0` to "netdev"[10].
- `ovs-vsctl set bridge br0 "other-config:datapath-id=000000000000000$id"`: set the datapath ID of `br0` to a unique value based on the assigned `id`.
- `ovs-vsctl add-port br0 eth*`: add `eth*` as a port to `br0`.
- `ovs-vsctl set-controller br0 tcp:10.0.1.3:6653`: set the controller for `br0`.
- `ovs-vsctl set controller br0 connection-mode=out-of-band`: sets the connection mode of the controller for `br0` to "out-of-band".
- `ovs-vsctl set bridge br0 protocols=OpenFlow13`: set OpenFlow protocol version to 1.3 for `br0`.
- `ip link set dev br0 up`: bring up the `br0` interface.

4.1.2 Servers

The code snippet 11 shows the configuration of the four servers. A custom GNS3 appliance was used for the servers, the appliance was created from a publicly available docker image[11] already configured with all the useful network utilities.

[9]The command has c 2 because it accounts for the newline
[10]It means that it should run Open vSwitch without Kernel support
[11]`https://hub.docker.com/r/praqma/network-multitool`

```
id=‘hostname|tail -c 2‘
ip addr add dev eth0 "10.0.0.2${id}/24"

echo -e ’#!/bin/bash\ncounter=1; while true; do echo $counter; counter=$((counter + 1)); sleep 1; done’ > /root/count
chmod +x /root/count

socat TCP4-LISTEN:1234,reuseaddr,fork EXEC:/root/count
```

Listing 11: Configuration of Servers

Here is a detailed explanation of each line of the script:

- `id=‘hostname|tail -c 2‘`: get the last character of the hostname and assign it to the variable ‘id‘.
- `ip addr add dev eth0 "10.0.0.2${id}/24"`: add the IP address ”10.0.0.2” + ‘id‘ to the `eth0` interface.
- `echo -e ’#!/bin/bash\ncounter=1; while true; do echo $counter; counter=$((counter + 1)); sleep 1; done’ > /root/count`: create a new script file named ”count” in the directory ”/root”. The script itself is a bash script that continuously increments a counter variable and prints its value every second.
- `socat TCP4-LISTEN:1234,reuseaddr,fork EXEC:/root/count`: start a TCP server on port 1234 using the `socat` utility. Any incoming connection to this port will execute the ”/root/count” script, allowing clients to interact with the counter script.

Overall, the script sets up an IP address on the `eth0` interface, creates a bash script that continuously counts and prints the value, and establishes a TCP server to run the script when clients connect to the specified port.

4.1.3 Clients

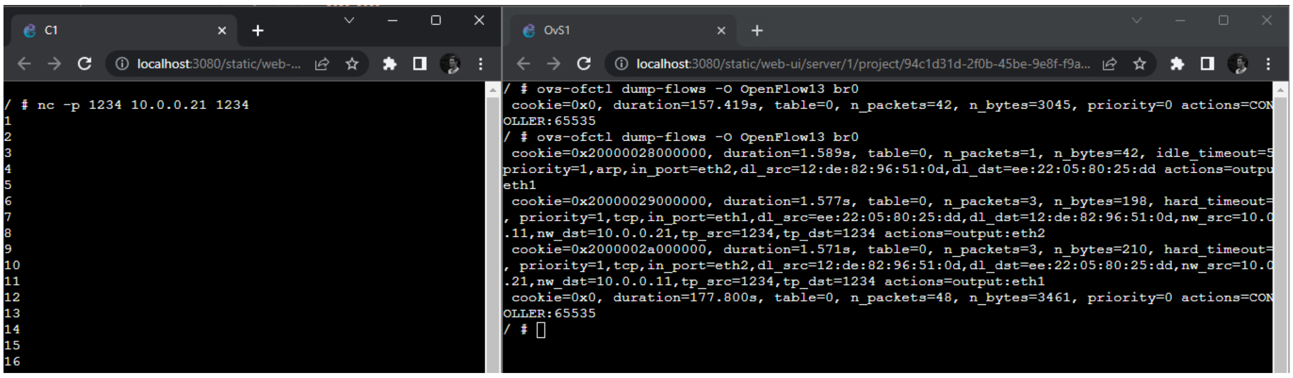
The code snippet 12 shows the configuration of the two clients. The same appliance created for the servers was used for the clients.

```
id=‘hostname|tail -c 2‘
ip addr add dev eth0 "10.0.0.1${id}/24"
```

Listing 12: Configuration of Clients

5 Conclusions

In conclusion, our approach made the code development process natural and intuitive by separating high-level intent from low-level packet creation and sending. Our solution is also extensible and seamlessly integrates with the existing modules of Floodlight. The implementation of ”servers” with time counters allows for a very intuitive demo of the functionality.


C1
localhost:3080/static/web-...
OvS1
localhost:3080/static/web-ui/server/1/project/94c1d31d-2f0b-45be-9e8f-f9a...

```
/ # nc -p 1234 10.0.0.21 1234
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

```
/ # ovs-ofctl dump-flows -O OpenFlow13 br0
 cookie=0x0, duration=157.419s, table=0, n_packets=42, n_bytes=3045, priority=0 actions=CON
OLLER:65535
/ # ovs-ofctl dump-flows -O OpenFlow13 br0
 cookie=0x20000028000000, duration=1.589s, table=0, n_packets=1, n_bytes=42, idle_timeout=5
priority=1,arp,in_port=eth2,dl_src=12:de:82:96:51:0d,dl_dst=ee:22:05:80:25:dd actions=outpu
eth1
 cookie=0x20000029000000, duration=1.577s, table=0, n_packets=3, n_bytes=198, hard_timeout=
, priority=1,tcp,in_port=eth1,dl_src=ee:22:05:80:25:dd,dl_dst=12:de:82:96:51:0d,nw_src=10.0
.11,nw_dst=10.0.0.21,tp_src=1234,tp_dst=1234 actions=output:eth2
 cookie=0x2000002a000000, duration=1.571s, table=0, n_packets=3, n_bytes=210, hard_timeout=
, priority=1,tcp,in_port=eth2,dl_src=12:de:82:96:51:0d,dl_dst=ee:22:05:80:25:dd,nw_src=10.0
.21,nw_dst=10.0.0.11,tp_src=1234,tp_dst=1234 actions=output:eth1
 cookie=0x0, duration=177.800s, table=0, n_packets=48, n_bytes=3461, priority=0 actions=CON
OLLER:65535
/ #
```

Figure 5: Demo of client that is connected to a server (left) and flow table of the switch (right)