

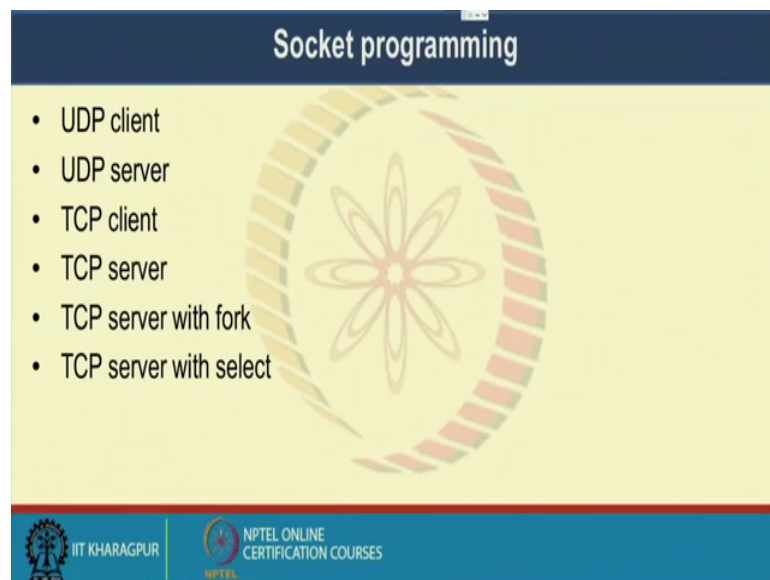
Computer Networks and Internet Protocol
Prof. Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 45
Software Defined Networking – III (Demo)

Welcome back to the course on Computer Network and Internet Protocols. So, in the last classes we are discussing about this software defined networking concept. So, today we will see then implementation of a Software Defined Architecture, Software Defined Networking architecture.

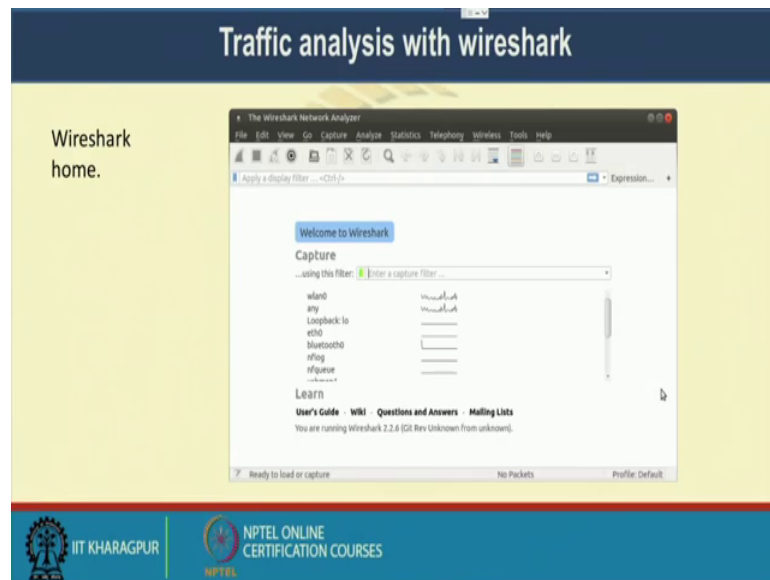
And in our network emulator platform called mini net and we are talking about these open flow protocol. So, we will see that how you can utilize this open flow protocol on top of our mini net architecture to send, or receive packets or to immolate our network topology inside your computer. So, let us have our journey on this mini net and open flow controllers.

(Refer Slide Time: 00:59)



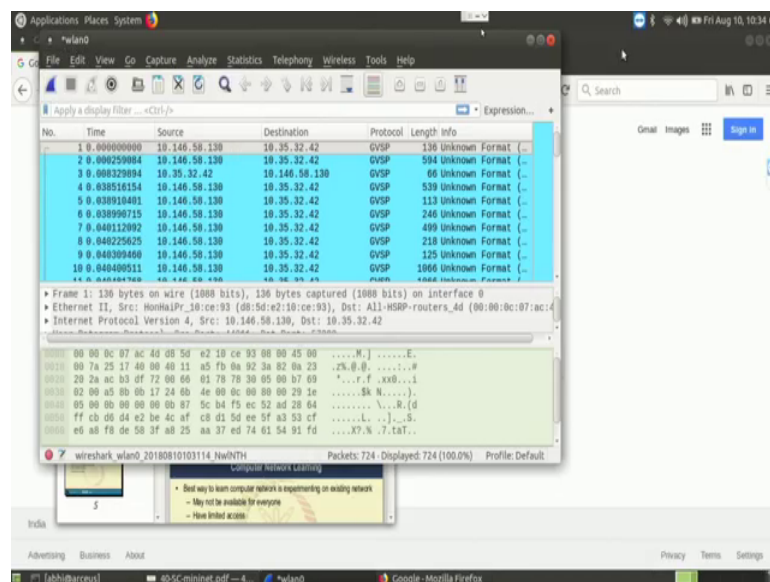
So, till now we have earlier looked into different socket programming aspects. So, you can actually in mini net you can run all these different socket programming and see that the packets are actually traversing in the network.

(Refer Slide Time: 01:15)



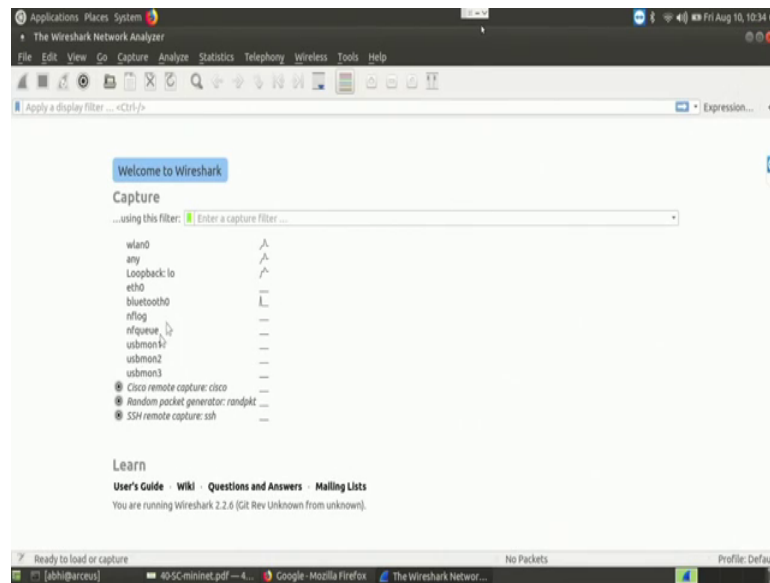
Now to capture the packets in the internet, we have a nice traffic analysis tool called Wireshark. So, let me first show you a demo of this Wireshark and see how you can actually capture the packets and analyze individual packets in the network.

(Refer Slide Time: 01:34)



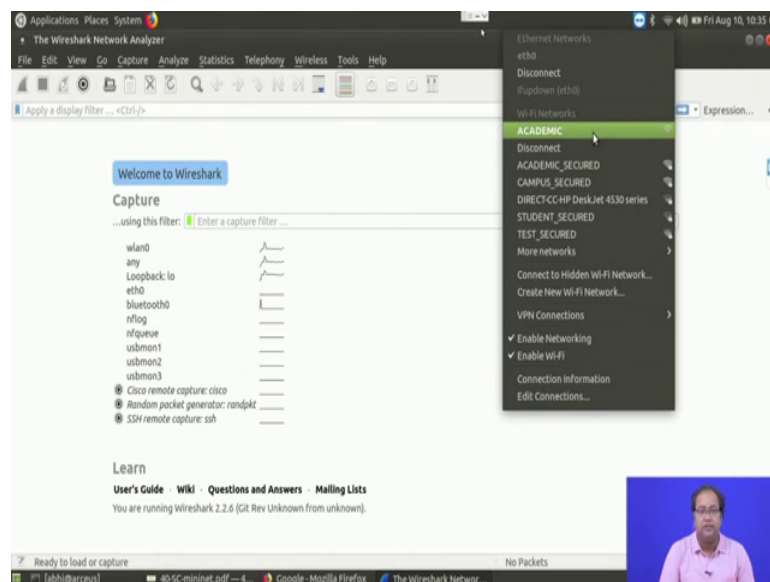
So, here that is this Wireshark interface. So, let me just open it from the scratch, so that the things become easier for you. So, we opened a Wireshark tool.

(Refer Slide Time: 01:54)



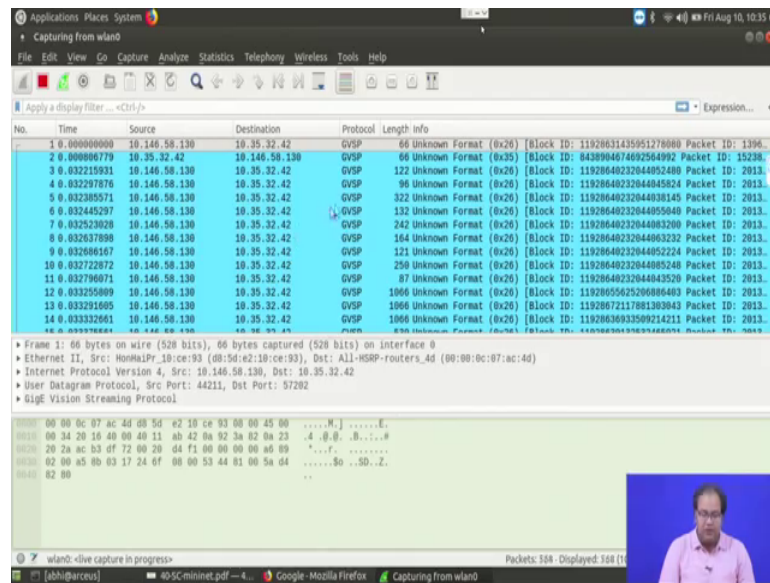
So, in that Wireshark tool this is the Wireshark home screen ok. So, here you can see all the interfaces which are there in this machine, where you will be able to capture the packets. Now this particular machine it is connected to the wireless LAN.

(Refer Slide Time: 02:15)



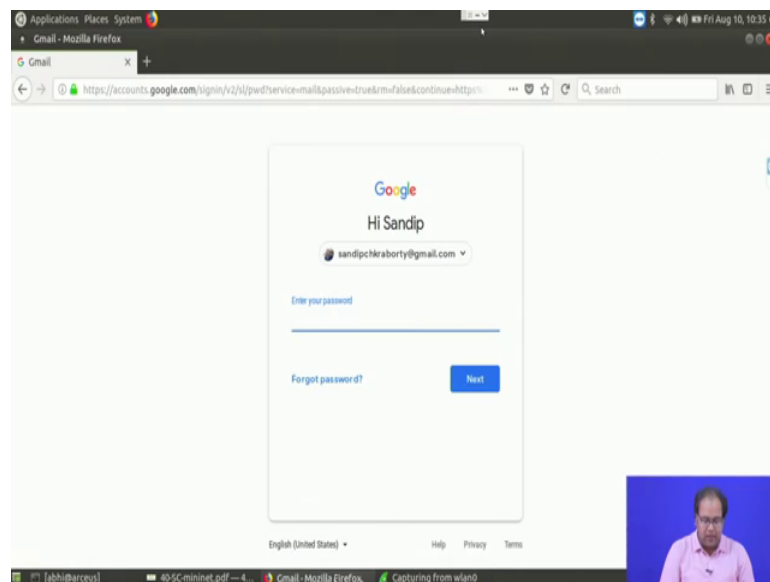
Here you can see it is connected to this academic SSID to the WI-FI router. So, we use this WLAN 0 interface, where it is receiving some packet here you can see that there is a small graph which is going on. So, it basically capture the packets which are there.

(Refer Slide Time: 02:40)



So, let us start capturing the packet in WLAN 0 interface. So, here it is capturing the packets in WLAN 0 interface.

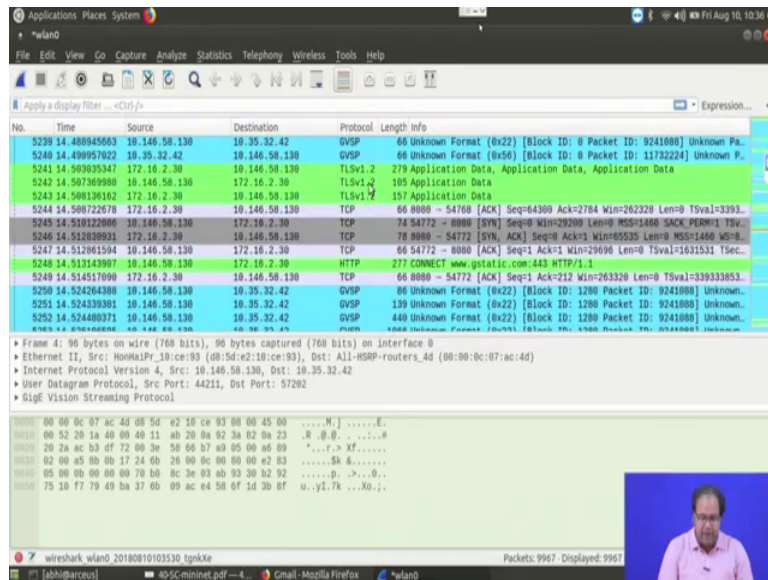
(Refer Slide Time: 02:48)



So, we will open some website. So, let us refresh this Google website or go to the Gmail website, so that we can get certain packets. Now come back to the Wireshark, stop the Wireshark interface and here you can see all the packet. So, you can see there are lots of packets where the protocol field.

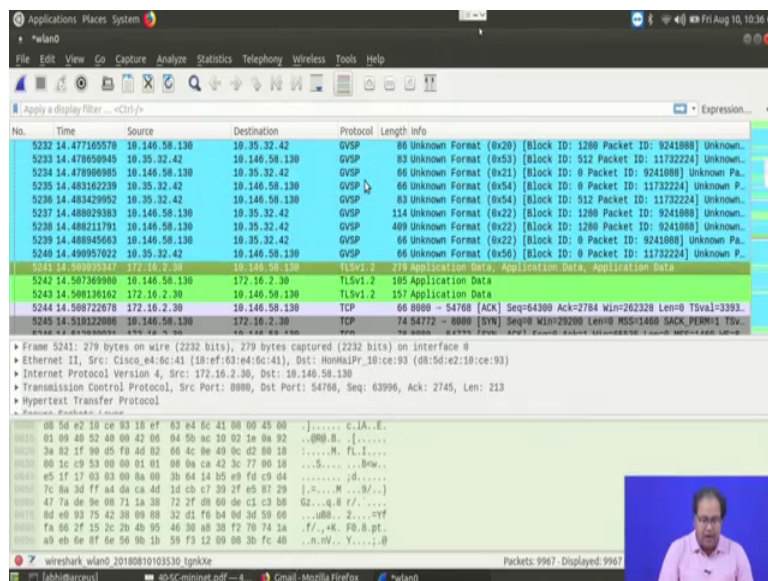
So, here we have the protocol field the protocol field is GBSP. So, this GBSP is something called GIE vision protocol which is used in tumblr kind of application which currently I am using for recording. So, it is capturing lot lots of such packets GBSP kind of packet it should also capture certain TCP packet.

(Refer Slide Time: 03:34)



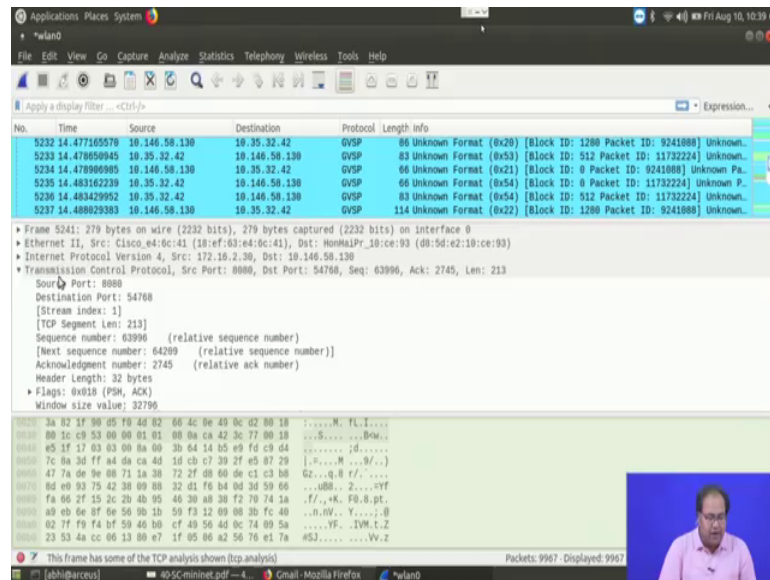
So, here is some TCP packets you can see so here are the TCP packets. Now whenever you are selecting one of these packets. So, let me choose one packet here.

(Refer Slide Time: 03:50)



So, the protocol it shows us TLS version 1.2 which is the transport layer security encrypted TLS encrypted packet. So, the Google whenever it sends the packet over the TCP protocol, it uses TLS to ensure the security at the transport layer.

(Refer Slide Time: 04:11)



Now, inside this packet if you look into this second window, this second window actually gives you the packet details at the different layers. So, this is a nice way to visualize the 5 layers of the TCP/IP protocol stack. So, here let us start looking into again this top down approach the way we are following the course.

So, where you can see that you have this SSL packet which is the encrypted data bits that we have, after that we have these HTTP header. So, in that HTTP header we are connecting to a proxy just contains the proxy information, because the packets that we are sending from this machine, it is sent to HTTP proxy server, and from that HTTP proxy server it is going to Google.

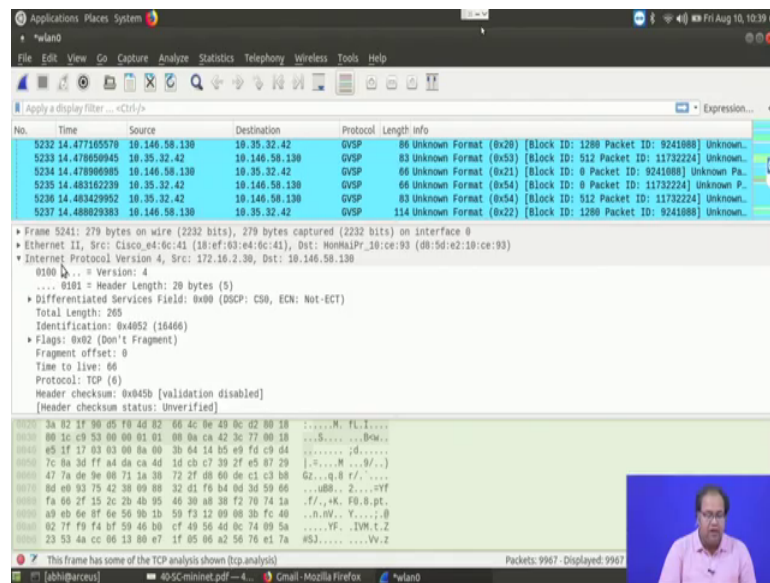
So, the packet which is sending to Google it is inside that secure socket layer that layer which is the encrypted data. So, you can see that it contains this application data protocol it says about HTTP over TLS. So, here it gives the application data the TLS version 1.2 the length, and the encrypted application data, so this is the encrypted part of the application data. So, there are three different TLS record blocks. So, the entire data is divided into three different TLS blocks and that contains the entire application data. Then this HTTP extension which contains the proxy information, then we have the

transmission control protocol at the TCP port. So, you can see that the TCP details are there.

So, here my source port is 8080, the destination TCP port is 54768, the stream index is something like 1, the single stream the segment lane it contains the TCP sequence number that we have seen for the transmission control protocol, the next sequence number, the acknowledgement number, and the header length, certain TCP flags.

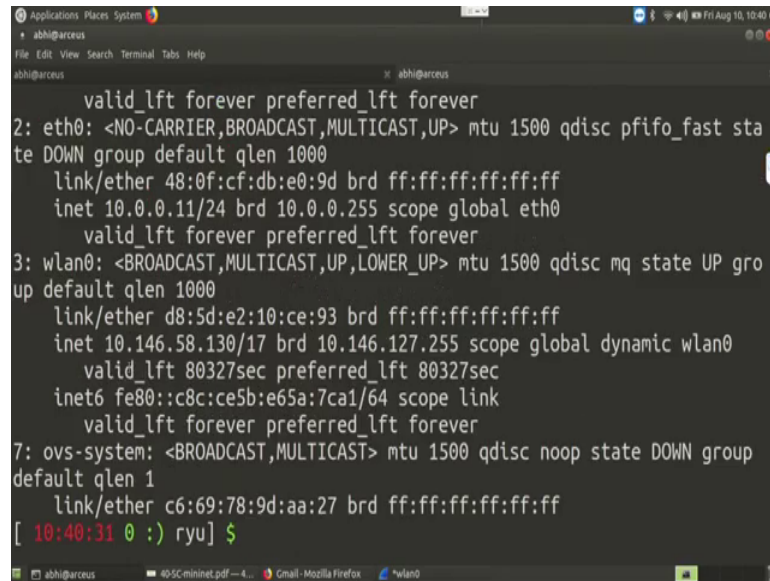
So, in the TCP header there were multiple flags. So, those flag bits are here the window size the receiver advertised window size, and accordingly the calculated window size. And the checksum field, the urgent pointer, then the TCP option field, and the sequence, and the acknowledgement field.

(Refer Slide Time: 06:38)



Then you can look into the IP header. The next layer is the IP header inside the IP header you can see that the source address and the destination address. Now the source address that I have 172 dot 16 dot 2 dot 30 that is the IP of the proxy address that we have in our IIT Kharagpur network. And the destination address is 10 dot 146 dot 58 dot 130 that is the address of this machine.

(Refer Slide Time: 07:07)

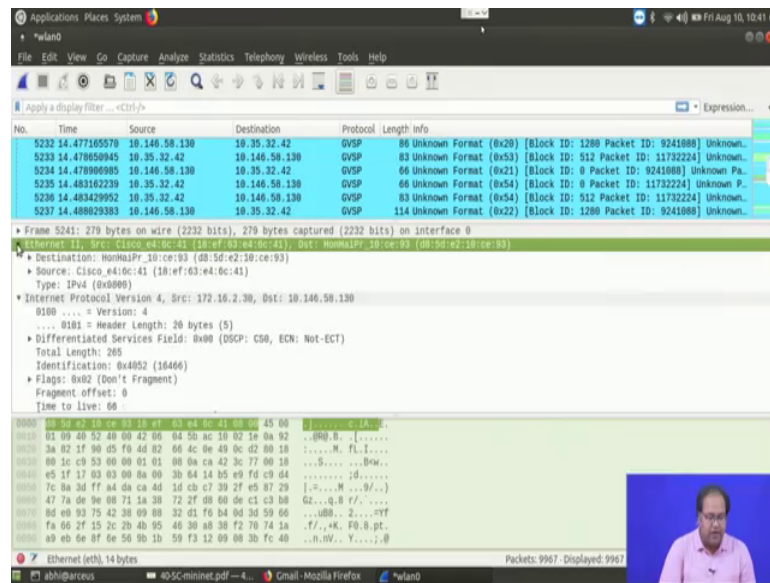


```
Applications Places System
abhi@arceus
File Edit View Search Terminal Tabs Help
abhi@arceus
valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast sta
te DOWN group default qlen 1000
link/ether 48:0f:cf:db:e0:9d brd ff:ff:ff:ff:ff:ff
inet 10.0.0.11/24 brd 10.0.0.255 scope global eth0
valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP gro
up default qlen 1000
link/ether d8:5d:e2:10:ce:93 brd ff:ff:ff:ff:ff:ff
inet 10.146.58.130/17 brd 10.146.127.255 scope global dynamic wlan0
valid_lft 80327sec preferred_lft 80327sec
inet6 fe80::c8c:ce5b:e65a:7ca1/64 scope link
valid_lft forever preferred_lft forever
7: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1
link/ether c6:69:78:9d:aa:27 brd ff:ff:ff:ff:ff:ff
[ 10:40:31 0 :) ryu] $
```

So, if you if you just try to see the IP of this machine you can see that say so in the ethernet address the loopback address well. So, here you can see that the WLAN address is it is connected to the wireless LAN interface. So, the address is 10 dot 146 dot 50 dot 130 the address of this machine. So, here also the destination address is 10 dot 146 dot 58 dot 130 the address of this machine.

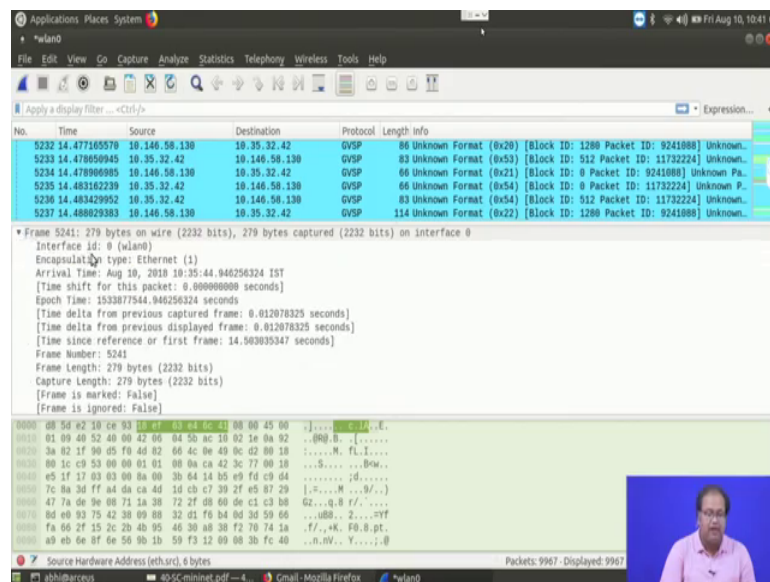
So, the packet has been received from the HTTP proxy that we have IIT, in IIT Kharagpur to this machine and the different field in the IP header. So the IP header length the flag bits in the IP header, the fragmentation information. Then the upper layer protocol, so it is using TCP protocol, then the source destination, this IP layer header information.

(Refer Slide Time: 08:28)



Then you have this ethernet information. So, the ethernet information you can see from here and finally, the link layer information.

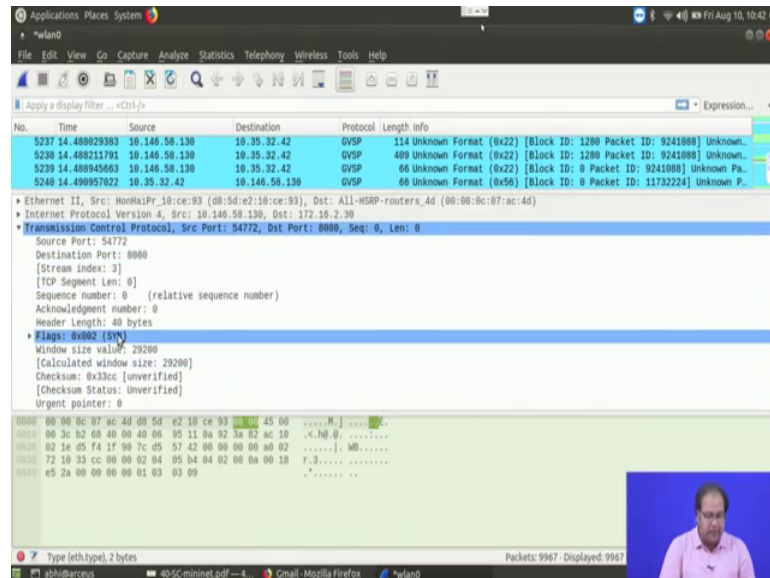
(Refer Slide Time: 08:34)



So, the data link layer has two sub part the logical link control and the Mac. So, this frame information that is coming from the Mac and this ethernet information coming from the LLC. It contains the packet arrival time, the epoch time, the frame length and different other fields which are there to indicate the link layer information. That way using Wireshark you can actually look into different type of packets say for example, you

can see that this is a TCP SYN packets. So, it is marked as a SYN. So, if you look into the TCP header for this packet, I will yeah the TCP header.

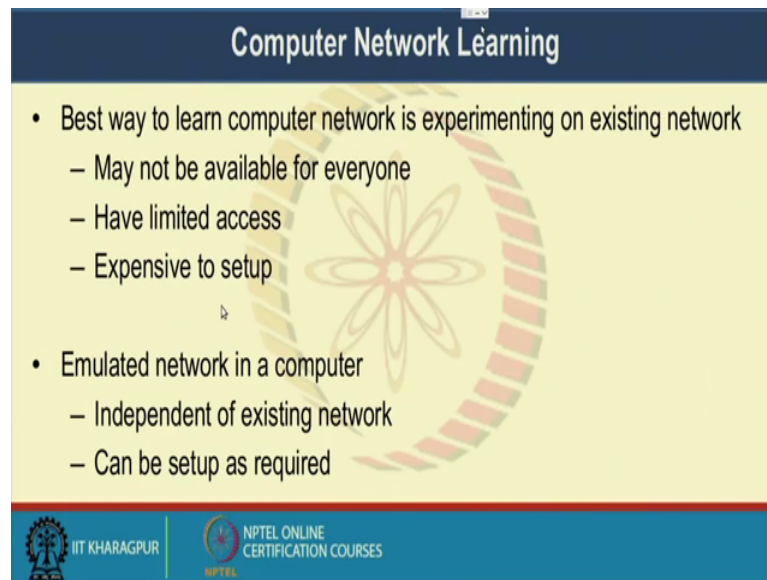
(Refer Slide Time: 09:24)



So, if you look into the TCP header for that packet you can see that the SYN bit is set. So, it is basically a SYN packet to initialize the TCP connection. So, you can see that a SYN is so here you can see interestingly the TCP three way handshaking mechanism. So, the SYN packet has been sent with sequence number 0 and certain window size, then you can see a SYN ACK; then followed by another ack. So, this three way handshaking is happening here.

So, that way using this Wireshark tool you can actually capture all the packets which is coming in your machine. And you can analyze them you can see what are the different packets coming to your machine and how to process those packet look into different header fields at a different layer of the protocol stack and explore it further ok. So, that is brief idea about how you can do the packet analysis using Wireshark ok.

(Refer Slide Time: 10:29)



Computer Network Learning

- Best way to learn computer network is experimenting on existing network
 - May not be available for everyone
 - Have limited access
 - Expensive to setup
- Emulated network in a computer
 - Independent of existing network
 - Can be setup as required

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will look into that how you can emulate a computer network in a single machine? So, that is the emulator platform which is again is done based tool that we are going to discuss in little details. So, in computer network the best way to learn a computer network is experimenting it on the existing network so that is always.

So, if you run your own protocol if you say design a protocol implement it and make a run on a on your network, so that is the best way to do. But the problem is that if this kind of existing network it may not be available for everyone. So, to get access to an existing network is a difficulty.

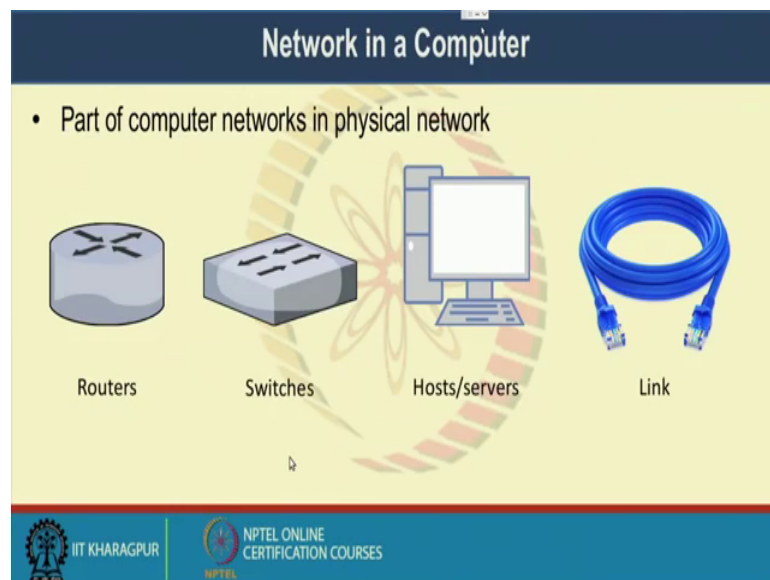
So, some time it may happen that you have a limited access to the network. For example, we have certain limited access in IIT Kharagpur network you cannot run anything over the IIT Kharagpur network. Because it is a public network and if you want to design your private network or want to set up your private network it is expensive to make a setup of your private network. So, that is why what we do? You try to emulate a network topology in a computer.

So, there are multiple simulation platform which has been used historically to understand the behavior of a computer network. But simulation platform has many limitation because it is not using the exact protocol stack which is running inside your machine. So, that is why many of the time a simulated network does not give you an ideal information

about how your protocol can perform in a real environment. But on the other hand the emulated network has that capacity.

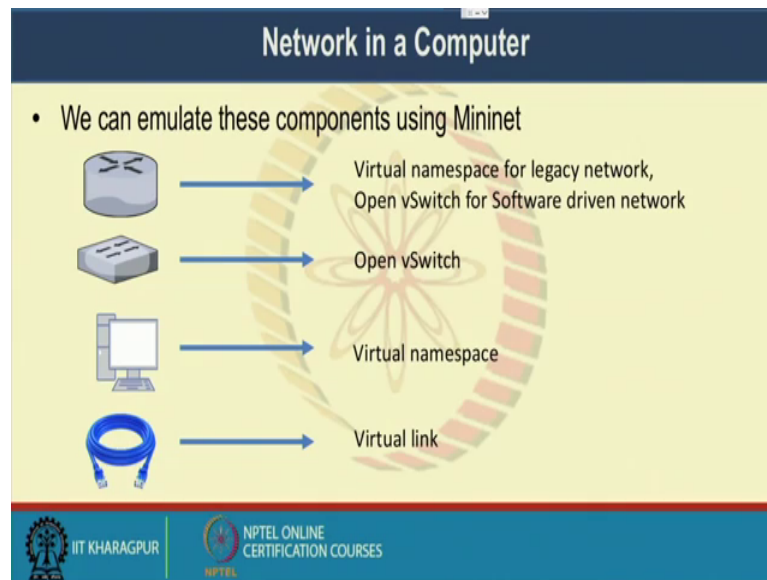
So, in a case of a emulated network the difference from the simulated network is that you are not simulating in a hypothetical, or a virtual environment rather what you are doing? You are utilizing the network protocol stack the implementation inside the kernel itself the actual implementation which is going to run in a real network. And on that immolated platform you are testing that how the performance of your network is going to be. So, the advantage is that it is independent of the existing network and it can be set up as required ok.

(Refer Slide Time: 12:38)



Now, here are the different parts or different components of computer networks in a physical network. So, you have the routers, you have the switches, you have different host and the server and you have the link.

(Refer Slide Time: 12:53)



Now, in a virtual domain, or an emulated domain whenever we are emulating it using this mini net platform. So, we call mini net as a network inside the computer, a emulated network inside the computer. So, these routers are implemented using called something called a virtual namespace for legacy network or open v switch for software different given network.

So, Open vSwitch is a tool chain which provides switch implementation in an open platform open platform, or open source platform. You have that Open vSwitch implementation and using Open vSwitch you can emulate a switch using the kernel protocol stack which is there in your Linux operating system. Then a switch can again be emulated using a Open vSwitch platform, host can be emulated using a virtual namespace a namespace is basically instance of the protocol stack which works like a individual hosts.

So, you have this entire protocol stack implementation inside your computer. Now you are creating a virtual instance of that protocol stack and emulating it at as an individual host. So, this entire architecture you can just think of the way we do the operating system level virtualization. So, I think that you have heard about this kind of virtual machine, and the tools like virtual box.

So, in a tool like virtual box what we do? We do the operating system level virtualization. So, you have this virtual box on top of you can have multiple vmswitch

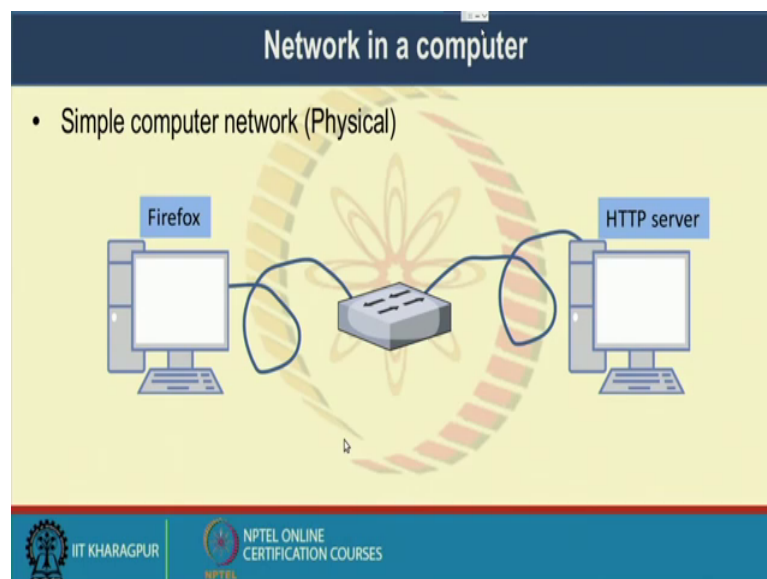
are running, and inside every vm you can run one different operating system. So, one vm can host a ubuntu operating system, another vm can host a say windows operating system, a third vm can host a feather operating system. And all this thing can run on top of a host operating system.

In a similar way here we are emulating the network using this virtual namespace, and a virtual switch concept where the network protocol stack implementation is there inside your kernel. And we are creating a virtualized instance of that protocol stack. So, whenever you are creating constructing a virtual host; that means, you are creating you are taking a virtual instance of the entire TCP IP protocol stack of the 5 layers and considering it as a virtual namespace. So, the term namespace actually indicates a virtual instance of this end to end protocol stack.

So, you are taking a virtual instance of that and considering it as an as an a individual host. Now if we are going to implement a switch or a router then at the layer three of the protocol stack you need to run the routing functionalities, or at layer two of the protocol stack you have to run the switching functionalities, or layer two functionalities, so that you can implement with the help of this Open vSwitch.

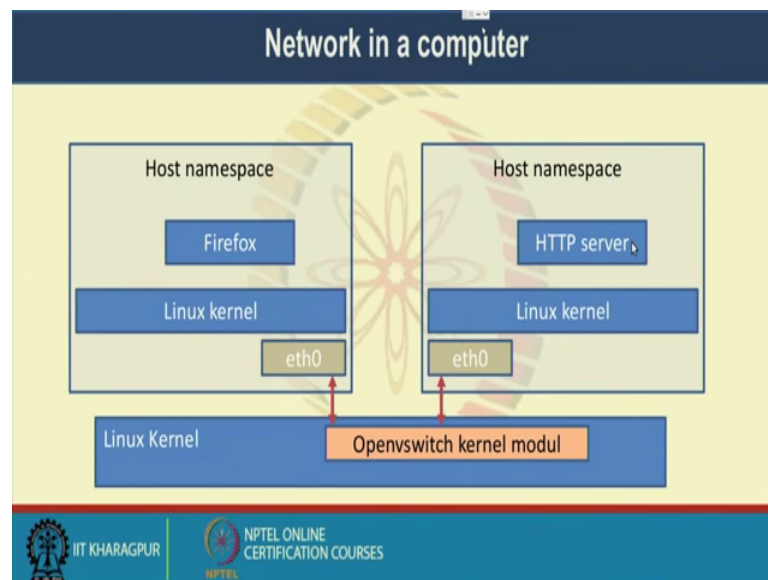
So, the Open vSwitch will adopt the virtual switching functionalities or the routing functionalities on top of that namespace the protocol stack namespace. And then you can emulate the links the physical links using virtual links.

(Refer Slide Time: 15:49)



Now, this is a kind of simple computer network in the physical domain you have one host which is running say a browser like Firefox, it is connected to a network switch or a router that is again connected to a HTTP server. So, using the browser you can browse the data from the HTTP server.

(Refer Slide Time: 16:08)


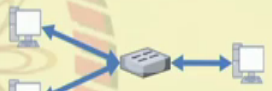
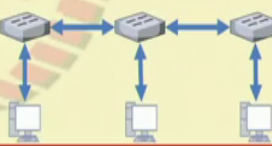




Now, the same thing you can implement inside a single machine. So, here you have your Linux kernel; in that Linux kernel you have this open vswitch kernel module which runs the switching functionalities by taking a virtualized instance of that TCP IP protocol stack and then you have two different namespaces host namespaces. So, these two different host name paces, again have a virtual instance of these 5 layers of the TCP IP protocol stack, and they are in the application site you are running a Firefox, then you have a Linux kernel which has this virtual implementation of the protocol stack and then the ethernet 0 which is a virtual link which is connected to this Open vSwitch kernel module.

So, it is connected with this logical switch at the virtual switch and the other host name space you have HTTP server running at the application and the remaining part of the protocol stack along with the virtual link through this eth 0 which is being connected. So, that way the physical network you can implement in a machine using this virtual instance instances of the network.

(Refer Slide Time: 17:16)

Network in a computer

- Use Mininet to create the network
 - \$ mn --topo single,2 
 - \$ mn --topo single,3 
 - \$ mn --topo linear,3 

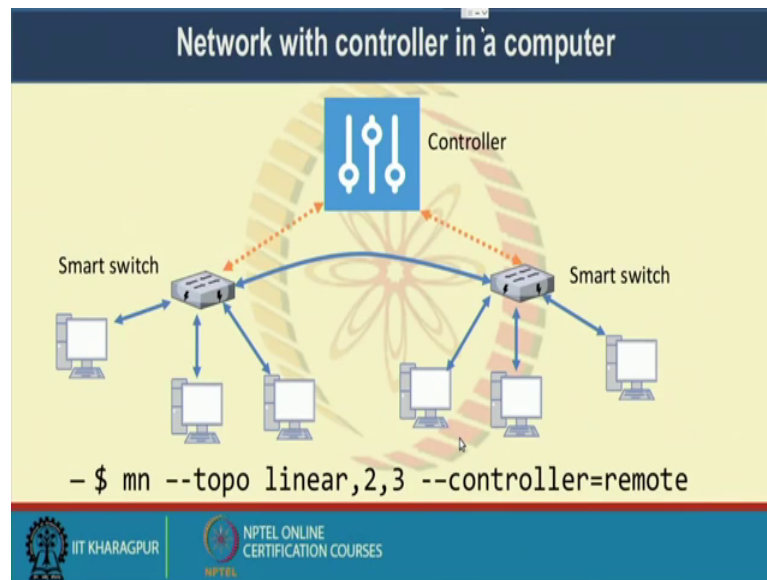
 

So, now how you can create such kind of topologies in a network in a computer, we can use the mini net tool I will show you a demo of that mini net tool, but before going to that just showing you some simple comments inside the mini net tool.

So, this mini net tool you can this is a open source tool, you can install it from the mini net website. So, from the mini net website you can even get the image under different kind of operating system or you can also get the source, you can compile it from the source and install it to your Linux based machine. So, in the mini net command if you type the command like mn; mn is corresponds to the mini net minus topo single 2 what it will do? It will create a topology like this; it will have a single instance of the switch and two different hosts.

So, if you make it mn minus minus topo, single 3, then you have a single switch with three different hosts. If you make it as mn minus minus topo linear 3, it will create a linear topology of the three switches and one host will connect with each of the switch. So, this is the topology corresponds to that.

(Refer Slide Time: 18:27)



And then if you want to create say a complicated topology, so, here what we are doing that we are creating a topology like this linear 2, 3 and this is a kind of SDN topology that we are going to implement. In the last lecture we have discussed about this SDN architecture we have the switches, and the switches are connected to a controller. So, that thing we are going to emulate here using this SDN mini net networking platform.

So, what we are going to do, we are having this mn minus minus topo; linear 2, 3, linear 2, 3 means you have a linear topology of two switches which are being connected and three host are connected with every individual switches, and then we have specifying minus minus controller equal to remote. That means, we are having a controller which is there in the remote machine and that controller is connected to the switches. Now in that controller you have to load individual controller software.

So, in the last class we are discussing that there can be multiple such controller platforms like ryu like that forks like open daylight, like floodlight, there are different kind of controllers you can pick up your favorite controllers and attach it with this virtual controller that you are designed. And then with that virtual controller you can actually try to do the experiments by setting up by writing your code inside the controller, by writing your network application inside the controller and then running it over this kind of emulated network.

So, now let us go for a demo of this entire procedure.

(Refer Slide Time: 20:04)

```
Applications Places System
+ abhi@barceus
File Edit View Search Terminal Tabs Help
abhi@barceus
valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast sta
te DOWN group default qlen 1000
link/ether 48:0f:cf:db:e0:9d brd ff:ff:ff:ff:ff:ff
inet 10.0.0.11/24 brd 10.0.0.255 scope global eth0
valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP gro
up default qlen 1000
link/ether d8:5d:e2:10:ce:93 brd ff:ff:ff:ff:ff:ff
inet 10.146.58.130/17 brd 10.146.127.255 scope global dynamic wlan0
valid_lft 80327sec preferred_lft 80327sec
inet6 fe80::c8c:ce5b:e65a:7ca1/64 scope link
valid_lft forever preferred_lft forever
7: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1
link/ether c6:69:78:9d:aa:27 brd ff:ff:ff:ff:ff:ff
[ 10:40:31 0 :) ryu] $
```

So, what we are going to do is first we.

(Refer Slide Time: 20:09)

```
Applications Places System
+ abhi@barceus
File Edit View Search Terminal Tabs Help
abhi@barceus
mn: error: no such option: --controller
[ 10:54:34 2 :( ryu] $ sudo mn --topo single,3 --mac --controller remote --switch ovs
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

So, So, first we will run mini net instance with we create a topology of a single switch and three different host. So, let us do it pseudo mn minus minus topo. So, you have to run it in the pseudo instances, because it run as a ryu you are going to access the kernel protocol stack.

So, that is you require the root access, single comma 3 the way I have shown you earlier like we have a single switch with three different hosts connected to that switch. Then

minus minus mac, minus minus controller remote, minus minus switch ovsk. So, here it says that I am going to have a controller which is now going to be connected with the switch and those which are of type ovsk switches.

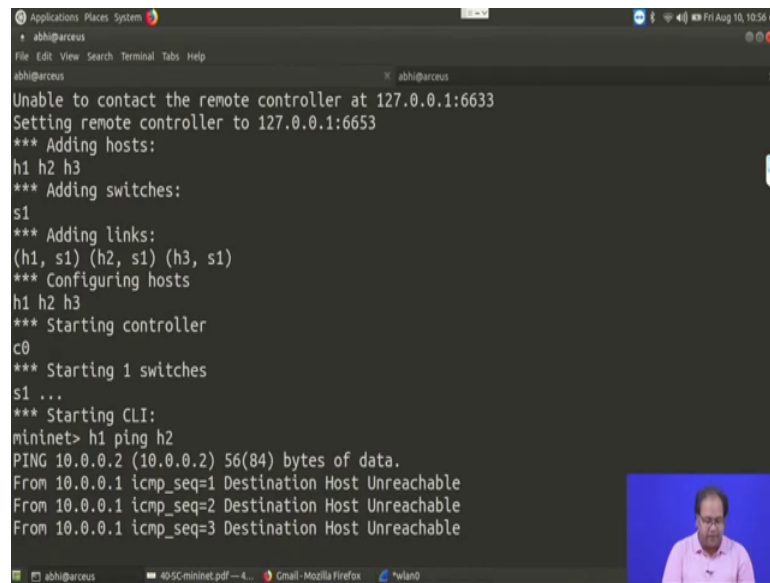
So, I have to give the root password oh sorry I have made a typo here it should be controller ok. Now you can see here what has happened first whenever it is trying to add the controller, it was not able to contact the remote controller at the local machine. So, we are saying that we are going to run the controller in the local machine.

So, the controller normally runs in two different out of the two different port 6653 or 6633. So, it is searching for the controller, but currently we have not executed any controller. So, it was not able to find the controller and it has added three different host h 1 h 2 and h 3 and added a switch called s 1 and the links are h 1 to s 1, h 2 to s 1 and h 3 to s 1 a kind of star topology. So, three hosts are connected to one switch.

So, it has configured the three hosts, started the controller, but the controller it was not able to connect and one switch has been started. So, now, you got the mini net console here, now from that mini net console if you try to say ping something. So, we make the command as h 1 ping h 2.

So, whenever we write h 1 ping h 2; that means, from the virtual namespace of h 1, the protocol stack which is there the actual protocol stack which is there from there we are going to execute the ping command and we are trying to ping the host h 2. So, here if you try to ping it you can see that it is not getting pinged.

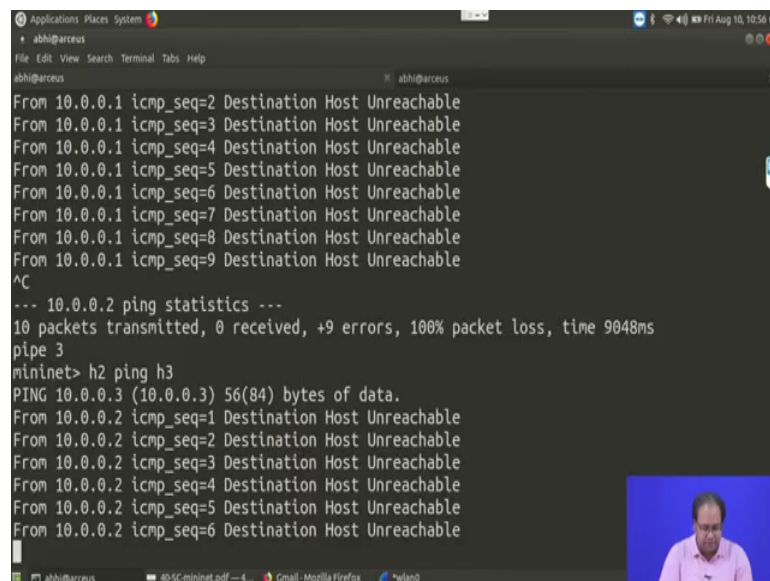
(Refer Slide Time: 23:22)

A terminal window showing the configuration of a Mininet network. The user sets a remote controller to 127.0.0.1:6653, adds hosts h1, h2, h3 and switch s1, and configures links between them. After starting the controller and switches, the user runs 'mininet> h1 ping h2'. The output shows three failed ping attempts from 10.0.0.1 to 10.0.0.2, all resulting in 'Destination Host Unreachable'.

```
abhi@arceus
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
```

So, it says that the destination host is unreachable.

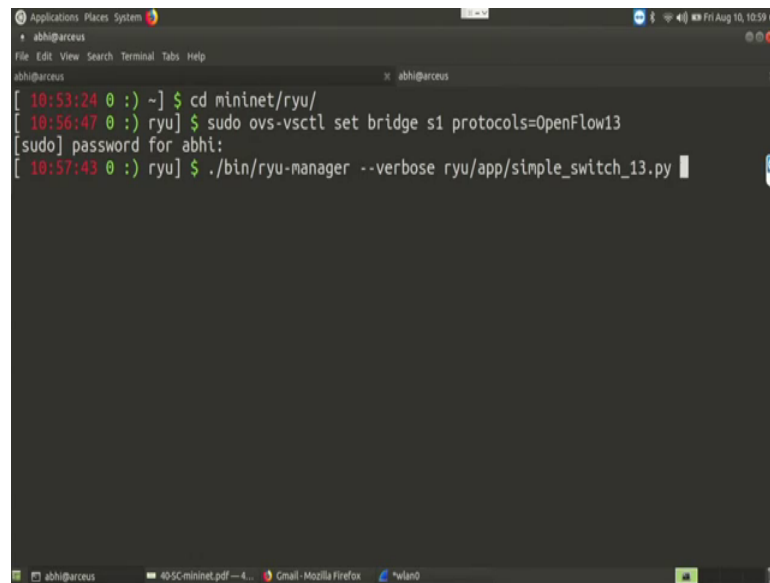
(Refer Slide Time: 23:29)

A terminal window showing the results of a ping command from host h1 to host h2. The output shows 10 failed ping attempts from 10.0.0.1 to 10.0.0.2, all resulting in 'Destination Host Unreachable'. The user then runs 'mininet> h2 ping h3'. The output shows six failed ping attempts from 10.0.0.2 to 10.0.0.3, all resulting in 'Destination Host Unreachable'.

```
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time 9048ms
pipe 3
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
From 10.0.0.2 icmp_seq=5 Destination Host Unreachable
From 10.0.0.2 icmp_seq=6 Destination Host Unreachable
```

Similarly, if you try to ping from say h 2 ping h 3, none of the machines will get pinged it says destination host unreachable.

(Refer Slide Time: 23:46)



```
abhi@arceus
[ 10:53:24 0 :) ~] $ cd mininet/ryu/
[ 10:56:47 0 :) ryu] $ sudo ovs-vsctl set bridge s1 protocols=OpenFlow13
[sudo] password for abhi:
[ 10:57:43 0 :) ryu] $ ./bin/ryu-manager --verbose ryu/app/simple_switch_13.py
```

Now, let us run the controller. So, what will do under this mini net directory, we have we are going to use the ryu controller.

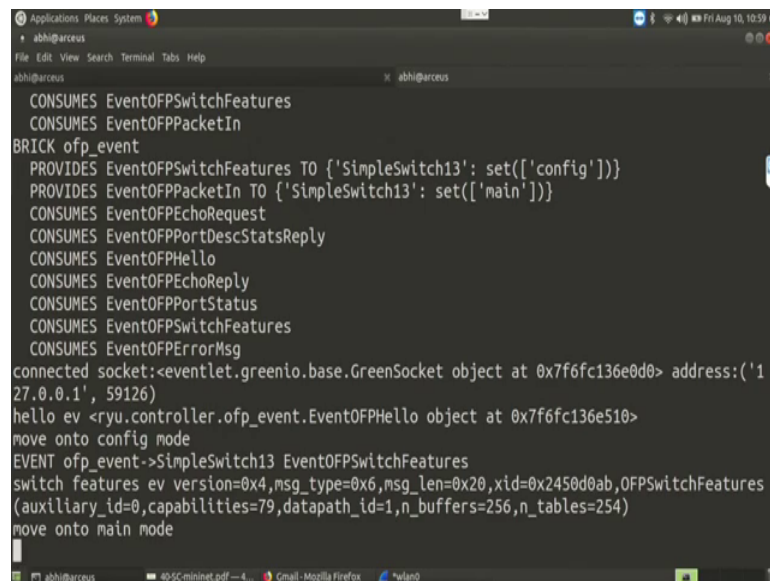
So, we are going to the directory ryu and starting the controller. So, ovs vsctl. So, this ovs vsctl command is used to start a controller and attach it with a corresponding ovs switch set bridge, we are trying to set the controller in the bridge mode and it will be connected with s 1. So, s 1 is going to work as a bridge mode with which the controller is getting connected 10 protocols equal to open flow 1 3.

So, we are specifying that we are going to use open flow version 1.3 at my protocol smrl. So, the password now we are going to start the controller. So, what we have done here we have with this ovs vsctl command with this s 1 which is working as the bridge mode with that we have configured t to hit this open flow version 1.3 protocol stack, now we are going to run the switch run the controller. So, to run the controller we are going to ryu manager, going to run ryu manager in the verbose mode.

So, that we can see what is going to happen here and the controller program that we are going to run. So, on the controller you have to run certain applications. So, that application will take care of configuring your switches that we have learned in the last class, it will configure the switch and it will install the forwarding rules inside that switch.

So, here we have written a python script, which is actually a default python script used inside the ryu controller and that python script actually works like application of a forwarding manager. It helps you to forward the packet from one machine to another machine. So, we are going to run that one, it is simple switch with version 13 dot p y. So, that is the python application which you have written or indeed it was a default application in ryu; once you install ryu you can get that as well. So, that particular application we are going to run here ok.

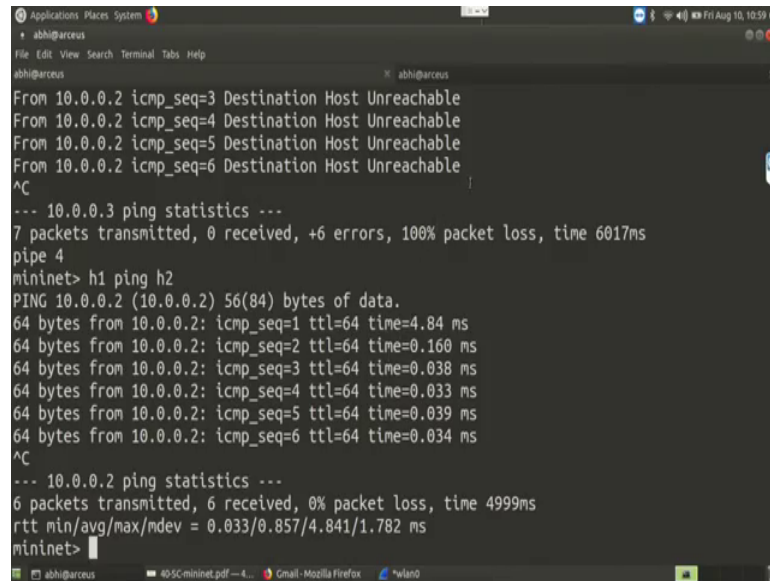
(Refer Slide Time: 26:33)



```
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPPacketIn
BRICK ofp_event
PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
CONSUMES EventOFPEchoRequest
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPHello
CONSUMES EventOFPEchoReply
CONSUMES EventOFPPortStatus
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPErrormsg
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f6fc136e0d0> address:('1
27.0.0.1', 59126)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f6fc136e510>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x2450d0ab,OFPSwitchFeatures
(auxiliary_id=0,capabilities=79,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
```

So, it has executed that one, and after that it has getting connected with the corresponding switch now, let us try to run it h 1 ping h 2.

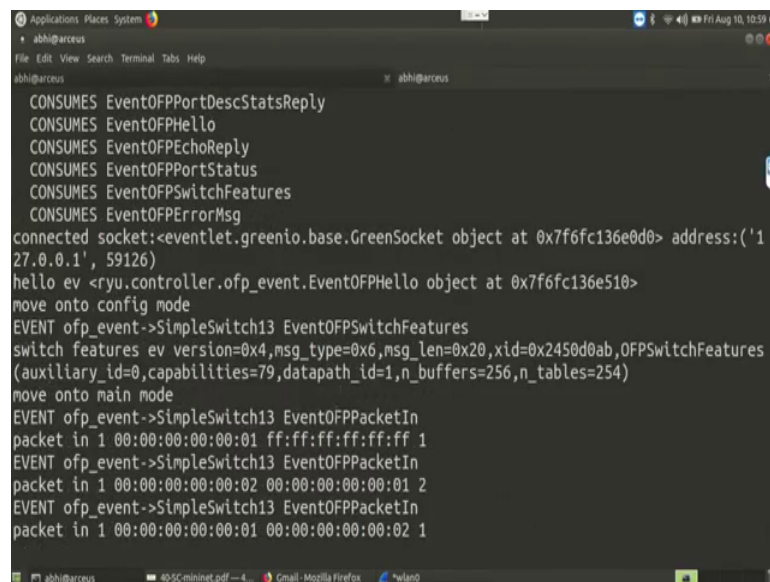
(Refer Slide Time: 26:48)



```
abhi@arceus
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
From 10.0.0.2 icmp_seq=5 Destination Host Unreachable
From 10.0.0.2 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.3 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6017ms
pipe 4
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4.84 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.160 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.034 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.033/0.857/4.841/1.782 ms
mininet>
```

Now you can see it is getting pinged and when it is getting pinged let us look into few events which are happening here.

(Refer Slide Time: 27:03)

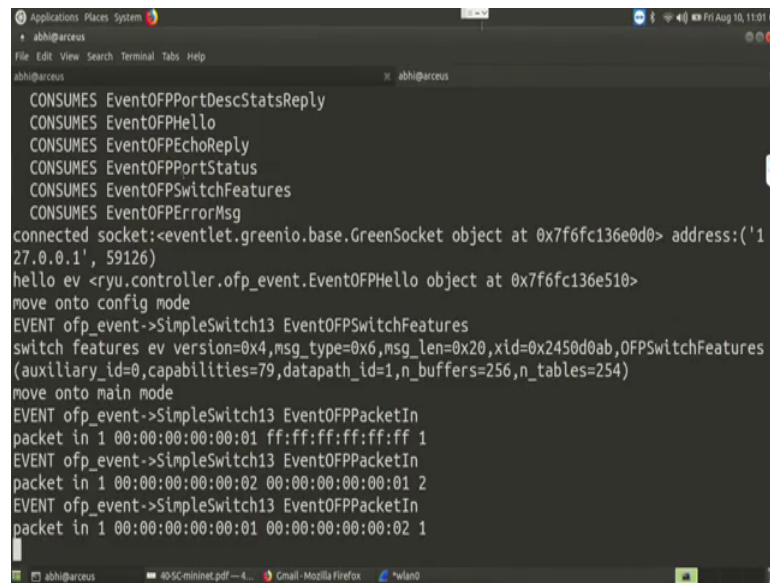


```
abhi@arceus
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPHello
CONSUMES EventOFPEchoReply
CONSUMES EventOFPPortStatus
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPErrormsg
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f6fc136e0d0> address:('1
27.0.0.1', 59126)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f6fc136e510>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x2450d0ab,OFPSwitchFeatures
(auxiliary_id=0,capabilities=79,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

Here you can see there was some event which has been locked. So, this event you can see certain packets are coming to the controller and based on that packet, it is configuring the corresponding switches.

So, the controller events are being logged here.

(Refer Slide Time: 27:23)

A terminal window showing network-related events. The output includes several 'CONSUMES' messages for event types like EventOFPPortDescStatsReply, EventOFPHello, EventOFPEchoReply, EventOFPPortStatus, EventOFPSwitchFeatures, and EventOFPErrormsg. It then shows a 'connected socket' message, a 'hello ev' message, and a 'move onto config mode' message. This is followed by an 'EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures' message with detailed parameters. Finally, it shows 'move onto main mode' and three 'EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn' messages, each with a timestamp and packet details like 'packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1'.

And here it is getting pinged and we have an interesting observation here, if you look into the response time of the switches. So, you can see that the first packet that was sent it has a longer time. It has taken a time of 4.84 milli second whereas, the remaining ping packet it has took around 0.16 millisecond and 0.03 millisecond, 0.02 milliseconds something like that, but the first one has taken certain more time why that is so?

If you remember in the last class that I have discussed, that how this entire controller architecture is going to work for the first packet whenever it reaches to the switch, the switch does not have any information about how to process that packet or how to forward that packet. So, what the switch has done the switch will send or generate a open flow event, which will reach to the controller. So, the event we can see in the other tab the open flow event that have been generated.

So, this open flow events will be generated and it will be sent to the corresponding switch, and then that particular switch will send that event to the corresponding controller, the ryu controller application that we are running. So, that particular application the switching application, it will generate the rules and configure the switch with that particular rule and then the packet will get forwarded and during that in between time, the packet will remain inside the buffer of the switch

So, for the initial packet we see a certain longer delay for, but for the remaining packet that delay shall less.

(Refer Slide Time: 29:01)

```
Applications Places System
+ abhi@arceus
File Edit View Search Terminal Tabs Help
abhi@arceus
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.033/0.857/4.841/1.782 ms
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.207 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.028 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.030 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.037 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.033 ms
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.028/0.052/0.207/0.052 ms
mininet> h2
```

Again if I run it, you can see that the delays is comparatively lesser. Only for the first time it has took that initial longer time. Similarly now in this case if I run it in a different case.

So, earlier I have done h 1 ping h 2, now say run it from the h 2 host.

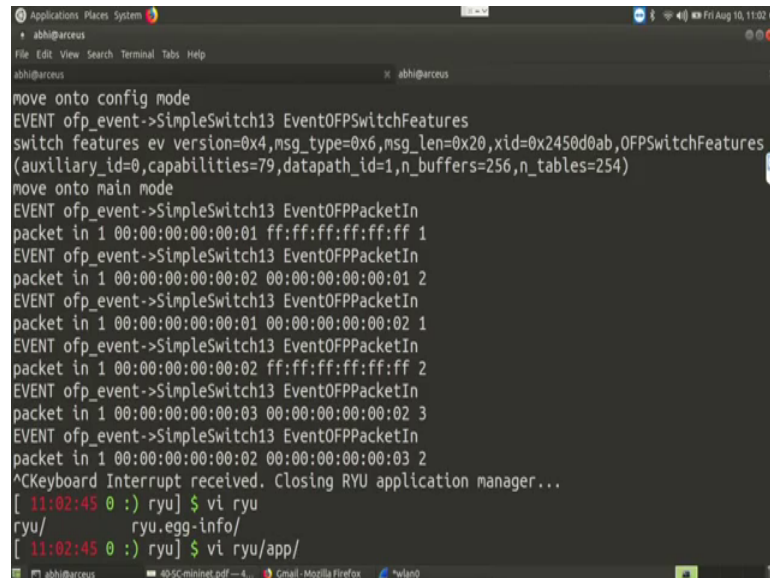
(Refer Slide Time: 29:25)

```
Applications Places System
+ abhi@arceus
File Edit View Search Terminal Tabs Help
abhi@arceus
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.028 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.030 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.037 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.033 ms
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.028/0.052/0.207/0.052 ms
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=4.21 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.155 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.033 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.055 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.030/0.896/4.211/1.658 ms
mininet>
```

So, if we run the ping from h 2 host to h 3, again you can see that the first packet has took some longer time to forward the things. So, that way you can actually run this entire

controller and the switches and emulate the topology by using this mini net emulator network emulator platform.

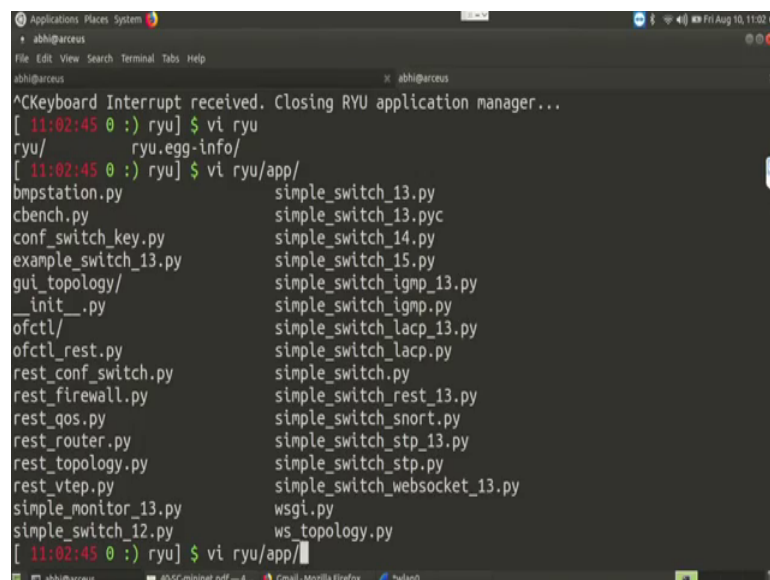
(Refer Slide Time: 29:44)



```
Applications Places System
+ abhi@barceus
File Edit View Search Terminal Tabs Help
abhi@barceus
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x2450d0ab,OFPSwitchFeatures
(auxiliary_id=0,capabilities=79,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:03 00:00:00:00:00:02 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:03 2
^CKeyboard Interrupt received. Closing Ryu application manager...
[ 11:02:45 0 :) ryu] $ vi ryu
ryu/ ryu.egg-info/
[ 11:02:45 0 :) ryu] $ vi ryu/app/
```

Now, briefly and you can see here that all these events have again executed for different notes. Now let us look into the application that we had written in python. So, I will quickly show you the application which is there.

(Refer Slide Time: 30:06)



```
Applications Places System
+ abhi@barceus
File Edit View Search Terminal Tabs Help
abhi@barceus
^CKeyboard Interrupt received. Closing Ryu application manager...
[ 11:02:45 0 :) ryu] $ vi ryu
ryu/ ryu.egg-info/
[ 11:02:45 0 :) ryu] $ vi ryu/app/
bmpstation.py simple_switch_13.py
cbench.py simple_switch_13.pyc
conf_switch_key.py simple_switch_14.py
example_switch_13.py simple_switch_15.py
gui_topology/ simple_switch_igmp_13.py
__init__.py simple_switch_igmp.py
ofctl/ simple_switch_lacp_13.py
ofctl_rest.py simple_switch_lacp.py
rest_conf_switch.py simple_switch.py
rest_firewall.py simple_switch_rest_13.py
rest_qos.py simple_switch_snort.py
rest_router.py simple_switch_stp_13.py
rest_topology.py simple_switch_stp.py
rest_vtep.py simple_switch_websocket_13.py
simple_monitor_13.py wsgi.py
simple_switch_12.py ws_topology.py
[ 11:02:45 0 :) ryu] $ vi ryu/app/
```

So, inside the app directory you can see that there are multiple applications which are there. So, you can actually play with these applications which are there and they will

start writing your own application using this python programming. So, simple switch 13 dot py ok.

(Refer Slide Time: 30:30)

```
Applications Places System
+ abhi@arceus
File Edit View Search Terminal Tabs Help
abhi@arceus x abhi@arceus
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        42,1 22%
```

So, here what we basically do? A simple switch thirteen class has been defined and inside that, we are defining different functionalities. The initial the initialized functionalities the a switch feature handler which handle different features inside the switch and then the interesting part is this add flow things.

(Refer Slide Time: 30:50)

```
Applications Places System
+ abhi@arceus
File Edit View Search Terminal Tabs Help
abhi@arceus x abhi@arceus
# correctly. The bug has been fixed in OVS v2.1.0.
match = parser.OFPMatch()
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

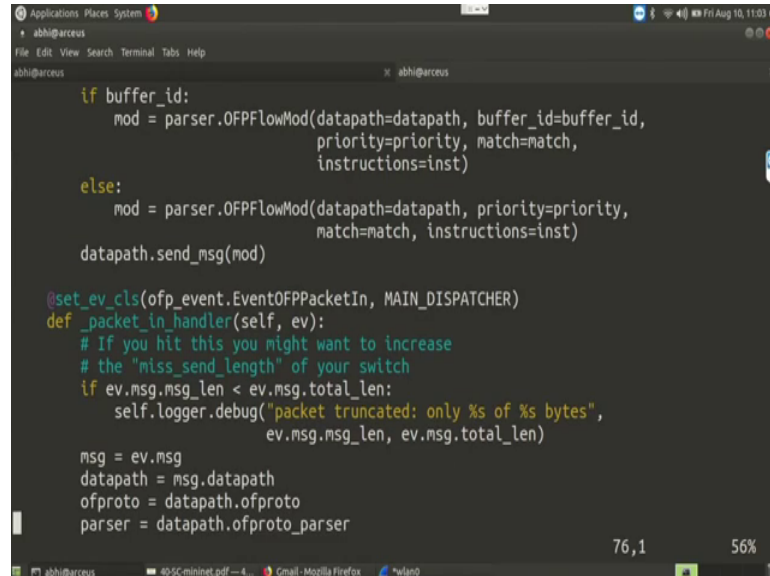
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                       actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)
55,1 44%
```

So, this add flow will add a new rule corresponds to a new flow. So, what it will do. So, this add flow it will call this packet in handler.

(Refer Slide Time: 31:07)



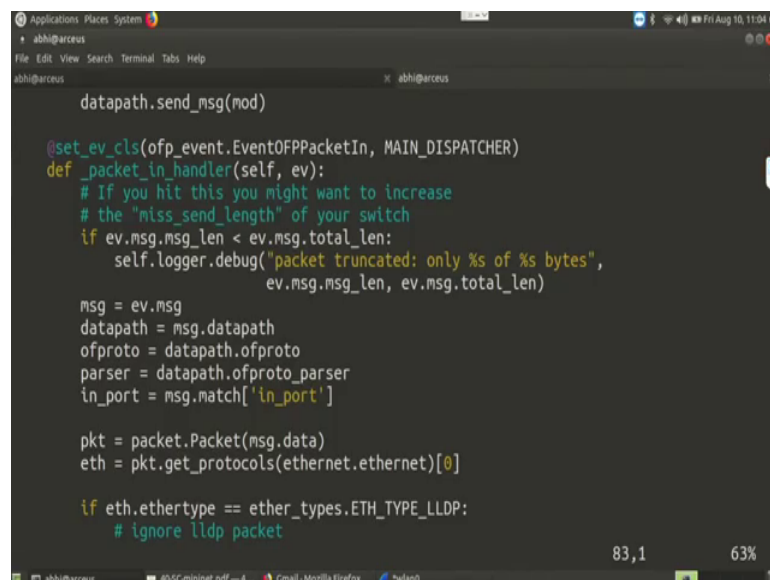
```
if buffer_id:
    mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                            priority=priority, match=match,
                            instructions=inst)
else:
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst)
datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

So, this packet in handler actually handles one open flow packet. So, whenever our packet in event occurs; that means, a packet is waiting at the switch and you have received that packet in event at the controller side.

(Refer Slide Time: 31:23)



```
datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

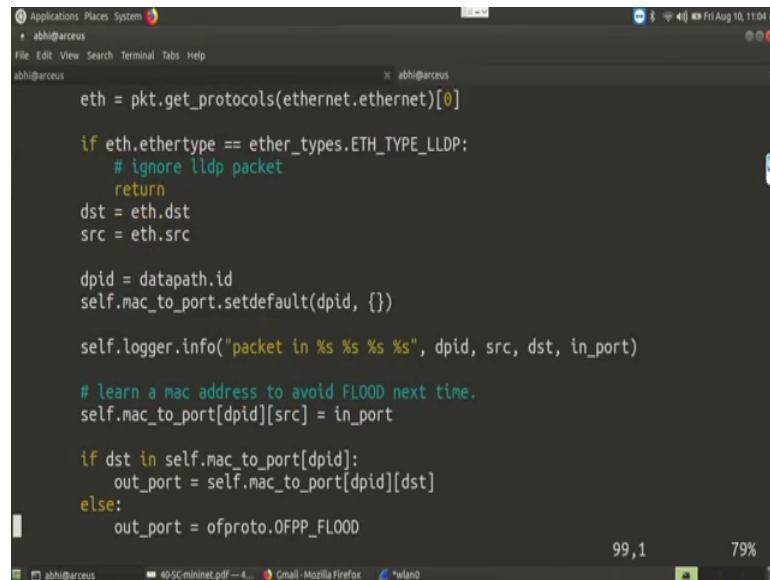
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
```

So, what we are actually doing. So, you can see that we are extracting the port; we are extracting the packet parameters, the packet equal to packet message the data then the ethernet header.

(Refer Slide Time: 31:35)

A screenshot of a terminal window on a Linux system. The terminal shows Python code for processing an Ethernet packet. The code starts by getting the protocols for the ethernet interface. It then checks if the packet type is LLDP; if so, it returns. Otherwise, it extracts the source and destination addresses. It then gets the datapath ID and sets a default for the mac_to_port dictionary. It logs the packet information and updates the dictionary with the source address. Finally, it checks if the destination address is in the dictionary; if so, it returns the output port, otherwise, it returns the flood port.

```
eth = pkt.get_protocols(ethernet.ethernet)[0]

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return
dst = eth.dst
src = eth.src

dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

self.logger.info("packet in %s %s %s", dpid, src, dst, in_port)

# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD
```

From the ethernet type, we are looking into the type and then we are taking the source address and the destination address from the ethernet interface. We are looking into the data part id and then generating the forwarding rule. So, here the forwarding rules are generated.

So, we are learning the MAC address. So, here we are doing the forwarding. So, the forwarding is dumped at the MAC address, in the later lecture you will learn how to how you can use MAC address for do the forwarding. So, these forwarding rules are being generated.

So, based on that we are generating the output port; so, where the packet will be forwarded next. And accordingly the action has been defined and then we are installing a new rule to in the switch.

(Refer Slide Time: 32:25)

```
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

actions = [parser.OFPActionOutput(out_port)]

# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
```

So, these new rules or this with this with this add flow command in the data path, the new rule is being installed inside the switch.

(Refer Slide Time: 32:34)

```
actions = [parser.OFPActionOutput(out_port)]

# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```

So, then we are making a packet out event.

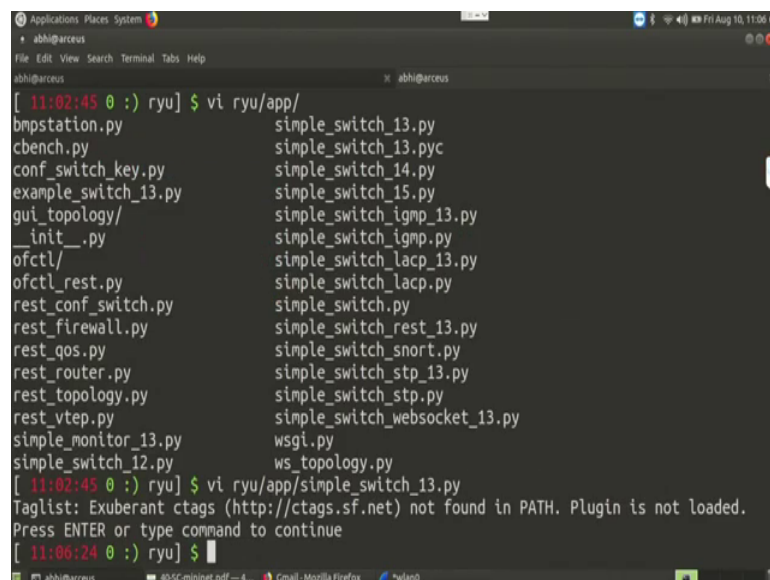
So, this packet out event will actually send the information from the controller to the switch and it will send that message to the data path from the data path. So, that way so, it has its own construct you have to learn that what are the different functions or

different classes which are being available in the mini net and accordingly try to learn this program.

So, So, I will suggest you to explore this further to explore this code if you are familiar with the python language, and start a, with playing with this kind of code. And if you are not familiar with python language, do not worry there are other controller as well.

So, for example, there are this open daylight controller which is written in java, you can try with those controller as well. So, you can choose your controller which is preferable to you and start playing with that.

(Refer Slide Time: 33:32)



```
abhi@arceus [ 11:02:45 0 :) ryu] $ vi ryu/app/
bmpstation.py          simple_switch_13.py
cbench.py              simple_switch_13.pyc
conf_switch_key.py     simple_switch_14.py
example_switch_13.py   simple_switch_15.py
gui_topology/         simple_switch_igmp_13.py
__init__.py           simple_switch_igmp.py
ofctl/                simple_switch_lacp_13.py
ofctl_rest.py         simple_switch_lacp.py
rest_conf_switch.py   simple_switch.py
rest_firewall.py      simple_switch_rest_13.py
rest_qos.py           simple_switch_snort.py
rest_router.py        simple_switch_stp_13.py
rest_topology.py      simple_switch_stp.py
rest_vtep.py          simple_switch_websocket_13.py
simple_monitor_13.py   wsgi.py
simple_switch_12.py    ws_topology.py
[ 11:02:45 0 :) ryu] $ vi ryu/app/simple_switch_13.py
Taglist: Exuberant ctags (http://ctags.sf.net) not found in PATH. Plugin is not loaded.
Press ENTER or type command to continue
[ 11:06:24 0 :) ryu] $
```

So, that is something we wanted to discuss in this particular class.

So, hope you got an idea about how to process these entire things and run a SDN controller in your local machine. So, I will I will suggest you to play with this mini net emulator platform and the different kind of protocol that you are learning execute it on top of that.

So, you can even execute a socket programming from this individual host, just like we have executed the ping application. You can run your socket programming application and run in tier. So, explore it further. So, hopefully you will get a nice understanding or nice insight of this network protocol stack so.

Thank you all for attending this class.