

A circular logo with a blue border and a light blue background. The text 'P4 Expert Roundtable Series' is centered. 'P4' is in purple and green, 'Expert' is in blue, and 'Roundtable Series' is in a larger blue font. A small white cartoon animal is to the right of 'Expert'. Below the text is the date 'April 28-29, 2020' and 'Hosted by:' followed by the ONF logo.

P4
Expert
Roundtable Series

April 28-29, 2020

Hosted by:



Building and Delivering a P4-based Network Tester

Ram Murthy

Software Lead

Keysight Technologies

AGENDA

- Introduction to Network Testers
- Case Study: BGP Convergence over ECMP
- A Deeper Dive into Implementation
- P4 vs FPGA product development
- Key takeaways and conclusions



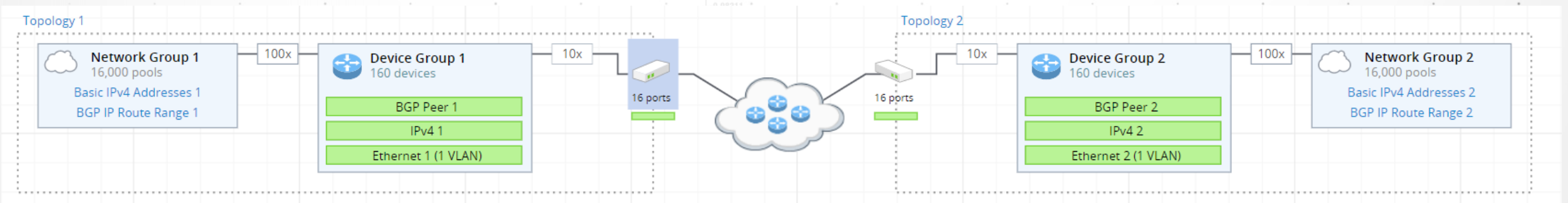
UHD100T32 32-Port 100GE Test System

Ultra-high-density 100GE QSFP28 test system



What defines a Network Tester?

- The main objective of a tester is to measure and characterize network devices and systems.
- Single device (DUT) metrics:
 1. Characterize throughput performance with flow tracking
 2. Measure latency of device
 3. Verify forwarding correctness with sequence checking
 4. Emulate traffic scenarios from the simplest point to point link all the way to fully-meshed
- System (SUT) metrics:
 1. Verify stateful protocols (i.e. BGP, OSPF, IS-IS)
 2. Validate routing performance and convergence (i.e. ECMP)



Can we leverage P4 to make a Network Tester?

KEY QUESTIONS

- Can we make a viable product which addresses use-cases and meets performance requirements?
- Can we use existing Ixia IP, maintain the same user-experience, and integrate a completely new dataplane technology?
- What are the resources required to complete the project?
- What is the developmental and build workflow?



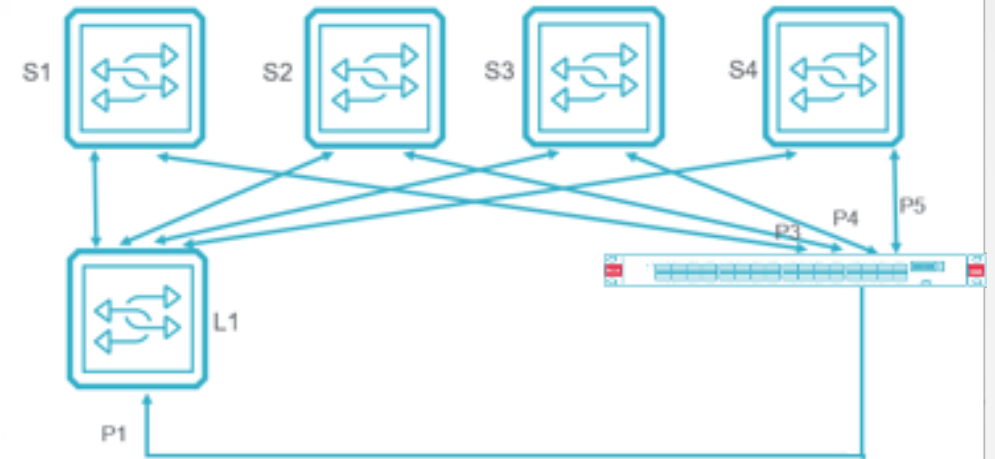
A Case Study: BGP RIB/FIB Convergence over ECMP

Test Objective

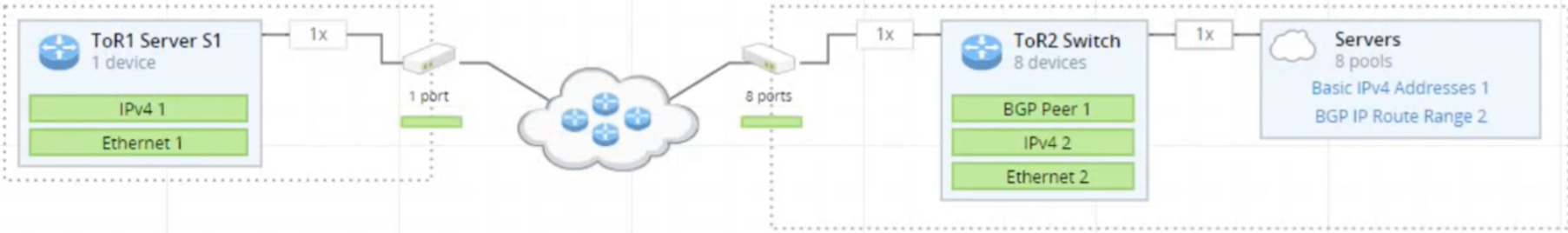
- Validate how quickly ECMP links can rebalance traffic after failures

Methodology

- Tester simulates a remote link failure
- Observe the number of lost packets after convergence
- Analyze traffic flows before and after the convergence
- Vary the number of ECMP links & 5-tuple



Sonic ToR1



Administrator: Command Prompt - python PythonApplication2.py

```
Status
Starting convergence tests ...
```

```
Tx Rx Load %
Ethernet - 001 100.00
Ethernet - 002 12.000
Ethernet - 003 12.200
Ethernet - 004 12.401
Ethernet - 005 13.001
Ethernet - 006 13.001
Ethernet - 007 12.700
Ethernet - 008 12.700
Ethernet - 009 12.000
```

```
Link State
Ethernet - 001 UP
Ethernet - 002 UP
Ethernet - 003 UP
Ethernet - 004 UP
Ethernet - 005 UP
Ethernet - 006 UP
Ethernet - 007 UP
Ethernet - 008 UP
Ethernet - 009 UP
```

```
ECMP Convergence Time
Conv DOWN Time [ms] 0
Conv UP Time [ms] 0
```

The Problem Statement

HOW TO IMPLEMENT USE CASE WITH P4?

Traffic Item	State	Enabled	Src/Dst Mesh	Route Mesh	Bi-Directional	Statistics Tracking	Sources	Destinations	Stack	Frame Size	Frame Rate	Payload
Traffic Item 1	UNAPPLIED	<input checked="" type="checkbox"/>	One - One	One - One	<input type="checkbox"/>	Traffic Item, Source/Dest Value Pair	Network Group 1	Network Group 2	Ethernet II VLAN IPv4 TCP	IMIX	100% Line Rate	Increment Byte

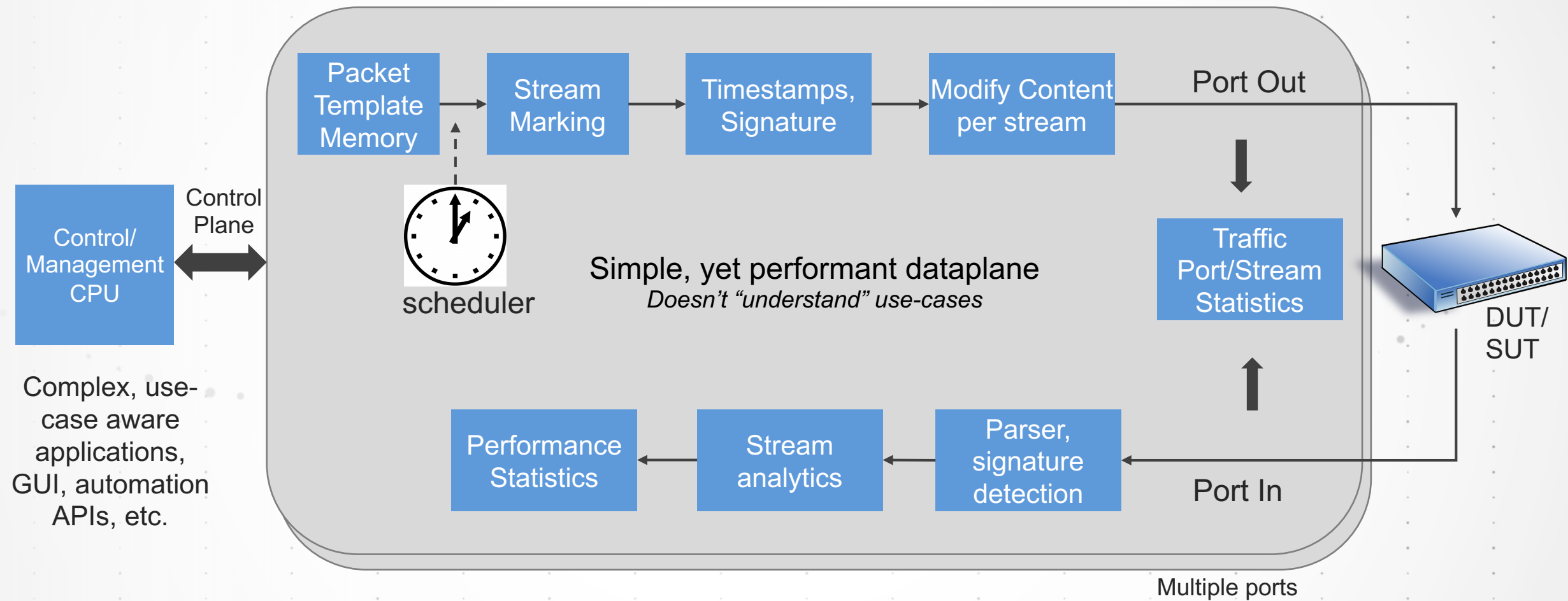
Primitives needed:

- Stateful BGP protocol emulation
- Precise traffic patterns over BGP routes for ECMP distribution
- Flow tracking and loss measurements
- Latency of flows on each BGP route
- Rate precision on each flow
- Precise measurements of live statistics per flow

VLAN-ID	Precedence	Source Address	Destination Address	TCP-Source-Port	TCP-Dest-Port
11	000 Routine	135.10.18.1	100.41.108.1	5140	564
11	000 Routine	139.21.37.1	104.214.120.1	5320	298
11	000 Routine	132.231.143.1	111.141.0.1	62996	528
11	000 Routine	135.32.60.1	117.247.225.1	5020	551
11	000 Routine	130.31.133.1	105.110.186.1	7000	347
11	000 Routine	132.22.43.1	102.252.220.1	62516	125
11	000 Routine	127.123.212.1	119.136.21.1	63596	664
11	000 Routine	132.147.251.1	104.40.174.1	5560	543
11	000 Routine	134.128.128.1	114.253.157.1	5740	486
11	000 Routine	131.184.40.1	118.144.176.1	62576	136
11	000 Routine	125.176.132.1	118.247.187.1	5380	184
11	000 Routine	122.128.210.1	115.137.3.1	7420	820

Tx Frames	Rx Frames	Frames Delta	Loss %	Tx Frame Rate	Rx Frame Rate	Tx L1 Rate (bps)	Rx L1 Rate (bps)
174,292,107,313	174,292,106,512	801	0	441,219,220.212	441,219,227.651	1,599,119,644,19...	1,600,004,572,71...
174,311,537,015	174,311,536,218	797	0	441,245,819.23	441,245,824.686	1,599,216,047,55...	1,600,037,227,10...

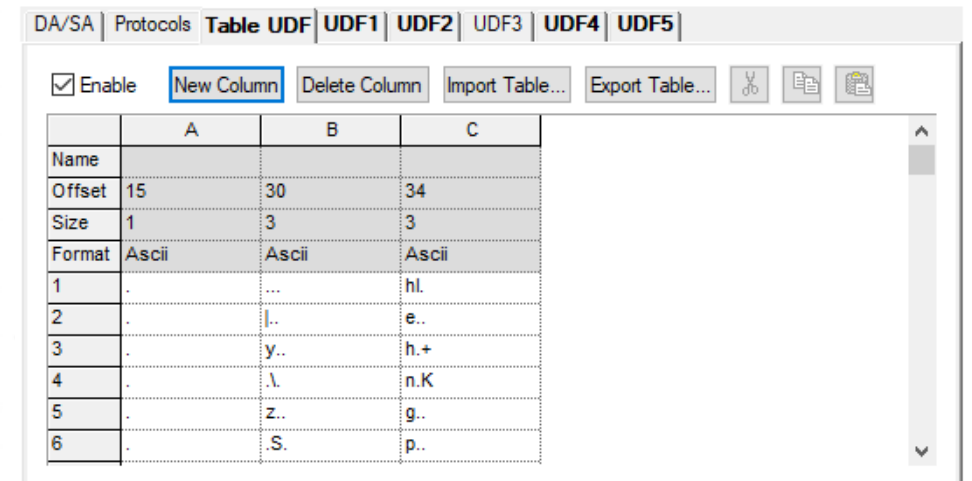
Block Diagram – Generic Packet Tester



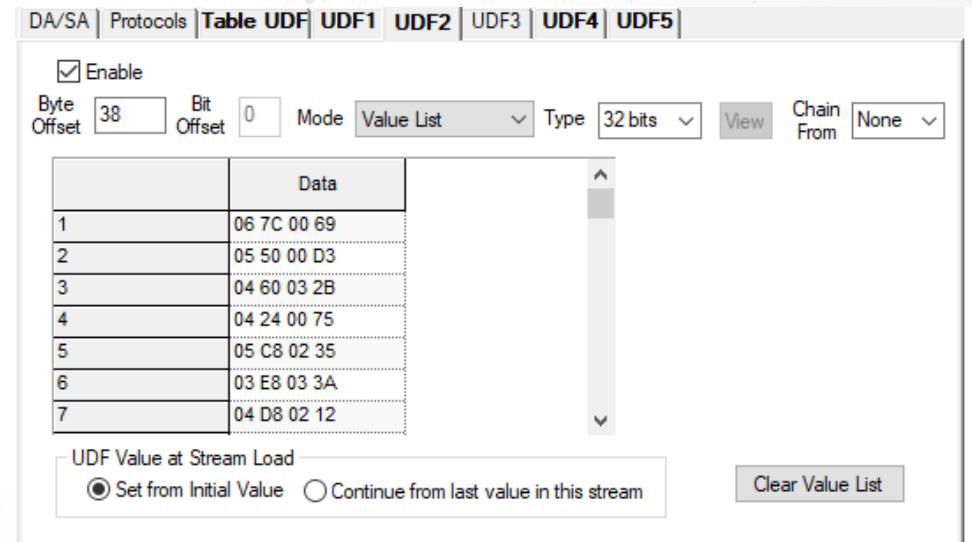
A Deeper Dive

FLOW GENERATION

- In typical FPGA based testers, products support user defined fields (UDFs) that can modify **any arbitrary offset** with a variety of patterns.
- This is a challenge in P4 from both a data plane and control plane perspective. The more natural API is protocol specific and based on field definitions in parser.
- Coming back to the use case: Is modifying arbitrary offsets absolutely necessary?
- In this scenario, the fields that need to vary are fairly specific (i.e. L2/L3 source/destination addresses, L4 source/destination ports, flow identifier)



	A	B	C
Name			
Offset	15	30	34
Size	1	3	3
Format	Ascii	Ascii	Ascii
1	hl.
2	.	l.	e..
3	.	y..	h.+
4	.	.\.	n.K
5	.	z..	g..
6	.	.S.	p..



Byte Offset: 38 Bit Offset: 0 Mode: Value List Type: 32 bits

	Data
1	06 7C 00 69
2	05 50 00 D3
3	04 60 03 2B
4	04 24 00 75
5	05 C8 02 35
6	03 E8 03 3A
7	04 D8 02 12

UDF Value at Stream Load
 Set from Initial Value Continue from last value in this stream

A Deeper Dive

FLOW GENERATION

- With a more limited approach it is fairly simple to implement the use case. Can simply use Match Action Units.
- Maintain a stateful packet index counter for each flow group
- MAU reads on a packet index counter and populates with appropriate values

VLAN-ID	Precedence	Source Address	Destination Address	TCP-Source-Port	TCP-Dest-Port
11	000 Routine	135.10.18.1	100.41.108.1	5140	564
11	000 Routine	139.21.37.1	104.214.120.1	5320	298
11	000 Routine	132.231.143.1	111.141.0.1	62996	528
11	000 Routine	135.32.60.1	117.247.225.1	5020	551
11	000 Routine	130.31.133.1	105.110.186.1	7000	347
11	000 Routine	132.22.43.1	102.252.220.1	62516	125
11	000 Routine	127.123.212.1	119.136.21.1	63596	664
11	000 Routine	132.147.251.1	104.40.174.1	5560	543
11	000 Routine	134.128.128.1	114.253.157.1	5740	486
11	000 Routine	131.184.40.1	118.144.176.1	62576	136
11	000 Routine	125.176.132.1	118.247.187.1	5380	184
11	000 Routine	122.128.210.1	115.137.3.1	7420	820

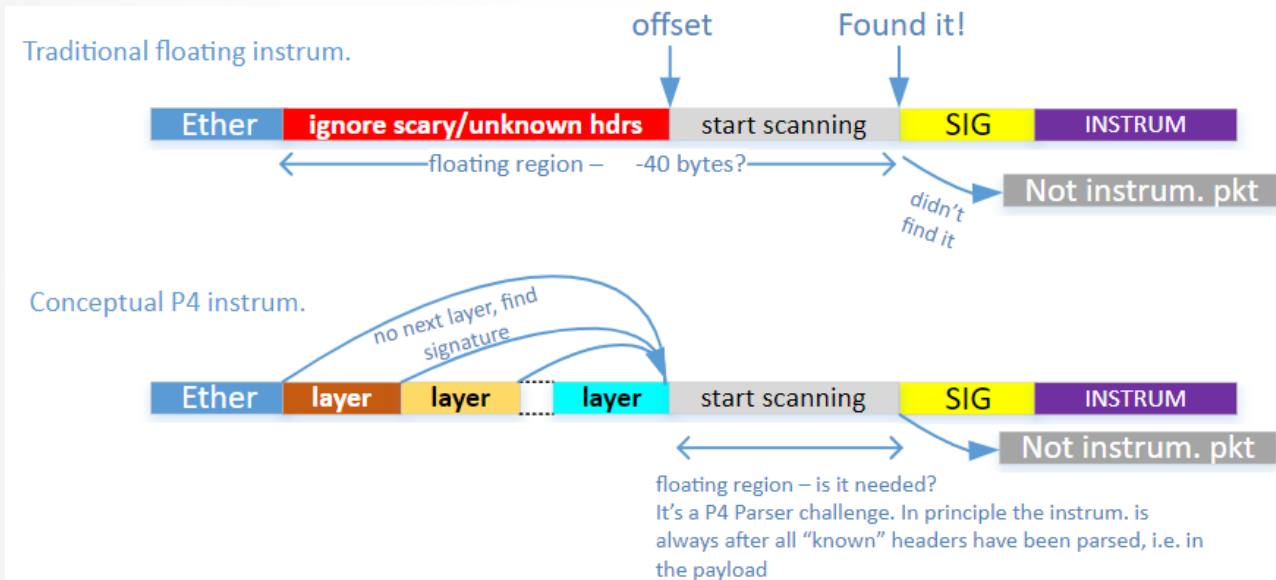
```
action do_modify_fields(dmac, smac, vid0,
vid1, dipv4, sipv4, dipv6, sipv6, pgid,
l4_dport, l4_sport)
{
    modify_field(outer_eth.dstAddr, dmac);
    modify_field(outer_eth.srcAddr, smac);
    modify_field(vlan_tag[0].vid, vid0);
    modify_field(vlan_tag[1].vid, vid1);
    modify_field(outer_ipv4.dstAddr, dipv4);
    modify_field(outer_ipv4.srcAddr, sipv4);
    modify_field(outer_ipv6.dstAddr, dipv6);
    modify_field(outer_ipv6.srcAddr, sipv6);
    modify_field(instrum.pgid, pgid);
    modify_field(tcp.dstPort, l4_dport);
    modify_field(tcp.srcPort, l4_sport);
    modify_field(udp.dstPort, l4_dport);
    modify_field(udp.srcPort, l4_sport);
}
```

```
table udf_vlist_tbl {
    reads {
        meta.stream: ternary;
        g_pkt_cntr.value: ternary;
        eg_intr_md.egress_port: ternary;
    }
    actions {
        do_modify_fields;
    }
}
```

A Deeper Dive

FLOW TRACKING

- In typical FPGA based testers, products support a floating signature and instrumentation header.
- This signature can typically be placed **anywhere** in the packet to provide the end-user maximum flexibility. Challenging to implement in P4 parser.



Automatic Instrumentation Signature

Start scan at

Signature Value

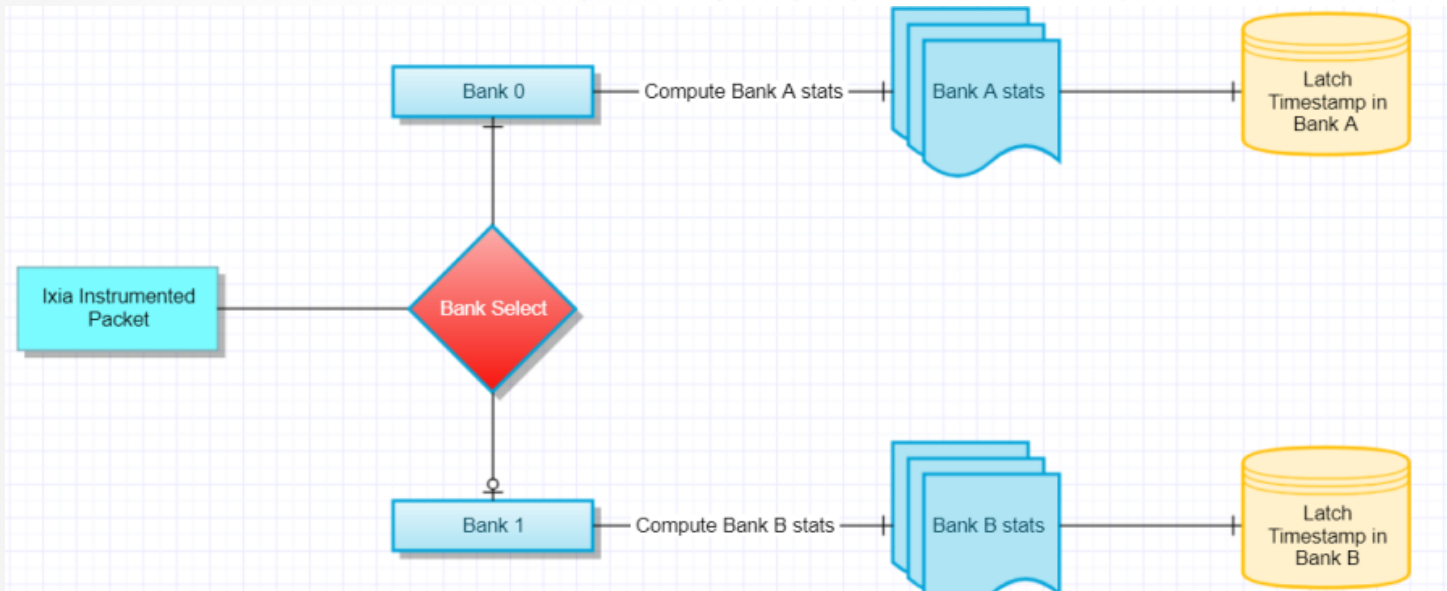
Mask

```
table rx_instrum_tbl {
    reads {
        big_sig.sig1: ternary;
        big_sig.sig2: ternary;
        big_sig.sig3: ternary;
        ig_intr_md.ingress_port :
        ternary;
    }
    actions {
        _nop;
        do_set_rx_instrum;
    }
    default_action: _nop;
    size: MAX_PIPELINE_PORTS;
}
```

A Deeper Dive

FLOW STATISTICS

- If the signature has been matched need to extract flow identifier (PGID) and compute statistics.
- Ideally should be able to get a time snapshot of all statistics and timestamps to effectively correlate and measure real time transmit and receive counters.
- FPGAs use ping-pong RAM buffers to create snapshots across the chip



P4 vs FPGA Product Development

BUILDS

FPGA

- Typical build times for FPGA images are 3-6 hours to route and 1.5 hours to synthesize
- Need on average 5 seeds for build strategies which is done in parallel
- Also need to take into account timing and utilization which adds additional components to development process

Route Results

Real-time: 01:25:22, CPU-time: 03:33:45, RAM: 14341.875

Synth Results

Real-time: 01:32:49, CPU-time: 00:53:44, RAM: 12557.008

P4

- Typical build times for P4 images + API bindings is about 5 minutes
- Allows for much quicker iterations on code, compile, deploy cycle
- Enables an agile development methodology using sprints for feature development and bug fixing

```
time p4_build.sh pktgen9.p4
real    4m40.359s
user    14m56.508s
sys     0m18.544s
```

P4 vs FPGA Product Development

APIS

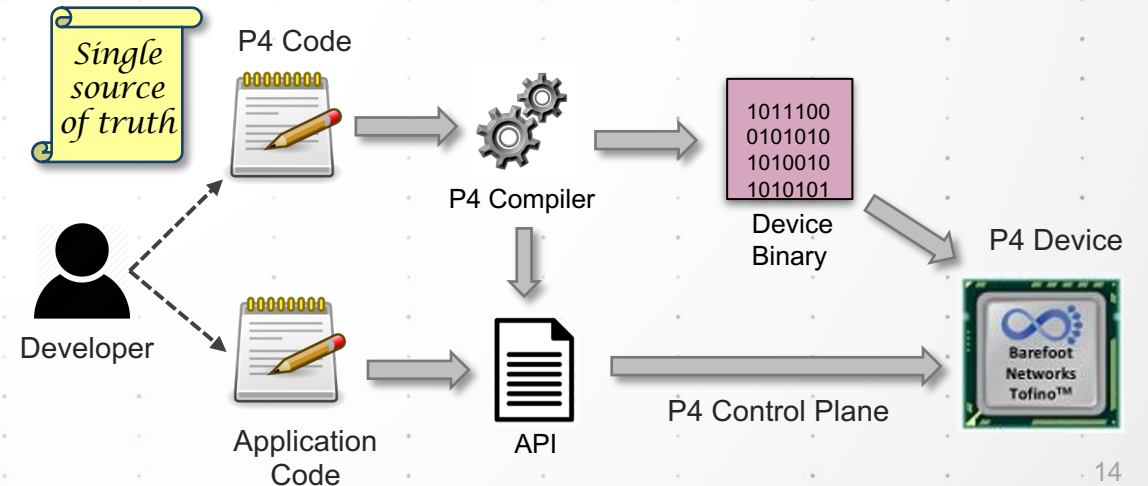
FPGA

- FPGA designers typically provide a design document of a module
- Design document will include register memory map of block as well as high level procedures to program and manage module. Passed to SW team to integrate into application stack.

Bit	Name	Description	R/W	Default
31:2	Reserved	Unused	RO	0
1				
20	Enable	'1' – Enable Insertion	R/W	X
19:6	UDF Byte Offset	Byte offset from start of packet for the start of the 32 bit overlay. Can be odd. Any offset in the packet is allowed.	R/W	X
5:4	Reserved	Unused	RO	0
3:0	Mask	'1' – Unrotated Byte Enable	R/W	X

P4

- P4 compiler auto-generates APIs for P4 program
- APIs can be exercised in a variety of ways such as RPC (i.e. gRPC) or directly via C APIs
- API integration into application stack is typically done by **the same P4 developer**.



P4 vs FPGA Product Development

SCALABILITY, PRECISION, ACCURACY

FPGA

- Typical products are specified prior to hardware development to achieve desired scalability.
 - i. Number of Trackable Flows – some products support up to 1M
 - ii. Number of Flow groups – some products support up to 1K per port
 - iii. Stateful Protocol Emulation scale
 - iv. Number of modifiers and depth of memory for them
- Rate precision through schedulers implemented in RTL
- Jitter minimization with timestamping logic at tail end of egress and head of ingress. Leads to high latency accuracy

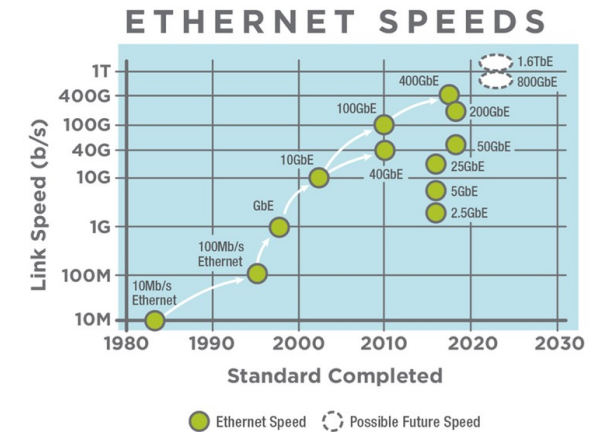
P4

- Scalability of platform determined by resources provided by ASIC and initially required trial and error with pipeline optimizations.
- Any precision and accuracy is subject to the underlying accuracy of fixed functions in the ASIC. P4 does not provide any kind of guarantees for this.
- At the end of the day, P4 is tailored for higher level programming of switch fabrics. Not nearly the same flexibility as RTL.

Key Takeaways and Conclusions

IS P4 THE FUTURE OF NETWORK TEST?

- The answer is a qualified yes 😊
- FPGAs will always be critical for L23 Network Testing in the following domains:
 - I. **New speeds and feeds** – Every increase in Serdes signaling rates will require Network Testers for ASIC validation. The P4 programmable ASICs at these speeds will always be released too late in the life cycle of a new Ethernet speed to address this market. The first Network tester to market in **800Gb or 1.6Tb will be FPGA based**.
 - II. **High Scalability Verification** – Due to the cost constraints of switching silicon, P4 programmable switches will most likely always lag in system resources to support use cases requiring high performance and scalability as stand-alone devices.
- P4 programmable ASIC + FPGA hybrid systems have a promising future as systems can leverage the programmability and throughput of these ASICs alongside the flexibility accessible through FPGAs.
- P4 enabled systems in general will spark innovation in the network test industry through its programming paradigm and ecosystem



Key Takeaways and Conclusions

P4 DRIVEN INNOVATION

- The ability to compile the data plane in 5 minutes is game changing for network test as the pace of innovation is the speed of typing
- Auto-generated P4 APIs make the P4 code the single source of truth. Separate specifications are not needed, effort is reduced, errors are avoided and turnaround times are shrunk.
- Uniform, simple, P4 APIs allow the dataplane to be continuously tested by automation pipelines using relatively simple test harnesses. Testing of the API and thus the P4 program can occur independently of the integration into existing application stacks.
- P4's level of abstraction, simplicity, and software-based approach accelerates feature velocity and enables individual developers to contribute across the application stack easily and own features end to end.
- Overall, P4 programmable platforms enable testing products to be developed iteratively and rapidly by self contained software teams in a much more responsive model to evolving customer needs. This directly reflects the change of scope from a hardware project to a software project.

Back to our Questions

KEY QUESTIONS

- Through our exploratory and product development cycles we found that we could definitely leverage a P4 programmable chip to deliver a viable Network Tester!
- We were able to leverage our years of IP to deliver the same user experience while abstracting the implementation details of the data plane. Ironically, the bulk of the work was in the control plane and software integration. The data plane portion was fairly straightforward to implement and unit test.
- From a resource perspective, we were able to rapidly prototype and bootstrap the product with a small self contained team.
- Leveraging P4 allowed us to transform a typical hardware project into a software project. Specifically, we were able to successfully transition to a modern agile development process with CI/CD pipelines across the application stack. This has us setup to be able to quickly mobilize and meet evolving customer needs.

Thanks and Acknowledgements

- Keysight UHD Team
- Intel/Barefoot Engineering and Support Team
- P4 Language Consortium and SONiC



ixia | A Keysight Business



UHD100T32

HIGHEST DENSITY
100 GIGABIT ETHERNET
TEST SOLUTION

BAREFOOT

NETWORKS | an Intel company



P4
Expert
Roundtable Series

April 28-29, 2020

Hosted by:



Thank You

ram.murthy@keysight.com

<https://www.linkedin.com/in/ram-murthy-699ab795/>

<https://www.ixiacom.com/products/uhd100t32-32-port-100ge-test-system>