Title Placeholder

Subtitle Placeholer

Networks and Distributed System
Group 1025 - 10th semester

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

Department of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø
http://es.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Title Placeholder
- *Subtitle Placeholer*

Theme:
Master's thesis

Project Period:
10^{th} semester
Networks and Distributed System
February – June, 2014

Project Group:
1025

Participants:
Martí Boada Navarro

Supervisors:
Jimmy Jessen Nielsen
Andrea Fabio Cattoni

Copies: 3

Page Numbers: 30

Date of Completion: May 14, 2014

Abstract:

Nowadays networks use different kind of mechanisms in order to give priority to a certain sort of traffic to process the packets in a different way depending on their application. The most common way to achieve this is using Differentiated Service, marking the packets depending on their application (giving higher preference to those that are more important or time sensitive such as Voice over IP, or Video on Demand). However, determining resource allocation per class of service must be done with knowledge about traffic demands for the various traffic classes, keeping a fixed amount of bandwidth for each class, which results in a poor utilization of resources.

In the last years, a new networking approach called Software-Defined Networking (SDN) is emerging fast. This approach is based on the separation of data and control planes. Such approach allows the network administrator to have a more dynamic control of the network behaviour. The purpose of this project is to analyse the possibilities that SDN provides to develop a more efficient resources allocation along the network.

Preface

This is a report of a student project conducted on the 4^{th} semester of Networks and Distributed Systems, at Aalborg University. This project serves as a Master Thesis as the requirements of the master's programme dictate. This is applied in the comparison of a designed application used for Cognitive Networks concepts over a Software-Defined Network (SDN).

In this project Floodlight is used as an SDN controller, and the network is simulated using mininet software.

Aalborg University, May 14, 2014

Martí Boada Navarro
<mboada14@student.aau.dk>

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem formulation . . . 2
- 1.3 Related Work . . . 3

2 Pre-analysis **5**
- 2.1 Traffic Engineering . . . 5
- 2.2 Software-Defined Networking . . . 5
  - 2.2.1 Advantages . . . 6
  - 2.2.2 Applications . . . 7
- 2.3 OpenFlow . . . 7
  - 2.3.1 Flow Tables . . . 8
  - 2.3.2 Messages . . . 11

3 Scenario Description **13**
- 3.1 Mininet Network Emulator . . . 13
- 3.2 Floodlight Controller . . . 14
- 3.3 Topology . . . 14

4 Algorithm Description **17**

5 Emulation and Evaluation **19**

6 Conclusions and Future Work **21**

Bibliography **23**

A Setting up Mininet and Floodlight **25**

B Mininet Topology Script **29**

1 Introduction

This introduction chapter tries to give a general view of the existing problems in the current networks, and how new emergent technologies can help to improve the network possibilities and capabilities in order to achieve a more flexible approach and adapt to todays needs.

1.1 Motivation

In the last 20 years networks requirements have been changing constantly, the amount of traffic have been increasing exponentially and more demanding end-to-end goals are needed. However, the networks architectures have been unchanged, increasing the complexity and hindering its configuration. In order to adapt to the new needs, new network paradigms such as Software-Defined Networking (SDN), cognitive networks or automatic networks are emerging fast due the interests of carriers and ISPs.

The first things to analyse are the problems of the existing networks. They have become a barrier to creating new, innovative services, and an even larger barrier to the continued growth of the Internet.

Analysing the data plane in the ongoing networks architecture we find well defined layers for different purposes, making them autonomous. For instance the physical layer (fibre, copper, radio...) is independent of the link layer (ATM, X.25, PPP...), the transport layer (TCP, UDP...) or the application layer (HTTP, FTP, telnet...), what allows the evolution of each of them independently. Thanks to the fact of dividing the problem in tractable pieces, networks have been able to evolve, increasing many magnitude changes in terms of speed, scale or diversity of uses.

On the other hand, there is the control plane which, unlike the data plane, has no abstraction layers at all. Within the control plane there are several protocols to configure the network elements, such as Multiprotocol Label Switching (MPLS)[1] or NETCONF[2]. There are also a bunch of routing protocols e.g. Routing Information Protocol (RIP)[3], Enhanced Interior Gateway Routing Protocol (EIGRP)[4] or Shortest Path Bridging (SPB)[5].

[1] MPLS architecture RFC 3031
[2] Network Configuration Protocol (NETCONF) RFC 6241
[3] RIP Version 2 RFC 2453
[4] CISCO Enhanced Interior Gateway Routing Protocol (EIGRP)
[5] SPB IEEE Std 802.1aq

Protocols tend to be defined in isolation, however, with each solving a specific problem and without the benefit of any fundamental abstractions. This has resulted in one of the primary limitations of today's networks: complexity. For example, to add or move any device, IT must touch multiple switches, routers, firewalls, Web authentication portals, etc. and update ACLs, VLANs, Quality of Services (QoS), and other protocol-based mechanisms using device-level management tools. In addition, network topology, vendor switch model, and software version all must be taken into account. Due to this complexity, today's networks are relatively static as IT seeks to minimise the risk of service disruption.

In order to deal with all that mess, hundreds of network monitoring and management tools[6] have appeared trying to help the networks managers to have a control of their networks.

At the same time that technology advances, the communications requirements also evolve. The claims that nowadays are needed in the different services that users demand, such as VoIP or streaming video in High Quality, where inconceivable when the architecture was designed.

All those factors have lead to an over-provisioning of the networks, increasing the cost, and wasting resources due the difficulty to optimise the control and adapt the physical resources available to the requirements of every instant. So it is time to take a step forward and evolve to a more optimal architecture, easy to manage, evolve and understand.

Here is where the SDN approach (explained in section 2.2) comes to play. Having a clear abstraction layers and a centralised control plane, it's much easier to make a more efficient and dynamic management and control of the network, since we have a global overview of the network, and control over its entirety.

The motivation of this Master's Thesis is to take advantage of the new centralised networking approach of SDN to develop a load balancing algorithm that adapt the route of each flow depending on the current state of the network in order to achieve a better resource allocation, reducing the overall cost and adapting its behaviour to the traffic growth.

1.2 Problem formulation

Data traffic in networks have been increasing exponentially since the beginning of networking at the same time that new kind of services and connection requirements appear. Those factors have led to a more complex networks that cannot satisfy the needs of carriers nor users.

As explained in this introduction chapter, the current networks have several limitations to adapt to end-to-end goals requirements for the different sort of services without wasting the existing resources.

Determining resource allocation per class of service must be done with knowledge about traffic demands for the various traffic classes, keeping a fixed amount of bandwidth for

[6] Standford Network Monitoring and Management Tools list.

each class, which results in a poor utilisation of resources. However, the traffic that the network has to carry change constantly in an unpredictable way.

Nowadays methodologies are static, or need specific network equipment, which leads to a non-scalable and expensive systems. A new centralised control plane paradigm approach, with a global view of every element in the network, can improve significantly the management of the route for every single flow, taking in account the characteristics of those in terms of bandwidth utilisation or other parameters, to accomplish a much better utilisation of the resources.

The aim of this project is to analyse the possibilities that Software-Defined Networking bring us to develop an application able to sense the state of the network and adapt its behaviour in order to achieve a better performance and better resource utilisation of the network.

However, resource utilisation is not the unique important factor. In order to guarantee a certain quality for the carried traffic, It's crucial to keep some parameters, such as delay or jitter, within a bounded limits.

Thus, this project tries to assess the possibilities brought by SDN to manage the route per each flow in order to take the maximum advantage of the available bandwidth in the network, and evaluate the following points:

1. Improvement in terms of resources utilisation.
2. Impact of rerouting the flows.
    - Delay
    - Jitter
    - Packet losses

1.3 Related Work

There is plenty of papers about traffic engineering [3] trying to adapt the routing rules to the network conditions with the aim of taking the maxim profit of the available resources, and avoid traffic congestion. One of the ways to readjust the networks is using OSPF, as described in [5], by adjusting the state of the links thus the traffic can be adjusted to the current conditions .In [4] they compare OSPF against MPLS. trying several ways to adjust the weights setting to achieve a performance closer to the optimal in MPLS routing, but getting as a result that MPLS can achieve better performance. However, they also mention some advantages of OSPF in front of MPLS, since it is much simpler due that the routing is completely determined by one weight of each link, what avoids making decision per each source-destination pair. Also, if a link fails, the weights on the remaining links immediately determines the new routing.

One of the features of MPLS is its flexibility to adapt

2 Pre-analysis

In order to investigate how to answer the initial problem stated in section 1.2, some background information is needed. This chapters gives a general overview about the concept of Software-Defined Networking, Openflow, and other related information that must be taken in account in order to develop and understand this project..

2.1 Traffic Engineering

Even though the project is focused on SDN, I think that I should include some introduction about Traffic Engineering (mainly with MPLS) since it's the starting point to understand what I want to accomplish with the algorithm.

2.2 Software-Defined Networking

As it is explained in section 1.1, the main problem of the current networks is that the control plane has no abstractions. That means that there is no modularity, which leads to limited functionality, since protocols with different proposals (such as routing, isolation or traffic engineering) coexist in the same layer.

One of the tendencies that seems to have more power for the next generation networking is Software-Defined Networking (SDN). One of the reasons to believe that SDN will be *the one* is that some big companies, such as Google, are already using it.[6][2].

SDN is a new network architecture that allows be programmed as if it were a computer. It provides an abstraction of the forwarding function decoupling the data plane from control plane, which gives freedom to manage different topologies, protocols without many restrictions from the physical layer.

SDN was designed to change that paradigm, dividing the control plane in three main layers: the forwarding model, the Network Operating System (NOS) and the control program.

Forwarding model: composed by the Network Elements (i.e. switches) and a dedicated communicated channel from each NE with the NOS which uses a standard way of defining the forwarding state.

Network Operation System: piece of software running in servers (controllers) which provides information about the current state of the network such as the topology or the state of each port.

Control program: express the operator goals, and compute the forwarding state of each NE to accomplish those goals.

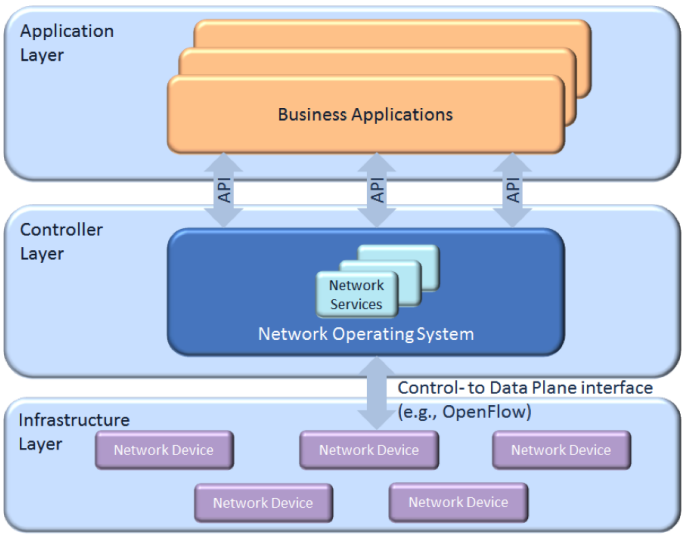

Figure 2.1: SDN architecture (image from Open Networking Foundation)

2.2.1 Advantages

SDN tries to improve the current networks. Bellow there is a list of the main advantages of the SDN paradigm.

OPEX reduction: centralised control helps to eliminate manual interaction with the hardware, improving the uptime of the network.

CAPEX reduction: separating the data plane of the control plane brings to a more simple hardware and increases the possibility of more competence between hardware manufacturers, since the devices don't depends on the proprietary software.

Agility: since the control layer can interact constantly with the infrastructure layer, the behaviour of the network can adapt fast to changes like failures or new traffic patterns.

Flexibility: having a separated abstraction for the control program allows to express different operator goals, adapting to a specific objectives. Operators can implement

features in software they control, rather than having to wait for a vendor to add it in their proprietary products.

2.2.2 Applications

To give an idea of how huge SDN is, the list below mentioned some of the applications which is related to.

1. **Appliance Virtualization**
   - Firewalls, Load balancers, Content distribution, Gateways
2. **Service Assurance**
   - Content-specific traffic routing for optimal QoE, Congestion control based on network conditions, Dynamic policy-based traffic engineering
3. **Service Differentiation**
   - Value-add service features, Bandwidth-on-demand features, BYOD across multiple networks, Service insertion/changing
4. **Service Velocity**
   - Virtual edge, Distributed app testing environments, Application development workflows
5. **'Traditional' Control Plane**
   - Network discovery, Path computation, Optimisation & maintenance, Protection & restoration
6. **Network Virtualization**
   - Virtual network control on shared infrastructure, Multi-tenant network automation & API
7. **Application Enhancement**
   - Specific SDN application, Reserved bandwidth for application needs, Geo-distributed applications, Intelligent network responses to app needs.

2.3 OpenFlow

In the current networks (i.e. non SDN), the routers and switches makes the forwarding (data path) and routing (control path) decisions by itself. In an OpenFlow switch, those decisions are divided between the network element (i.e. the switch) and the controller.

Such division keeps the data path on the network element, while high-level routing are moved to the NOS.

OpenFlow is the protocol used to communicate the NOS (controller) with all the Network Elements (NE). Is an open standard that provides a standardised hook to allow researchers to run experiments, without requiring vendors to expose the internal workings of their network devices. OpenFlow is currently being implemented by major vendors, with OpenFlow-enabled switches now commercially available.

OpenFlow is the most common protocol used in SDN networks, and is often confused with the SDN concept itself, but they are different things. While SDN is the architecture dividing the layers, OpenFlow is just a protocol proposed to convey the messages from the control layer to the network elements. There is a bunch of OpenFlow based projects[1], including several controllers, virtualised switches and testing applications.

2.3.1 Flow Tables

Each NE has a flow table which contains a set of flow entries. Each flow entry is written by the controller, which computes the routing decisions sending the corresponding rules to each switch. Those flow entries consist of a determinate actions (e.g forward packet through specified port) to do with packets which headers contains a particular values (e.g. packets with a specific destination).

If no match is found for an incoming packet, this is forwarded to the controller over a secure channel, and it is the controller who manage the flow entries of the NEs adding or removing flow entries in order to accomplish the stablished rules.

Each flow entry consists of three fields: header fields, counters and actions. The process by which each incoming packet is matched against is explained in the next points.

Header Fields

Aiming to determine each flow, there are several fields that can be matched. The header fields that incoming packets are compared against are the following ones:

- Ingress Port
- Ethernet Source
- Ethernet Destination
- Ethernet Type

[1] List of OpenFlow Software Projects.

- VLAN ID
- VLAN Priority
- IP Source
- IP Destination
- IP Protocol
- IP ToS bits
- TCP/UDP Source Port
- TCP/UDP Destination Port

Counters

Each switch maintains per-table, per-flow, per-port and per-queue. The counters for use are listed in Table 2.1. Those values can be obtained through polling requests to the network elements.

Table 2.1: Openflow Counters

Per Table	**Per Flow**	**Per Port**	**Per Queue**
Active Entries	Received Packets	Received Packets	Transmit Packets
Packet Lookups	Received Bytes	Transmitted Packets	Transmit Bytes
Packet Matches	Duration (sec)	Received Bytes	Transmit Overrun Errors
	Duration (nano)	Transmitted Bytes	
		Receive Drops	
		Transmit Drops	
		Receive Errors	
		Transmit Errors	
		Receive Frame Alignment Errors	
		Receive Overrun Errors	
		Receive CRC Errors	
		Collisions	

Actions

Each flow entry is associated with zero or more actions that determine the behaviour of the network element when there is a matching in a incoming packet. There are several types of actions. However, just some of them are not supported in all the OpenFlow switches. When the NE connects to the controller (see Figure fig:OFHandshake), it indicates which

of the optional actions it supports.

The required actions are:

Forward: forwarding the packet to physical ports.

- ALL: send the packet to all the interfaces less the incoming one.
- CONTROLLER: send the packet to the controller.
- LOCAL: send the packet to the switches networking stack.
- TABLE: perform actions in flow table.
- IN_PORT: send the packet through the incoming port.

Drop: if there is no specified action, the flow entry indicates that the matching packets should be dropped.

And the optionals:

Forward: forwarding the packet to physical ports.

- NORMAL: Process the packet using the traditional forwarding path supported by the switch (i.e. traditional L2, VLAN, and L3 processing.)
- FLOOD: Flood the packet along the minimum spanning tree, not including the incoming interface.

Enqueue: forwards a packet by some specific queue of the port. Used to provide basic QoS

Modify-Field: this action increases the usefulness of an OpenFlow implementation, allowing to modify the following fields of an incoming packet:

- VLAN ID: adds, modifies, strips or replace the VLAN header.
- Ethernet Source MAC: modifies source MAC address.
- Ethernet Destination MAC: modifies destination MAC address.
- IPv4 source: modifies source IPv4 address.
- IPv4 destination: modifies destination IPv4 address.
- IPv4 ToS bits: replaces the existing IP Type of Service bits.
- Transport source port: replaces the existing source port.
- Transport destination port: replaces the existing destination port.

2.3.2 Messages

The protocol, described in [1], defines three sort of messages, *controller-to-switch*, *asynchronous* and *symmetric*. For instance, when the controller detects a new switch connected, it starts a handshake process (illustrated at Figure 2.2) starting with a symmetric *Hello* message, and followed by two controller-to-switch messages: *Features* and *Configuration*.

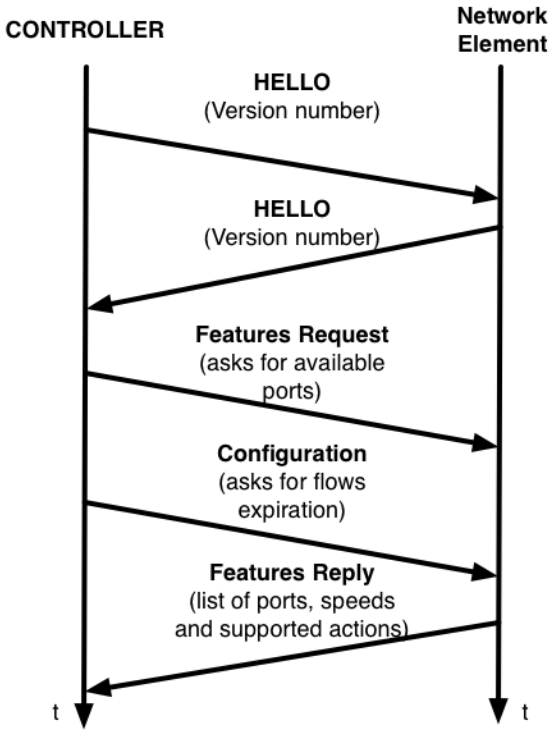

Figure 2.2: OpenFlow messages for Controller-Switch Handshake

The different types of messages are described briefly in the next points.

Controller-to-switch

Controller-to-switch messages are initiated by the controller. Their finality is to get parameters from the NEs or manage them. The different kinds are listed in the next points.

Features: Requesting the specific capabilities of the switch.

Configuration: Set and query configuration parameters in the switch.

Modify-State: Their primary purpose is to add or delete flows in the flow tables and to set switch port properties.

Read-State: Used to collect statistics from every switch.

Send-Packet: Used to send packets out of a specified port on the switch.

Barrier: Used to ensure message dependencies have been met or to receive notifications for completed operations.

Asynchronous

Asynchronous messages are sent from the network elements to the controller when some event occurs. The four different types of asynchronous messages are described beneath.

Packet-in: When a packet arrives to the network element and this doesn't match with any entry of the flow table, a message is send to the controller asking what to do with. By default it includes the first 128 bytes of the packet header.

Flow-Removed: Message sent when a flow entry expires or it is removed from a network element.

Port-Status: Sent by the network element when state configuration state (such as port brought down directly by a user) changes.

Error: Messages indicating some errors in the network element.

Symmetric

Symmetric messages are sent without solicitation, in either direction. The 3 different sort of messages are the following ones:

Hello: Sent when connection startup as a handshake.

Echo: Used to indicate latency, bandwidth or liveness of a controller-NE connection. It must return an echo reply.

Vendor: Pretends to offer additional functionality. Meant for future OpenFlow revisions.

3 Scenario Description

In this chapter

3.1 Mininet Network Emulator

Mininet is an open source software that allows to emulate an entire network. It is the main tool for Software-Defined Networking testbed environments to design, undergo and verify OpenFlow projects.

Mininet provides a high level of flexibility since topologies and new functionalities are programmed using python language. It also provides a scalable prototyping environment, able to manage up to 4000 switches on a regular computer. This is possible thanks to a OS-level virtualization features, including processes and network namespaces, which allows to have different and separate instances of network interfaces and routing tables that operate independent of each other.

Unlike other simulators, like ns-2[1] or Riverbed Modeler[2], which lack realism and the code created in the simulator needs to be changed to be deployed in the real network, Mininet needs no changes on code either configuration when applied to hardware-based networks, offering a realistic behaviour with a high degree of confidence. Another advantage in front of other simulators is that allows real time interaction.

There are four topology elements that Mininet can create:

Link: emulates a wired connection between two virtual interfaces which act as a fully functional Ethernet ports. Packets sent through one interface are delivered to the other. It is possible to configure Traffic Control for the links importing the TCLink library[3] via the Python API.

Host: emulates a linux computer. Is simply a shell process moved into its own namespace, from where commands can be called. Each host has its own virtual Ethernet interface.

[1] The Network Simulator-ns-2
[2] Riverbed Modeler
[3] mininet.link.TCLink Class Reference

Switch: software OpenFlow switches provide the same packet delivery semantics that would be provided by a hardware switch.

Controller: Mininet allows to create controller within the same emulation or to connect the emulated network to an external controller running anywhere there is IP connectivity with the machine where Mininet is running.

Figure 3.1 shows the elements in a simple network with 2 hosts connected to one switch.

The topology script used for this project is shown in Appendix B

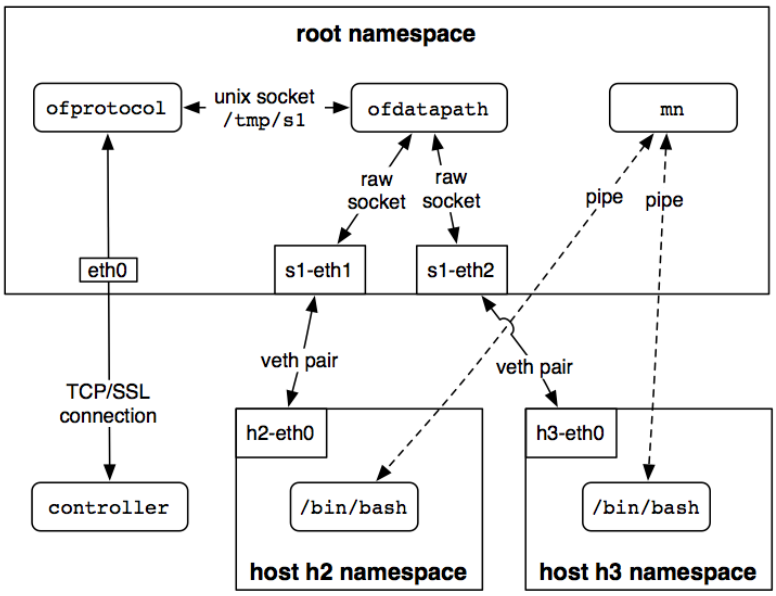

Figure 3.1: Mininet internal architecture with 2 hosts, 2 links, 1 switch and 1 controller (image from openflowswitch.org)

3.2 Floodlight Controller

- *what is it?*
- *Why Floodlight? (justification of selection)*
- *how it works? (brief explanation about the structure and it's behaviour)*
- *main parts involved with the algorithm (listeners of events, forwarding module...)*

3.3 Topology

Explanation about the topology used, justification of why I use this topology

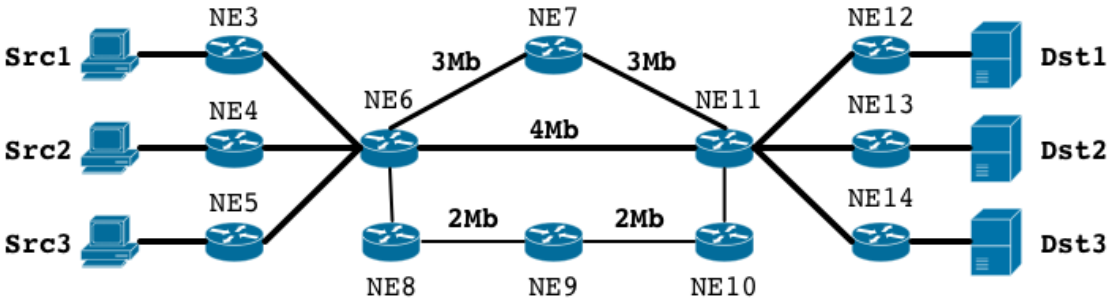

Figure 3.2: Topology used.

4 Algorithm Description

In this section I'll describe the algorithm. First a general overview, with a pseudo-code and a flow chart of its behaviour, and then a more deep explanation about each of the processes, such as how it detects new flows, how find all the routes, how makes the rerouting decision, etc.

5 Emulation and Evaluation

Here I'll explain the emulations done. I'm using MGEN to generate traffic, which allows to generate different patterns (burst, Poisson, ON-OFF) with UDP and TCP, and other parameters. It also introduce a timestamps in the packets. With the results of those traffics I can measure delay, bandwidth, and packet losses end-to-end.
The emulation will consist of generating different traffic patterns.
The evaluation wi evaluate end-to-end performance (delay, bandwidth, and packet losses), plus the network resources utilisation (links).

6 Conclusions and Future Work

Conclusions about the results exposed in the previous chapter, plus some future work to improve the algorithm based on the results obtained

Bibliography

[1] (2009). Openflow switch specification. Technical report, Open Networking Foundation.

[2] (2012). Inter-datacenter wan with centralized te using sdn and openflow. White paper, Google, Inc.

[3] Awduche, D., Chiu, A., Elwalid, A., Widjaja, I., and Xiao, X. (2002). Overview and principles of inter- net traffic engineering. RFC 3272 (Informational).

[4] Fortz, B. and Thorup, M. (2000). Internet traffic engineering by optimizing ospf weights. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 519–528.

[5] Katz, D. Kompella, K. Y. D. (2003). Traffic engineering (te) extensions to ospf version 2.

[6] Sushant Jain, Alok Kumar, S. M. (2013). B4: Experience with a globally-deployed software defined wan. Technical report, Google Inc.

A Setting up Mininet and Floodlight

In order to simulate the scenario with SDN, this project uses Mininet[1] software, which is developed by Stanford University and released under a permissive BSD Open Source license, to simulate the network. And Floodlight[2] as a controller of the network. Others software tools used are Virtual Box[3] to run the Mininet virtual network and Eclipse[4] as a IDE. This appendix explains the steps needed in order to setting up the environment needed for the study.

A.1 Mininet

1. The easiest and recommended way to setting up mininet is downloading a Virtual Machine (VM) image, which is provided in the following link: *Mininet 2.1.0*.
2. Open VirtualBox and create a new VM Ubuntu type and use the download file in the previous step as a virtual hard disk for the new VM.
3. Select "*settings*", and add an additional host-only network adapter so that you can log in to the VM image.
4. Run the new VM.
5. **Login:** mininet

   Password: mininet
6. Start Mininet with the command "*sudo mn*". In the case we want to load a specific topology and connect Mininet to the controller, we have to add some parameters to that command. To load the topology specify in the file *topo.py* and connect to the controller which is running in the host with the IP: *172.26.20.23* the command will be the following:

[1]Mininet website: http://mininet.org
[2]Floodlight website: www.projectfloodlight.org/floodlight
[3]VirtualBox website: www.virtualbox.org
[4]Eclipse website: www.eclipse.org

```
sudo mn --custom topo.py --topo mytopo --controller remote,ip=
172.26.20.23,port=6633
```

7. For further information about Mininet and how to use it go to: Mininet Walkthrough page.

A.2 Floodlight

Floodlight is an open-source software which source code is published on GitHub: github.com/floodlight/floodlight. However, in order to add new modules it is easier to use a Integrated Development Environment (IDE) such as Eclipse. The following steps are needed in order to download Floodlight and integrate it with Eclipse.

1. Type the following commands:

```
sudo apt-get install build-essential default-jdk ant python-dev eclipse
git clone git://github.com/floodlight/floodlight.git
cd floodlight
ant eclipse
```

2. Launch Eclipse
3. "File" → "Import" → "General" → "Existing Projects into Workspace" → "Next"
4. From "Select root directory" click "Browse" and select the parent directory where you placed floodlight. → click "Finish"
5. Create the FloodlightLaunch target:
   (a) Click "Run" → "Run Configurations"
   (b) Right Click on "Java Application" → "New"
      i. For "Name" use *"FloodlightLauncher"*
      ii. For "Project" use *"Floodlight*
      iii. For "Main" use *"net.floodlightcontroller.core.Main"*
   (c) Click "Apply"

A.2.1 Adding a module to Floodlight

In order to add a module with new functionalities, it is needed to change the default startup modules:

`src/main/resources/floodlightdefault.properties`
`src/main/resources/META-INF/services/net.floodlight.core.module.IFloodlightModule`

Adding the new class created.

B Mininet Topology Script

```
#!/usr/bin/python

#Allow to pass arguments
import sys

from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.topo import Topo
from mininet.node import CPULimitedHost
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

from mininet.link import Link, TCLink

def emptyNet():

    "Create an empty network and add nodes to it."
    net = Mininet( controller=RemoteController, link=TCLink, autoSetMacs =
        True )
    info( '*** Adding controller\n' )
    net.addController( 'c0' , controller=RemoteController,ip=sys.argv[1],
        port=6633 )
    print 'connecting to:', sys.argv[1]

    info( '*** Adding hosts\n' )
    srcHost1 = net.addHost( 'h1' )
    srcHost2 = net.addHost( 'h2' )
    srcHost3 = net.addHost( 'h3' )
    dstHost1 = net.addHost( 'h4' )
    dstHost2 = net.addHost( 'h5' )
    dstHost3 = net.addHost( 'h6' )

    info( '*** Adding switch\n' )
    switch3 = net.addSwitch( 's3' )
    switch4 = net.addSwitch( 's4' )
    switch5 = net.addSwitch( 's5' )
    switch6 = net.addSwitch( 's6' )
    switch7 = net.addSwitch( 's7' )
    switch8 = net.addSwitch( 's8' )
    switch9 = net.addSwitch( 's9' )
    switch10 = net.addSwitch( 's10' )
    switch11 = net.addSwitch( 's11' )
    switch12 = net.addSwitch( 's12' )
```

```
44      switch13 = net.addSwitch( 's13' )
45      switch14 = net.addSwitch( 's14' )
46
47      info( '*** Creating links\n' )
48      srcHost1.linkTo( switch3 )
49      srcHost2.linkTo( switch4 )
50      srcHost3.linkTo( switch5 )
51      switch3.linkTo( switch6 )
52      switch4.linkTo( switch6 )
53      switch5.linkTo( switch6 )
54      net.addLink( switch6, switch7, bw=3 )
55      net.addLink( switch7, switch11, bw=3 )
56      net.addLink( switch6, switch11, bw=4  )
57      switch6.linkTo( switch8 )
58      net.addLink( switch8, switch9, bw=2 )
59      net.addLink( switch9, switch10, bw=2 )
60      switch10.linkTo( switch11 )
61      switch11.linkTo( switch12 )
62      switch11.linkTo( switch13 )
63      switch11.linkTo( switch14 )
64      switch12.linkTo( dstHost1 )
65      switch13.linkTo( dstHost2 )
66      switch14.linkTo( dstHost3 )
67
68      info( '*** Starting network\n')
69      net.start()
70
71      info( '*** Running CLI\n' )
72      CLI( net )
73
74      info( '*** Stopping network' )
75      net.stop()
76
77  if __name__ == '__main__':
78      setLogLevel( 'info' )
79      emptyNet()
```

Listing B.1: Topology Script