Universität Würzburg
Lehrstuhl für Informatik III
Prof. Dr.-Ing. P. Tran-Gia

Sommersemester 2015

Unterlagen zum Hardwarepraktikum „Internet-Technologien"

Aufgabe B: OpenFlow

Christopher Metter und Anh Nguyen-Ngoc

Stand: 2. April 2015

Inhaltsverzeichnis

1 Einführung

Als Grundlage für diese Aufgabe dient das OpenFlow Tutorial 2012 des OpenFlow-Konsortiums.

Mit einem Netzwerkhub verbindet man verschiedene Netzelemente in einem Netzwerk. Die gängigen Modelle haben mindestens 4 Anschlüsse (auch Ports genannt). Ein Hub arbeitet auf der physikalischen Übertragungsebene und sendet alle Pakete, die er von einem Netzgerät bekommt, auf allen Ports an alle andere angeschlossenen Geräte. Die Geräte selbst entscheiden anschließend, ob sie die Pakete annehmen und weiterverarbeiten oder direkt verwerfen. Durch den Einsatz eines Hubs im Netzwerk wird im physikalischen Sinne eine Stern-Topologie aufgebaut. Der logische Aufbau ist allerdings ein Bus-System, bei dem alle Informationen alle Teilnehmer erreichen. Sämtliche Teilnehmer befinden sich in der gleichen *Kollisionsdomäne*.

Die Weiterentwicklung eines Netzwerk-Hubs ist der Switch. Im Gegensatz zu einem Hub wird bei einem Switch explizit entschieden, auf welchem Port welche Pakete gesendet werden. Dadurch wird die Kollisionsdomäne reduziert und das Netzwerk kann effizienter arbeiten. Die Entscheidung, welche Pakete wohin gesendet werden, trifft der Switch durch eine Adressierungstabelle. In der Tabelle speichert er für jeden Port die MAC-Adressen der angeschlossenen Geräte.

OpenFlow ist eine Schnittstelle zur Steuerung der Adressierungstabellen in Netzwerk-Switches und Routern. Indem diese Logik aus der Hardware heraus extrahiert wird, lässt sich dies auf einem speziellem Rechner, dem Controller, per Software steuern. Diese Schnittstelle ermöglicht es beispielsweise Forschern, mit einfachen Mitteln Netzwerke mit komplexen Strukturen und Eigenschaften zu realisieren. Doch nicht nur in der Forschung ist Software-Defined-Networking (SDN) interessant: Erst vor wenigen Tagen hat Google bekannt gegeben, dass dort nicht nur im internen Netz, sondern auch in den Rechenzentren großflächig auf OpenFlow gesetzt wird.

Mehr zu OpenFlow findet ihr unter:
`http://www.openflow.org/wp/learnmore/`.

Wenn ihr zum ersten mal von OpenFlow hört, solltet ihr auf jeden Fall das "OpenFlow Whitepaper" mit allgemeinen Informationen lesen:
http://www.openflow.org/documents/openflow-wp-latest.pdf

Frage: Was ist der Unterschied zwischen Packet- und Flow-Switching? Unterstützt OpenFlow das Packet-Switching?

2 Versuchsumgebung

Nicht nur die Netzwerksteuerung lässt sich in Software umsetzen, auch der Switch lässt sich virtualisieren: Mittels Open vSwitch können wir in einer Linux-Umgebung ein ganzes OpenFlow-Netzwerk simulieren. Hierzu stellen wir für die Durchführung des Praktikums ein VirtualBox-Image bereit, das bereits alle Softwarekomponenten vorinstalliert enthält.

Das Passwort des Benutzers *ubuntu* lautet *ubuntu*.

2.1 Durchführung am eigenen Rechner

Nach der VirtualBox-Installation ladet ihr euch das Image unter folgender URL herunter: http://www3.informatik.uni-wuerzburg.de/staff/christopher.metter/download/HWP_2015_-_Aufgabe_B_-_OpenFlow.ova.

Nach dem Download kann die 3,0 GB große Datei in VirtualBox über den Menüpunkt "Appliance importieren" importiert werden. Mit etwas Glück funktioniert das Experiment auch mit VMware oder anderen Virtualisierungslösungen, wobei wir hierfür keine Hilfestellung anbieten können. Sofern wir im Verlauf des Praktikums ein aktualisiertes Image anbieten, werden wir das über das Nachrichtenforum in WueCampus verkünden.

2.2 Durchführung im Windows-Pool A201

Auf den Rechnern am Lehrstuhl ist VirtualBox bereits installiert und alles vorbereitet. Nach dem Login mit Benutzer/Passwort *hwp* lässt sich die virtuelle Umgebung über die Desktopverknüpfung mit dem OpenFlow-Logo starten. Bitte beachtet, dass die virtuelle Maschine bei jedem Start zurückgesetzt wird, um anderen Gruppen eine unangetastete Umgebung zu bieten.

3 Versuchsdurchführung

3.1 Einleitung

In den folgenden Teilaufgaben werdet ihr einen OpenFlow-Controller für einen Netzwerk-Hub in einem intelligenten Controller für einen Switch verwandeln. Anschließend erweitern wir die Funktionalität um die Möglichkeit, Flow-Regeln direkt im Switch anzulegen. Auf diesem Weg werdet ihr alle OpenFlow-Debugging-Werkzeuge kennenlernen. Unter anderem werdet ihr:

- Flow-Tabellen mit `dpctl` betrachten
- OpenFlow-Nachrichten mit Wireshark analysieren
- Ein *multi-switch-*, *multi-host*-Netzwerk mit Mininet simulieren
- Einen Benchmark für euren selbst geschriebenen Controller mit `cbench` durchführen

In diesem Abschnitt beschreiben wir die Entwicklungsumgebung und die Werkzeuge, die für die Programmierung des intelligenten Switches nötig sind. Wir werden allgemeine und OpenFlow-bezogene Debugging-Werkzeuge berücksichtigen.

Vorweg definieren wir ein paar Terminologien und starten mit den Terminals:

Lokales Terminal: Dies ist das Terminal der virtuellen Linux-Maschine, auf der ihr arbeitet. Es lässt sich über die Desktop-Verknüpfung *Terminal* öffnen. In den Listings dieser Aufgabe werden Eingaben in dieses Terminal durch ein vorangestelltes `$` symbolisiert.

Mininet Terminal: Das Terminal der Netzwerkemulationsumgebung Mininet, das die virtuelle Topologie (Hosts, Switches) erzeugt. Listings sind durch ein vorangestelltes `mininet>` gekennzeichnet.

xterm Terminal: Mit diesem Terminal wird die Verbindung zwischen einem PC im virtuellen Netzwerk, das in der virtuellen Maschine erzeugt wird, hergestellt. Dies wird im nächsten Abschnitt näher erläutert. Der Fenstertitel eines xterm Terminals enthält den Namen des Hosts und im Folgenden wird das xterm durch ein vorangestelltes `xterm@hostname$` symbolisiert.

Die OpenFlow-Tutorial-VM beinhaltet eine Anzahl von OpenFlow-spezifischen und anderen Netzwerkprogrammen. Es folgt eine kurze Beschreibung der Programme:

OpenFlow-Controller: nutzt die OpenFlow-Schnittstelle. Die OpenFlow-Referenzimplementierung beinhaltet einen Controller, der sich wie ein intelligenter Netzwerk-Switch verhält. Diesen werden wir im Folgenden genauer untersuchen. Im nächsten Abschnitt werdet ihr einen eigenen Controller basierend auf RYU schreiben.

OpenFlow-Switch: basiert in unserem Fall auf dem Open vSwitch und stellt die OpenFlow-Schnittstelle bereit. Es gibt die *Kernel-Space*-basierte Variante und die *User-Space*-basierte Variante. Die Kernel-basierte Variante arbeitet im Betriebssystem als eigenständiges Modul, die letztgenannte Variante arbeitet wie ein normales Programm, das vom Benutzer gestartet werden kann. Bei der User-Space-Variante muss der Datenverkehr durch die Kernelschicht in den Benutzerkontext weitergereicht werden. Für den Produktionseinsatz haben einige Hersteller (z.B. Broadcom, HP, NEC, ..) OpenFlow-kompatible Hardware-Switches im Programm.

dpctl: ein Kommandozeilenprogramm, das OpenFlow-Nachrichten versendet. Dies ist nützlich um Switch-Port- und Flow-Statistiken anzuschauen. Des Weiteren können damit manuell Flow-Einträge angelegt werden.

Wireshark: in grafisches Programm, um Netzwerkverkehr zu betrachten. Die OpenFlow-Distribution beinhaltet einen Wireshark-Dissector. Dieser analysiert OpenFlow-Nachrichten, die an den OpenFlow-Port (6633) gesendet werden, und stellt die Nachrichten lesbar dar.

iperf: ein Kommandozeilenprogramm, um die Geschwindigkeit einer einzelnen TCP Verbindung zu testen.

Mininet: eine Netzwerk-Emulationsplattform. Mininet erzeugt ein virtuelles OpenFlow-Netzwerk: Controller, Switches, Hosts und Verbindungen. Das Ganze geschieht auf einem einzigen realen oder virtuellen PC. Weitere Details zu Mininet befinden sich auf der Mininet Homepage.

cbench: Programm zum Testen der Performance von OpenFlow-Controllern.

3.2 Netzwerktopologie erzeugen mit Mininet

Das Netzwerk für die ersten Aufgaben besteht aus drei Hosts und einem Switch:

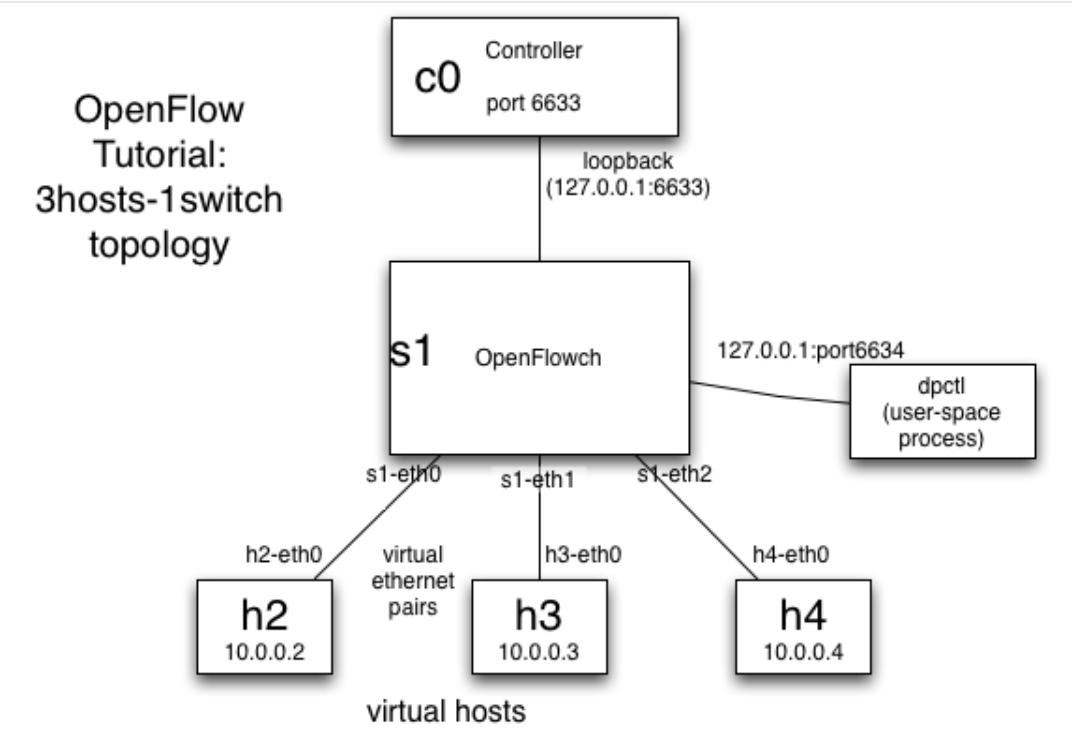

Abbildung 1: Netzwerk-Layout bestehend aus drei Hosts und einem Switch

Um mittels Mininet (`mn`) das Netzwerk in der VM zu erzeugen, schreibt in die lokale Konsole:

```
$ sudo mn --topo single,3 --mac --switch ovsk \
        --controller remote
```

Durch die Eingabe des obigen Befehls wurden durch Mininet folgende Aktionen durchgeführt:

- eine neue Topologie anlegen
- einen einzelnen Switch (`--topo single,3`) vom Typ *Open vSwitch* im Kernelkontext (`--switch ovsk`) erzeugen
- drei virtuelle Rechner der Topologie hinzufügen und IP-Adressen zuweisen
- MAC-Adressen der Interfaces (zur besseren Lesbarkeit) vereinfachen und an

die IP-Adressen angleichen (`--mac`)

- jeden virtuellen Host mit dem Switch über ein virtuelles Netzwerkkabel verbinden
- den OpenFlow-Switch so konfigurieren, so dass er sich mit einem Remote-Controller verbindet (`--controller remote`).

Als eine kleine Einführung in Mininet, stellen wir zunächst die wichtigsten Befehle, die alle in der Mininet-Konsole eingegeben werden müssen, vor:

Liste der verfügbaren Befehle ausgeben:

```
mininet> help
```

Liste der verfügbaren Knoten ausgeben:

```
mininet> nodes
```

Um einzelne Befehle auf einem Host auszuführen, fügt man vor den Befehl den Namen des Hosts ein. Ein Beispiel um die IP eines virtuellen Hosts anzuschauen:

```
mininet> h2 ifconfig
```

Eine Alternative wäre ein xterm-Terminal für einen odere mehrere virtuelle Hosts zu starten. Dies ermöglicht interaktive Eingabe und zeigt Debugging Informationen an:

```
mininet> xterm h2 h3
```

Sollte Mininet nicht richtig funktionieren oder abgestürzt sein, verwendet den `clear`-Befehl:

```
$ mn -c
```

und startet Mininet anschließend neu.

Weitere Informationen und Befehle zu Mininet findet ihr im Walkthrough: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/MininetWalkthrough

3.3 Ping-Test

Nun gehen wir zurück zur Mininet-Konsole und versuchen den Host `h3` vom Host `h2` aus zu pingen:

```
mininet> h1 ping -c3 h2
```

Bemerkung: Der Hostname `h3` wird automatisch auf die IP-Adresse 10.0.0.3 aufgelöst, sofern der Befehl in der Mininet Konsole ausgeführt wird. `-c3` bestimmt die Anzahl der Pings, in diesem Fall 3.

Frage: Bekommt `h2` eine Antwort? Warum? Warum nicht?

3.4 Flow-Tabelle füllen

Wie ihr sehen konntet ist die Flow-Tabelle des Switches leer und es ist kein Controller zum Switch verbunden. Dies verursacht einen Fehler, da der Switch schlichtweg nicht weiß, wohin er die Pakete weiterleiten soll.

Damit der Switch die Pakete verarbeiten kann werden wir nun manuell Flows in die Flow-Tabelle des Switches einfügen:

```
$ ovs-ofctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
$ ovs-ofctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
```

Diese Flow-Einträge sorgen dafür, dass Pakete von Port 1 an den Port 2 weitergeleitet werden und umgekehrt. Um zu verifizieren, dass die Einträge auch wirklich in die Flow-Tabelle geschrieben wurde:

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

Wiederholen des Ping-Tests:

```
mininet> h1 ping -c3 h2
```

Frage: Bekommt `h1` nun eine Antwort? Kontrolliert die Flow-Tabelle und schaut euch die Statistiken für jeden Eintrag an. Entspricht dies den Erwartungen?

Bemerkung: Sollten keine Antworten sichtbar sein, ist es möglich, dass die Flow-Einträge schon verfallen sind bevor wir den nächsten Ping-Test durchgeführt haben. Mit dem Befehl `ovs-ofctl dump-flows s1` kann man die `idle_timeout`-Zeit für jeden Eintrag einsehen. Ihr könnt die Einträge erneut anlegen oder die Flow-Einträge mit einer längeren Timeout-Zeit mittels folgendem Befehl versehen:

```
$ ovs-ofctl add-flow tcp:127.0.0.1:6634 \
        in_port=1,idle_timeout=120,actions=output:2
```

3.5 Netzwerkverkehr beobachten mit Wireshark

Wireshark ist für Netzwerktechniker ein sehr nützliches Tool, um den Datenverkehr in Netzwerken zu beobachten. Ihr werdet es im Folgenden zur Analyse des OpenFlow-Protokolls nutzen und es kann euch eine große Hilfe bei der Fehlersuche sein.

Über das Menü *Capture* → *Interfaces* könnt ihr auswählen, von welchem (virtuellen oder physikalischen) Netzwerkinterface ihr den Netzwerkverkehr mitschneiden wollt. Benutzt den *Start*-Knopf direkt neben der Schnittstelle `lo` (loopback). Es sollten nun gelegentlich Pakete im Wireshark-Fenster zu sehen sein.

Aus dem gesamten Datenverkehr filtert ihr die OpenFlow Kontrollpakete heraus, indem ihr einen Filter auf das OpenFlow-Protokoll *openflow_v1* im Eingabefeld setzt. Anschließend aktiviert ihr den Filter für den aufgezeichneten Datenverkehr mit einem Klick auf *Apply*.

Startet nun den POX-Controller im Terminal:

```
$ ~/pox/pox.py forwarding.tutorial_l2_hub
```

Mit diesem Befehl startet der Controller, der sich wie ein Hub verhält, und demnach keine Flow-Einträge anlegt. Der Controller ist bereit nachdem folgende Ausgabe erscheint:

```
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Im Verlauf dessen sollten von Wireshark (start in einem Terminal mit `wireshark &`) einige Pakete angezeigt werden, beginnend mit dem *Hello exchange* des Controllers und Switches. Mit einem Klick auf ein Paket erscheinen unter der Liste von Paketen Details zu dessen Inhalt und Bedeutung. Klickt auf die Dreiecke vor den einzelnen Einträgen um die Inhalte aufzuklappen, insbesondere interessant der Layer *OpenFlow Protocol*. Schaut euch die empfangenen Pakete genauer an.

Frage: Welche Pakete sind zu sehen und welche Bedeutung haben sie?

Da von Mininet alle Daten über *localhost* verschickt werden ist es in Wireshark nicht offensichtlich, woher die Pakete stammen. Der Controller nutzt fest den OpenFlow-Port 6633, während die Pakete des Switches von einem anderen, dynamischen Port

stammen.

Im Folgenden sollt ihr den Datenverkehr betrachten, der bei einem *Ping* erzeugt wird. Bevor ihr dies tut, erweitert den Wireshark-Filter, um die *echo-request* bzw. *echo-reply*-Nachrichten, die zur Aufrechterhaltung der Verbindung zwischen Controller und Switch dienen, zu ignorieren. Verändert den Filter von oben wie folgt:

```
openflow_v1 &&
!((openflow_1_0.type == 2) or (openflow_1_0.type == 3))
```

Um zu sehen, welche OpenFlow-Nachrichten erzeugt werden, sendet einen einzelnen Ping von `h1` an `h2`:

```
mininet> h1 ping -c1 h2
```

In Wireshark sollte nun eine Reihe von neuen Paketen erscheinen. Dokumentiert und erklärt diesen Datenverkehr!

Führt den Ping erneut aus. Benötigt der Ping die gleiche Zeit wie zuvor, solltet ihr den Ping erneut ausführen - eventuell sind die Flow-Einträge abgelaufen/gelöscht.

Dieser Datenverkehr ist ein Beispiel für den *reaktiven Modus* von OpenFlow, wenn Flow-Einträge als Antwort auf individuelle Pakete angelegt werden.

Eine Alternative ist der *proaktive Modus*, bei dem die Flow-Einträge gezielt vor dem (erwarteten) Eintreffen eines Pakets angelegt werden, um Verzögerungen durch die Benachrichtigung des Controllers und der Eintragung des Flows in die Flow-Tabelle vorzubeugen.

3.6 Controller-Benchmark mit iperf

`iperf` ist ein Kommandozeilen-Programm, um die Geschwindigkeit zwischen zwei Rechnern im lokalen Netz oder Internet zu testen.

Zunächst führt ihr einen Benchmark mit dem POX-Controller durch. Anschließend vergleicht ihr die Ergebnisse des mitgelieferten Hubs mit denen des selbstgeschriebenen, auf Flow-Einträgen basierenden, Switches.

In der Mininet-Console führen wir Folgendes aus:

```
mininet> iperf
```

Dieser Mininet-Befehl führt in dieser Umgebung gleich mehrere Schritte automa-

tisiert aus: Es startet einen `iperf`-TCP-Server auf einem virtuellen Rechner sowie einen `iperf`-Client auf einem zweiten virtuellen Rechner. Nachdem die Verbindung vom Client zum Server hergestellt ist, werden Pakete zwischen Server und Client ausgetauscht und die Ergebnisse ausgegeben.

$admin-hiwi will nun einen OpenFlow Software Switch im Linux User-Space implementieren. Wie würde sich das zuvor beobachtete Ergebnis verändern? Warum?

4 Programmieren des intelligenten Switches

Im folgenden Abschnitt werdet ihr endlich euren eigenen Switch programmieren. Hierzu stellen wir euch den Code des Standard-Hub-Controllers zur Verfügung. Nachdem ihr euch mit der Funktionsweise dieses stupiden Netzelements vertraut gemacht habt sollt ihr diesen so erweitern, dass er sich wie ein intelligenter Switch verhält. Um dies zu bewerkstelligen muss sich der Switch jedes Paket anschauen und zunächst die Zuordnung zwischen Ausgangsport und MAC-Adresse lernen. Ist zu einer Ziel-MAC-Adresse anschließend der Port bekannt, so kann das Paket direkt an den gegeben Port verschickt - andernfalls wird das Paket an alle Ports des Switches verschickt.

Anschließend erweitert ihr euren intelligenten Switch zu einem auf Flow-Einträgen basierenden Switch. Dieser legt einen Flow-Eintrag an, anstatt das Paket direkt an den Port zu verschicken. Nach der Installation des Flow-Eintrags wird das Paket nicht mehr an den Controller geschickt.

Wir werden hierzu die Ryu-Plattform und die Programmiersprache Python benutzen.

4.1 Ryu-Controller mit Python

Ryu ist eine alternative Softwareimplementierung eines OpenFlow-Controllers. Sie bietet Kontrolle auf den Netzwerkverkehr auf Flow-Level. Das bedeutet, dass die Wege der Pakete durch das Netzwerk können pro Flow einzeln festgelegt werden.

Vor dem Start von Ryu müsst ihr sicherstellen, dass der Referenz-Controller nicht mehr läuft und unser Ryu-Controller stattdessen auf Port 6633 lauschen kann:

```
$ sudo killall controller
```

Anschließend startet ihr Mininet neu, wobei wieder der Open vSwitch (`ovsk`) verwendet werden soll.

In einem zweiten Terminal wechselt ihr dann in das Verzeichnis des Ryu-Source-Codes (`~/ryu/`) und startet den Basis-Hub-Controller, der in dieser Übung als Ausgangspunkt dient und in Python implementiert ist:

```
$ PYTHONPATH=. ./bin/ryu-manager --verbose ryu/app/hwp.py
```

Dieser Befehl startet eine Beispiel-Anwendung in der Ryu-Plattform. Der Parameter `--verbose` bewirkt die Ausgabe von Debug-Informationen. Nach dem Start wartet der Controller auf Port 6633 auf eine eingehende Verbindung von einem Switch.

Wartet also, bis der OpenFlow Switch selbstständig mit unserem Controller verbindet. Das Signal für eine erfolgreiche Verbindung ist folgende Ryu-Nachricht:

```
connected socket:<...> address:('127.0.0.1', 33237)
hello ev <...>
move onto config mode
switch features ev version: ...
move onto main mod
```

4.2 Verifizieren des Verhaltens mittels tcpdump

Nun stellt ihr sicher, dass alle Rechner sich gegenseitig pingen können und jeder Rechner den gleichen Datenverkehr sieht - das typische Verhalten eines Netzwerk-Hubs. Dafür öffnet ihr `xterm`-Terminals für jeden Rechner und schaut euch den Datenverkehr an. In der Mininet-Konsole startet ihr drei `xterm`-Terminals mit:

```
mininet> xterm h2 h3 h4
```

In den Terminals für die Rechner `h3` und `h4` startet ihr `tcpdump`:

```
xterm@h3$ sudo tcpdump -XX -n -i h3-eth0
```

und

```
xterm@h4$ sudo tcpdump -XX -n -i h4-eth0
```

Im `xterm`-Terminal für `h2` schickt ihr einen Ping los:

```
xterm@h2$ ping -c1 10.0.0.3
```

Die Ping-Pakete laufen durch den Controller und werden auf allen Ports weitergeschickt - mit Ausnahme des Ports, auf dem die Pakete ankommen sind. In der tcpdump-Ausgabe für `h3` und `h4` sollten identische ARP- und ICMP-Pakete zu sehen sein.

Nun schauen wir uns an, was passiert wenn ein nicht-existierenden Rechner gepingt wird:

```
h2$ ping -c1 10.0.0.5
```

Ihr solltet nun drei nicht beantwortete ARP-Anfragen auf `h3` und `h4` sehen. Sollte dies später der Fall sein, ist das ein Zeichen für verlorene Pakete.

Alle `xterm`-Terminals können nun geschlossen werden.

4.3 Benchmark des HUB-Controllers

Nun werdet ihr einen Benchmark mit dem Hub-Controller durchführen. Zunächst stellt ihr sicher, dass alle Rechner auf Ping-Nachrichten antworten. Mininet sowie der Ryu-Controller müssen gestartet werden. In der Mininet-Konsole führt ihr folgenden Befehl aus:

```
mininet> pingall
```

Sollte dies erfolgreich sein, startet ihr den automatisierten iperf-Benchmark in der Mininet-Konsole.

Frage: Vergleicht das Ergebnis mit den Ergebnissen zu dem Referenz-Controller zuvor. Was fällt euch auf?

4.4 Programmierung des Controllers

Stoppe in den laufenden Controller mit `Strg+C`. Die zu editierende Datei befindet sich in
`~/ryu/ryu/app/hwp.py`.

Die meisten Anpassungen werdet ihr in der Funktion `packet_in_handler()` durchführen. Wie ihr sehen könnt, ist hier bisher kaum Code zu sehen. Dieser Code ist ein minimales Beispiel für eine Ryu-Application und bildet die Funktionalität eines Standard-Hubs ab.

Jedes mal wenn ihr Anpassungen der Datei speichert, stellt sicher, dass ihr den Ryu-Controller neu startet. Um den Hub oder das Verhalten des intelligenten Switches zu verifizieren, nutzt die Ping-Funktion. Rechner, die kein Ziel sind, sollten keinen

Datenverkehr via `tcpdump` mehr anzeigen, nachdem einmal die initiale Broadcast-ARP-Nachricht beantwortet wurde.

4.5 Einführung in Python

In diesem Abschnitt werdet ihr für dieses Tutorial alle notwendigen Befehle von Python erklärt bekommen, jedoch raten wir euch einen Blick in das *Python Tutorial* (http://docs.python.org/tutorial/) zu werfen, um einen besseren Überblick zu haben.

Python:

- ist eine dynamisch interpretierte Sprache, bei der Kompilieren nicht erforderlich ist. Ihr müsst den Code abändern und das Script (bzw. in unserem Fall den Controller) nur neu starten.
- benutzt Einrückung, um den Code zu trennen. Die aus anderen Programmiersprachen gewohnten geschweiften Klammern sind in Python nicht erforderlich. Achtet auf genaue Einrückung und nutzt 4 Leerzeichen anstelle von Tabs.
- ist dynamisch typisiert. Man muss den Typ einer Variablen nicht deklarieren, dies wird automatisch gemacht.
- hat von Haus aus Tabellen, genannt *dictionaries*, und Vektoren, genannt *lists*.
- ist objektorientiert und introspektiv.

Ein Dictionary zu initialisieren:

```
mactable = {}
```

Diesem ein Element hinzufügen:

```
mactable[0x123] = 2
```

Prüfen ob ein bestimmtes Element vorhanden ist:

```
if 0x123 in mactable:
    print "element 2 in mactable"
```

Um Debug-Nachrichten in Ryu auszugeben:

```
self.logger.debug("saw new MAC!")
```

Um Fehler-Nachrichten in Ryu auszugeben:

```
self.logger.error("unexpected packet causing system meltdown!")
```

Um Eigenschaften (*member*-Variablen) und Funktionen eines Objekts auszugeben:

```
print dir(object)
```

Kommentare:

```
# Prepend comments with a #; no // or /**/
```

4.6 Die Ryu-API

Die folgenden Abschnitte stellen euch Details der Ryu-APIs vor, die nützlich für das Tutorial sind.

OpenFlow-Nachrichten mit Ryu senden:

```
datapath.send_msg( ... ) # send a packet out a port
self.add_flow( ... ) # install a flow
```

Die Funktion `send_msg()` ist Bestandteil der Ryu-API und in `~/ryu/ryu/controller/controller.py` zu finden.

```
 def send_msg(self, msg):
```

sendet ein OpenFlow-Paket an den Datapath. Das Object `msg` besitzt folgende Parameter:

datapath Datapath an den das Paket geschickt werden soll

buffer_id id des Puffers an den geschickt wird

inport Datapath-Port, der als Source markiert wird (Default: Controller Port)

actions Liste der auszuführenden Aktionen oder Datapath Port an den verschickt werden soll

packet Daten, die im OpenFlow Paket eingebettet werden

Im Folgenden wird ein Beispiel aus dem Code `tutorial.py` vorgestellt, das zeigt, wie man diese Funktion benutzt:

```
actions =
[datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
 out = datapath.ofproto_parser.OFPPacketOut(
datapath=datapath, buffer_id=msg.buffer_id,
in_port=msg.in_port,actions=actions, data=data)
```

Dieser Aufruf schickt das im Switch gespeicherte Paket (mit der gegeben `bufid`) an alle Ports ausgenommen dem Input-Port. Ersetzt `openflow.OFPP_FLOOD` durch eine Port-Nummer, um ein Paket an einen speziellen Port zu verschicken.

```
def add_flow(self, datapath, in_port, dst, actions):
```

Installiert einen Flow-Eintrag im Datapath und hat folgende Parameter:

datapath Datapath, in dem der Flow installiert werden soll.

in_port Der eingehende Port, damit das Paket im Fall von Flooding (`OFPP_FLOOD`) nicht am eingehenden Port wieder gesendet wird.

dst Beschreibt die Kritierien für das Matching des Flows.

actions eine Liste, in der jeder 2-elementige Eintrag eine *Action* repräsentiert. Element 0 der Action sollte ein `ofp_action_type` sein und Element 1 das Aktionsargument (falls erforderlich). Für `OFPAT_OUTPUT` sollte ein weiterer 2-elementiger Eintrag mit `max_len` als erstes Element und die `port_no` (Portnummer) als zweites Element benutzt werden.

Für weitere Details zum Format bitte die Ryu API in `ryu/ryu/controller /controller.py` beachten! Informationen über OpenFlow-Konstanten finden sich in `types/enums/structs` bzw. `~/openflow/include/openflow/openflow.h`.

4.7 Pakete parsen mit den Ryu Packet Libraries

Die *Ryu Packet Parsing Bibliotheken* werden automatisch ausgeführt, wobei die Pakete analysiert und aufbereitet werden, so dass alle Protokollfelder in Python zur Verfügung stehen.

Die Parsing-Bibliotheken findet ihr in `~/ryu/ryu/lib/packet/`. Jedes Protokoll hat dort seine eigene Datei.

Für die erste Aufgabe werdet ihr nur den Zugriff auf die Felder Ethernet-Quelle (`src`)

und -Ziel (`dst`) benötigen. Um auf die String-Felder zu zugreifen benutzt die in Python übliche Punkt-Notation:

```
packet.src
```

Eine ausführliche Dokumentation findet sich unter http://ryu.readthedocs.org/en/latest/library_packet.html

Um alle Variablen eines geparsten Pakets anzuzeigen:

```
print dir(packet)
```

Dies liefert folgende Ausgabe:

```
['_MIN_LEN', '_PACK_STR', '_TYPE', '_TYPES',
'__abstractmethods__', '__class__', '__delattr__', '__dict__',
'__div__', '__doc__', '__format__', '__getattribute__',
'__hash__', '__init__', '__len__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'_abc_cache', '_abc_negative_cache',
'_abc_negative_cache_version', '_abc_registry',
 '_class_prefixes', '_class_suffixes', '_decode_value',
'_encode_value', '_get_decoder', '_get_default_decoder',
'_get_default_encoder', '_get_encoder', '_get_type',
'_is_class', '_restore_args', 'cls_from_jsondict_key',
'dst', 'ethertype', 'from_jsondict', 'get_packet_type',
'obj_from_jsondict', 'parser', 'protocol_name',
'register_packet_type', 'serialize', 'set_classes',
'src', 'stringify_attrs', 'to_jsondict']
```

Viele Eigenschaften des geparsten Pakets können ignoriert werden, jedoch ist diese Ausgabe ein schneller Weg, um einen Ausflug in die Dokumentation zu umgehen.

4.8 Controller testen

Bevor ihr den Controller-basierenden Netzwerk-Switch testet, stellt zunächst sicher, das alle Pakete den Controller erreichen sowie nur Broadcast-Pakete (z.B. ARP-Broadcasts) und Pakete mit unbekanntem Ziel (wie das erste gesendete Paket eines Flows) an alle Switch-Ports geschickt werden. Dies könnt ihr mit `tcdump` auf jedem Rechner (mittels `xterm`-Terminal) überprüfen.

Wenn der Switch sich nicht mehr wie der Standard-HUB verhält wird eine Regel für

jeden Flow installiert. Sofern Quell- und Ziel-Port bekannt sind könnt ihr mit `dpctl` die Flow-Einträge überprüfen und die Statistiken zu den Einträgen einsehen. Wenn der Ping-Befehl wesentlich schneller ausgeführt wird, ist es ein Zeichen dafür, das die Pakete nicht mehr an den Controller geschickt werden. Eine weitere Möglichkeit, um den Controller zu testen ist es, `iperf` in der Mininet-Konsole zu benutzen und zu kontrollieren, dass keine neuen OpenFlow-Nachrichten verschickt werden.

4.9 Unterstützung mehrerer Switches

Bis hierher musste euer Controller nur einen Switch unterstützen und hat nur eine MAC-Tabelle. In diesem Abschnitt werdet ihr den Controller erweitern, so dass dieser mehrere Switches verwalten kann.

Starte Mininet mit einer anderen Topologie neu:

```
$ sudo mn --topo linear --switch ovsk --controller remote
```

Die erzeugte Topologie sieht wie folgender Abbildung dargestellt aus.

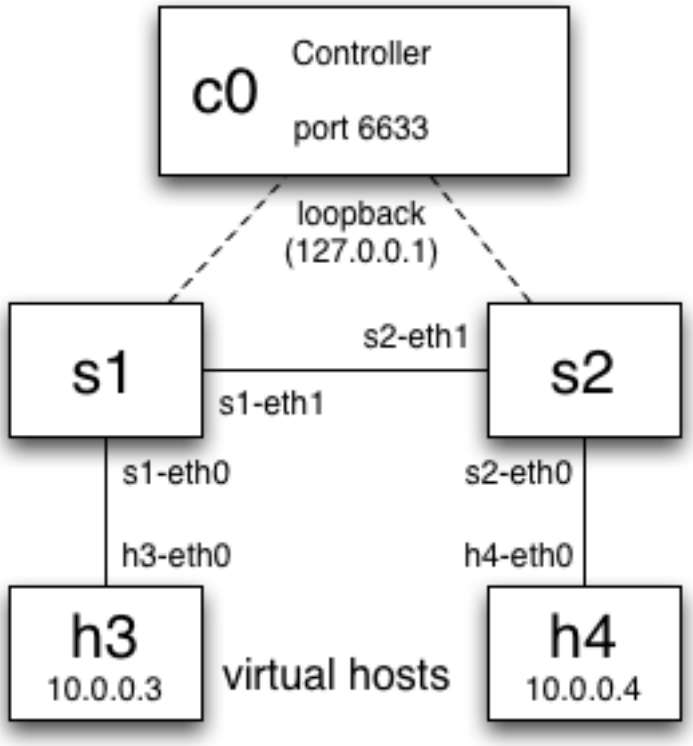

Abbildung 2: Multiple switches

Wie zu sehen ist erzeugt der Befehl eine 2-Switch-Topologie mit jeweils einem Host verbunden mit einem Switch.

Nun modifiziert ihr den Switch so, dass für jeden *Datapath In-Port* (DPIP) eine *MAC-to-port*-Tabelle existiert. Diese Strategie funktioniert nur für *Spanning Tree*-Netzwerke (Spanning Tree Netzwerke besitzen redundante Verbindungen, die im Fall eines Fehlers aktiviert werden) und die Verzögerung, um einen neuen Pfad zu erstellen, ist proportional zu der Anzahl der Switches zwischen den Rechnern. Andere Module von Ryu, die Routing beinhalten, agieren bei dieser Problemstellung sinnvoller. Ryu Routing beinhaltet eine *all-pairs shortest path*-Datenstruktur und legt sofort alle benötigen Flow-Einträge in den unterschiedlichen Switches zu bekannten Quell- und Ziel-Verbindungen an.

Nach der Modifizierung des Controllers verifiziert ihr die neuen Funktionalität in der Mininet-Konsole mit:

```
> pingall
```

4.10 Benchmark des eigenen Controllers

`cbench` testet den OpenFlow-Controller, indem das Programm *Packet-In*-Ereignisse für jeden neuen Flow erzeugt. Cbench emuliert eine Reihe von Switches, die sich zum Controller verbinden. Zusätzlich wird auch der Controller über eingehende Pakete benachrichtigt. Dabei betrachtet cbench Flow-Modifikationen (Flow-Mods), die zum Switch versendet werden.

Um euren auf Flows basierenden Controller zu benchmarken schließt ihr zunächst Mininet sowie den noch laufenden Controller. Letzteren startet ihr ohne die *Verbose*-Option neu, da nun ein Performance-Test erfolgen soll und die korrekte Funktionsweise bereits vorher gezeigt wurde.

```
$ PYTHONPATH=. ./bin/ryu-manager --verbose ryu/app/hwp.py
```

Startet `cbench` in einem weiteren Terminal:

```
$ cd ~/oflops/cbench
$ ./cbench -l5
```

Dieser Befehl durchläuft 5 Schleifen mit jeweils 16 virtuellen Switches und gibt die Anzahl der *Flow-Setups/Sekunde* aus. Der angezeigte Wert ist die Zahl neuer Flows sowie die messbare Latenz. `Cbench` verschickt Test-Nachrichten standardmäßig nacheinander.

`Cbench` kann wie folgt im *Durchsatzmodus* betrieben werden, um parallel zu senden:

```
$ ./cbench -l5 -t
```

Um diese Werte mit dem POX-Controller vergleichen zu können, schließt ihr den Controller und startet den POX-Controller:

```
$ ~/pox/pox.py forwarding.tutorial_l2_hub
```

Startet cbench erneut:

```
$ ./cbench -l5 t
```

Frage: Was ist schneller?

Um den integrierten POX-C++-Switch zu starten führt folgendes aus:

```
$ ~/pox/pox.py forwarding.l2_learning
```

Vergleicht die Testläufe mit `cbench` mit und ohne den Parameter `-t`.

Frage: Was fällt im Vergleich auf?

Frage: Welche Anpassungen sind notwendig, um den Switch in einen Router umzuwandeln?