**ABSTRACT**

Software defined networking is an approach to computer networks where the data plane resides in the network devices, and the control plane in a centralized controller. The data plane helps in forwarding the traffic and the control plane decides about where the traffic is sent. The communication between the controller and the network devices is through the Open flow protocol. Controller studies the network topology and updates the forwarding table on routers and switches. The switches perform the action mentioned in the rules in the forwarding table.

In college networks like Anna University, the students are provided access to sites based on Access Control List. ACL is a predefined list, which has a set of sites with illegal content. If any user tries to access such sites, the user is denied access. But few sites which are blocked have legal content too, which are often knowledgeable. This project identifies such sites as ambiguous. But the college network blocks the entire site.

Through this project, a system is designed that implements SDN which provides partial access to such ambiguous sites. The user can access the legal content in such sites, but not the illegal content. This is done with the help of access control lists, which is similar to that in traditional network, but has separate lists of legal, illegal and ambiguous sites in SDN. The user is given access in case of legal request and denied access in case of illegal request. In case of sites identified by the project as ambiguous, the access right is based on the content of the site requested.

The system is concerned with implementing SDN within a LAN. A centralized controller controls the switches within the LAN. If the destination is within the same LAN, the controller in SDN calculates the shortest path between the source and the destination. The rules are updated accordingly, based on the access right decided with the help of ACL implemented in SDN.

**CHAPTER 1**

**INTRODUCTION**

Software defined networking (SDN) is a revolution in the world of networks. It introduces software in a hardware oriented networks through the use of a controller.SDN helps in controlling the switches through a program called the controller that performs the function of a control plane in a traditional switch. The controller is a centralized program that reads the entire network topology and formulates rules that would be updated on to the 3 layer switches in the network using the OpenFlow protocol [8] , which is a communication protocol that gives access to the forwarding plane of a network switch over the network.

SDN proves to be very useful for a dynamically changing topology as the point of variance is the centralized controller. SDN allows [network administrators](http://en.wikipedia.org/wiki/Network_administrator) to manage network services through [abstraction](http://en.wikipedia.org/wiki/Abstraction) of lower level functionality.

This is done by decoupling the system that makes decisions about where traffic is sent (the [control plane](http://en.wikipedia.org/wiki/Control_plane)) from the underlying systems that forwards traffic to the selected destination (the [data plane](http://en.wikipedia.org/wiki/Data_plane)). In SDN, the controller performs the functions of the control plane.

* 1. **PROBLEM DOMAIN**

The college network (Anna University) uses a traditional network.

In the campus network, the administrator maintains an access control list that focuses on blocking unauthorized sites. But few sites which are blocked in college network contain legal content too, that often proves to be knowledgeable and resourceful. Such sites are termed as ambiguous sites. For instance, websites like YouTube are ambiguous that contain legal as well as illegal content. A traditional network like the campus network blocks the entire site ( YouTube ) thus cutting down on the access to legal content like lectures and guest talks. Hence access is denied even to the useful resource in that site. To overcome this disadvantage, this project focuses on displaying the legal content from an ambiguous site by the use of SDN, thus providing a wider access to legal information.

* 1. **PROBLEM STATEMENT**

The system is concerned with SDN that helps providing selective access to ambiguous sites, as considered by the project and placing rules in the switches accordingly for a particular LAN. If multiple paths are available between the source and destination in the LAN, the shortest path is chosen. And if any link in that path goes down, the packet is transmitted through the next optimal path. As the process continues and if the switch gets overloaded and exceeds the threshold (maximum number of rules in each switch), few rules are flushed before entering new rules.

**1.2 OBJECTIVE**

The objective of the project is:

* Access control with the help of SDN
  + Providing access to legal sites
  + Denying access to illegal sites
  + Providing access to legal content in ambiguous sites
* Rule placement in switches
* Shortest path calculation within the LAN
* Managing rule space overhead

**1.3 PROJECT DESCRIPTION**

The system makes use of SDN to provide access control. Similar to access control list used in traditional network, the system has three different lists for legal, illegal and ambiguous sites in SDN. If the requested destination is a legal site, the user is given access. If it is illegal, the user is denied access. If it is ambiguous site as considered by the project, the user is given access initially and for the further requests, the page source of the requested site is retrieved and parsed. A database of words for sites with illegal content is created. The parsed file is compared with the database and if the file has any illegal word, the user is denied access. If not, the user is given access.

Once if the access right is decided, SDN calculates the shortest path between the source and the destination within the LAN using the shortest path algorithm and ports calculation algorithm. Then the rules are updated in the switches along the path. If any link along the shortest path calculated goes down, the controller is notified and the next optimal path is calculated to transmit the packet to the destination.

Every time before updating rules in the switches, the number of rules already present is calculated. If it exceeds the threshold (maximum number of rules), few rules are flushed before installing new rules.

**1.4 CONTRIBUTION**

This project has its contribution in :

* Shortest path Algorithm: Takes the destination host as input and

formulates the shortest path within the LAN

* Access control: The user is given access rights based on the authenticity of the destination requested.

**1.5 ORGANISATION OF REPORT**

This report is organized as follows:

Chapter 2 of the report describes the basic architecture of SDN, the OpenFlow protocol and work related to the project. Chapter 3 explains the functional and non functional requirements of the projects and the various components of the project. Chapter 4 is the system design which explains the various modules in the project and the algorithms used. Chapter 5 describes the structure of the simulator used and implementation details of the system. Chapter 6 explains the performance evaluation and various test cases that the system satisfies. Chapter 7 is the conclusion, future work and the limitations of the project.

**CHAPTER 2**

**LITERATURE SURVEY**

This chapter focuses on the architecture of SDN and the work that has been done in the field of Software Defined Networking. It discusses the various applications of SDN that has been deployed so far.

**2.1 SDN Architecture**

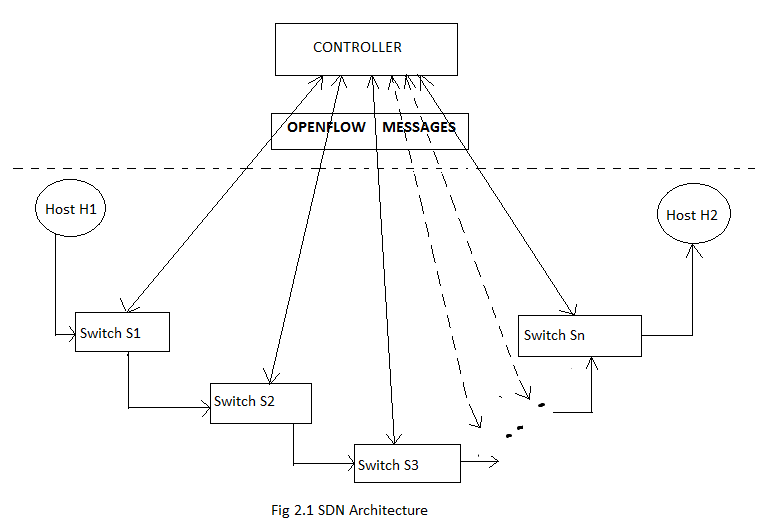


Figure 2.1 shows the basic architecture of SDN. Software Defined Networking helps in programming a centralized controller that performs functionalities similar to a control plane in a traditional network. The centralized controller in SDN communicates with the switches through the OpenFlow messages. OpenFlow messages provide access to the forwarding plane of a switch or router over the network. The OpenFlow protocol [8] sends out messages from the controller to place rules in the switch. Every switch in the network is connected to the controller in addition to being connected to the adjacent switch. There can be numerous switches connected between two hosts. The centralized controller is designed in such a way that it has information on the entire network. A new packet arriving at the switch is forwarded to the destination host according to the rules that the controller places in the switches. Every switch checks its connectivity with its neighbouring nodes and reports to the controller the network topology.

**2.2 Related works**

Software Defined Networks has various applications in the field of networking. Various new possibilities and configuration methods [15] have been introduced and enhanced in networking with the implementation of SDN. SDN has also been introduced in mobile networks [16] to monitor the increasing traffic demands. Numerous research has been done for successful implementation of SDN in handling dynamic traffic requirements [19] and deploying it in various scenarios especially in cloud computing. SDN also addresses its own problems [17] and seeks to solve it by programming the controller efficiently. An IEEE SDN initiative [18] to facilitate professional and academic exchange of SDN related ideas was set up.

Internet has large number of hosts connected in various topologies. Good models of the topological structure of a network are essential for developing and analyzing internetworking technology. Thus numerous methods [1] help in generating various topologies for performing networking operations. These approaches use graph-based models to illustrate the locality and the hierarchy that exists in the internet.

One such model that emerged successful for simulating Software Defined Network is described by Casado [2] where a centralized controller performs the work of the control plane. This model, known as Ethane, works fine with wired and wireless hosts. Its working shows its compatibility with existing switches and hosts. With the model to simulate SDN, the APIs of the traditional network required modification. Hence SDN requires its own APIs [11] for the centralized controller to work.

SDN helps in programming the controller with efficient and optimized algorithms. OpenFlow protocol helps in establishing communication between the controller and the switches in the network. SDN works efficiently for a dynamically changing topology. There are various predefined controllers [4],[3] ,[10] that help in simulating SDN. These controllers are programmed to work on various environments. The Nox controller [4] proves to be efficient in creating an operating system that forwards packets to the switches based on the program that runs on the NOX controller. The beacon controller [3] uses openflow messages to communicate with the switches.

The use of OpenFlow protocol for a campus network [8] has proved to be very useful in programming the flow tables in switches. In addition to accessing the flow tables, the controller can also place optimal number of rules in the switch as defined by the Palette algorithm [7] . This algorithm ensures that no switch is overloaded with rules. Thus it increases the efficiency of networking. The algorithm gives a detailed analysis on decomposing large SDN tables into small tables and distributing them over the network. It helps share resources across connections.

DIFANE [12] and VCRIB [9] leverage all switches in the network to enforce an endpoint policy. The policy ensures that all the traffic is in the data plane for better performance and scalability and also avoid a bottleneck at the controller.

The controller needs to be optimized. Blocking illegal content is generally provided by a firewall. However configuring a firewall is not an easy task. As discussed by Yuan,Mai,Su [13],static analysis has decreased the difficulty in configuring a correct firewall. A centralized firewall policy is distributed at the ingress switches of the packet [13] [14]. Though firewall proves to be sufficient in blocking illegal sites, the aim of retrieving legal content in illegal sites through a firewall is difficult. This project aims at providing a controller that helps block illegal content and place optimal number of rules in the switch for a campus network.

Thus the project focuses on implementing SDN for transmission of packets by

formulating rules for the switches [6]. The controller is provided with an access control list that helps it to demarcate legal and illegal sites. Thus the project retrieves legal content, as authenticated by the network administrator, from illegal sites [5]

**CHAPTER 3**

**REQUIREMENT ANALYSIS**

This chapter describes the various components of the system and the functional and non functional requirements of the system.

3.1 COMPONENTS OF THE SYSTEM

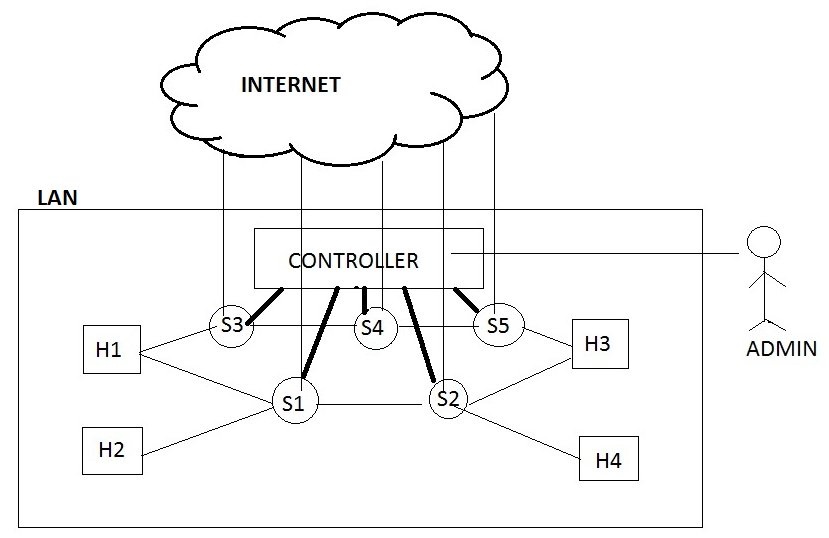


Figure 3.1 describes the components of the proposed system.

**H1...H4** : Hosts within the LAN

**S1 …S5 :** Switches within the LAN with OpenFlow support

**Controller:** The centralized controller that controls the switches within the LAN

**Admin:** The one who configures the controller based on the requirements

The switches are connected to the adjacent switches and the hosts nearby. All the switches within the LAN report to the same controller. The switches are connected to the Internet, to retrieve the page source of the site requested.

3.2 FUNCTIONAL REQUIREMENTS

The various functions performed by the system are,

1. Providing access to legal requests.
2. Denying access to illegal requests.
3. Deciding the access rights to ambiguous requests, based on the content of the site requested.
4. Calculating the shortest path within the LAN, if multiple paths are present.
5. Placing rules in the switches accordingly along the path calculated, based on the access right decided with the help of access control list.
6. Flushing rules in the switches within the LAN in case of overhead.

The use case diagram depicts the various functional requirements

of the proposed system, as shown in Figure 3.1. The actors are user, server, controller and switch.

The site requested by the user can be either legal, illegal or ambiguous.

Based on the destination address, the request is classified at the

first switch of the network. If the requested site is illegal, the request is denied. If it is either legal or ambiguous, initially the user is given access.

Once the decision to permit is made, rules are placed accordingly in

the switches, which is done after calculating the shortest path between

the source and destination within the LAN. If the requested site is ambiguous, the page source of the requested site is retrieved and parsed. The parsed file is compared with a database of illegal words. The debugging module is

used to find another optimal path between the source and destination, if

any link along the first optimal path is down. The cover and replace is

used to remove rules from switches if they get overloaded and exceed

the threshold.

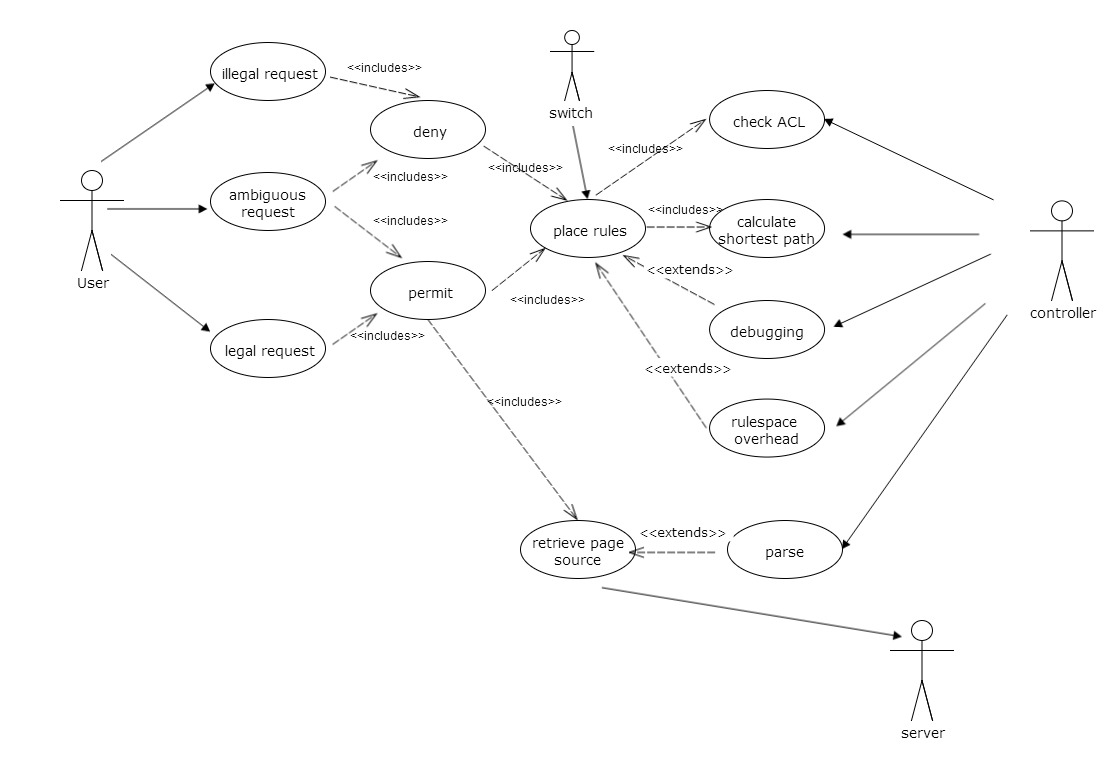


Figure 3.1 Use case diagram

3.2 NON FUNCTIONAL REQUIREMENTS

3.2.1 Simulator

The system can either be deployed into a real time network or simulated with the help of a simulator. This project aims at simulating SDN and perform access control using it. A simulator that can be used should support a centralized controller program that communicates with the switches connected to it with the help of OpenFlow Protocol. The switches used should be openflow switches, to support openflow messages. One simulator that has these features is MININET and is used in this project.

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. The mininet VM includes all OpenFlow binaries and tools pre-installed, and tweaks to the kernel configuration to support larger Mininet networks. The mininet VM is run using the VMware

workstation. The VMware wokstation enables users to set up one or

more virtual machines on a single physical machine, and use them simultaneously along with the actual machine. Each virtual machine can

execute its own operating system.

3.2.2 Scalability

The system makes use of a controller, to control switches within a LAN. The simulator, Mininet supports 4096 switches. However the topology used has 4 hosts and 5 switches with all the nodes reporting to a single controller within a LAN.

**CHAPTER 4**

**SYSTEM DESIGN**

This chapter deals with the various modules used in the project and their functionalities. It gives an idea about the inputs and outputs of the modules and the usage of the components in the working of the system.



Figure 4.1 is the basic block diagram of the proposed system. It depicts the various modules that are involved in the system. When a packet arrives at a switch, a look up of the forwarding table is done. In case of a match, the packet is forwarded to the destination. Otherwise the packet is given as input to the Access Control Module. The Access Control Module helps in determining whether the user has requested for legal, illegal or ambiguous site, as considered in the project. If the request is for an ambiguous site, the Content Inspection module helps in displaying the legal content alone in the site. However if the user requests for an illegal site the packet is dropped. In case of a legal site, the Path Calculation module is invoked to provide the optimal path between the user and the server. The Rule Placement module helps in avoiding overloading of the switches.

The proposed system consists of 4 modules, namely,

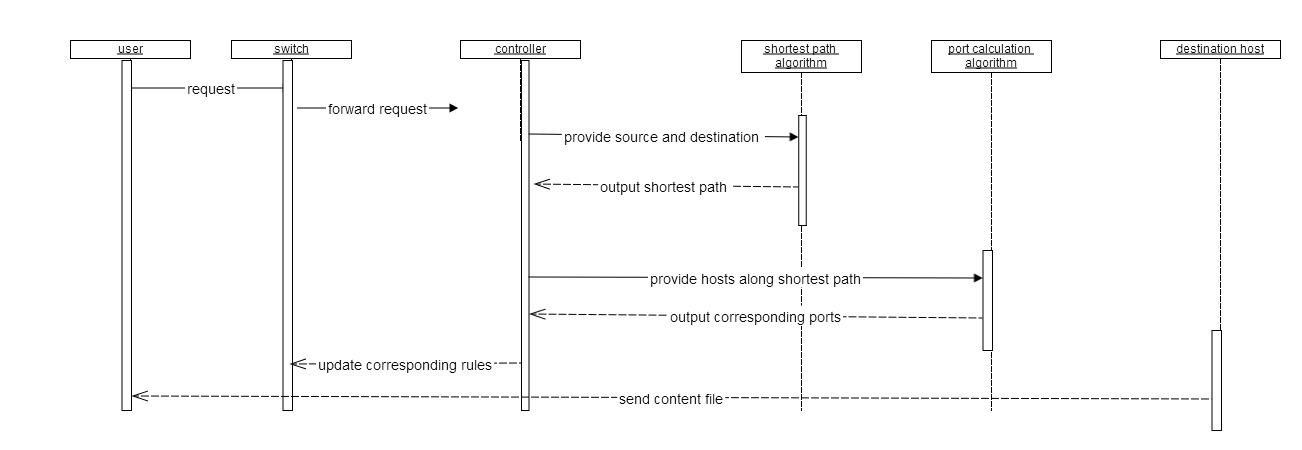
* Access Control Module.
* Path Calculation Module.
* Content Inspection Module.
* Rule-Placement Module.

**4.1 ACCESS CONTROL MODULE :**

The Access Control Module specifies whether to provide or deny access to the destination host by implementing the extended ACL. A user may request access to a site which could be legal, illegal or ambiguous. Based on the access permissions of the requested site, the module implements three functions to allow access to a legal site, to deny access to an illegal site and to inspect the contents of the site in the case of an ambiguous site.

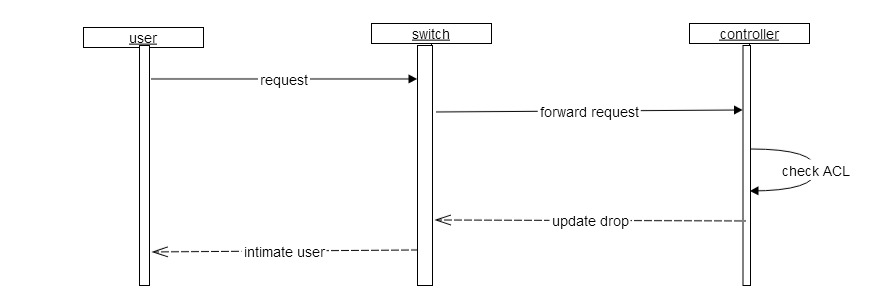
**4.1.1 LEGAL ACCESS FUNCTION**

When a request reaches a switch, it checks if there is a rule in its forwarding table to reach the destination. In case of a miss, the request is forwarded to the controller. The controller then uses the Extended ACL (access control list) to check if access can be provided to the particular destination host. The endpoint policy in the controller is invoked for the authorization of the request. If it is an authorized request, the controller invokes the path calculation module and uses the rule placement module to update the rules in the switches. When the rules are updated in the switch, the client can now have access to the server and can retrieve information from the site.



**4.1.2 ILLEGAL ACCESS FUNCTION**

Similar to the previous function, once a request to an unauthorized site reaches a switch, it checks for a corresponding entry in the forwarding table. In case of a miss, the request is forwarded to the Controller. The Controller checks if access can be provided for the destination host. Once there is a match in the ACL for an illegal site, the Controller adds rules in the switch (ingress) to instantly drop the packet thus blocking the access to the site. The user is then intimated about the packet drop. Further requests to the site will be blocked automatically by the switch.



**4.1.3 AMBIGUOUS ACCESS FUNCTION**

Ambiguous sites are those sites which contain both legal and illegal

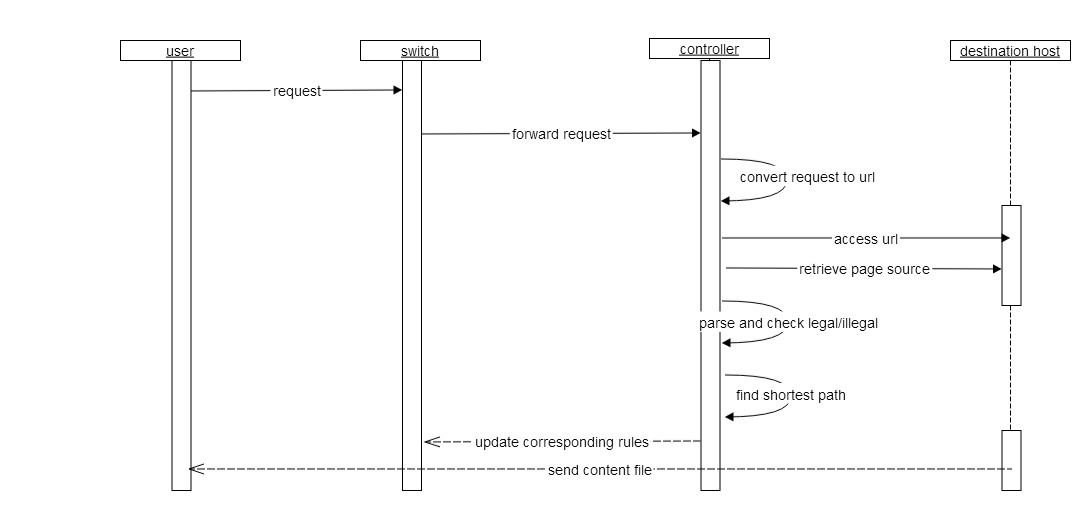
content like youtube. In this case, access has to be provided only for the legal content and has to be blocked for the illegal content. This module is very much similar to the legal access module where in the initial handshake happens similar to those of legal sites. But on further requests, the controller does content inspection and only then provides access.

Thus the input and output for this module, for all the three possible requests are,

INPUT : Destination host

OUTPUT: Access grant or deny based on ACL.

PROCESS : Legal access function, Illegal access function, ambiguous access function, ACL verification, connection establishment or connection drop.



**4.2 PATH CALCULATION MODULE:**

The module helps in calculating the optimal path between two hosts within the same LAN network. To implement this, three algorithms have been used, namely,

• The shortest path algorithm

• The ports calculation algorithm

• The debugger.

**4.2.1 SHORTEST PATH ALGORITHM:**

This algorithm computes the best path(shortest) to reach the destination host within the LAN network, when provided with the source, destination and the graph topology as inputs.

**4.2.2 PORTS CALCULATION ALGORITHM:**

Once the path is found, the controller needs the input and output ports along the path to update the rules in the switches. To find the ports, the port calculation algorithm is used. This algorithm provides the input and output ports of the switches in the calculated path when the source, destination and a list of ports connections are given as input.

**4.2.3 THE DEBUGGER :**

Whenever a link, which is being used by a client to connect to a legal destination host, is down, the access to the destination host is denied to the client. In order to avoid this, the debugger module checks if all the links along the path generated by shortest path algorithm are connected and only then runs the rule placement algorithm. If it finds that some link along the path is down, it immediately calculates the next optimal path within the LAN network and updates the rules in the switches along the alternate path so that the client is always able to get access to any legal content.

INPUT: Source, destination, network topology, list of ports.

OUTPUT: Optimal path between source and destination hosts within the same LAN network.

PROCESS: Shortest path calculation, ports calculation, debugging.

**4.3 CONTENT INSPECTION MODULE:**

The content inspection module is used in the case of an ambiguous site to verify the content of the page based on which it decides whether the access can be granted or not. The controller retrieves the page source of the destination host and then writes it to a file. This file is then parsed and is matched against a database of illegal contents. In case of a match, the controller denies access to the page by updating the rules to drop the packet. Else, the access is provided through the shortest path found during the initial handshake.

INPUT : Page source of the destination site.

OUTPUT: Content validated as legal or illegal.

PROCESS: Parsing and Database validation.

**4.4 RULE PLACEMENT MODULE:**

The rule placement module is concerned about the placing the appropriate rules for legal, illegal and ambiguous access in corresponding switches along the path to the destination in an efficient manner such that the switch does not get overloaded by rules (beyond rule space) at any instant of time . To enable this, the module makes use of the cover and replace algorithm, routing policy and the rule placement algorithm. The routing policy and the rule placement algorithm together make sure that the packet reaches the destination through the path specified by the shortest path algorithm and thus ensuring that the network obeys the endpoint policy E , forwards the packets over the paths specified by R.

To further improve upon the efficiency and to ensure that the rules in the

switch do not exceed the rule space, the cover and replace algorithm isused. This algorithm checks if the rules in any switch exceeds the rulespace. If any, then it tries to minimize the number of rules in the overloadedswitch. It places only the recently used rules in the overloadedswitch and transfers the rest to another idle switch along the path . Acommon rule is written in the overloaded switch to seek the idle switch(the one to which the rules are transferred) in case of need.

INPUT: Endpoint policy, Routing policy, Network topology and the maximum number of rules each physical switch can hold.

Endpoint policy (E) :It views the entire network as one big switch. The policy specifies which packets to drop or to forward to specific egress port as well as any modification of specific header fields.

Routing policy(R) :The routing policy specifies what paths traffic should follow between the ingress and egress ports.

OUTPUT: Efficient placement of rules in the switches.

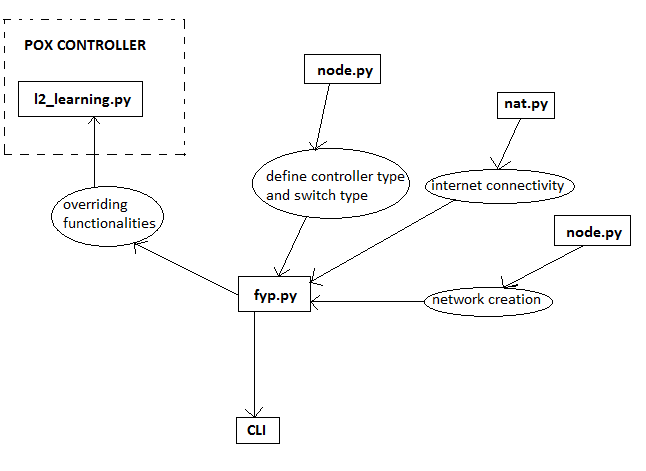
PROCESS: Rule placement algorithm and Cover and replace Algorithm.

**CHAPTER 5**

**SYSTEM DEVELOPMENT**

The chapter describes the implementation of the project. It helps in understanding the structure of the project in detail with the help of simulator structure and code snippets.

**5.1 Simulator description**

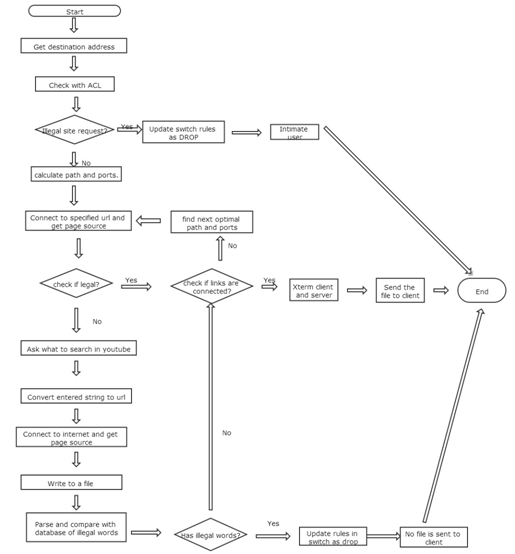


Software defined networking can be simulated through various simulators. Each of these simulators help in creating a virtual network and user defined controller. Thus mininet is one such network simulator that adheres to the requirements of the project and helps in having a better understanding of the network technology. It has pre-defined packages installed that helps it to simulate software defined networks (SDN) . The POX controller behaves as a remote controller that helps in handling the functions of the control plane. The controller can be programmed according to the administrator’s need.

Figure 5.1 defines the structure of mininet. A network of hosts and switches can be created with the help of net.py python file that is pre-installed in mininet. Every node is connected to the internet via the nat.py python file. The IP address and the MAC address of every host is retrieved from the node.py file that includes numerous pre-defined functions describing the routine of various controllers and different types of switches. Every switch is converted to a learning switch that is used in SDN networks. The l2\_learning.py file contains the functions of the POX controller. Thus fyp.py file imports all these files and overrides the functionalities of the controller defined in l2\_learning.py file according to the needs of the network administrator. The fyp.py file also contains commands to update rules to the switch.

**5.2 CONTROL FLOW**

Figure 5.2 depicts the control flow of the entire project. The network topology is the input to the overall proposed system. The simulator, Mininet, creates a network topology given the hosts and the switches. The simulator provides pre-defined functions for network creation. Every host in the created network is given access to the internet. The client then requests access to any server. The switch in the path of the client and server, performs a look-up in its forwarding table. When there is no rule in the switch, the packet is sent to the controller. The controller performs the shortest path algorithm to determine the optimal path for packet transmission. The controller then performs a look-up into the ACL ( Access Control List ) to determine the authorization of the requested server. In case of an illegal site, the first switch in the path is updated with a denial rule. In case of a legal or an ambiguous site, the port calculation algorithm is invoked. Data about the port connectivity of every switch is maintained in a list. The controller looks up into the list to determine the in-port and out-port of every switch in the path. The controller then updates the rule as permit in every switch along the path, performing the rule

****

placement algorithm. The rule placement algorithm ensures that every switch is not overloaded with rules at any point in time. In case of a legal site, the rules in the switch, once updated as permit, allows continuous access of the site by the client. In case of an ambiguous site, the controller retrieves the source page of the requested content by the client to the same server. The source page is parsed for identifying any illegal content against a database that stores some illegal keywords. Even a single match makes the controller update the rule as deny to all the switches along the path. However in case of no match, the controller ensures the page as a legal one and updates the switches along the path with a permit rule. Subsequent access to an ambiguous site invokes the controller to parse the source page for every access. Thus the project ensures complete access to legal content in an illegal site.

There are 4 modules that the project focuses on :

* Access Control Module
* Path Calculation Module
* Content Inspection Module
* Rule-Placement module

**5.3 IMPLEMENTATION**

**5.3.1 Access Control Module**

The access control module helps in demarcating between the various types of sites thus providing the user access to legal content in all sites. The module defines 3 functions for the 3 types of sites : legal sites ,illegal sites and ambiguous sites. The hosts are appended once the network is created in hosts\_list. The user is prompted to enter the destination host that is saved in the variable dest. The source and the destination stored in variables src and dest respectively are given as input to the Path Calculation Module. The output of the Path Calculation module is stored in the variable path. The ACL contains 3 variables defined as list type: legal-sites, illegal-sites and ambig-sites, each of which contains various site url. The ACL is checked against the variable dest. Depending on the list that the variable dest matches, the function defined in this module is implemented. subprocess.call command is used to update rules in the switches involved in the shortest path.

**5.3.1.1 Legal Access Function**

A list of legal sites is maintained in legal-sites variable. The ACL is checked if the entered destination host is legal. If so the IP and Mac addresses are calculated for every host in the path and appended in the ip\_addr and mac\_addr lists. The page source of the requested site is retrieved and stored in a file called page and is sent to the user.

//contains list of legal sites

legal-site.append(h3)

//get the server to be connected to

dest=raw-input()

//calculate the shortest path between source and destination within the same LAN

path=bfs(graph,src,dest,q)

//calculate the IP address and MAC address for rule updation

ip-addr.append(mininet.node.Node.IP(src))

ip-addr.append(mininet.node.Node.IP(dest))

mac-addr.append(mininet.node.Node.MAC(src))

mac-addr.append(mininet.node.Node.MAC(dest))

//check the ACL and connect to internet

For x in legal-site

If x==dest

//retrieve and write the source code to a file

page=urllib2.urlopen(http://stackoverflow.com).read()

file=open(stack,w)

file.write(page)

//call to the ports calculation algorithm

ports=portscal()

//rule updation to every switch along path.

for i in range(0,len(sl))

subprocess.call([ovs-ofctl,add-flows,%s%(sl[i]), in-port=%s,vlan-tci=0x0000, dl-src=%//sl is the list of switches in the path

**5.3.1.2 Illegal Access Function**

A list of illegal sites is maintained is maintained in illegal-sites list. The ACL is checked if the entered destination host is illegal. If so, the rules are updated as drop in the first switch along the path.

//contains list of illegal sites

illegal-site.append(h2)

//get the server to be connected to

dest=raw-input()

//check the ACL

For x in illegal-site

If x==dest

Deny request

//call to the ports calculation algorithm

ports=portscal()

//rule updation to the first switch along path.

subprocess.call([ovs-ofctl,add-flows,%s%(sl[0]), in-port=%s,vlan-tci=0x0000, dl-src=%//sl is the list of switches in the path

**5.3.1.3 Ambiguous Access Function**

A list of ambiguous sites is maintained in ambig-sites list.

The ACL is checked if the entered destination host is an ambiguous site. If so, the page source of the requested site is retrieved and stored in a file called page and is sent to the user. After the file is sent to the user , the file is cleared. Then the user is asked what he/she wants to search further in the site specified already. The content that the user enters is converted into a url, and the page source of that specified page is retrieved and written to the file called page. This file is given as input to the Content Inspection Module for determining the legal content in that file. The ouput of the module determines whether the page requested is legal or illegal and the rules are updated to the switch accordingly.

//contains list of ambiguous sites

ambig-site.append(h4)

//get the server to be connected to

dest=raw-input()

//calculate the shortest path between client and server

path=bfs(graph,src,dest,q)

//calculate the IP address and MAC address for rule updation

ip-addr.append(mininet.node.Node.IP(src))

ip-addr.append(mininet.node.Node.IP(dest))

mac-addr.append(mininet.node.Node.MAC(src))

mac-addr.append(mininet.node.Node.MAC(dest))

//check the ACL and connect to internet

For x in ambig -site

If x==dest

page=urllib2.urlopen(http://youtube.com).read()

file=open(stack,w)

//write the source code to a file

file.write(page)

//call to the ports calculation algorithm

ports=portscal()

//rule updation to the first switch along path.

subprocess.call([ovs-ofctl,add-flows,%s %(sl[0]),in-port=%s,vlan-tci=0x0000,dl-src=%//sl is the list of switches in the path

//implements the Control Inspection Module

**5.3.2 Path Calculation Module**

The module helps in calculating the optimal path between two hosts. The path is calculated with the help of shortest path algorithm and ports calculation algorithm. A test case that is implemented in this module is Debugger function.

Inputs to the module are two variables, src and dest. The source host is stored in the variable src and the destination host is stored in the variable dest.

**The shortest path algorithm:**

The variable graph contains the network topology defined by the network administrator. The variable src is stored in the variable start and the variable dest is stored in the variable end. The output is the shortest path between the start and the end nodes. The output is stored in the variable tmp-path.

def spath(graph,start,end,path=[]):

temp-path=[start]

add tmp-path to queue

while q is not empty

dequeue q to tmp-path

last-node=tmp-path[len(tmp-path)-1]

print tmp-path

if last-node == end

tm-path is a valid path

for link node in graph[last -node]

if link node not in tmp-path

newpath = tmp-path+ link-node

q.enqueue(new path)

return tmp-path

**The Ports Calculation Algorithm**

The algorithm helps in calculating the in-port and out-port of the switches along the path between the source and the destination.

The variable path is the path calculated by shortest path algorithm. l1[]and l2[] are the list of hosts and switches having connection between each

other. l3[] is the list of port number through which l1[i] (each node) is connected to l2[i] (each node )

ports=[]

for j in range(0,len(path)):

for i in range(0,len(l1)):

if l1[i]==path[j]:

if l2[i]==path[j+1]

ports.append(l3[i])

return ports

**The Debugger:**

This algorithm is a test case to the project. The algorithm helps in calculating the next optimal path between the same pair of source and destination hosts when any of the links along the shortest path calculated is down. The source and the destination stored in the variables src and dest respectively are given as input to the shortest path algorithm. All possible path between the source and destination is calculated using the shortest path algorithm and stored in a file called spath. The path length of every possible path in the file spath is calculated and stored in the list pathlen. When a link is down, the pathlen is traversed to get the next optimal path length. The corresponding path is traversed in the spath file. The path is then retrieved from the file and stored in the variable, path. The switch\_list function is invoked to retrieve the switches along the path. Input to the function is the variable, path. A pattern match is done and the list of switches is stored in the variable s\_list. The rules are updated in the switches stored in s\_list.

def spath(pathlen):

maxim=100

a=0

print pathlen

for i in range(0,len(pathlen)):

if pathlen[i]<maxim:

a=i

maxim=pathlen[i]

print "min index is "

print a

pathlen[a]='100'

fo=open('sp','r')

lines=fo.readlines()

path=re.findall('[a-z][0-9]',lines[a])

print "path of min length is "

print path

path1=path

return path

def switchlist(path):

s\_list=[]

for x in path:

k=re.search('[s][0-9]',x)

if k!=None:

y=re.search('[0-9]',x)

print y.group(0)

s\_list.append(y.group(0))

print s\_list

sl=['s'+str(x) for x in s\_list]

print sl

sl1=sl

return sl

**5.3.3 Content Inspection Module**

This module helps in parsing through the retrieved source file of ambiguous sites. The source page that the user requests is written to a file called page. The file is parsed word by word and compared against a database of illegal words. These illegal words are stored in a file called database. Any match between the two files indicates illegal content and the switches in s\_list are updated with drop . While there is no match, the switches are updated with permit.

//enter the content to be searched

query=raw-input()

//connect to the internet

url.append(http://youtube.com/resultssearch-query=)

page=urllib2.urlopen(url).read()

//write to a file

file.write(page)

//compare the file with the database file

for i in file1:

thisline1=i.split()

for i in database:

thisline2.split()

for k in thisline1:

for k1 in thisline2:

if k==k1:

//update the rule as drop and set flag as 1

If flag==0:

update the rule as permit

**5.3.4 Rule Placement Module**

The rule placement module avoids overloading of switches with rules. A variable n helps in keeping count of the number of rules in the switch. Subprocess.call() function helps in implementing python commands. The rules in every switch in s\_list is written to a file called cover. The number of rules is calculated and the value of n is incremented. If the value exceeds the threshold value ,8 ,the rules in the switch are flushed and new rules are added. Edst[] is the list of visited sites.

def count(y):

n=0

fz=open('cover','w')

subprocess.call(["ovs-ofctl","dump-flows","%s"%(y)],stdout=fz)

fm=open('cover','r')

line1 =fm.readlines()

for k in line1:

m=re.search('cookie',k)

if m!=None:

n=n+1

print n

return n

//call for count function

n=count(y)

if n>=8:

edst=[]

subprocess.call(["ovs-ofctl","del-flows","%s"%(y)])

**CHAPTER 6**

**RESULTS AND DISCUSSION**

This chapter describes the performance evaluation measures and the different test cases along with screen shots.

According to the network topology used in this project,

* Host h1 client
* Host h2 Site with illegal content alone
* Host h3 Site with legal content alone
* Host h4 Site with both legal and illegal content.

The network created for this project is tested with upto 20 nodes

and the various test cases are also experimented.

|  |  |
| --- | --- |
| **If requested destination is:** | **Process done:** |
| H2 | Request dropped and rule updated in first switch as drop |
| H3 | Shortest path is calculated and rules updated in all switches along the path as permit |
| H4 | Page source is retrieved and parsed to find the authenticity of contents. The rules are updated accordingly. |

* 1. **PERFORMANCE EVALUATION**

The performance of the project can be evaluated by verifying the access rights decided by the controller for different requests.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **REQUESTS** | **ACCESS RIGHT IN TRADITIONAL NETWORK** | **ACCESS RIGHT IN SYSTEM DESIGNED** |
| 1 | Google.com | Permit | Permit |
| 2 | Stackoverflow.com | Permit | Permit |
| 3 | Songspk.com | Deny | Deny |
| 4 | Tubetamil.com | Deny | Deny |
| 5 | Film songs in youtube.com | Deny | Deny |
| 6 | Kurukshetra guest lecture in youtube.com | Deny | Permit |

The requests 1 & 2 are legal. Hence such requests are given access in both traditional network and the system designed using SDN Access Control. The requests 3&4 are illegal, hence denied in both the cases. The requests 5&6 are to the site youtube.com. Youtube being an ambiguous site, the access right is decided based on the content requested. If the user requests film songs from youtube.com, the access is denied in both the cases. If the request is for kurukshetra guest lecture, which is a legal content, it is denied in case of traditional network but the user is given access in the system designed.

* 1. **TEST CASES**
     1. RULE SPACE OVERHEAD

A threshold, which specifies the maximum number of rules in each

switch, is determined when the network is created. For each request,

before the switch is loaded with new rules, the number of rules already

present is compared against the threshold. If it exceeds the threshold, the

rules are flused before loading new rules for the current transmission.

* + 1. LINK FAILURE

Once the shortest path is calculated, the links along the path are

checked if they are up. If not, the current calculated path is neglected and

a second optimal path is calculated between the source and destination

and the process is continued.

* + 1. LEGAL CONTENT IN AMBIGUOUS SITE

The main difference between traditional network and SDN is that

the controller can be programmed according to the user wish. In our

college scenario, few sites are blocked as they contain illegal content.

But few such sites have some legal content in them, yet they are not

given access. The controller in this project is programmed in such a

way that based on the content of the page requested by the user, access

rights are given. Content inspection is done, if request to an ambiguous

site is made and the access is given if it is legal, else access is denied.

**6.3 SCREEN SHOTS**

Every rule updated in the switch has the following major fields to be filled:

In\_port : the incoming port number of the switch

dl\_src : the MAC address of the source host

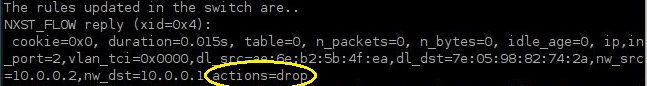
dl\_dst : the MAC address of the destination host

nw\_src : the IP address of the source host

nw\_dst : the IP address of the destination host

6.3.1 Rules updated in the switch for illegal access

The action field is updated as drop. This indicates that any packet from the source host and the destination host specified in nw\_src and nw\_dst is dropped by the switch.



6.3.2 Rules updated in the switch for legal access

The action field is updated with the output port number. The output port number is obtained from the ports calculation algorithm. This indicates that any packet from the source host and the destination host specified in nw\_src and nw\_dst is forwarded through the specified output port number by the switch.

D:\916483ftbcky\Figures\legal_site_rules.JPG

**CHAPTER 7**

**CONCLUSION**

This chapter explains the goals that are achieved through this project, the future works that can be done and the limitations found in this simulator.

**7.1 GOALS ACHIEVED**

Thus in this project, a network is simulated and the client host is allowed to connect with the remaining hosts in the LAN. The various goals achieved are,

* A request to legal site is given access
* A request to illegal site is denied access
* A request to ambiguous site is given access based on the content requested
* An alternate path is chosen, if any link along the path calculated goes down
* The rules in a switch are flushed before installing new rules, if the count exceeds the threshold decided.

**7.2 FUTURE WORK**

* The system designed is concerned with hosts and switches within a LAN, and they are controlled by a single controller. This system can be extended in such a way that multiple LANs are involved and each LAN being controlled by multiple controllers, if there are large number of nodes.
* This project simulates the idea of access control in SDN with the help of simulator called MININET. This can also be deployed in a real time scenario with the help of switches, host machines and controller.

**REFERENCES**

[1] Kenneth L Calvert, Matthew B Doar, and Ellen W Zegura, “Modeling internet topology”, Communications Magazine, IEEE, vol. 35, num. 6, pp. 160–163, 1997.

[2] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, and Scott Shenker, “Rethinking enterprise network control”, IEEE/ACM Transactions on Networking (TON), vol. 17, num. 4, pp. 1270–1283, 2009.

[3] David Erickson, “The beacon openflow controller”, In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 13–18. ACM, 2013.

[4] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Mart´ın Casado, Nick McKeown, and Scott Shenker, “Nox: towards an operating system for networks”, ACM SIGCOMM Computer Communication Review, vol. 38, num. 3, pp. 105–110, 2008.

[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazi`eres, and Nick McKeown, “Where is the debugger for my software-defined network?”, In Proceedings of the first workshop on Hot topics in software defined networks, pp. 55–60. ACM, 2012.

[6] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker, “Optimizing the one big switch abstraction in software-defined networks”, Proc. ACM CoNEXT, vol. 2, 2013.

[7] Yossi Kanizo, David Hay, and Isaac Keslassy, “Palette: Distributing tables in software-defined networks”, In INFOCOM, 2013 Proceedings IEEE, pp. 545–549. IEEE, 2013.

[8] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, “Openflow: enabling innovation in campus networks”, ACM SIGCOMM Computer communication Review, vol. 38, num. 2, pp. 69–74, 2008.

[9] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan, “vcrib: virtualized rule management in the cloud”, Hot-Cloud12, vol. 4, 2012.

[10] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrˆea, Sidney Cunha de Lucena, and Robert Raszuk, “Revisiting routing control platforms with the eyes and muscles of software-defined networking”, In Proceedings of the first workshop on Hot topics in software defined networks, pp. 13–18. ACM, 2012.

[11] Myung-Ki Shin, Ki-Hyuk Nam, and Hyoung-Jun Kim, “Software-defined networking (sdn): A reference architecture and open apis”, In ICT Convergence (ICTC), 2012 International Conference on, pp. 360–361. IEEE, 2012.

[12] Minlan Yu, Jennifer Rexford, Michael J Freedman, and Jia Wang, “Scalable flow-based networking with difane”, ACM SIGCOMM Computer Communication Review, vol. 40, num. 4, pp. 351–362, 2010.

[13] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, andP. Mohapatra, FIREMAN: A toolkit for Firewall modeling and analysis.," in IEEE Symposium Security and Privacy, pp. 199-213, 2006.

[14]S. Ioannidis, A. D. Keromytis, S. M. Bellovin, andJ. M. Smith, Implementing a distributed Firewall," in CCS, (New York, NY, USA), pp. 190-199,ACM, 2000.

[15] [Hyojoon Kim](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Hyojoon%20Kim.QT.&newsearch=true), [Feamster, N.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Feamster,%20N..QT.&newsearch=true),” Improving network management with Software Defined Networking” in [Communications Magazine, IEEE](http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=35)  (Volume:51 ,  [Issue: 2](http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6461169) ), February 2013

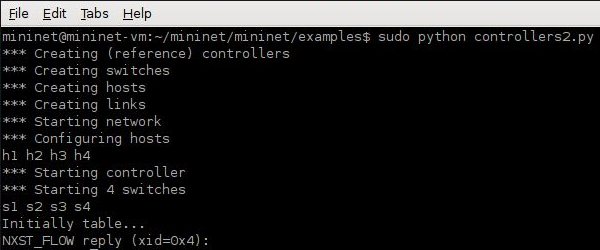
[16] Ali-Ahmad, H.; Cicconetti, C.; de la Oliva, A.; Draxler, M.; Gupta, R.; Mancuso, V.; Roullet, L.; Sciancalepore, V. "CROWD: An SDN Approach for DenseNets",  *Software Defined Networks (EWSDN), 2013 Second European Workshop on,* On page(s): 25 – 31

[17] Valdivieso Caraguay, A.L.; Barona Lopez, L.I.; Garcia Villalba, L.J. "Evolution and Challenges of Software Defined Networking",  *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for,* On page(s): 1 – 7

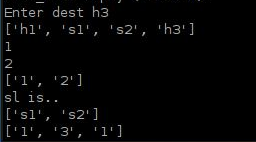
[18] Ulema, M.; Amogh, N.; Boutaba, R.; Buyukkoc, C.; Clemm, A.; Xie, J.L.; Vuran, M.C.; Manzalini, A.; Saracco, R. "IEEE Software Defined Network Initiative",  *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for,* On page(s): 1 – 6

[19] Arslan, Z.; Alemdaroglu, A.; Canberk, B. "A traffic-aware controller design for next generation software defined networks",  *Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on,* On page(s): 167 - 171

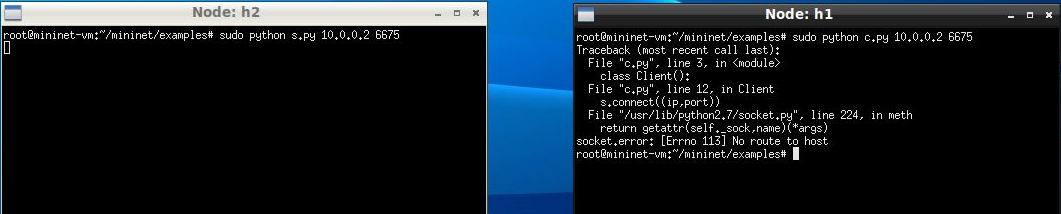
**APPENDIX**



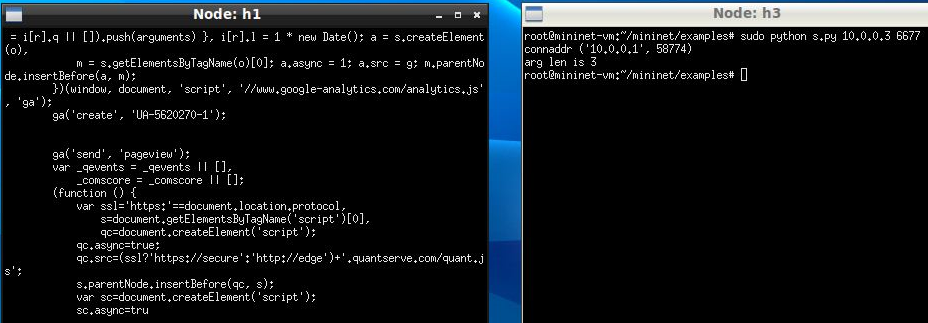
Network creation in the Mininet simulator



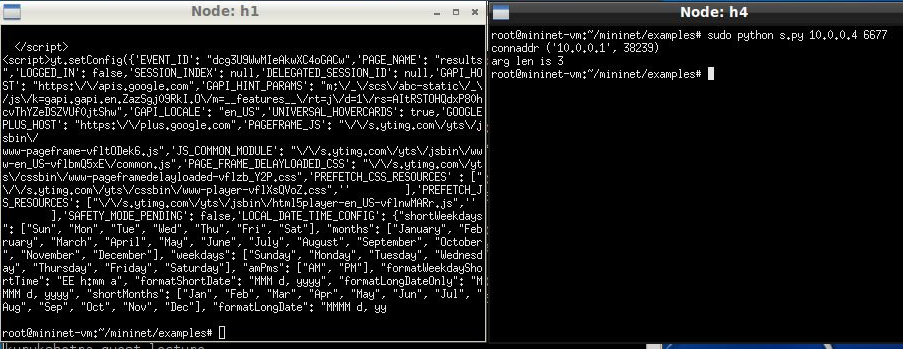
Shortest path algorithm and ports calculation algorithm.



Illegal site access denied



Legal site access granted



Ambiguous site as considered by the project displays the legal content