

Advanced Computer Networks
Professor Sameer Kulkarni
Department of Computer Science Engineering
Indian Institute of Technology, Gandhinagar
Lecture 45
Programmable Networks – Data plane
Programmability – Overview II


(Refer Slide Time: 0:17)

MOTIVATION FOR SOFTWARE DEFINED NETWORKING (SDN)


- Networks:
 - Notoriously difficult to manage
 - Evolves very slowly

Abstraction is the key to extracting simplicity: easier to write, maintain and reason about the programs that manage and control the network.

Ref: Scott Shenker, et.al. The Future of Networking, and the Past of Protocols, Open Network Summit, 2011



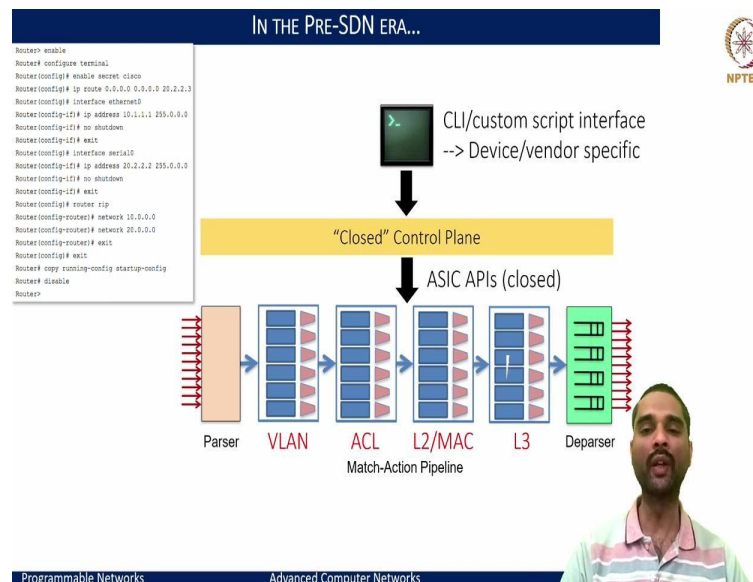
Programmable Networks Advanced Computer Networks



So first, let us now try to understand what motivated us for the SDN in the first place. One were the control plane, where the networks were really difficult to manage, and we wanted them to be handled and made programmable in one aspect. The others were basically to break the ossification and to provide better means to innovate that means to enable any evolutions that happened, which were typically slow to be made much faster. And this, again, requires abstractions.

In essence, we want to extract out the simplicity so that we are able to ensure that we can manage to build the right means to give the flexibility for what needs to be done versus what can be hidden as the key complex aspects underneath the abstraction layer. And in essence, we want to maintain and reason about the programs that manage and control the network. And how is the key aspect that was missing in the first part or exactly saying what functionalities you want versus how they would want to be done?

(Refer Slide Time: 1:41)



And here, we want to look at and understand the transitions that have been done, how they have been achieved, and how you want to take it forward. So, to add to what we just discussed, we have this typical match action pipelines with a parser and deparser, the packet processing capabilities in the data plane or the forwarding plane of the devices, which typically resembled the structure where the entire aspect of what the parser would do, and what kind of headers that you are going to extract, what are the kinds of tables that we are going to deploy, support for what headers that we can match on? were all decided by the ASIC vendor within the ASIC.


And what we meant is if there were a switch, that would say that it has support for VLAN for L2 MAC, for L3 for ACLs, each comes with its own set of tables and the parser that was pre-baked to support these headers. And this meant that the ASIC APIs were closed, were not open for innovation not open for experimentation either. And the way that these tables will be updated and configured and the means to do that, as a control plane, was also a closed aspect.

And on top of this, whatever the user had, at a bare minimum, was to work with the command line interpreters or the custom script interfaces, which the OEMs the vendors would provide. And you would configure, for example, if we had a typical router, you are trying to interface and set up certain configurations, you log in onto the router, maybe SSH, maybe just TCP depending on what kind of router you have. And you would have custom commands like config interface configure the IP addresses, setup the routing protocols, and likewise is on a switch in terms of

what information. But you never had any control over how all of this would translate to setting up the actual data plane functionality. And this is where the SDN started to come and deossify or disentangle these rigidities by bringing in the right abstraction.

(Refer Slide Time: 4:18)


BEFORE OPENFLOW ...



- Open Signaling (1990s)
 - Make network control functions more open, extensible, and programmable
 - Separation between hardware and control software
 - Access to the network hardware via open programmable network interfaces.
- Focuses on connection-oriented network services in the early days
 - [IETF RFC3294](#) (2003) General Switch Management Protocol (GSMP)
 - [IEEE P1520](#) (1998) standards initiative for programmable network interfaces

Programmable Networks

Advanced Computer Networks



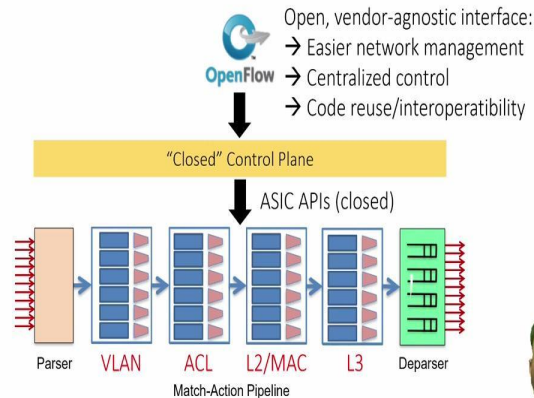
With those custom ASICs or custom capabilities that were closed or proprietary. Aspects of dealing with the control or the data plane started to create a lot of troubles, especially as barriers in telecommunications and network infrastructures. And this is where an international research and industry community known as OPENSIG started in early 1995 with a concept of Open Signaling and network programmability. In essence, to open up the aspect of how things can be brought for really making these networks to be programmed or customized per need. And interestingly, there were many of the aspects that got pulled in terms of how to make these network control functions more open, extensible, and programmable.

And also, like we saw with SDN, the core aspect, there was a separation between the hardware and control software that you would want in terms of how the control software could dictate the hardware in terms of how to control the device. But now also as an aspect, to see how this control over hardware in terms of what hardware needs to do. And that is where the open programmable network interfaces were thought of. And there was a general switch management protocol as an IETF RFC that came out in the early 2000s in terms of what should be the key APIs with respect to which a switch kind of device can be managed.

And there were also initiatives from the IEEE, IEEE P1520, which discussed the need for standard software interfaces, that is, basically for the programmability of the networks, specifically for service and signaling control. And when we speak about telecommunications networks, in the early era, the call setup, call teardown, handover, all of these signaling required a lot of management aspects and control that you wanted to establish as a programmable entity. And this is where this IETF P1520 started, where the objective was to enable the development of open signaling, control, and management applications, as well as like high-level multimedia services that wanted to be run on the networks.

And the scope here was also confined to the ATM switches and circuit switches or the IP routers. But these aspects, when we think, are also applicable to the traditional circuit-switched or packet-switched devices that we would want to work on like the switches and routers. So, the basic idea of how we can have a process of development as a standard for application programming interfaces for networks started to emerge with the early open signaling works and these IEEE P1520 as a means. And what it really envisioned was to have the telecommunications network not as a bunch of hardware infrastructure but think of it as a giant computer that can be fully programmed and controlled, which would help enable the delivery of voice, data, and video services globally, the way we would want, that means it is more tangible to control and manage these devices. And the key intelligence here would vest with the networks in terms of how you would signal, how you would translate the algorithms to control, how to manage the data, and how to manage the forwarding plane of these devices. And the essence is carried forward in what we see as OpenFlow.

(Refer Slide Time: 8:32)

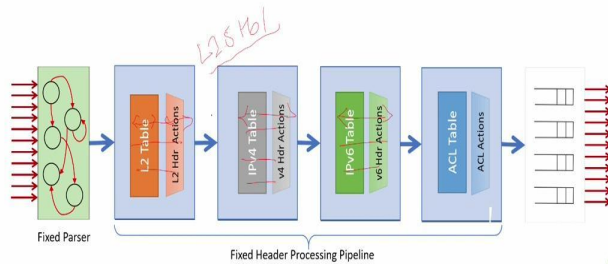


And the aspect of looking into core networking devices, which we saw earlier and discussed about the key changes. And now, if we push back on what really did the OpenFlow really do in terms of working on this switching hardware, it did not change anything on the hardware pipeline in terms of what contents you can parse and what contents you can match or take specific actions, OpenFlow had no control over how these things will be done. This would still be done as if it is a closed vendor, as an API that will dictate it.

But what it really opened up is the aspects about the control plane that could now be managed, or externalized, or centralized and be updated in a standardized fashion rather than having custom means to interact with specific devices. Now, it opened up a common vendor-agnostic interface, which would allow for easier network management and centralized control.

And whatever you would write on top of this as a vendor-agnostic interface could be reused and made interoperable across different vendors to make sure that things would work the same way. So, that is where code reuse and interoperability started to emerge with OpenFlow. And this pattern was although applicable just at the interface of the control plane but nothing underneath for the hardware.

(Refer Slide Time: 10:15)

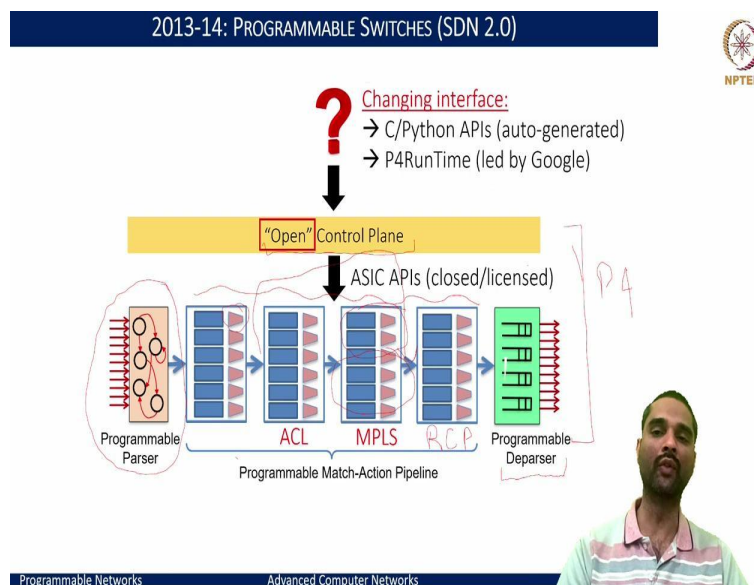
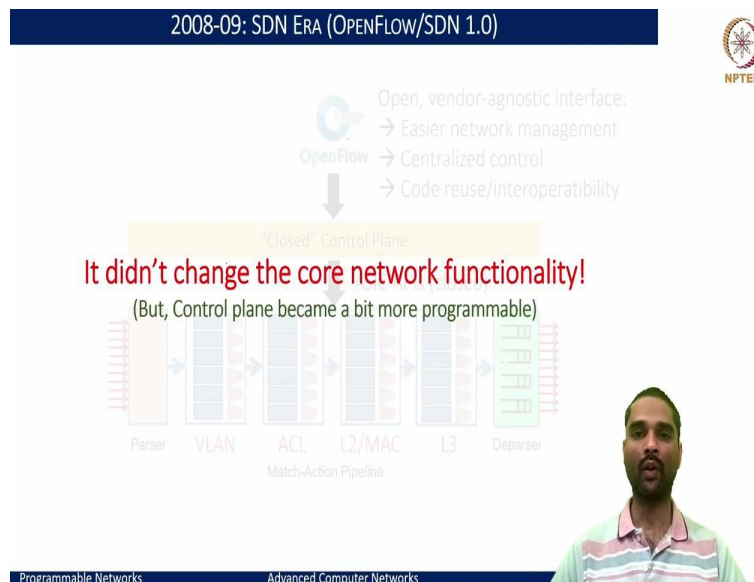


And what that meant is we still had these fixed-function pipelines. And the capabilities for us would then still invest with fixed parser means, if I have OpenFlow, the OpenFlow headers, whatever we want to match and set up, they have to be supported by the underlying hardware; only then would the OpenFlow variants work. And this is where like, initially, when the OpenFlow started, we saw that earlier, we just had our own 12 header fields, the standard L2, L3, and L4 headers that you would match and take specific actions and actions set was also pretty limited.

In a sense, we would only swap particular header, take, drop the packet, or forward a packet on a particular port, and so on. And this is what we refer to as the multiple match tables, which you could basically push, and aspect of how these tables and actions would correspond in each of these tables, as we go along, could be programmed through OpenFlow saying what specific header match that you want, and what specific actions that we could do was fixed. But I could not change, in any way, the configuration of L2 table or add, let us say, a new layer 2.5 table if it is not supported; we cannot change this. So, this was fixed.

But the entries that we want to populate in each of these tables is where the OpenFlow allowed to add the visibility. Otherwise, everything would have been done just by the device. And that is where OpenFlow started to bring more of visibility, and more of control over what match entries and what action entries we would populate in each of these pipelines. But pipelines themselves again, note that they were not something that could be changed with the earlier OpenFlow.

(Refer Slide Time: 12:33)



So, it did not change the core network functionality, but it made sure that the control plane now became a bit more programmable. And going forward with SDN 2.0, what we would really want is not just the control plane becomes programmable but the control plane becomes open. In essence, we should have the control to dictate what should reside in these programmable parser entries and what should be the match that we would want. What kind of actions, if any, that we would want to support should also be there. And how many of these tables that we want, and what kind of entries we want to build, need to be programmed and controlled on these devices.

That means end to end, when we look at the headers that we want to match and parse, that is the programmable parser, and what kind of actions that we want to take, how we want to pipeline and process any of the packets, we want full control over that. And that also needs to be a programmable match action pipeline. And once we do spay, take specific actions, how we want to deparse, or basically assemble back the packet and emit the packet out on a given port, the control should also be there on all of these entities. And this was exactly the vision of programmable switches.

Here, basically what this meant is to say that we can change mix and match which tables would look for what headers we could have table 1 lookup for layer 2, table 2, 3 lookup for the ACL, table 4 lookup for MPLS, or any of the custom headers that we want to bring in. We can bring in and say I had that particular functionality on any of the tables so we can dictate the number of tables or match actions that we would want to perform. How many of those that we want to perform all of this flexibility needs to be provided.


In essence, we are making the control plane now completely open and also the ASIC APIs that would then match for making this aspect on the give ASIC somewhat licensed or a closed even though closed or a licensed fashion, we could still have target specific applications to translate these open control plane activities into this ASICs through either target specific or somewhat licensed APIs through which we could build this functionality. And to enable this is now the question of what exactly we need. And we can think of, like, the key changing interfaces that we want to bring or either C or Python APIs that could auto-generate and build what we call us to program the open control plane to ensure that we can meet.


And this is where the P4 run time, which was the effort led by Google early on and is now being Consortium in its own and trying to provide the means to facilitate the interfaces for really dictating all these elements. And in essence, what we have now made is the control plane to be fully open and ensure that the customizations can be pulled in just as a user writes a program and executes them on the hardware; you would have the programs that are written by the user and execute them on hardware. And for this, the hardware also needs to ensure that the P4 runtime support to be provided. And they are able to do both target-agnostic and target-specific actions within the given hardware.

(Refer Slide Time: 16:50)

NEED FOR FLEXIBLE PACKET FORWARDING

- Modify the set of packet fields
 - Network virtualization: new tunneling formats
 - Support for new flags
 - Disabling existing fields
- Use tables more flexibly
 - Different environments require tables of different shapes and sizes
 - Enterprises: ACL-heavy
 - Core: IPv4-heavy
 - **Resource wastage**: can't use another table's hardware





Programmable Networks

Advanced Computer Networks

And what this would bring is, provide us the core flexibility for doing how to do the packet forwarding and what kind of packet forwarding we want to do. And the reasons are quite simple. Like if I want to modify the set of packet fields, especially when we thought of network virtualization and overlay networks, it called for a lot of tunneling formats like we have NVGRE, VXLAN, and so many ETH in ETH, IP in IP, and STP, all of these need to be supported. That means we cannot be confined to just one set of specific headers, we may have to look in recursively into the packets, VLAN in VLAN all of these.

And this required us to have the need for flexible packet forwarding, we could not have these tunneling supports unless the ASICs support and we want to build custom tunnels techniques; we want to have your ASIC support them readily. And also, the support for new flags in terms of what way we would want to support like DSCP bits if we were to enable them and try it out. We should be able to do like DSCP bits or TOS bits that we want to verify change adapt for any of the use cases, we should not be able to do that.

And in fact, the other way of looking at it is also if we disable specific fields, and I want to bypass a custom layer 4 and jump directly to layer 7, maybe we do not want any TCP or UDP being there, just try out some other custom layer 7 on top of layer 3, would that be possible? and that meant that we should need the means to disable the existing fields, bypass them, and still support forwarding the packets or processing the packets with customized headers. And this is

where the need for flexible packet forwarding arrived, and catering towards this where all the changes where we needed the aspects to be looked in a different perspective.

And also like, when we build specific tables like, think of I am deploying a router that is the core versus a router that is within a campus or an enterprise network where especially when we look at the campus or enterprise networks, they are really more particular about the access controls in terms of which flows should be permitted, which flows should be denied. And what are the actions that need to be taken in each of these access control modes? But when we look at the core networks, they are really heavy on just forwarding based on the IPv4 or IPv6 headers.

So, if we had such match action tables given by one vendor where you had the tables given for ACL tables given for IPv4, you may not be utilizing the resources equally either because core networks, you will not be using ACL tables. That means you are wasting those resources when in fact, we could have used the same for much more rules at the IPv4, and likewise at the enterprise, we may not be using layer 2 or layer 4 or layer 3 of the types but looking primarily at the ACL rules.

And if we want to increase the ACL rules at the cost of other header other tables, we will not be able to do that means, again, we are wasting the table space that otherwise the hardware would have been allocating for ACLs. And this was where again, we needed more flexibility in defining not just the kinds of tables but also define the size of the tables in terms of what should be width they should have, what should be the depth in terms of how many rules they can accommodate, and how many number of bits, they should match in terms of the width and so on. So, the characteristic of actually having full control over the tables was also the need to ensure that the resources will be used appropriately.

(Refer Slide Time: 21:30)

WHY OPENFLOW ISN'T ENOUGH



- In the beginning, OpenFlow was simple: Match-Action
 - Single rule table on a fixed set of fields (12 fields in OF 1.0)
- Needed new encapsulation formats, different versions of protocols, additional measurement-related headers
- Number of headers ballooned to 44 in OF 1.5 specification!
 - With multiple stages of heterogeneous tables



Programmable Networks

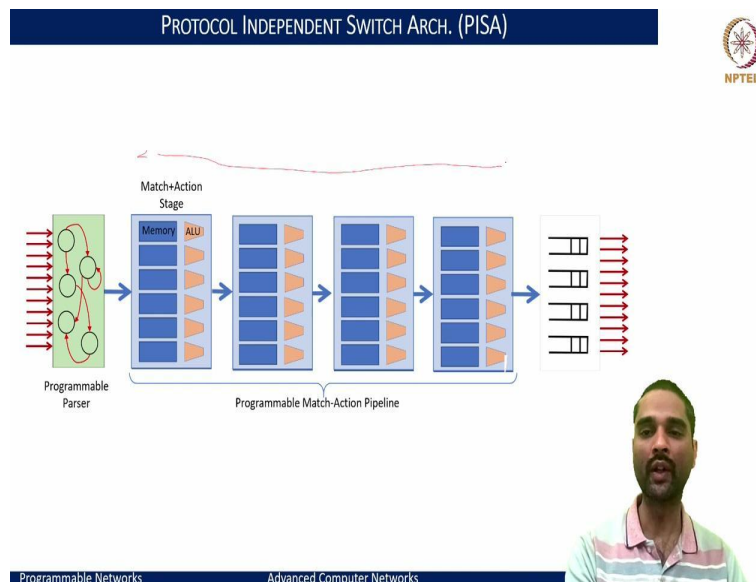
Advanced Computer Networks

And again, if we reflect back and see why OpenFlow was not sufficient in these aspects, we can see that OpenFlow started with just the simple match action wherein the predefined set of fields that would be supported by the ASIC was only to be programmed. And initially, it all started with a single rule table wherein, what we call as the single match action tables or SMTs. Where all the sets of rules would be just baked in a single field in the early OpenFlow 0.9 and 1.0.

And then, it progressed and evolved into multiple match action tables. But OpenFlow was not able to facilitate or support any encapsulation formats or different versions of protocols, which we could readily build like any of the layer 4 newer protocols. But if the action tables were not supported in the OpenFlow headers, it will not be able to support or configure such rules. And also, any additional measurement-related headers, if we had to add, like if you want to track or approve certain aspects of a particular kind of packet, it would be difficult unless it was supported in the OpenFlow rules in terms of what could be matched.

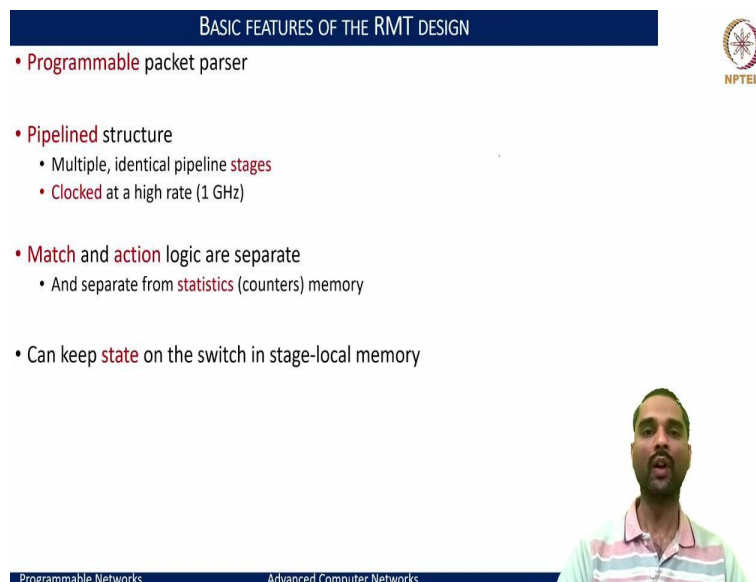
And these requirements, in fact, led to various versions of OpenFlow being evolved in terms of OpenFlow 1.0, which was just having around 12 fields, to OpenFlow 1.5, which came up to support around 44 such match tables. And having these different matches that means you need multiple kinds of tables to be supported within the hardware. For each stage, you need different kinds of tables. So, all of these led to say just the OpenFlow approach of trying to match on a specific field and take certain actions would not be feasible unless we really controlled the way we would want to configure the tables themselves.

(Refer Slide Time: 23:54)



And this is where again, this protocol-independent switch architecture came to say we now not only control just the programmable parser match action pipeline but we would want to specify for each match action stage what should be the table width, table depth, and for each of this processing, how many numbers of tables that we would want to support.

(Refer Slide Time: 24:25)




And these basic features were the call-in for what we call as the reconfigurable match action table design wherein the first component is the programmable packet parser, where the parser

block could be configured and updated with respect to what you want to see, what headers you will want to extract, and the pipeline structure in terms of the match actions that we would want to do. We could have multiple of these pipeline stages, and each of these stages could be done one after the other at a very high clock rate so that we are able to meet the line rate packet processing. And the match action logic in each of these tables could be completely distinct. And whenever we are trying to build these match action logics, we would also want to have support for specific counters or statistics for certain kinds of a match in terms of how many layer 2 packets were with this particular destination MAC or how many of the packets for a particular layer 3 address on a particular port? All of these counters etc, could be managed within the switch in terms of even how many packet losses were observed for each of the ports, what was the overflow, or what is the port utilization? All of these counters could be built up.

And what this also meant is we would also want to keep the state on the switch in somewhere a stage local memory for each stage when you process your change, you maintain information and then carry it forward. And these are exactly the aspects that we are looking for in trying to define what we call as a reconfigurable match table.

(Refer Slide Time: 26:24)


BENEFITS OF DATAPLANE PROGRAMMABILITY



- Flexible Parsing and matching on non-standard fields:
 - Faster and easier network evolution: new protocols/headers
 - Traditionally a new protocol addition takes 4-5 years!!
 - Hardware upgrades → software upgrades: protection on investment
- H/w goes beyond this
 - Exposing other datapath processing primitives (existing + new)
 - Accessible and programmable via P4 (high-level DSL)
 - Realize new functions (not fully arbitrary) in the datapath
 - **Researchers:** Propose an ASIC-level solution for new/existing problems and readily "realize" it in production hardware

Programmable Networks

Advanced Computer Networks



And with this, what we would really end up seeing is to have a flexible parsing and be able to match on non-standard fields. That means we will be able to program the parser to match the different kinds of fields which may be custom-built and not necessarily standardized. And this

will also make the faster and easier network evolution because now, we can have the ability to test out the newer protocols on the hardware in a much more quicker fashion because now we can write parsers in minutes and then run them out on the devices much more quickly.

And likewise, when we see this would really break what we saw earlier as a traditional protocol design that would take several years. And now, we are no more inclined to saying there is a need for hardware upgrades to facilitate support for certain protocols. It is just a software upgrade that would enable the new feature to be made baked-in into the device and make it ready to operate. And this also is a key for breaking down the investment or lowering the investments that you would have on the hardware where you are trying to reuse the same hardware with any of the software updates that we can make as long as the resource constraints are met. And through this, the hardware would go beyond on providing the data path processing primitives that is whatever are the existing primitives, and we may want to build newer primitives, which may be a combination of the existing ones to build or bake-in the new functionalities. And to do this, what we would need is a domain-specific language like a P4 that we will look into.

So, that the resources are made accessible and programmable through a construct of a high-level programming language or a domain-specific language for the switches. And it is just as a compiler abstracts the high-level language and the target-specific aspects, we want the P4 to abstract out the target-specific aspects and present the rich interfaces for what a programming language would cater to and meet the target-specific requirements as necessary. And overall, like this is how the researchers proposed an ASIC level solution for new and existing problems which would enable them to realize it very readily in the production hardware.

(Refer Slide Time: 29:19)

- Transactional Memory (SRAM) + Stateful ALUs
 - Stateful operations across multiple packets
 - Simple computations: add, subtract, approx. multiply/divide
- Queuing Telemetry Information
 - Enqueue/dequeue queue depth
 - Time spent in the queue
- High-resolution Timestamping
 - nanosecond-scale time stamps
 - Ingress/egress MAC timestamps, ingress/egress pipeline timestamps, etc.
- Packet cloning/replication
 - Flexible mirroring or conditional multicasts (at run time)



And what this would also call for, in addition, is the need for transactional memory and stateful ALUs wherein we want to persist certain data as we process the packets, as we parse the packets, and as we configure the operations on these packets. So, the stateful operations that we would want to do across multiple packets, if we want to tag and tailor the counter or you want to do the segmentation, we want to do defragmentation all of these would require you to maintain a specific state across the processing of multiple packets, and that is where the need for a transactional memory would be useful.

And then you would want the ALU, which would do basically the set of simple computations, it would be add, subtract, or an approximation of multiply or divide within just an integer multiply or integer divide would be able to work out, or you want to compute certain averages or like what we use DWMA kind of aspect, how you can do the approximate of these computations within the data plane of the hardware?

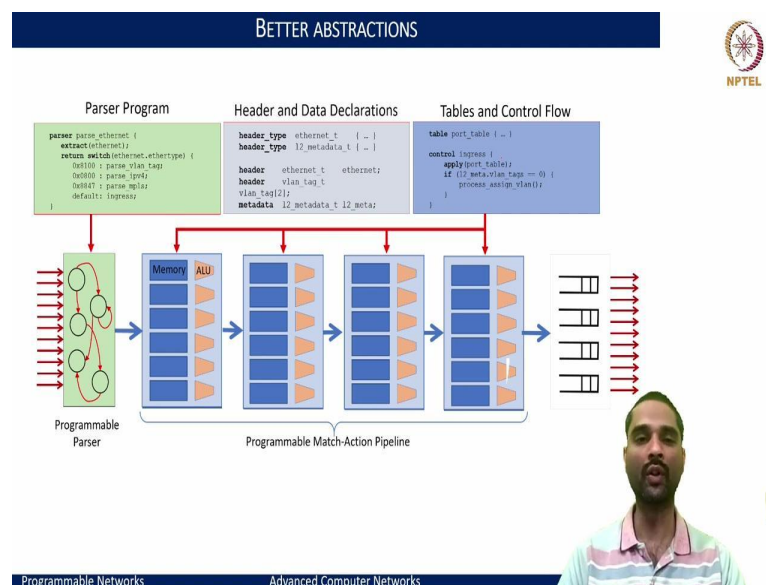
And also, what this would enable is also provide what we call as telemetry information in terms of how much of the time the queues were occupied for a queue for given port, or what was the depth in which the queue or the queue occupancy is being seen, what is the average queue occupancy, what is the time spent for packets in each of the queues, all of this information could be extracted.

And even, we could present very high-resolution timestamping within the devices to say when exactly the packets arrived, at what time the packets went on a nanosecond precision. And this

way, we could basically enhance the functionalities that device the data plane primitives were having but not exposed, to but now being exposed for the programmer to utilize them and build specific characteristics.

And in fact, when we want to monitor the traffic and want to see what kind of activities are going on, you may want to have packet cloning and replication capabilities so that we can mirror the traffic and see what is going on at a different kind of analytics or monitoring device. And we could also do conditional multicast, so that the packets can be sent out on different ports.

(Refer Slide Time: 32:08)

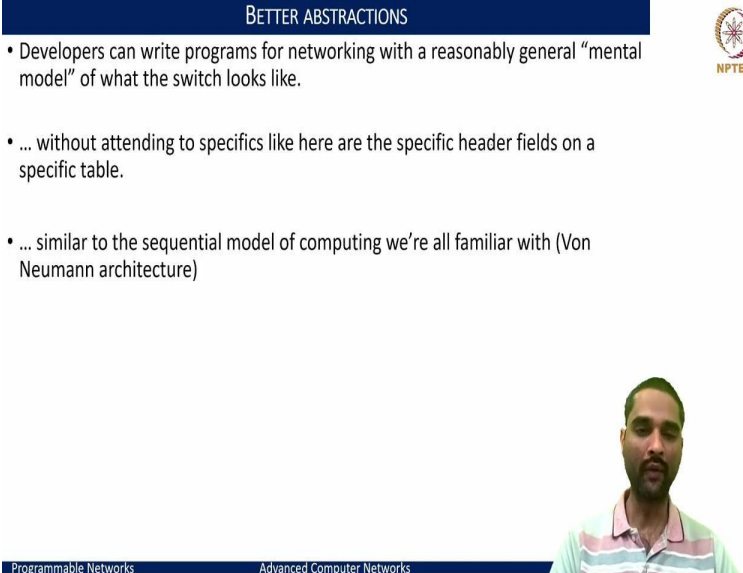


So, all of these calls for what we see as better abstractions that can be built in terms of abstractions for what way the parsers need to be built, what are the headers we want to argument, and what are the kinds of tables and control flow we would want to dictate on each of these devices? where in this example, we can see that if I have to build a parser, the parser would start with layer 2 or typical Ethernet. And then, from the ether type, you would want to extract what would be the next higher layer that you would want to look for be it the VLAN be it IPv4 MPLS, or if nothing matches, you want to just fall on with a default ingress approach wherein any packet that has come, you just treat it with respect to the port, and maybe this is a packet that is some informative thing that is meant only for a switch, you will want to handle it locally. So, all of these functionalities could be baked-in, and the ability to add custom headers thereof, like if I say that a switch, by default, does not have any notion of what is layer 2, layer 3, layer 4 header

but the programmer dictates the switch to consider Ethernet as a layer 2, IPv4 as a layer 3, MPLS on top of Ethernet as a layer alternative layer 3 to look for and then process.

The intelligence of what to parse and the intelligence of what tables or what actions to take is all being baked-in into these devices from the program, rather than the device coming up with its own intelligence that we want to tamper. And with this flexibility, what we would want as an abstraction to be provided for the user to build a device with P4 constraints.

(Refer Slide Time: 34:10)



The slide is titled "BETTER ABSTRACTIONS" in a dark blue header. It contains three bullet points: "Developers can write programs for networking with a reasonably general 'mental model' of what the switch looks like.", "... without attending to specifics like here are the specific header fields on a specific table.", and "... similar to the sequential model of computing we're all familiar with (Von Neumann architecture)". An NPTEL logo is in the top right. A video feed of a man in a striped shirt is in the bottom right. A footer bar at the bottom contains "Programmable Networks" and "Advanced Computer Networks".

- Developers can write programs for networking with a reasonably general "mental model" of what the switch looks like.
- ... without attending to specifics like here are the specific header fields on a specific table.
- ... similar to the sequential model of computing we're all familiar with (Von Neumann architecture)

NPTEL

Programmable Networks Advanced Computer Networks

And these better abstractions would lend readily enable the developers to write programming for the networking. And we can now reasonably think of how we write a program on the general CPUs to be the same way as writing what we want to do with each packet and express that in a programmatic fashion. And to do this, the programmer does not have to address or cater to the specifics of what header fields or what specific tables when these can be matched out on the target-specific aspects. But a programmer should be able to dictate what headers he would want to look for and what way that when there is a match on the header what actions to take.


So, the intent aspect would come from the user or a programmer and how things are to be realized when a particular ASIC behind it by the target-specific component in the compiler. And this is exactly what we have been very familiar with the Von Neumann architecture, where we


write the programs and let the CPU execute them for us. And the same way now, we write the programs and let the hardware execute it for us.

(Refer Slide Time: 35:38)

BETTER TOOLS

- Enable **compilers** to translate abstractions to actual hardware
 - Algorithms to allocate resources
 - Implemented once, implemented well
- Enable **formal verification** of behaviors existing in the network
 - Think of the network program is a contract between the control and the data plane
- Brings network software one step closer to “real” software





Programmable Networks

Advanced Computer Networks

And this would also enable us over time to build better compilers that can translate these abstractions to actual hardware in a much more efficient performance-oriented fashion. And algorithms also can be tweaked to allocate the right resources so that they are optimized on memory, and optimized on processing. And these can be done automatically behind the scenes as the compilers evolve. And this will also present a nice case to say that we implement the code once, implement it well, but executed on different hardware in different types much more readily, just as what we would do with very high-level languages, the programs that we write in high-level language.

And further, this would also enable us to build what we can see as having the verifications of the code that we would build to establish the behaviors within the network. Now, think of this network program as a contract between the control and data plane, and we want to verify whether this contract had been done correctly or not.


So, we could even bake-in the programs to do this kind of verification. And this would enable us to even rationalize and see what could be the problems and how they can be tackled in a programmatic fashion. And this brings us closer to saying, we are now interacting not with a


network as a hardware, but network as a software that is being programmed and able to provide the agility of how things can be brought into the networks.

(Refer Slide Time: 37:22)

REDUCED COMPLEXITY

- Network operators can remove the features they don't need
 - And the associated bugs
 - Avoid wasting time waiting for switch vendors to fix those bugs
- Network operators can add the features they need
 - New ideas and software are owned by the operators
 - ... rather than switch vendors
- More innovation in networking!





Programmable Networks

Advanced Computer Networks

And this would readily help minimize the complexity for many of the network operators for any of the features that they would want to bring in, and any of the bugs that they would have to track, and debug; all of this can now become much more easier to handle. And also, it would be open for innovation in terms of bringing in new ideas or software and deploying them on the hardware rather than waiting for the switch vendors to build the new things. And this is how innovations in networking can sprawl.

(Refer Slide Time: 38:05)

TELEMETRY



- Debugging correctness and performance issues in a live network is challenging
 - Packets not going where they're supposed to
 - Packets experiencing significant queueing or drops
- Measuring networks effectively requires collecting a lot of data
 - Recall number of packets every second even on a single 10 Gbit/s network
- Flexible packet formats allow you to put useful information directly on the original packet
 - In-band Network Telemetry (INT)



Programmable Networks

Advanced Computer Networks

And one of the other important aspects that really has come up is the telemetry we did mention briefly about it, and what it really enables is to debug precisely if my packet is lost, why it was lost, if my packet is experiencing delay, for what reasons for what aspects are really accounting to this delay, why is there anomaly with respect to certain behaviors of certain switches or routers, which links are creating trouble? all of this, if we would want to monitor on a per packet basis, if we have visibility at how each packet is processed within and we are able to control what each packet goes through, then we can definitely achieve this kind of monitoring and extract this information much more readily.

So, measuring the networks more effectively and collecting lots of data that we can build and process becomes much more easier. And this is where again certain specific aspects also started, we will look into this in a bit about what we call as In-band network telemetry wherein we want the networks to program and if we want networks to dictate when things go wrong or when things are right in terms of events that can be propagated in terms of messages that can get communicated, what is the means to do it, and how can we achieve it? All of this can be realized through this when we have control over the forwarding plane in full. And that is what the P4 tries to achieve. And through INT, we would also be able to achieve a fine-grained network telemetry.

(Refer Slide Time: 40:00)

How do we know if a programmable switch chip has the same power, performance and cost as a fixed function switch chip?



So, if we think of all of this, what we have now said is we are trying to go towards a programmable switching chip and get rid of the customer ASIC that otherwise would come with the closed APIs, but rather make those programmable. But then we also have to think of what is the cost, performance, and power overheads that these programmable switching chips bring.

(Refer Slide Time: 40:31)

	P4 Programmable "Tofino"	Fixed Function
L2/L3 Throughput	6.4Tb/s	6.4Tb/s
Number of 100G Ports	64	64
Availability	Yes	Yes
Max Forwarding Rate	5.1B packets per sec	4.2B packets per sec
Max 25G/10G Ports	256/256	128/130
Programmability	Yes (P4)	No
Typical System Power draw	4.2W per port	5.3W per port
Large Scale NAT	Yes (100k)	No
Large scale stateful ACL	Yes (100k)	No
Large Scale Tunnels	Yes (192k)	No
Packet Buffer	Unified	Segmented
Segment Rtg/Bare Metal	Yes/Yes	No/No
LAG/ECMP Hash Algorithm	Full entropy, programmable	Hash seed, reduced entropy
ECMP	256 way	128 way
Telemetry and Analytics	Line-rate per flow stats	Sflow (Sampled)
Latency	Under 400 ns	450 ns

Otherwise, both systems are identical:

- # of Ports
- CPU
- Power Supplies



Source: Nick McKeown "SDN 3.0", ONF connected 2019



And like I said, this evolution happened over the last few years to say almost around 10 years. And now, we are having these devices like P4 programmable Tofino, which was basically a barefoot Tofino device now acquired by Intel. Here, we can observe that the forwarding rates that

we are able to achieve with these P4 programmable Tofino devices are on par, or in fact, better like 5.1 billion packets per second on a programmable switching chip, as opposed to the fixed function switching chip that is providing around 4.2 billion packets per second. So, the forwarding rates, you can see our requests are comparable and, in fact, better with these programmable chips.

And when we see from the power that these devices take again, we can see that the typical system power drawn 4.2 Watt per port seems to be much more reasonable than the fixed function switching chips with 5.3 Watt per port. That means we are good on power, we are good on the performance. And likewise, if we consider the latency of packet processing that is happening, you can see that they are both around 100, 400 nanoseconds. That means the performance metrics and the power metrics are all on par with the fixed function switches that we otherwise see.

And in fact, now because these are programmable, we are able to support many more of the functionalities than just providing the switching capabilities like it can account for NAT within the programmable switch with roughly around 100k rules that we could support format. And likewise, access control lists that could be configured for again, the same so we could use either of like the NAT rules, or ACL rules within an enterprise network, or you want to use the same for just supporting the P4 tables within the core networks. That customization allows you to use the same device in the way that the real utility is for a given device and analytics again, like if you look as a key aspect, we can do line rate per flow statistics, as opposed to just having a S flow, which is basically a sampling method for one out of N packets that you would want to capture and try to predict the behavior. So, you would be able to do better fine-grained monitoring as well with these kinds of devices. And this has happened primarily because of lot of evolutions that happened over the recent past.


And in every other aspect, if you consider the number of ports that you could support 25 Gig, 10 Gig ports, the CPUs that you would use, or the power supply that you would plug, all of these aspects remain the same. So, the number of 100 Gig ports is 64 in both cases. And so, there is nothing that we would consider that the programmable switches would have degradations in performance or in terms of power, are no more true.


And this is where the case for many of these programmable switches has already made inroads into many of the data centers and enterprise networks. In fact, Microsoft Azure, they came with the smart NICs that they deployed in early 2015 and now, they are moving towards these programmable switches.

(Refer Slide Time: 44:45)

LIMITATIONS IN DATAPLANE PROGRAMMABILITY

- **High-level constraint:** all processing MUST maintain line rate
- No “loop” constructs
- No floating point computations
 - Only approximate computations possible
- Single Ported, per-stage SRAM memory
 - Single memory entry can be read/updated in one pkt pass





Programmable Networks

Advanced Computer Networks

Nonetheless, there are specific limitations when it comes to the data plane programmability that we also need to keep in mind. One is there are specific high-level constraints in terms of if the device needs to meet the line rate processing, like when we speak offline rate I mentioned in the last lecture link, if it is a 10 Gig NIC with the very small packets of 64 bytes, you would end up with having to support almost roughly 14.8 or 15 million packets per second. And as we scale from 10 Gig to 200 Gigs, you are speaking about like 150 roughly 150 million packets per second rate, or at least around 50 million packets per second, when we consider roughly 200 or 256-byte packets and likewise.

So, this means whatever the match actions, whatever the parsing that we want to build, should ensure that it ends, and there should not be any loop construct, which will make the things run in an infinite fashion. So, support for loop construct is currently not supported. And we also know that floating-point computations are intensive. And hence, adding like it could be possible that we can add a floating-point co-processor and do the computations. But in fact, it is much more


easier to do, if we can do approximate computations or integer computations within the switch so that we are both time-sensitive and be able to do with less of power.

So, the floating point of computations is currently not supported in the P4 part. And also, if there is an entry that is going to be read and updated, the way we access the memory can add up to the latency, especially if you are trying to write reupdate and read the changes, then you may end up having a lot of cycle stalls for accessing that memory. Hence, the accesses right now are basically single-ported per stage SRAM memory where we could add multiple ports, but that also comes with a cost and overheads.

And hence, keeping it simple, having a single memory entry for read and update in one packet parse. So, that means that you would want to read updates in each of the stages in as you go rather than having to do read write updates in a simultaneous fashion, which could be addressed in the later stages.

(Refer Slide Time: 47:39)


SDN FORWARDING ABSTRACTIONS – FUNDAMENTAL RESEARCH PAPERS



- Programmable data plane

1. "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN", SIGCOMM'2013.
2. "P4: Programming Protocol-Independent Packet Processors", SIGCOMM Computer Communications Review, July 2014.

Programmable NetworksAdvanced Computer Networks



And these aspects, essentially in a nutshell define the overall aspects of the programmable forwarding data planes. And to understand them in a lot more detail, we would have to look at these two papers that is Forwarding Metamorphosis that is Fast Programmable Match Action Processing in hardware for SDN, which was exactly the RMT paper. And follow-up is the P4 programming Protocol Independent Packet Processors, which was a SIGCOMM CCR paper in

July 2014. That would lay the foundations for a better understanding of the programmable forwarding planes.