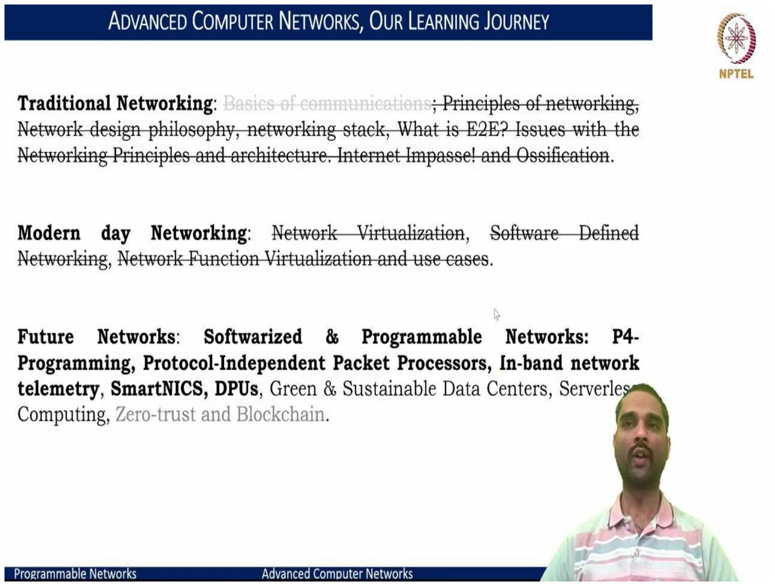
Advanced Computer Networks

Professor Sameer Kulkarni
Department of Computer Science Engineering
Indian Institute of Technology, Gandhinagar
Lecture 44
Programmable Networks – Data plane Programmability – Overview I

(Refer Slide Time: 0:19)



ADVANCED COMPUTER NETWORKS, OUR LEARNING JOURNEY

Traditional Networking: ~~Basics of communications; Principles of networking, Network design philosophy, networking stack, What is E2E? Issues with the Networking Principles and architecture. Internet Impasse! and Ossification~~.

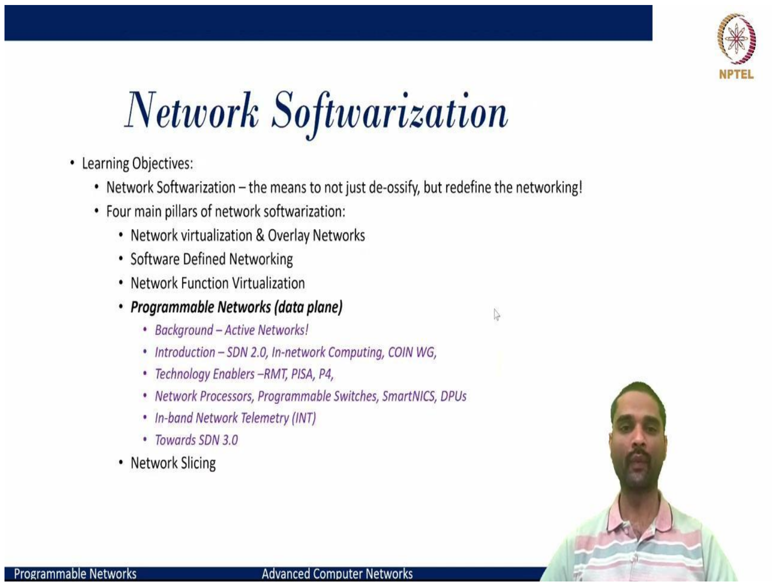
Modern day Networking: ~~Network Virtualization~~, ~~Software Defined Networking~~, ~~Network Function Virtualization and use cases~~.

Future Networks: **Softwarized & Programmable Networks: P4-Programming, Protocol-Independent Packet Processors, In-band network telemetry**, **SmartNICS, DPUs**, Green & Sustainable Data Centers, Serverless Computing, Zero-trust and Blockchain.

Programmable Networks Advanced Computer Networks

In the last 3 weeks, we looked at how network virtualization, software-defined networking, and network function virtualization and the associated use cases led towards the softwarization of the network. This week, we will look into another major change where networking devices can be programmed to bring loads of customizations and behavioral updates to the traditional networking data plane elements. In this aspect, our primary focus for this week would be programmable networks. So, the data plane networks, especially the P4 and in network computing.

(Refer Slide Time: 1:02)



Network Softwarization

- Learning Objectives:
  - Network Softwarization – the means to not just de-ossify, but redefine the networking!
  - Four main pillars of network softwarization:
    - Network virtualization & Overlay Networks
    - Software Defined Networking
    - Network Function Virtualization
    - ***Programmable Networks (data plane)***
      - *Background – Active Networks!*
      - *Introduction – SDN 2.0, In-network Computing, COIN WG,*
      - *Technology Enablers –RMT, PISA, P4,*
      - *Network Processors, Programmable Switches, SmartNICS, DPUs*
      - *In-band Network Telemetry (INT)*
      - *Towards SDN 3.0*
    - Network Slicing

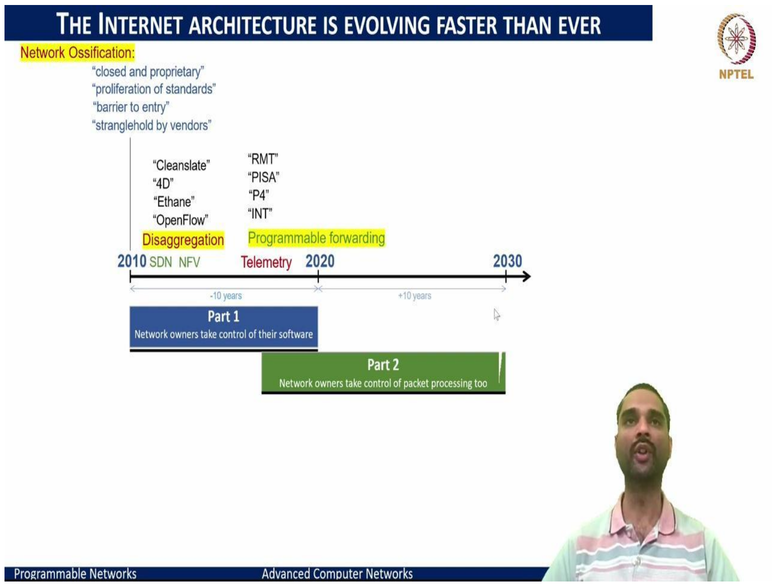
Programmable Networks   Advanced Computer Networks

And in this, we will try to cover the basic aspects of how it all started, from the introduction of SDN, to a new generation of SDN, SDN 2.0, leading to what we call key technology Enablers like RMT reconfigurable match tables, and PISA, P4 in this context. We will also look into the key aspect of network processors, programmable switches, SmartNICS, and emerging DPUs in the context of a programmable networking data plane. We will also look into In-band Network telemetry or INT in terms of how it is helping monitor better and even in addressing several of the network telemetry aspects.

Then we will summarize with what we could see as the programmable networks, starting from just opening up the programmable control plane to the programmable data plane and then to automation of this programmable data and control plane aspects put together to how they can better help build these softwarized networks. And in this, what we need to basically understand is that a programmable data plane represents a versatile way to forward the packets and support various formats of protocols, not just the ones that we traditionally accept or established ones.

We could also build custom protocols with much more ease and have a lot more flexibility in working them out and testing them on real hardware. And this is the case where the P4 language that is used to configure these for parsing and forwarding actions for on the switches helps a lot.

(Refer Slide Time: 3:01)



The Internet architecture is evolving faster than ever

Network Ossification:
"closed and proprietary"
"proliferation of standards"
"barrier to entry"
"stranglehold by vendors"

"Cleanslate"
"4D"
"Ethane"
"OpenFlow"
Disaggregation

"RMT"
"PISA"
"P4"
"INT"
Programmable forwarding

2010 SDN NFV Telemetry 2020 2030

~10 years +10 years

Part 1
Network owners take control of their software

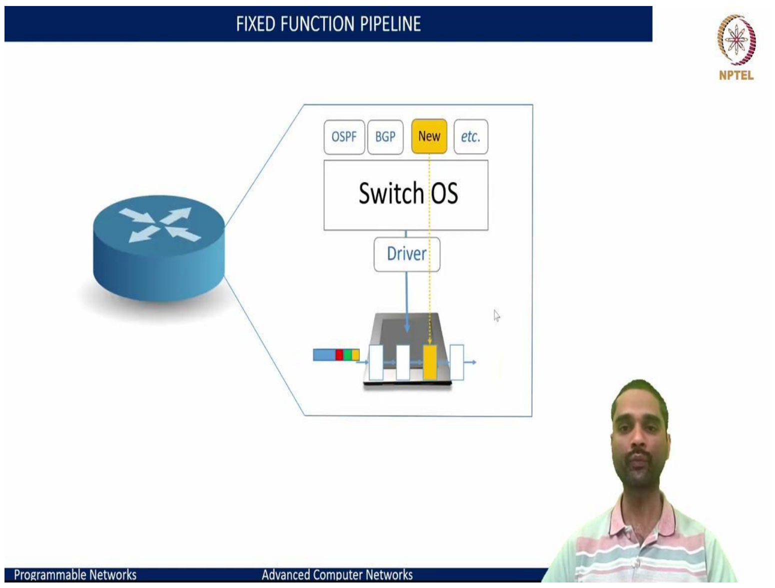
Part 2
Network owners take control of packet processing too

Programmable Networks Advanced Computer Networks

So, to look at, in a quick summary, what we have seen is we started with a network classification where the entire merchant silicones were all closed and proprietary, where the proliferation of standards had been just really not possible unless it took several years to bring in the innovations. And these were the key barriers to entries. And going forward, what we saw was a clean-slate architecture, 4D approach, and OpenFlow as the key areas resulting in SDN and NFV kind of mechanisms to desegregate this and deossify the networking infrastructure, where the network owners were now able to take control of their software.

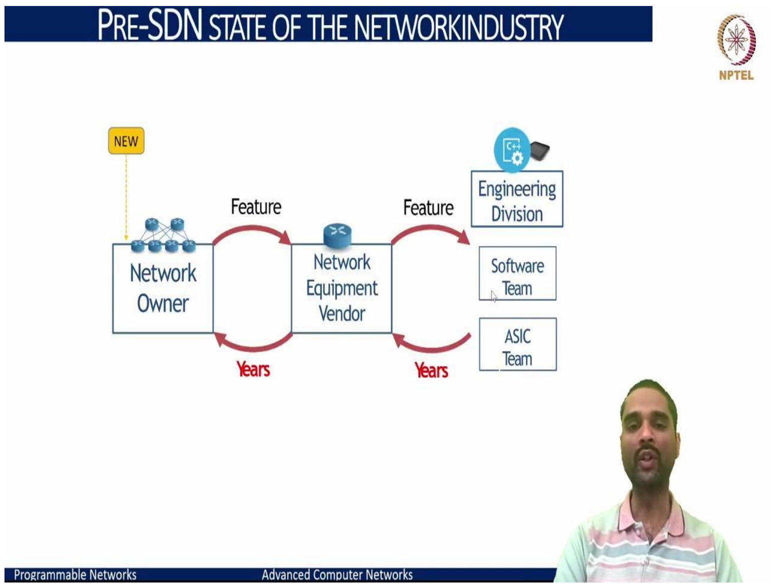
And our journey will continue to learn about part 2, which is where the network owners can take control of the packet processing too. That means they are going to have now better control and be able to deploy how the programmable forwarding can be realized. And be better able to tell, in essence, what is happening with respect to the packets as they are processed by different networking elements within the network.

(Refer Slide Time: 4:23)


FIXED FUNCTION PIPELINE

So, let us look at what we typically call as these fixed plain function pipelines or like if we look at a traditional router, what it typically has is a control plane and data plane embedded within a device, where it comes with its own firmware or Switch OS which drives in terms of how the functionalities would be run within such a device. So, if we take OSPF and BGP as some of the routing algorithms. Then they would implicitly derive the tables and then push the rules onto them to say how to forward and process the packets. And if we take an example of, we want to build something new characteristic onto these devices, that means a new control functionality would mean a new means to add or process the packets in the forwarding plane.

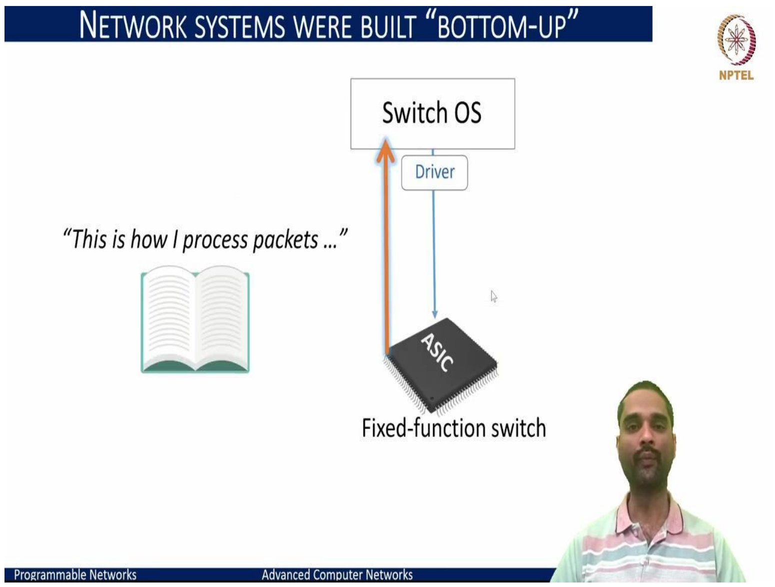
(Refer Slide Time: 5:19)



And to do this, what we have seen also in the earlier case was that, as the network owners try to bring these innovative ideas in the plane, they have to go to the network equipment vendors to support those features so that they can be embedded and take control and data plane as necessary. And once the network equipment vendors bring in those features that are deployed through their engineering division team, either as software or in the ASIC and the hardware, and then push back these prototype equipments that could now be tested by the network owners, we have seen this kind of cycle takes years together.

And this was where exactly the things were really difficult to bring in the innovations. And if we had to break this, we saw earlier, one way that we can softwarized the entire network components. If we softwarized, we are now running them on commodity hardware. And we say we also saw that there is a tradeoff between performance, price, and power that we have to match up with. But in this context, now, if we want to think back, there is an issue in terms of how these features are going to be supported. But what if we are having control over the equipments rather than falling back on the OEMs to make these devices and this is exactly the other kind of state and transition that we are now talking about.

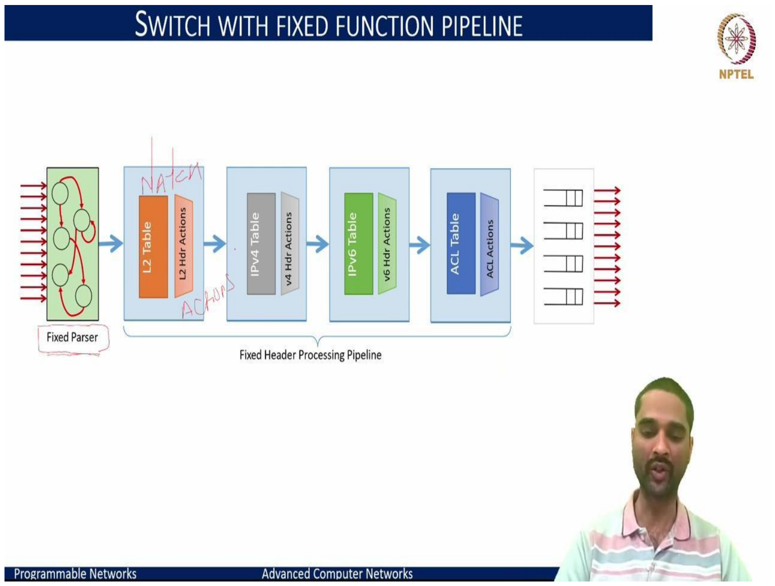
(Refer Slide Time: 6:55)


Network systems were built "bottom-up"

So, going forward, to see what needs to be changed, if we have to make these ASICs to behave the way we want. So, to look at what really was happening is the OEMs would have the ASIC, which provides a fixed set of functionalities. And these functionalities are exported out through the switch OS, which is also, again, proprietary, and the way the APIs will be given out for us to configure will tell what exactly the ASIC will do. In terms of what all are its capabilities that have been baked in into the ASIC and what are all the configurations that we can do for this particular ASIC?

And the OEMs would facilitate the firmware and drivers to ensure that things can be updated on these custom ASICs. And these, in essence, dictated how they would process the packets rather than allowing the user or the operator to say what an ASIC should do to process a packet.

(Refer Slide Time: 8:06)


SWITCH WITH FIXED FUNCTION PIPELINE

And if we put this in perspective of how the actual processing within an ASIC would occur as the packets arrive. The first thing that they will do is parse the packets. By the notion of parse, what we are trying to say is as the packets arrive, you look at the layer 2 header that Ethernet if it has supported, layer 3 headers, the IPV4, IPV6, and if the ASIC did not have the IPV6 match, it would fail. And likewise, at layer 4, if it would match up for TCP, UDP. And if there were any other protocols that ASIC did not understand, it would again fail. So, these parsing capabilities for what protocols that packet headers would be matched to were confined in this by the ASIC developers. So, it was not easy for me if I say that I want to add a new protocol layer 4 protocol; that would not work unless the functionality was baked-in into this fixed set parser to match that header and process. And once you match, what typically happens in the processing is, if I am having an L2 switch, it would look up the L2 table to match the source and destination MAC address and then update the L2 header.

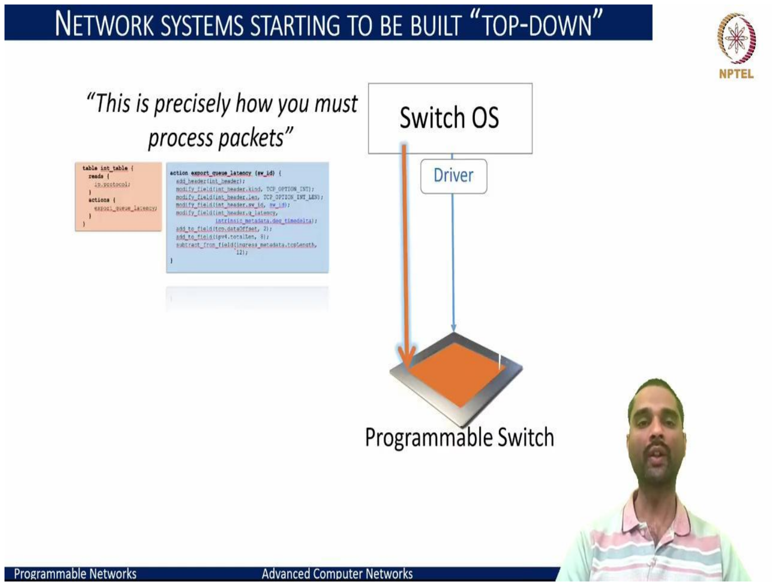
So, if it was a switch that is trying to forward and then say, what is the next destination header, so it would update. So, this is where basically you would have the match on the entries, and then what are the actions that you would want to take once you match a particular rule. So, it could be a layer 2 table that you would look up for all the source destination MAC for each of the combinations of packets that you may expect for source and destinations to arrive and then take certain actions.

And likewise, once you process, you will then process the layer 3 headers, that is, IPV4 table, to match up the IPV4 like in routers, when we saw the destination match that have the destination IPs, which are being mapped to say what actions you take, like what output port that you would want to send the packet out on.

And likewise, it could be IPV6 tables where you would want to look up for the IPV6 entries and then take corresponding actions. And these are what we call as the fixed header processing pipelines. And then you would have the access control tables, which would say whether you should permit at particular kinds of flows or with particular kinds of rules to be allowed to be processed or to drop.

And this is how typical fixed function pipelines were baked-in into the ASICs. And once the packet process is getting processed through all of these chains, it will then be rendered onto a particular output or dropped silently or discarded within the device based on what actions have been defined at each of the match action pipeline stages. And this is where the rigidity of the hardware comes in. And if we have to think of breaking this, what we would really need is to have the ability to customize these specific functionalities.

(Refer Slide Time: 11:35)



NETWORK SYSTEMS STARTING TO BE BUILT "TOP-DOWN"

"This is precisely how you must process packets"

```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency,
               intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength,
                      12);
}
```

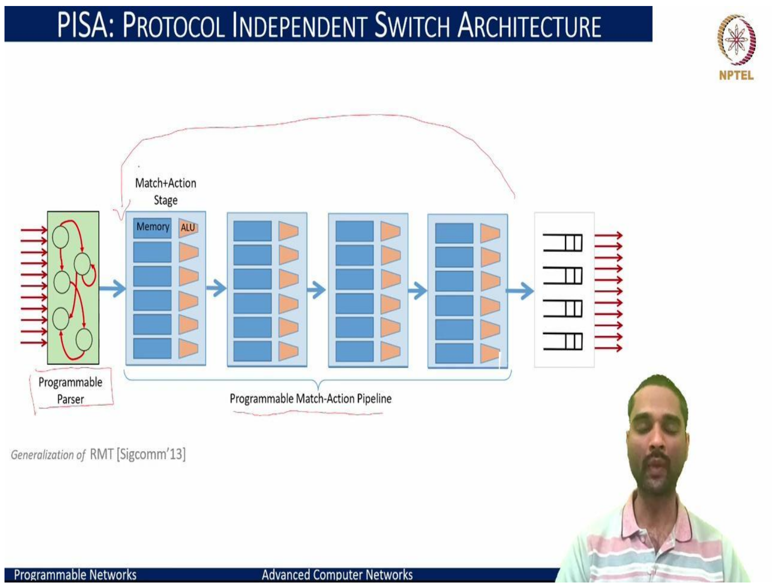
And what that really means is, we would want to ensure that we dictate to the switch what exactly it needs to do? In essence, what we are trying to do is now program a switch saying what it should parse for, what actions it should take, and how it should take a particular action. In essence, we would now be baking in the functionalities for the switch to behave the way we would want it to behave. And this by notion, we are saying that this is precisely how the switch should now process the packets rather than the OEM telling what a switch will do, the network operator or a programmer would tell you how the ASIC and I want this ASIC to do these functionalities.

For example, I want the switch to understand there is a new protocol header that I have added. And that is maybe VLAN or VXLAN; if that capability exists, it should see. If it sees that it is a VXLAN it should decapsulate and try to process the inner header and process the packet rather than just looking at the first layer, 2-layer, 3-layer, 4, and change.

And it could be that I want to create something new protocol at layer 4. So, in an RCP protocol, then I want that functionality to be made understandable by the switch to say it is not just TCP or UDP but some other intermediary that is sitting between TCP and UDP. And you want to process that kind of functionality.

And then take necessary actions as well, like once we match particular fields, we want to add or translate particular entries like if it was just a forwarding, we would want to change just the L2 MAC. If we would want to do like a NAT kind of application, we would want to change source IP, source port, destination IP, and destination port combinations to public and private as we would want to allow the NAT rules. So, likewise, we want these configurations to be applied on the switch ASIC on as we intend them to do and perhaps on the fly. Rather than saying I programmed it once and it remains for its lifetime, I want it to be programmed as and when you see the need.

(Refer Slide Time: 14:12)


PISA: Protocol Independent Switch Architecture

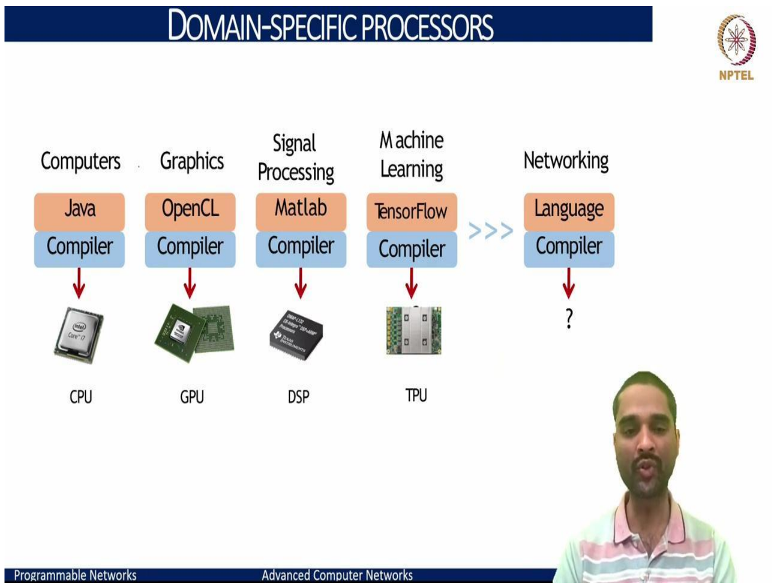
NPTEL
Match+Action Stage
Memory
ALU
Programmable Parser
Programmable Match-Action Pipeline
Generalization of RMT [Sigcomm'13]
Programmable Networks
Advanced Computer Networks

And to do this is where the protocol-independent switch architecture or the PISA comes in. We will look into this in much more detail, but to understand in a nutshell, we are now defining a programmable parser that means a user can dictate what packet headers should be matched along. So, if I build a custom header, I can build the parser to say that it can record the device can now recognize the custom header and match these functionalities within its parser's capability to detect and process as the packets arrive.

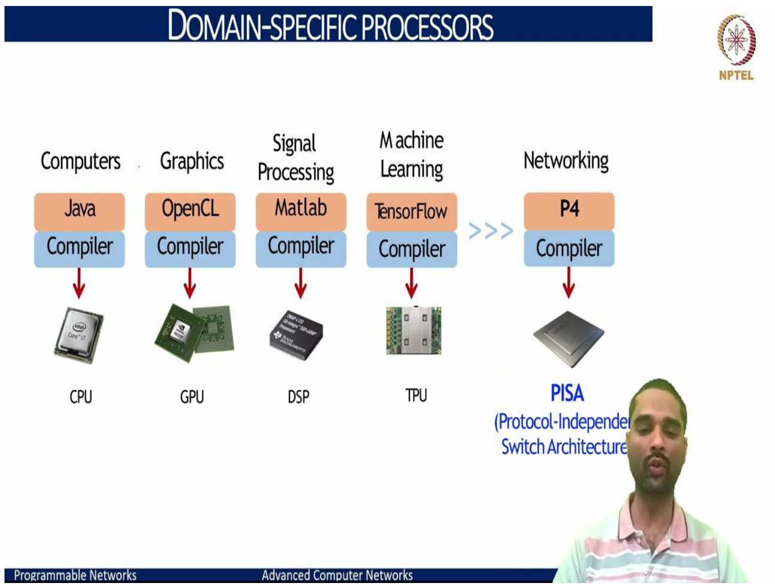
And as a programmer, I would also want to dictate not just that, I parse but what actions do I take when I see a particular set of headers that I will encounter? And that means I want to also dictate what are the programmable match action pipelines that I want the switch to do. That means, I may want to do the layer 2 header match and layer 2 actions to be taken, I may want to do layer 3, layer 2.5, layer 4 or encapsulation, decapsulation, all of these aspects we would want to bake in. And say that I dictate based on once I receive a particular IP packet, how would it look go for TCP? How would it go for a UDP based on the next header field in the parser?

And so, we want to build the aspect or a notion of the ability to parse the packet, the ability to match custom headers, and for each of the matches ability to take certain actions. And that means we would want to be programming both the parser as well as the a match action pipelines. And at times, if we see that, we have an encapsulation IP in IP or ETH in ETH, we want these to be reparsed. So, that means the ability to also redo specific actions based on whatever the aspects that we would have seen for a particular packet would also be an aspect that we look for in designing this switch architecture.

(Refer Slide Time: 16:41)



DOMAIN-SPECIFIC PROCESSORS



DOMAIN-SPECIFIC PROCESSORS

And, if we put this analogy to what we have seen, like when we can consider our computers, what we have are the CPUs, where you have a high-level programming language and the specific compilers that would allow your high-level language to be run by these CPUs. So, if we take intel like O7 then ASIC architecture, the ISA is defined, and this high-level program like Java, if

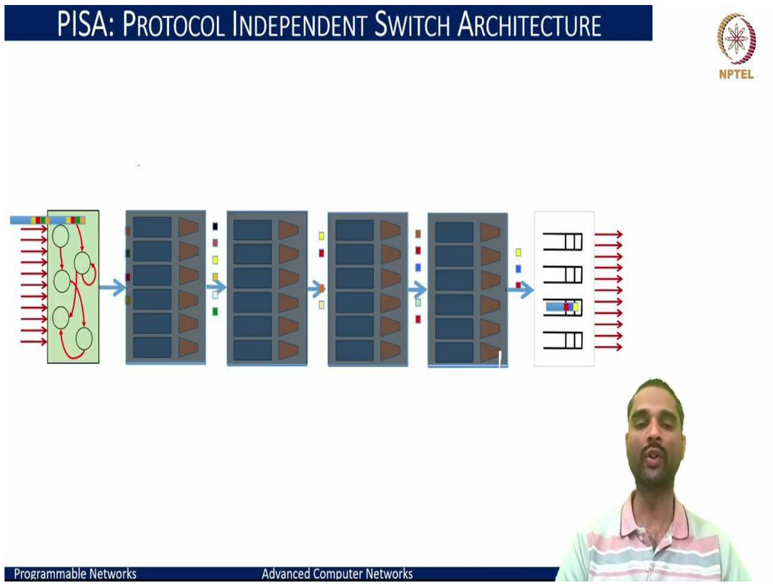
I have a Java compiler, it will generate a bytecode, and the interpreter will then run the exact instructions, what a user wants, but run by the hardware, that is the processor.

And likewise, on the GPU. If we look at the way we use OpenCL or CUDA, this specifies what exactly is the user's intent of what GPU needs to do as a program, and that when it is compiled with OpenCL, or CUDA and pushed onto the GPU, GPU would exactly do what a user wants. While it is all done on hardware. And likewise, if we look at the DSPs, and we use MATLAB as a tool to generate signal processing applications and use MATLAB as a programming construct to compile and run the specific code on the DSPs.

And even if we take on the TPUs, or for machine learning use cases, we would have the TensorFlow code that we would run on the TensorFlow compilers and run them on the TPUs the way the user intends to do. So, we would want similar constructs to be made available for networking. That means we might have what we would need is specific hardware, which is generalized, which is going to take the intents of how you would want a network to behave as a program. And that program compiled for a specific language would then allow the ASIC that we just discussed, user processor that would do the processing, just as any program would run on a CPU or on GPU and likewise.

And this is where exactly the P4 fits into the space, saying P4 has a language for a user to express what a switching device should do and how it should parse the packet, how it should match, and what actions it should take. All of this as a P4 language and then user P4 compiler to generate these instructions that could be run on this switch device, and this is what we will look into understand how overall this goal can be realized as we go along.

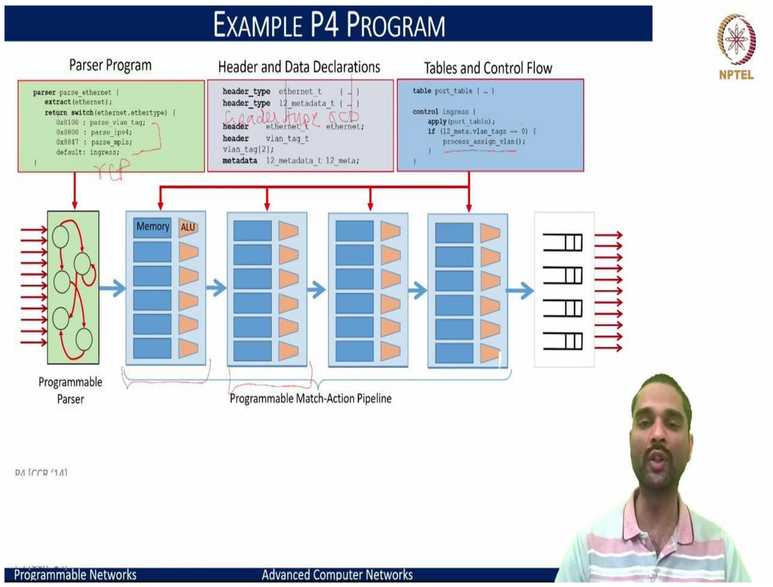
(Refer Slide Time: 19:35)



PISA: Protocol Independent Switch Architecture

And what this really means is for each and every packet when we receive on the switch hardware, it should be able to do independent parsing. And as it would parse the contents of the packets, it would want each of these packets to be applied with their corresponding match and actions. So, that different kinds of packets can take different paths within the same processing hardware, be executed with different actions, and then emitted out.

(Refer Slide Time: 20:13)



Example P4 Program

Parser Program

```
parser parse_ethernet {
   extract(ethernet);
   return switch(ethernet.ethertype) {
      0x8100 : parse_vlan_tag;
      0x0800 : parse_ipv4;
      0x8847 : parse_mpls;
      default: ingress;
}
```

Header and Data Declarations

```
header_type  ethernet_t    { … }
header_type  l2_metadata_t { … }

header    ethernet_t    ethernet;
header    vlan_tag_t
vlan_tag[2];
metadata  l2_metadata_t l2_meta;
```

Tables and Control Flow

```
table port_table { … }

control ingress {
   apply(port_table);
   if (l2_meta.vlan_tags == 0) {
      process_assign_vlan();
   }
}
```

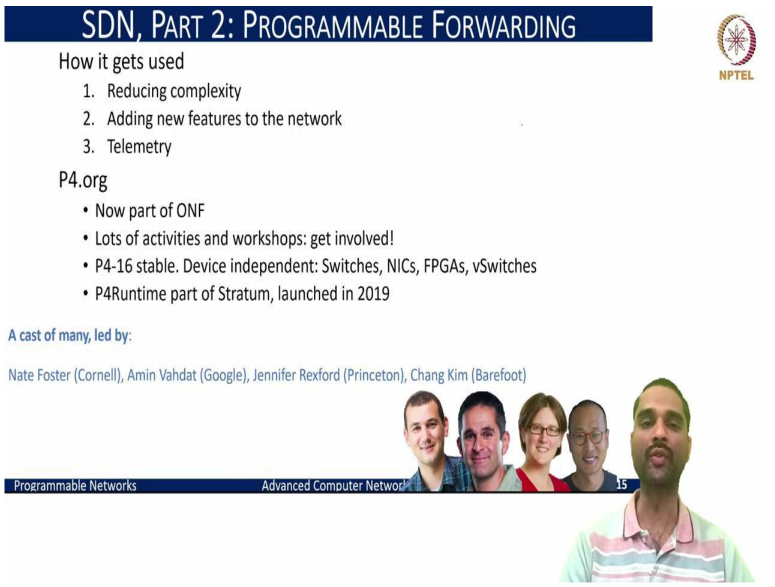
And to put it in simple perspective, for any packet that comes, we would want to first parse the Ethernet header. And once we extract the Ethernet header based on the type of Ethernet type, what would be the next parsing that we would want to do? Maybe we have a VLAN tag or IPV4 or MPLS header? We would want to operate and process extracting the details for each of these independently. And we may want to add any other custom things we could add and then process them here. And if nothing is matching, by default, we could say that just take the regular path or ingress pipeline as a common fallback.

And for each of these parsing, we would want to dictate how these tags would look, not necessarily what has been standardized, we could also build any custom header types. So, that if I had RCP as my next header, I would want to build what it means for the header type RCP. And likewise, so, if we are able to dictate bit by bit, what it should be able to match, and what it should be able to extract from the packet match and then take necessary action.

So, we will define the header and data types. And these header and data types will then be used for the parser to dictate, what you are looking for, and what is the exact next field that you would want to look up and match. And once we have these matches, how or what kind of actions we would want to take or the control parts that we would want to build? So, we typically build these as a set of tables, each of these tables where we want specific matches to correspond to specific actions that we would want to take.

Like, if we encounter, in this case, the VLAN tags, we will want to assign whatever the VLANs that you would see based on the port table of where the packet came and where the packet needs to be sent out. And then, correspondingly, apply the VLAN tags and send them out. So, this is where we want the flexibility in parsing, in defining the data types for such parsing, and defining what kind of actions and controls we would want on each of these packets in processing. And this is what would make open ASIC just flexibility for programming any of the custom types and any of the custom actions onto these devices.

(Refer Slide Time: 22:54)



SDN, Part 2: Programmable Forwarding

How it gets used

1. Reducing complexity
2. Adding new features to the network
3. Telemetry

P4.org

- Now part of ONF
- Lots of activities and workshops: get involved!
- P4-16 stable. Device independent: Switches, NICs, FPGAs, vSwitches
- P4Runtime part of Stratum, launched in 2019

A cast of many, led by:

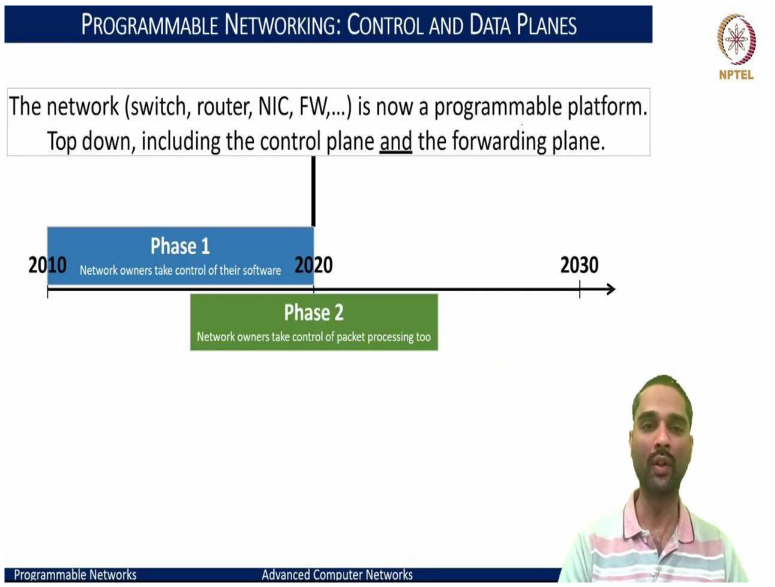
Nate Foster (Cornell), Amin Vahdat (Google), Jennifer Rexford (Princeton), Chang Kim (Barefoot)

Programmable Networks Advanced Computer Networks 15

And this is what was termed as SDN 2.2, which is basically programmable forwarding. And what it really adds to is to reduce the complexity in terms of what needs to be done onto the hardware to achieve this. That is the goal of looking forward to what can be done so that this becomes much more simpler. And how this can enable us to add different kinds of newer features onto the network itself. Like if we said we have lots of encapsulated headers that we have to process and parse, we should know we could be able to do multiples of these encapsulations and decapsulations within the hardware rather than waiting for the hardware OEMs to build those support to encapsulate or decapsulate the packets. Further, the most crucial part as the networks grow more and more complex is also about how we could monitor and build the aspects of what exactly happened with the packets as and when they are processed within the network. And all of these were, in essence, the key aspects that led to SDN 2.0 and the major contributor to this is the P4.org which is now part of the open network foundations.

But there were a lot of activities and workshops that started on starting from P4, and now we are into what we call as the P416 specifications, which dictate what is means to build device-independent switches, what is the construct of the programming language, what is the specifications for dictating such switches? So, now we are able to create these custom functionalities within the well-baked ASIC as long as we are able to support these P4 specifications. And P4 runtime as a part stratum was launched in 2019, which now enables for any of the users to build the programs onto these switches. This was all led by basically Nate Foster, Amin Vahdat, Jennifer Rexford, and Chang Kim, which was a barefoot network, now taken up by Intel. But all of this has happened in a very rapid time in the last 7 to 8 years. And our essence would be to get to what or each of these means, what were the foundations and how we have arrived to this stage, and what is the way forward in a lot more detail.

(Refer Slide Time: 25:45)



PROGRAMMABLE NETWORKING: CONTROL AND DATA PLANES

The network (switch, router, NIC, FW,...) is now a programmable platform. Top down, including the control plane <u>and</u> the forwarding plane.

2010
Phase 1
Network owners take control of their software
2020
2030
Phase 2
Network owners take control of packet processing too
NPTEL

Programmable Networks   Advanced Computer Networks

In a nutshell, what this would leave us is to have a network element like the switches, routers, even including our network interface cards that we plug into our host devices to be thought as not just hardware but as a programmable platform where we could dictate what kinds of packets that our NICs or our switches and routers would look up what exactly we can make them to process in terms of dictating how the control and forwarding both of these planes need to behave on these kinds of devices.