Advanced Network Architectures and Wireless Systems Project

Adaptive Flow-Table Management

Lorenzo Catoni, Leonardo Giovannoni, Marco Lucio Mangiacapre

Roadmap

Roadmap

Project Aim

- Implement a Floodlight module[1]
- Network scenario: computing nodes connected through an SDN-based network

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

Objectives

- Collect utilization statistics of Flow Tables using OFTableStatsRequest[2]
- Provide a RESTful interface for setting expected duration of flows
- Process Packet-In messages
- Generate a new flow rule with hard-timeout value
- Test the system using GNS3 and Floodlight

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

Additional Information

- Flow Table utilization: number of installed flows / maximum number of flows
- The flow reservation REST API operates out of band

Network Layout

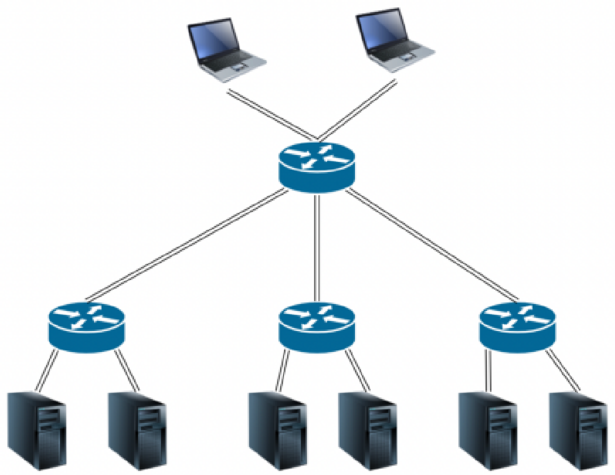

Figure 1: Required network layout

Roadmap

Controller Approach

- Opted for an **out-of-band** controller approach
- Each SDN switch has an interface connected to the controller's subnet
- All other interfaces are connected to the hosts' network
- Both networks are L2 networks - only switching is involved, no routing is necessary

Network Layout

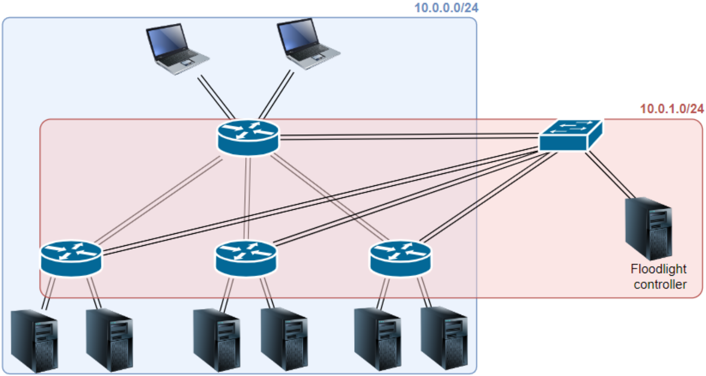

Figure 2: Network layout with emphasis on the two different subnets

Controller Approach - Reactive

- Chosen a **reactive** approach for the controller
- Ensures necessary flow rules are in place when the flow starts
- Allows effective control over the hard timeout of the packet
- Proactive approach would trigger the hard timeout counter prematurely

Flow of Packets and Roles

- Client sends the first packet (1)
- Switch sends the packet to the controller (2)
- Controller sends the flow rule to switches to allow the packet (3)
- 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, switches consult the controller again
- The controller instructs the switches to drop the flow

Flow of Packets Diagram

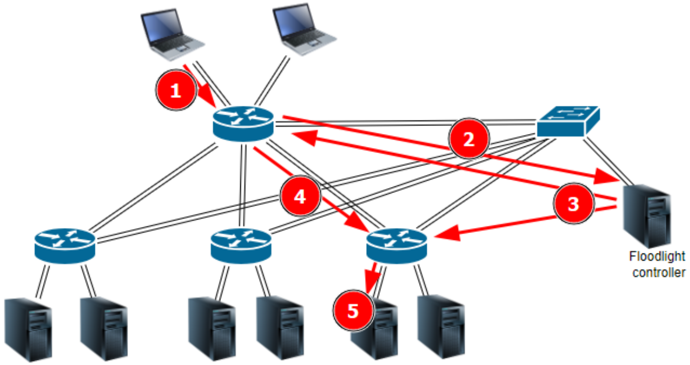

Figure 3: Flow of packets during a request

Design Decisions

- By default, all ARP traffic and ICMP traffic are allowed
- For TCP and UDP traffic, access is granted only if a corresponding flow rule is requested
- Other protocols are dropped
- ARP and ICMP traffic are allowed by default due to their importance for network functionality and debugging

Roadmap

Floodlight Exploration

- Floodlight modules examined to discover potential for leveraging the existing forwarding module
- `RoutingDecision` object key for inter-module communication
- Decided to align with design pattern of firewall module
- Modifications needed on the forwarding module to accommodate our needs

Firewall Module Example Code

```
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
    decision.setDescriptor(ALLOW_BCAST_COOKIE);
    decision.addToContext(cntx);
  } else {
    decision = new RoutingDecision(sw.getId(), inPort,
        IDeviceService.fcStore.get(cntx, IDeviceService.CONTEXT_SRC_DEVICE),
        IRoutingDecision.RoutingAction.DROP); // <- drop
    decision.setDescriptor(DENY_BCAST_COOKIE);
    decision.addToContext(cntx);
  }
  return Command.CONTINUE;
}
```

Modifying ForwardingBase

- Modified the abstract base class `ForwardingBase`
- Introduced the ability to specify idle and hard timeouts
- Adjusted the `pushRoute` function to accommodate the timeouts

ForwardingBase Changes Code

```
@@ -160,9 +160,10 @@ public abstract class ForwardingBase implements IOFMessageListener {
    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) {
    ...
    fmb.setMatch(mb.build())
-   .setIdleTimeout(FLOWMOD_DEFAULT_IDLE_TIMEOUT)
-   .setHardTimeout(FLOWMOD_DEFAULT_HARD_TIMEOUT)
+   .setIdleTimeout(idleTimeout)
+   .setHardTimeout(hardTimeout)
```

Changes in the Forwarding Module

- Additional parameters incorporated into the `PushRoute` function call
- Default values are used for idle and hard timeouts if routing decision object is null

Forwarding Module Changes Code

```
@@ -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);
...
@@ -510,7 +516,9 @@ public class Forwarding extends ForwardingBase implements IFloodlightModule, IOF
  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);
...
@@ -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);
```

DROP_TCP Routing Action - Part 1

Differences made to the forwarding module

```
@@ -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());
```

DROP_TCP Routing Action - Part 2

Function added to correctly drop tcp packets

```
FUNCTION doDropFlowTcp(packet_in, eth, decision)
    ipv4 = getIPv4Payload(eth)
    tcp = getTCPPayload(ipv4)

    IF tcp.FLAGS.RST == 1
        doL2ForwardFlow(packet_in, decision)
        RETURN
    ENDIF

    inPort = getInPort(packet_in)

    eth_out = newEthernet(eth.destination, eth.source, EtherType.IPv4)
    ipv4_out = newIPv4(ipv4.destination, ipv4.source, Protocol.TCP)
    tcp_out = newTCP(tcp.destination, tcp.source, RST | ACK, tcp.ack)

    ipv4_out.payload = tcp_out
    eth_out.payload = ipv4_out

    packet_out = buildPacketOut(eth_out, inPort)

    write(switch, packet_out)
ENDFUNCTION
```

The Exam Module - Part 1

Exam module interface

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

The Exam Module - Part 2

A simplified version of the FlowRequest class

```
@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 ...
}
```

Packet-In Handling

Pseudocode for the processPacketInMessage function

```
packet gets payload of the PacketIn
IF packet is ARP
    ALLOW
ELSIF packet is ICMP
    ALLOW
ELSIF packet is UDP OR packet is TCP
    IF flow request matching packet is present
        timeout gets calculate_timeout()
        ALLOW_with_timeout
    ELSE
        DROP
    ENDIF
ELSE
    DROP
ENDIF
```

Gathering Statistics

- Enabled by the Floodlight statistics module[3]
- Automated querying and updates
- Querying intervals: 10 seconds (configurable)

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

Customizable Parameters

- Maximum number of flows, default short-lived timeout, utilization threshold
- Configurable in the `floodlight.properties` file

```
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
```

calculate_hard_timeout Function

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

Roadmap

1. Project Requirements
2. Our Approach
3. Floodlight
   - The Forwarding Module
4. GNS3 Implementation
5. Conclusions

GNS3 Implementation

- Utilized GNS3 within Docker[4] for project sharing
- Created two Docker images: GNS3 and Floodlight
- Floodlight controller accessible via the cloud object in GNS3
- Three switches used for performance reasons

[4] https://github.com/jsimonetti/docker-gns3-server

GNS3 Layout

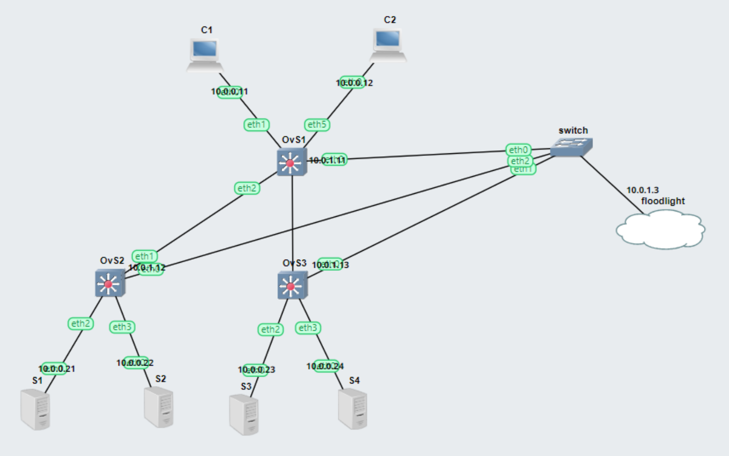

Figure 4: GNS3 network layout

Device Configuration

- Utilized official GNS3 Open vSwitch appliance[5]
- Configured three SDN switches
- Used custom GNS3 appliance for servers and clients

[5] https://gns3.com/marketplace/appliances/open-vswitch

Open vSwitch Configuration

```
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

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 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
```

Server Configuration

```
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
```

Client Configuration

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

Roadmap

Conclusion

- Approach separated high-level intent from low-level packet creation and sending, enhancing code development process.
- Solution seamlessly integrates with existing Floodlight modules, and it is extensible.
- Implementation of "servers" with time counters enabled intuitive demonstration of functionality.

Demonstration

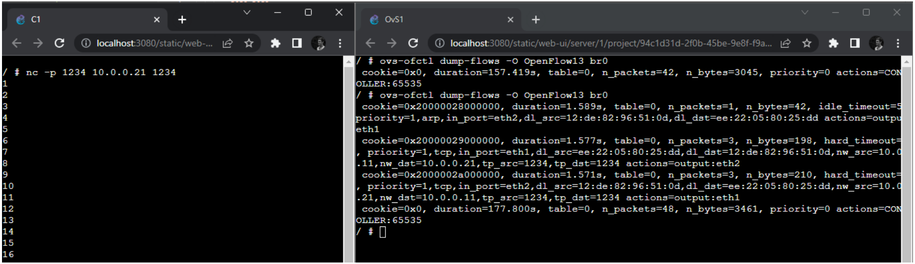

Figure 5: Demo of client that is connected to a server (left) and flow table of the switch (right)