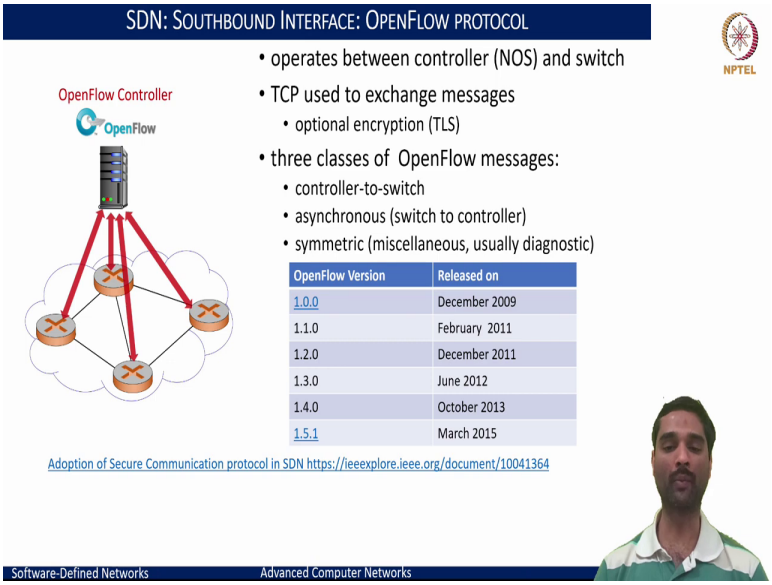
Advanced Computer Networks
Dr Sameer Kulkarni
Department of Computer Science Engineering
Indian Institute of Technology, Gandhinagar
Lecture 35
OpenFlow

(Refer Slide Time: 00:16)

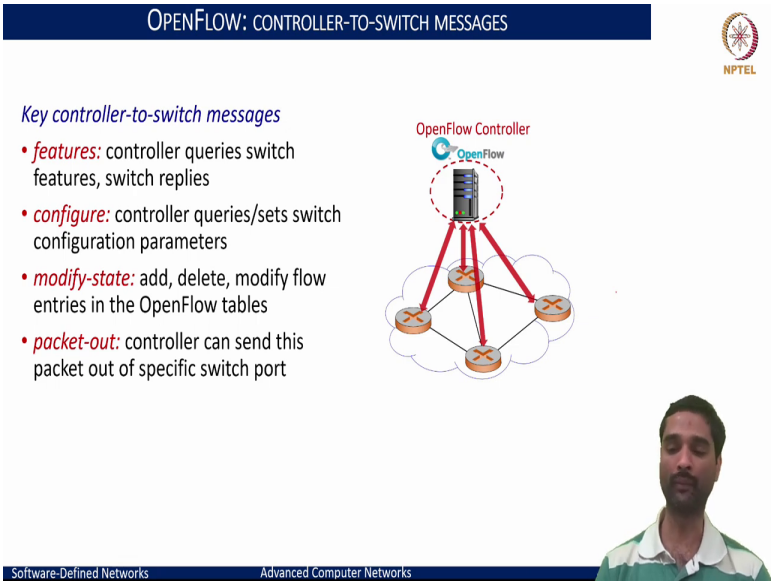


Let us now try to look at the OpenFlow protocol, which is the southbound interface for the SDN to communicate with the network devices. And this operates between the SDN controller or the network operating system and the switch or the routing devices. And to facilitate this communication, TCP protocol is used to exchange the messages; often, this can also be encrypted, and we could use any of the TLS variants to facilitate secure communication for these southbound APIs.

This basically abstracts all the interfaces that are necessary to communicate from a switch to the controller and vice versa. This means that it has three classes of messages: one when the controller wants to communicate with a switch, and that is the controller to switch messages, two when the switches want to communicate some sort of events or updates back to the controller, and almost these are basically asynchronous event notification types. And three, there will be a query from the controller and the response back from the switch to the controller or vice versa. And these are typically what are known as symmetric messages and are used for the exchange of various information and for diagnostics purposes.

And as we speak of OpenFlow, like I said, it started with the Nicira networks in the early 2008. And then, the OpenFlow as a standard specification 1.0 was set up in December 2009. Thereafter, it continued to evolve based on the interaction from the networking community, and lot more updates started to emerge. And by March 2015, we have version 1.5.1 of the OpenFlow specification that describes what are the exact messaging constructs for each of these cases. And what is the sequence of operation in essence to the overall protocol for communication between the network operating system or the SDN controller with the OpenFlow devices? Let us look at some of these aspects.

(Refer Slide Time: 02:36)



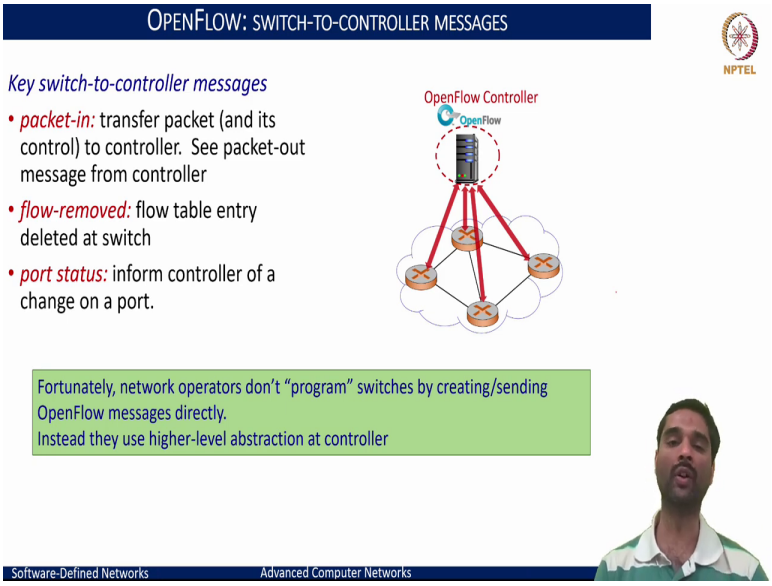
For the first of the kind is a controller-to-switch message. And this is quite essential for a controller to establish communication with the switch and try to gather the features of the switch in terms of what kind of device it is, what is the OpenFlow version it is supporting and how many number of ports, what is the active status of the links that it has, what is the bandwidth, all of these aspects as features of a device, whether it supports secure communication or not.

And various other aspects can be queried from the controller to extract the features of the devices. And also, there is a configure aspect where the controller can configure the parts of the switch, especially when you want to configure specific ports to be active inactive, you want to bring down specific links, or want to set some VLAN attributes for particular switches, those can all be done through the configure message.

And then, you have the modified state operation, which basically is to update the forwarding rules on each of these networking devices. We will look into this in more detail.

And the packet-out is another sense where if a controller wants to send a specific update for the switch to send specific kinds of a packet, it could use this packet-out message.

(Refer Slide Time: 04:03)



So, the other kinds of messages include the switch to controller messages, where the switches would initiate a message, try to reach out to the controller and get the response back and act accordingly. And the most common one that we will see is a packet-in message wherein as the data packets arrive onto the switch, if it is not able to decide what to do with a particular packet, it will query the controller, and controller would run the intelligence that it would do and say wait what should the switch do with this packet and it will install back the rule with the earlier seen message of either a packet-out or a modified state and then let the switch take particular action on that packet.

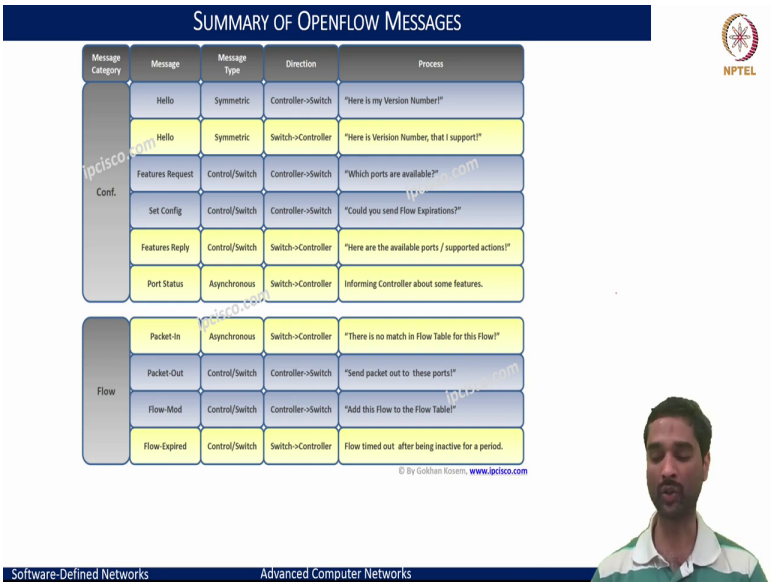
And likewise, once the rules are installed by the controller, the timeout or age out if no more matches have been on the data plane. And once the timeout, those routes will be removed from the switches forwarding table. In such cases, the switch would then notify back to the controller, saying a particular flow entry that was being set up has been deleted at the switch.

And likewise, if the switch sees there is a change in the port status or change in the link, link flapping when the link goes down or comes back up, the notifications will be sent from the switch all the way to the controller indicating the change in the port status for a particular device on a particular port. Now looking at these, we already start to see that there are a lot more messaging templates that need to be understood.

But then we said we wanted to simplify in the beginning for network operators to operate with. But if network operators are now made to learn all of this, this would actually make it a very difficult task for anyone to operate with. So, the best part here is that network operators need not have to interact with the OpenFlow or understand the OpenFlow specifications in greater detail, but the network OS or the SDN controller would abstract out all of this and handle it internally.

So, that way, it has the right separation so that we can see where the NOS would really fit in trying to abstract out the complexities. And what the network operators, in turn, use is exactly the high-level abstractions that are facilitated by the network OS.

(Refer Slide Time: 06:23)



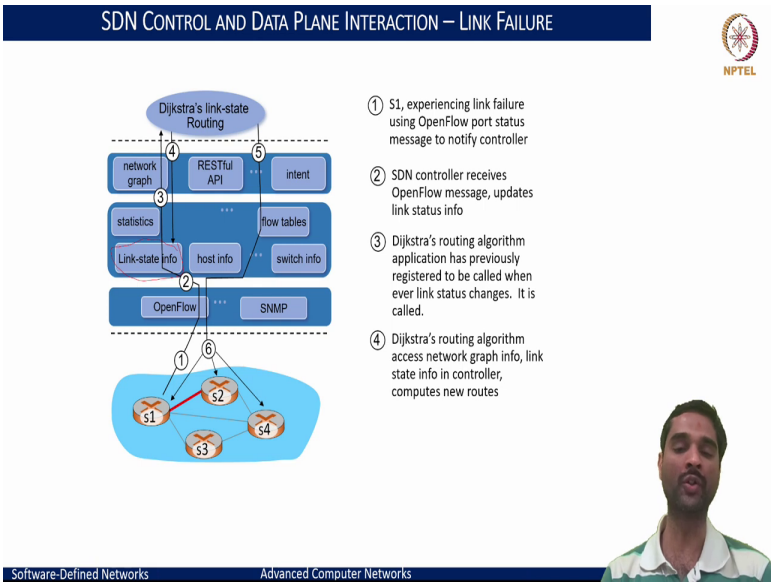
And the table here summarizes the set of kinds of messages that we just discussed. And on a broad basis, we can categorize them as per flow basis, wherein the data path entries as the packets arrive, there would be a packet-in that would go to the controller from the switch, and

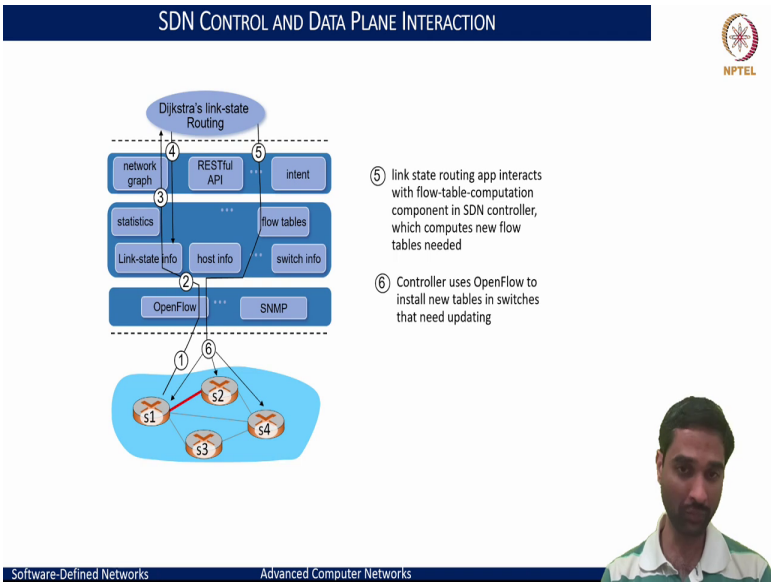
then in response, the controller would either do a packet-out or a flow-mod updates on to the switch.

And once these kinds of rules that have been set by the controller expire, the flow expired rule would be sent out from the switch to the controller, and the others include the configuration kind of messages, wherein both the switch and the controller can exchange the Hello messages to keep their status update and show that they are alive. And what is the information that they can exchange for the feature request.

And if the controller wants to configure particular aspects, then it will use the set-config onto the switch. And likewise, if there is a change in the status of the network topology, some links go up or go down or come up, then the port status is communicated back to the controller. This way, the network OS and switch devices can keep in sync with respect to the status that is going on and enable the network applications to take into account the changes that have happened and modify the characteristics back onto the switch in terms of how the packets should be processed.

(Refer Slide Time: 08:05)





Or let us try to put this into perspective and see one of the most fundamental network applications that is the link state routing. So, the diagram here is trying to show topology with four switches or routers, s1 to s4, and an application that is written on top of the network OS that is Dijkstra's link state routing, which is trying to configure and set up the routing tables. So, that all of these routers would do what a link-state routing would otherwise do.

So, we can clearly see now that unlike the typical link state routing where all of the routers have to flood, now there is no need for these routers to flood any of the packets amongst themselves and try to build the topology information; instead, they would all need to communicate their status in terms of the port status and the connectivity link information to the controller.

So, all the s1, s2, s3, and s4 would share their link state information just with the controller, and then the controller would be able to map, build the network graph and know what exactly is the topology without the need for any flooding. And once the topology is being built, the application Dijkstra's link state routing algorithm can be used to update the routing tables to be populated at each of these routers.

So, let us see how the sequence of operations would happen considering that we have a base state of link states that are being set, and let us say one of the links, s1 to s2, fails. Now in the traditional approach, whenever there is a failure of the link, again, these routers have to flood the information what has changed with respect to a link, and then everyone has to collect this update and then reconstruct the table routing tables.

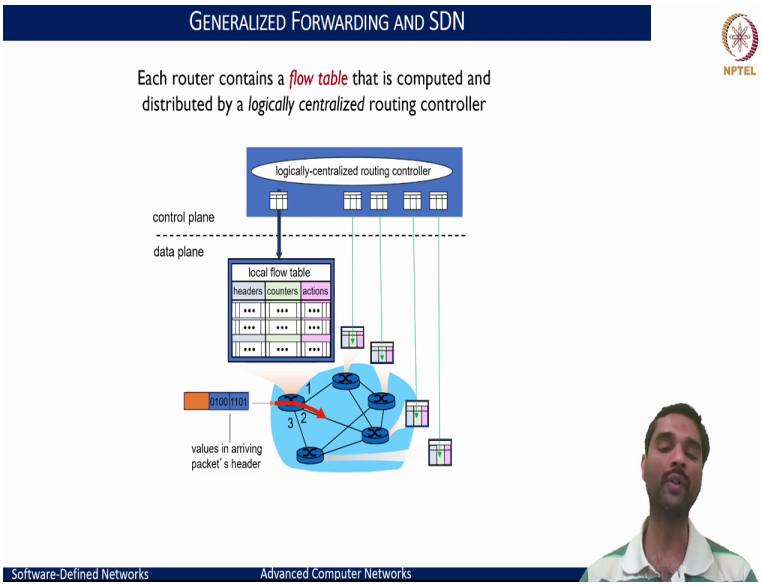
But here there is again no such need. What will really happen when the link, let us say s1-s2 goes down, is first, as soon as the switch experiences the link down, let us say s1 or s2, it would send a link failure message back to the SDN controller and through the OpenFlow interface, and once this controller gets notified, it will then process this message and update the link state info table indicating that link s1 s2 is now down.

And once the link state info has changed, that means the topology or the graph characteristics of the current network has changed. And if we see that, on top, there is a Dijkstra's link state routing algorithm that has basically the application that has dependent on this link state information where it would have registered for any changes to be notified back to it, then there would be a notification from the link state change triggered to the Dijkstra's link state routing application.

And once the application receives this notification that there is a change in the link state, it would go ahead and try to query the change in the link state and get the current network graph the way it looks. And then it can now centrally compute what would be the routes that would then be functional and what updates that need to be done at each of the s1, s2, s3, s4. If any of them were dependent on the route of s1-s2 link, then it would pass on this information back to the link state info to the SDN controller using the northbound interface.

Now, once there is an update that has happened on the link state info from the networking application, this link-state routing application will then pass on these updates. And using the southbound interface down to all the routers s1 to s4, the controller could update the OpenFlow tables or install new rules, or delete the rules that are stale or no more applicable and update the data plane features.

(Refer Slide Time: 12:18)



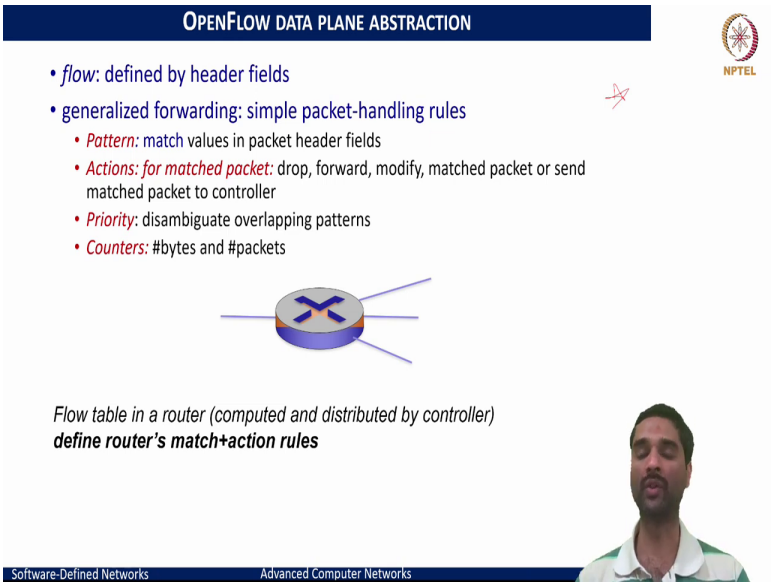
And once this is done, we will be able to ensure that the routing goes as normal. So, having conceptualized how the operations really occur at the level of the application's northbound and southbound interface, let us now try to look more closely into what goes in into the southbound interface. And what are the means to update specific rules in the switches and routers?

And the major construct like we said, as an abstraction 2 for SDN, was to prepare the generalized forwarding scheme, wherein you would prepare the kind of flow tables that will be updated on each of the routers and switches so that the incoming packets would then match on these flow tables and process the packets accordingly to route forward the packets onto the next hop links.

And this is done by the control plane, which dictates what should be the match action entries that need to be set in each of the router's tables using the southbound API. So, the main construct here is what we call as a flow table, which is an aggregate of the flow rules that are going to be set on each of the routers, which are then going to be used for matching the incoming packets.

So, you have learned in the earlier class about how the TCAMs operate, and you have the longest prefix match that would happen for all the incoming packets, and the semantics remain the same here. But what is the exact content on which the longest prefix match needs to be done is what gets dictated by the control plane or the network operating system. In essence, the NOS would compute these forwarding tables, and then update them on each of the routers in the network.

(Refer Slide Time: 14:11)



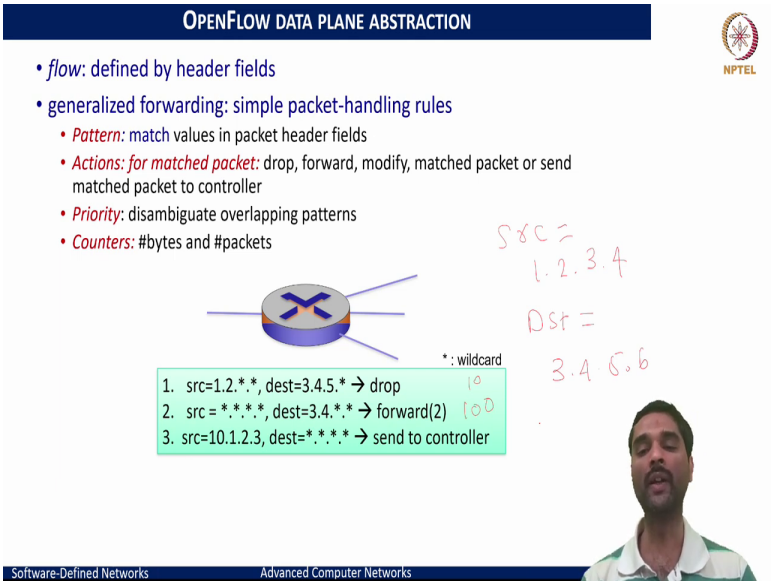
So, let us try to look at what each of these updates would look like, and what is the notion of a flow table entry; what we call as a flow is basically a set of header fields that are going to be matched onto the incoming packet. Anything that matches a rule that says which fields to be matched on to the incoming data for a particular set of header fields. And if there is a match, we call that a flow.

And every entry in a flow table corresponds to a set of flows. And you have a wild card that you typically use a * to say that a single entry can match any of the bids for that particular field. And that way, multiple flows can match into a single rule. And this is what gives us more flexibility. In defining how you would want to define the forwarding entries, now you are no more confined to just the destination IP or the destination MAC headers, you could change the patterns of the bits that you want to match on the headers and then take corresponding actions, and you are also not confined to just saying drop or route, you could even modify the packet headers itself. And that is the power of having this generalized match and generalized action basis.

And to disambiguate if there are multiple matches that can happen. And you want to disambiguate the overlapping patterns, we can use the priority bits to see which of the rules when matched or a subset of the rules that match should be picked for processing.

And besides, whenever there is a match, and you take certain actions, we can also keep specific counters, which would enable us to basically track how many bytes or how many packets are being processed by a particular rule, and how many packets have passed to a particular port destined towards a particular IP. All of these statistics can also be easily attained through this match action pattern. And the core here is defining the router's match plus action routes.

(Refer Slide Time: 16:20)



In here, we can see if we have to say match and action as a common thing, we can say you will match the source IP with 1.2., I do not care the remaining two doubles. But if the destination happens to be 3.4.5. whatever the last, I would want to drop the packets. And the * is basically a wildcard that would say you can match on any it does not matter whether it is 1.2.1.0, or 1.2.10.255; any of those would be a match when we say 1.2.*.*

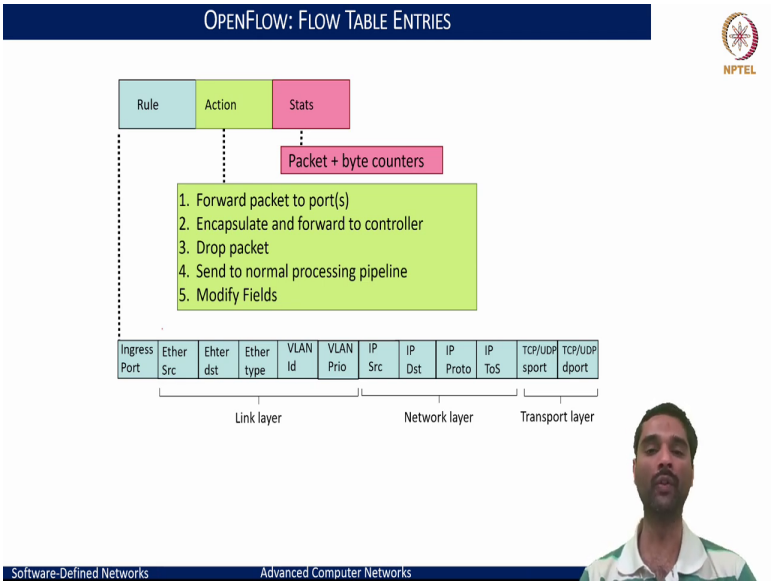
And likewise, we could have a very generic match that any of the IP sources. And if there is a destination that is going to be 3.4.*.*, then I want to forward. So, now the reason I have chosen these two is to say that now any packet that comes with 1.2., let us say the source is equal to 1.2.3.4 and destination equal to 3.4.5.6. In this case, what you would see is both rules 1 and 2 would match.

But if you see the longest prefix that has matched, that is rule 1. And if you go by the LPM approach, then this packet would be dropped regardless, even if rule 2 says that you would want

to forward. So, that is how you have the flexibility. And now, if you want to override this and say that no, I want to forward it on port 2, then you will want to use the priority and say this has a priority 100 while this has a priority 10.

And then you can flip to say I can go by the priority match i this case besides just the prefix match. And that is how flexibility comes to the net in terms of managing and configuring the rules.

(Refer Slide Time: 18:11)



So, let us look into how these flow table entries are defined. And what are the aspects of a flow table entry in OpenFlow mean? And the first part and the most important one is the rule, which dictates what is it that you would want to match on. And what this says is basically, like when a packet comes in, you want to know on which particular port did the packet come for that particular device. And that is called as the ingress port.

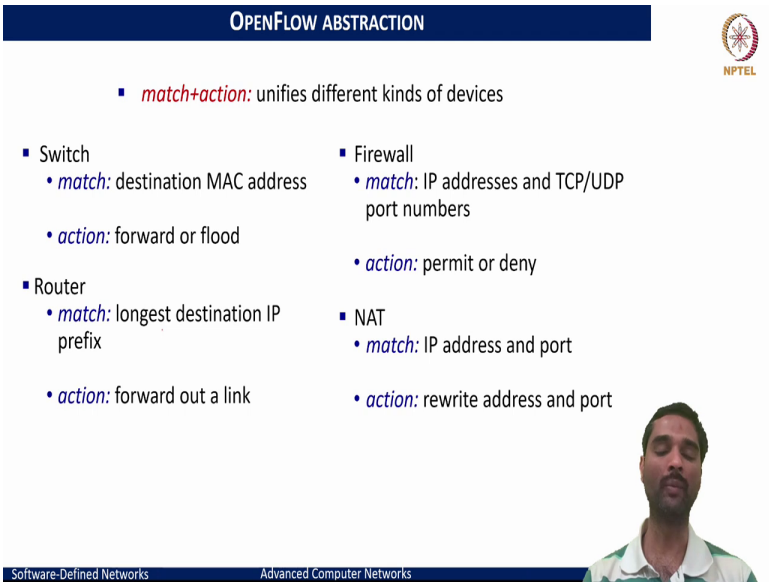
And when a packet came, what is its layer 2 header look like, the Ethernet source, Ethernet destination, Ethernet type, and including layer 2.5, like the VLAN ID and VLAN priority bits can also be checked to see whether there is a match on any of these fields. And unlike the switches, which are confined just to layer 2, we can also look at layer 3 header fields, like IP source IP destination IP protocol, which is the next header that you will see, and then the IP type of service bits.

So, at layer 3, we can match on any of these IP headers, not just that, unlike the layer 3 routers will just look at these fields, we could also look at layer 4, which is TCP or UDP source port and destination port at the transport layer. And we could match on any of these fields to now say what specific actions we would want to take.

And the list of actions includes one to forward packets to a particular port, that is, you receive a packet on one of the links, and you want to send a packet on another link, which is basically, you receive on port 1, send it out on port 2 or vice versa, or you would want to encapsulate that packet and forward it back to the controller itself so that if network operating system or the controller wants to look at a packet and do some processing, we could always send the packets back to the controller.

And if we do not know what to do, or if you think there is some malicious packet that you do not want to process, we could as well drop the packet. And if everything happens seems to be there, and the routes are already set, you want to just do the normal packet processing pipeline to say what actions need to be done on that. And typically, this was either forward or modify specific fields on that packet header. Like when you think of NAT, you would want to change if the packet is an outgoing packet; you would want to change from a private IP private port to a public IP port details and send the packet out before sending it on to the internet. Or when you receive the packet back, you would want to do the exact inverse. So, we can specify all of this using just these flow table entries.

(Refer Slide Time: 20:59)



So, this OpenFlow abstraction of match action helps unify different kinds of devices and helps achieve different kinds of functions in just the commodity hardware. Let us look at how this can be achieved.

First is the switch, and if you think of a switch, what it will do is match on the destination MAC address. And then, if it finds an entry in its table, it would forward it to one of the outgoing ports, and if it does not have any match, it will just flood on all the other ports other than the incoming port, and with the match action abstraction, we can specify the rules to say match on the destination MAC address and then if there is a perfect match you forward if there is no match you flood and this serves the role of a switch.

So, we can now implement the switch on commodity hardware or just on our bare Linux machines and taking off the routers, which are the layer 3 devices; what we have seen earlier is that they do the longest prefix match on the destination IP and once there is a match on the destination IP they will forward out on a specific link. And now, we can specify through this generalized forwarding to match on a particular destination IP, and as the match is found, you can forward it out to route the packet appropriately.

And if there is no match, typically, the routers would then just send it out on a default port or drop in this case, now we have a mechanism to even forward such packets to the controller, and when you forward the packets to the controller, it can then find out what is the route that you need to set and then set the corresponding action so that the packet can be forwarded on the

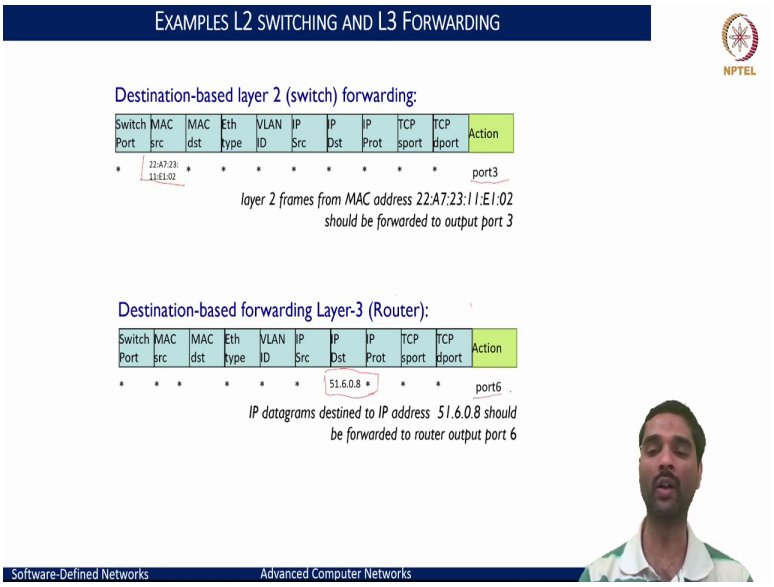
outgoing link. Thus we can achieve the functionalities of the switch and router using the OpenFlow abstraction.

Further up, we can even implement specific middleboxes right within the construct of OpenFlow, like think of a firewall, what it would typically do is try to match on specific IP addresses either the source IP or destination IP, and try to match on the specific TCP or UDP source and destination port numbers and either it would allow the actions to be permitted that is, you forward the packets to go and reach the intended destination, or if you would want to block certain packets, then you would drop or basically deny the packets from going forward.

So, now with the match action aspects using the action as either forward or drop and specifying the match with respect to the IP addresses and TCP or UDP port numbers as an exact match, we can implement the firewall functionality as well. Also, I mentioned earlier about the NAT, we could do the same wherein we basically rewrite the IP address and port numbers according to whether the packet is outgoing, then private to public, and whether it packet is incoming, and you can do public or private and then send the packets out.

So, thus a simple single abstraction with the match action forwarding mechanism or a generalized forwarding allows us now to implement different networking elements in one go; that is, on the middlebox functionalities, the primitive middleboxes, and the routers and switches can be implemented very easily.

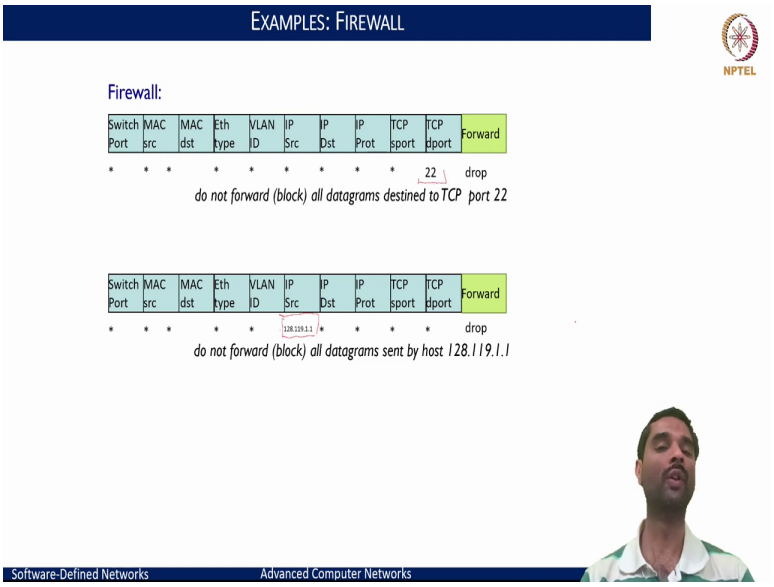
(Refer Slide Time: 24:30)



So, let us consider the examples for L2 switching and L3 router forwarding. So, the idea is now like we want to set up these fields, wherein we want to match this particular source MAC and if there is a match for this, I would want to say that the action that needs to be taken is sent it out on port 3. This way, whatever the rule that we match for the source MAC, it says you can send it out on a particular output port.

Thus, we will be able to do the L2 forwarding. And likewise, if we have to do the L3 forwarding, which is a router's role, we would match on the IP destination, which can be the longest prefix match or exact match for a particular IP, and dictate the action to forward it on a specific port.

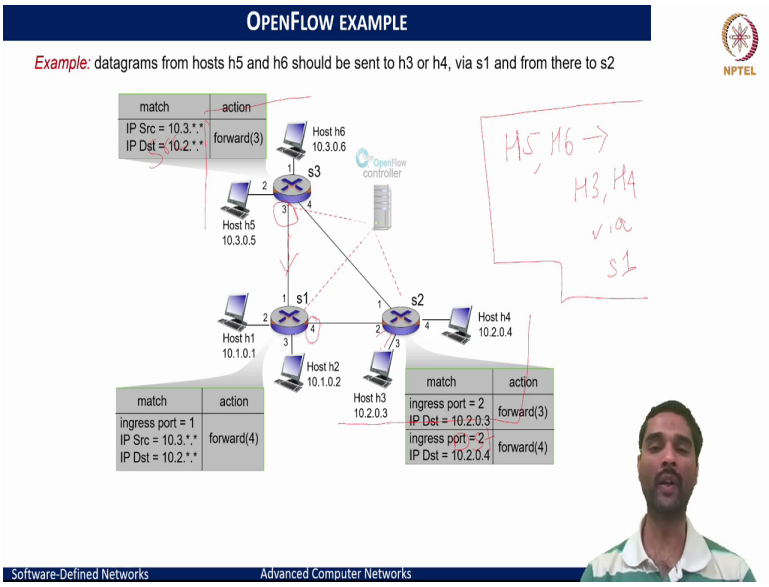
(Refer Slide Time: 25:20)



Similarly, the firewall we can say that we want to deny any incoming traffic for SSH. And we know that the SSH port happens to be 22, then any of the rules, regardless of what is the MAC ID, what is the destination MAC ID or the IP source or IP destination, we want to block all the SSH incoming traffic. And that way, if the destination port happens to be 22, we just drop the packet, so we have essentially denied any SSH connections for the servers in our network.

And likewise, if we want to block a particular source, which is seen to be malicious, we can say that we want to watch this particular IP, let us say 128.119.1.1. And if this is a malicious IP, we do not want to admit any traffic from that particular IP, then the rest of the fields become immaterial. If there is a match on the IP source, we would want to drop any such packets coming from the source. Thus, it could be also very easy to implement the firewall functionality.

(Refer Slide Time: 26:23)



Now, let us try to see in operation how this OpenFlow works as an example. And what we can see here is that we have a simple topology, where we have three of the routers and these six of the host connected to it. And like I discussed before about how we would want to achieve the waypoint routing, or the essence of trying to dictate particular hosts packets would go through a particular switch, regardless of whether the path is the least cost or not.

And if we want to have this control, and let us say we want to send the packets from either of the host 5, or from host 6, which are intended for either of the host 3 or host 4 as destination machines. So, from a source here and a destination here in this field, we would want the traffic to go through this particular switch s1. And if this is our intent, what a network operator should be able to say is just that I have these host 6 and host 5, any flows coming from this has to go to the switch s1. And if we define this rule, and then see how it will translate and update would happen in each of the cases, then the network operator can simply push an intent saying any traffic from h5, h6, that is destined towards h3 or h4 should be routed via s1. And how this would reflect back and updates on the OpenFlow rules from the network operating system, let us try to look at this.

So, if we have such an intent given, then the OpenFlow controller needs to set the corresponding rules on each of the intermediate routers, that is primarily at the s3 and s1, and s2. And at the s3, we want to say that if the IP source happens to be 10.3. which is basically matching both h5 and h6, and if the IP destination happens to be 10.2.* that is matching the h3 and h4 on the

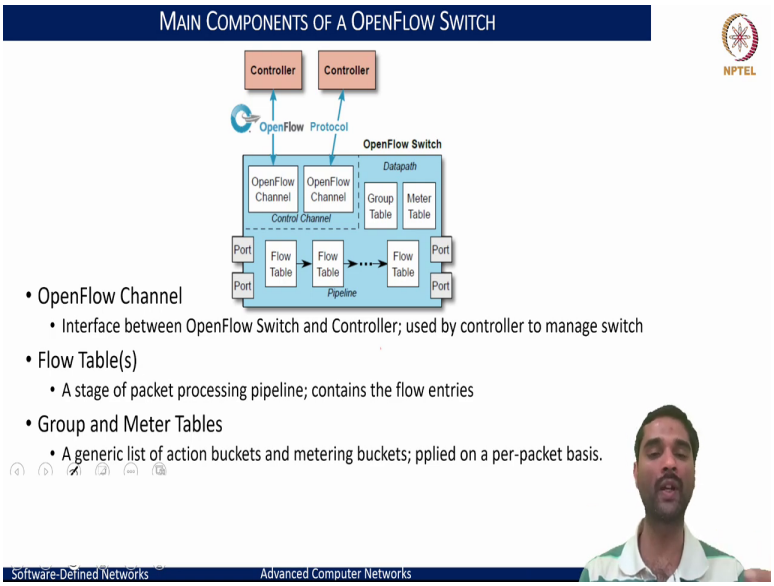
destination side, then we would want to forward such packets on this port 3, that is this particular link.

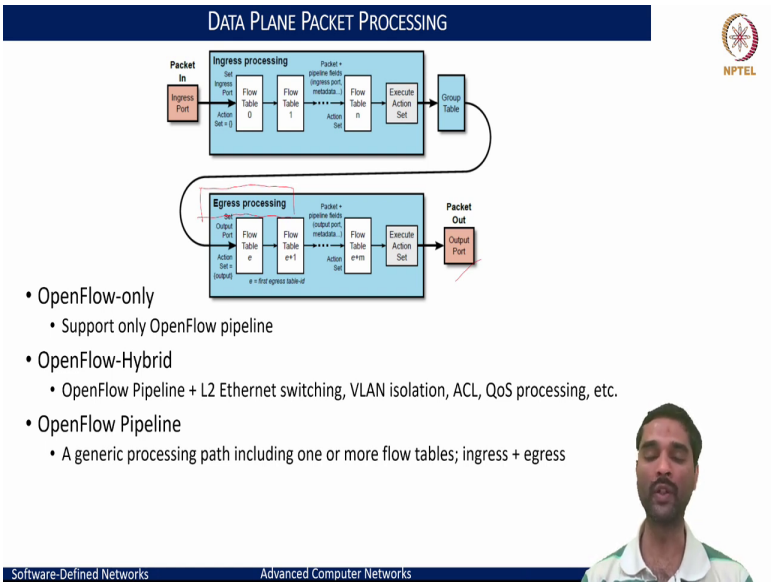
And that would ensure that s3 would always match these rules and then forward the packets on the output port 3 to make sure that the packet reaches the s1 and as packets arrive at the s1, they would arrive from an ingress port, that is, the incoming port s1 and the source and destination fields remain the same. Now, our intent is to forward these packets onto the switch s2, that is basically the output port 4, so you would say the action as forward on port 4.

And once the packet will reach the s2 router, they would reach ingress port 2, and then we would want to send the packets back based on the destination IP, that is, if the intent is for host 3 in the IP destination happens to be 10.2.0.3, we would want to send out the packets on this port 3. If it is intended for h4 or 10.2.0.4 as an IP destination, then we would want to send it out on port 4 to make sure that the packets reach the corresponding host.

And this is how the abstractions and the means to configure specific rules at each of the routers are presented from the OpenFlow point of view to the network operating system or the SDN controller.

(Refer Slide Time: 30:27)





So, I have tried to summarize the essential aspects of how the SDN controller could behave and what is the OpenFlow, and how that can be used to set specific rules. But what does it mean to look at an OpenFlow switch if we have to understand it in a very broad sense now, the key components include for any OpenFlow switch is first the control channel or the OpenFlow channel on which the controller could establish communication with the corresponding switch device.

And any communications would then happen on this channel. And as we said, this channel can be secured using TLS communication. Otherwise, it would be it could be just the bare TCP to ensure that the communication is reliable. And not any of the packets that come, the three kinds of messages that we discussed, would then update on the data path or make some configurations on these flow tables.

And the flow tables are basically a set of tables where different rules are going to be matched based on a given set of rules, and then specific actions are going to be taken. So, a flow table consists of many of the flow entries which are going to be matched. And this is what a TCAM would actually be implemented so that you are able to match all the entries with the wildcards supported.

And as the OpenFlow specifications grew, multiple tables were supported, the pipeline of having moving the packets from one flow table to the other, taking multiple sets of actions were supported. And also, the addition of group tables and meter tables were added. So, that these

common actions could be defined. In the meters like if you want to do the rate-limiting then you would use a meter table setup like what is the number of byte counts on which you would want to stop processing specific flow packets, those updates can be done on the group and meter table.

And over the years, like I said, there have been a lot of additions that have happened in this OpenFlow world. We will not have all the time to dig into the details of what all have changed. But this should help us understand as see what all updates have gone through and to think of what happens when a packet arrives it is called the data plane packet processing or the forwarding plane packet processing model.

Here as the packets arrive on any ingress port, that is the incoming port, they would go through first the ingress processing pipeline, which consists of a set of flow tables on which the packets are going to be looked, and match in terms of the rules that have been set. And if there is a match, you would want to keep adding what actions you would want to take for that match. And the action itself can be: forward to the next tables like, if I have a match on table 0, I would want to send it out to table 1.

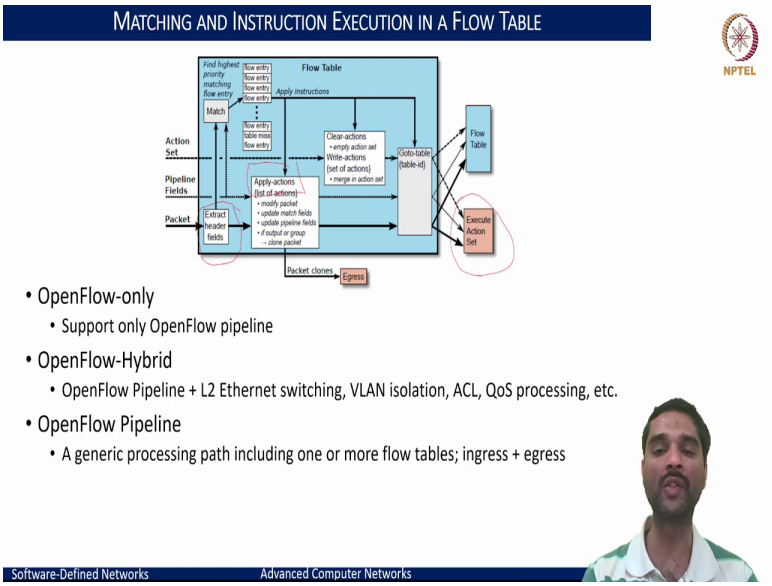
And if there is a match on table 1, you would want to send it out on a flow table n and likewise and then execute the set of actions, it could also be that you would want to send it to a common group table and process these packets. And once the processing happens in this pattern and you have updated specific things on a common group, you want the packet to be sent out on a particular port, then we have what we call as egress processing or exit path.

Here again, you will have a set of new tables and match actions, which are equivalent to what you would see in the ingress, but this is the post-processing on the packet that you would want to do before you send out that packet from the corresponding device onto the output port. And this is how the data plane packet processing model is in the OpenFlow switches. And this OpenFlow facilitates two kinds of processing one is called the OpenFlow only, where this pipeline is strictly defined by the OpenFlow constructs.

And the other is a hybrid model that was added in much later versions of the OpenFlow, where you could also add specific of the switch processing characteristics like L2 Ethernet switching, VLAN isolation, access controls, QoS processing, which may be specific functions or

applications that are there within the switch and you would want to exercise. Thus this constitutes how the packet processing happens within the data plane of the OpenFlow switches.

(Refer Slide Time: 35:02)



And to look at when a packet comes for a particular device, you can think of it enters a packet on a particular ingress port. And it is going to be first as the packet enters you want to extract all the header fields that are relevant for you to match on a particular flow entry in a given table. And once you parse and extract these header fields, you would want to do a match and if there is a match in any of these flow entries, you would want to apply the corresponding set of actions and these could be like modify packet or update packet or even drop the packet and so on.

And if there is also a set of actions that you would want to take later like clear specific actions like if you want to empty an action set that can be done through the clear actions as an action type. And once you empty you would want to do some normal common processing with respect to some specific table and within each of the tables as you match there is also a metadata that can be passed to keep track of what all tables it has tried to match and how it has been processed, and eventually execute all the set of actions that have been accumulated over the set of flow tables and then eventually send a packet out.

And in some cases, you may also want to have a support to clone a packet and send the clone of a packet on a particular egress port. Those kinds of actions can also be facilitated now in the newer versions of OpenFlow.