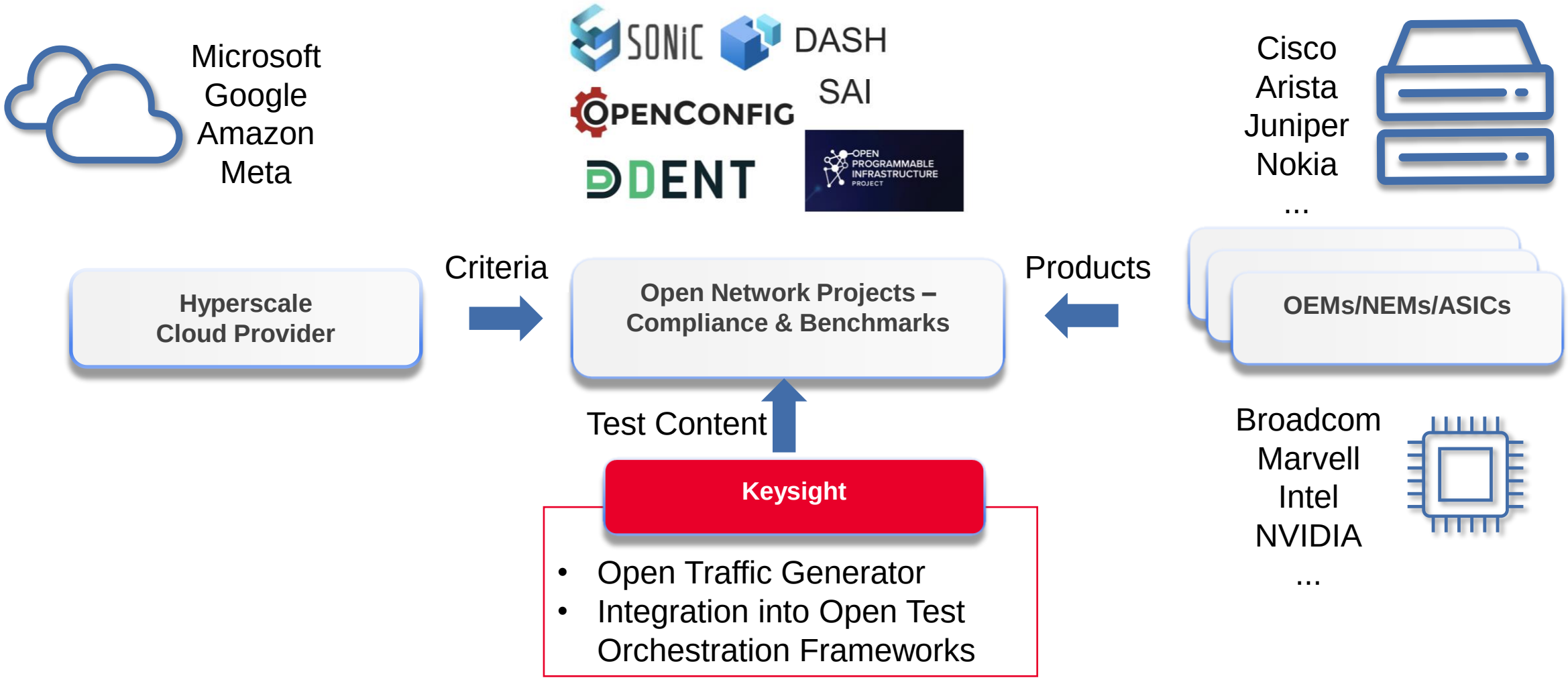
KEYSIGHT

SONiC DASH testing

Keysight team
13th June 2023

Open networking eco-systems



Keysight Community Engagements & Contributions

SONiC

NOS for data center

Test working group contribution with OTG / snappi and performance / resiliency test cases

KEY SPONSOR

Microsoft

SONiC-DASH

Disaggregated API for SONiC hosts/DPU

Test working group chair

Architect for test framework & keeper of community test bed

KEY SPONSOR

Microsoft

OpenConfig

Vendor-neutral layer for managing network

Partner with Google to co-design OTG model and contribute FeatureProfiles test suite

KEY SPONSOR

Google

DENT

NOS for retail and edge networks

Test working group chair and keeper of system integration test lab

KEY SPONSOR

amazon

OPI

Open programmable infrastructure

Founding member and member of Board and TSC

KEY SPONSOR

KEYSIGHT TECHNOLOGIES

Keysight Community Engagements & Contributions

2017

SONiC OS released by Microsoft

Standardized network operating system

2020

SONiC Whitebox validation

Keysight tools integrated into sonic-mgmt framework

Contributed RDMA, BGP performance and resiliency test content in partnership with MSFT

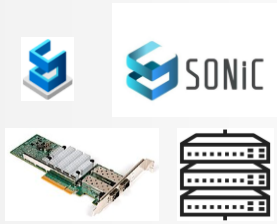
2021

SONiC-DASH announced

Keysight leads test working group

Design test framework and testbed

Engaging directly with MSFT and five DPU vendors



2022

Partnership in DASH WG and Chassis WG

Built and demonstrated DASH hero test

Enhance SAI test framework with Keysight test tools.

Enabling chassis test coverage with RDMA and MACSec test

Part 1 **Chris Sommers**

Distinguished SW Engineer

Shift-Left Testing and the Importance of a Behavioral Model

Part 2 **Mircea Dan Gheorghe**

Director, System test

SONiC-DASH performance benchmark testing using “Hero Test”

KEYSIGHT

The SONiC-DASH Project: Shift-Left Testing and the Importance of a Behavioral Model

June 13, 2023

Chris Sommers, Distinguished SW Engineer
Keysight Technologies
chris.sommers@keysight.com

Shift-Left & DevOps workflows – the New Normal

Industry Drivers:

- Development and test roles are merging into one.
- Has become the default best-practice, because it works: faster TTM, agility, better quality, reduced headcount.
- This demands tooling – build and test frameworks, source control, CI/CD pipelines, etc.

DASH Project Drivers:

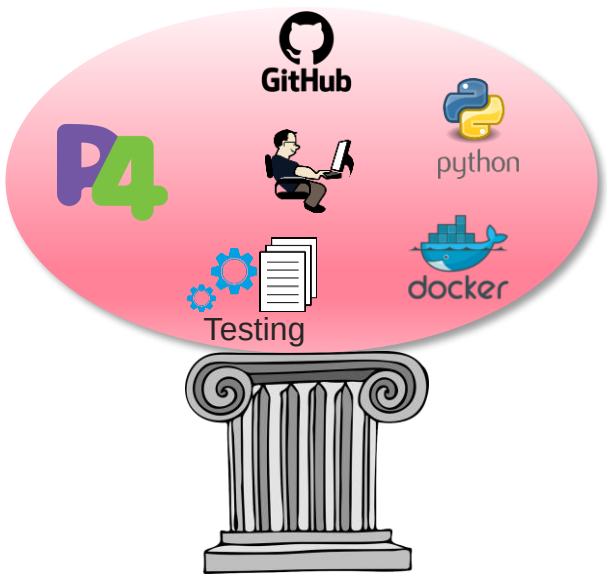
- We want a definitive model of the dataplane to serve as the reference.
- Final implementations can take months/years (esp. hardware) – how to begin testing earlier?
- The community needs a common test-bed, but vendors need to guard their IP. What can be openly shared?

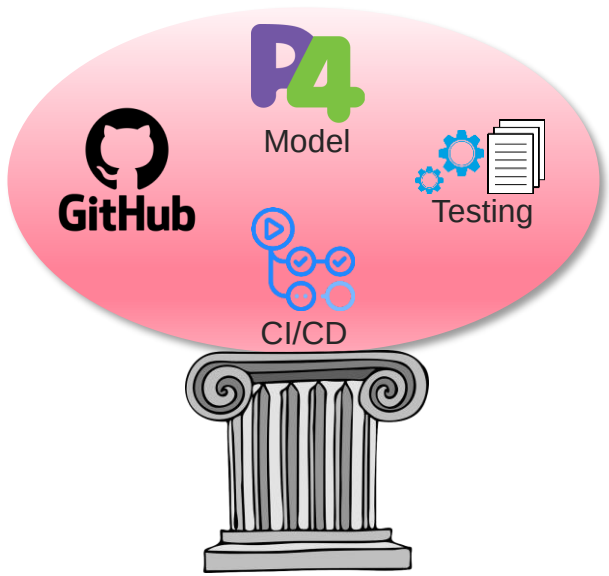
Solution:

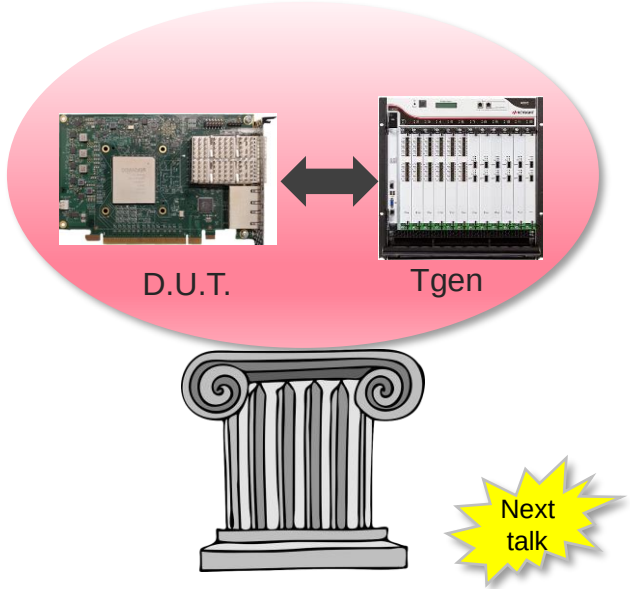
- Digital Twins are gaining popularity. In DASH, we have a P4 Behavioral Model, providing:
  - ✓ Human-readable, precise definition of the dataplane – ACL rules, packet transformations, etc.
  - ✓ Vendor neutral, open implementation – not dependent upon SoC architectures or SDKs
  - ✓ Machine–executable “digital twin” available early in the program.
  - ✓ Single source of truth, artifact generation – SAI headers, test harness, etc.

The DASH Community invested heavily in creating viable, SW-based, DevOps workflows and tooling. The following slides will describe some of our work.

The four pillars of SONiC-DASH Community DevTest







Next talk

1. P4 Behavioral Model

- Community-developed, functional behavioral model, written in P4.
- Human-readable, machine executable
- Single source of truth.
- Generates many artifacts: SAI headers, BMv2 libsai and test framework.

2. Dev environment, Functional Tests

- Standardized build & test environment.
- Easy on-ramp: ***git clone, make all***. Up and running in minutes!
- Compiler, test environment and tests which can be run on behavioral model or *real hardware*.

3. Continuous Integration

- New P4 code is immediately tested in GitHub using CI/CD pipelines.
- Regressions are caught, new features are validated.

4. Performance / Scale Tests

- “Hero” test specification – minimum production performance criteria
- Hardware test-bed and test-cases verify performance and scale of vendor implementations.
- Easy to compare figures-of-merit

What Keysight has worked on

P4 Code – Well-suited to describe and define a dataplane as *software*

Example: Validate "PA_ADDRESS" to ensure only valid packets are processed

P4 Source:

```
table pa_validation {
    key = {
        meta.vnet_id: exact;
        hdr.ipv4.src_addr: exact;
    }

    actions = {
        permit;
        @defaultonly deny;
    }

    const default_action = deny;
}
```

In English:

"Validate the PA (provider address - internal datacenter resource routing) as follows:
- Compare the incoming packet's VNET ID and IPv4 source address with a configured list of valid combinations.
- If such an entry is in the list, permit the packet. If not, deny it."

Actual photograph of a DASH community design review

- P4 Code is "relatively easy" to read. It's concise and straightforward.
- The community can use it as a precise "specification" of the DASH dataplane, as an interpretation of the plain-English specs.
- It can be executed in software models (BMv2) using packet in/out tests to validate the design.

P4 Behavioral Model DevOps-style Workflow – Quick ‘n’ Easy!

Numerous pre-built images to save time

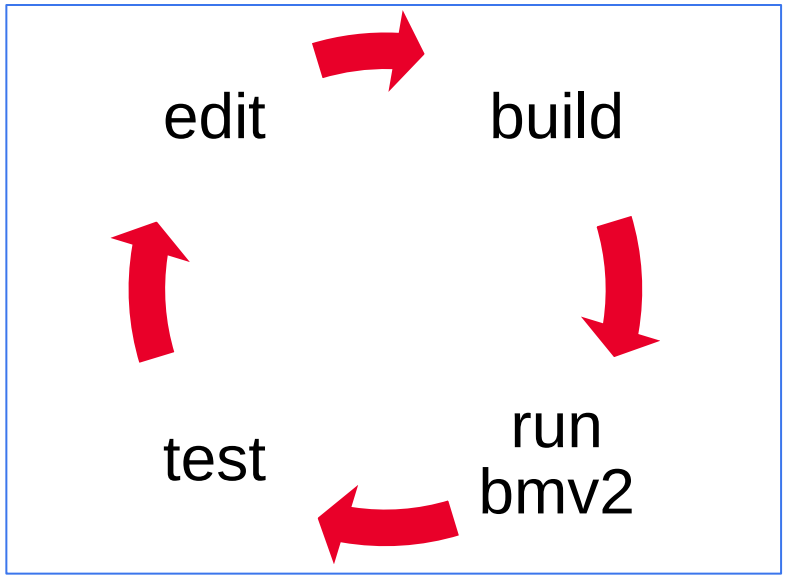
Azure Container Registry (ACR)

Implicit docker pull during build

(docker pull)

1. Import project
2. Develop code & run tests
3. Submit back to the repo
4. Validate in CI pipeline

2 Develop



Takes just minutes!

1 Install

git clone

git push

3 Publish changes

https://github.com/sonic-net/DASH

GitHub Actions

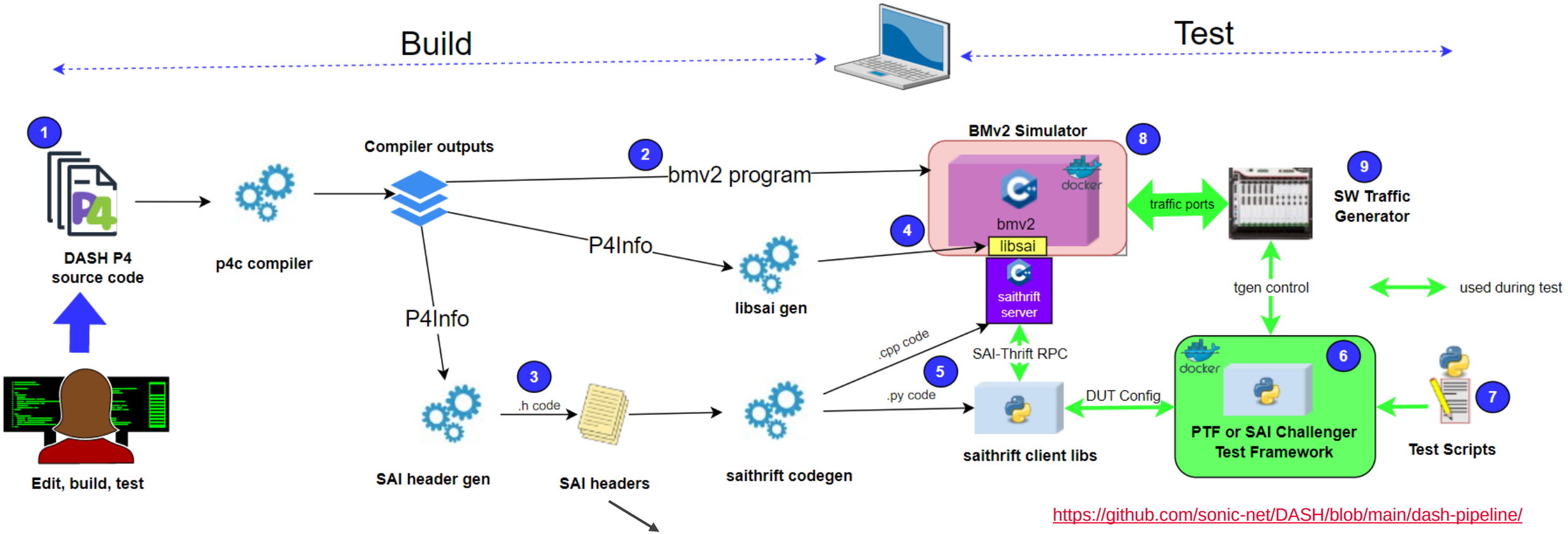
4 The CI Pipeline builds and tests code on every push or pull request

DASH P4 Code Generates Numerous Artifacts

Build: With a single “make” command, the P4 source code (1) is processed to yield many artifacts, including:

(2) an executable model, (3) SAI headers, (4) libsai interface to the bmv2 model, and a (5) sai-thrift test harness.

Test: Python test frameworks (6) execute test-scripts (7) in the P4 model (8) using a SW traffic generator (9)



https://github.com/sonic-net/DASH/blob/main/dash-pipeline/

API integrates with SONiC

P4 Code Transformations

Code generators translate dataplane code into APIs (SAI header files) and test harness (Thrift RPC)

P4 Source:

```
dash_pipeline.p4
table pa_validation {
      key = {
         meta.vnet_id: exact;
         hdr.ipv4.src_addr : exact;
      }


      actions = {
         permit;
         @defaultonly deny;
      }


      const default_action = deny;
   }
```

The dataplane model.

Generated SAI header:

```
typedef struct _sai_dash_pa_validation_api_t
{
   sai_create_pa_validation_entry_fn
create_pa_validation_entry;

   sai_remove_pa_validation_entry_fn
remove_pa_validation_entry;

   sai_set_pa_validation_entry_attribute_fn
set_pa_validation_entry_attribute;

   sai_get_pa_validation_entry_attribute_fn
get_pa_validation_entry_attribute;

   sai_bulk_create_pa_validation_entry_fn
create_pa_validation_entries;

   sai_bulk_remove_pa_validation_entry_fn
remove_pa_validation_entries;
```

The southbound interface to the dataplane

Generated SAI-Thrift Python Client:

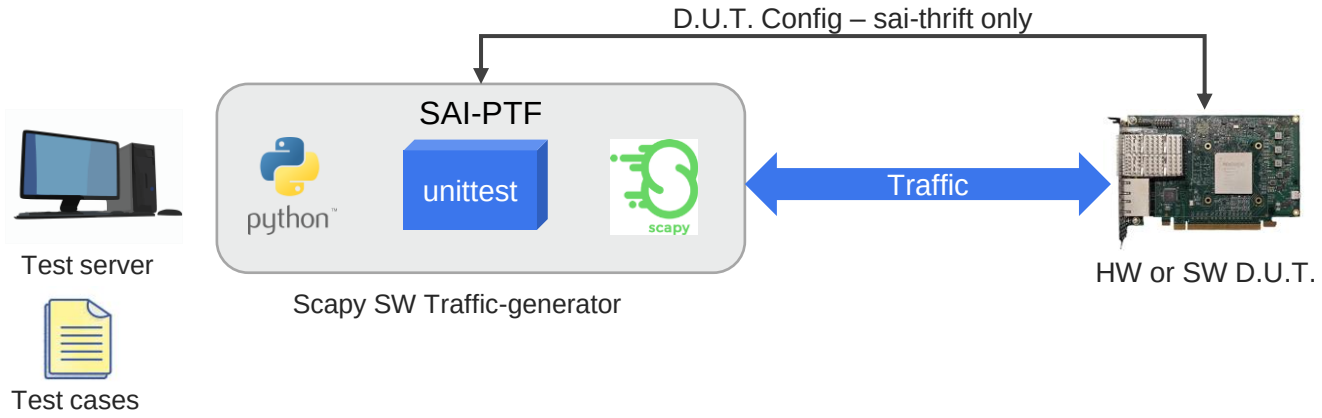
```
def sai_thrift_create_pa_validation_entry(self,
pa_validation_entry, attr_list)


def sai_thrift_remove_pa_validation_entry(self,
pa_validation_entry)


def sai_thrift_set_pa_validation_entry_attribute(self,
pa_validation_entry, attr)


def sai_thrift_get_pa_validation_entry_attribute(self,
pa_validation_entry, attr_list)


def sai_thrift_bulk_create_pa_validation_entry(self,
pa_validation_entry, attr_count, attr_list, mode)


def sai_thrift_bulk_remove_pa_validation_entry(self,
pa_validation_entry, mode)
```

An RPC interface to SAI for testing

Functional Test Frameworks used by DASH

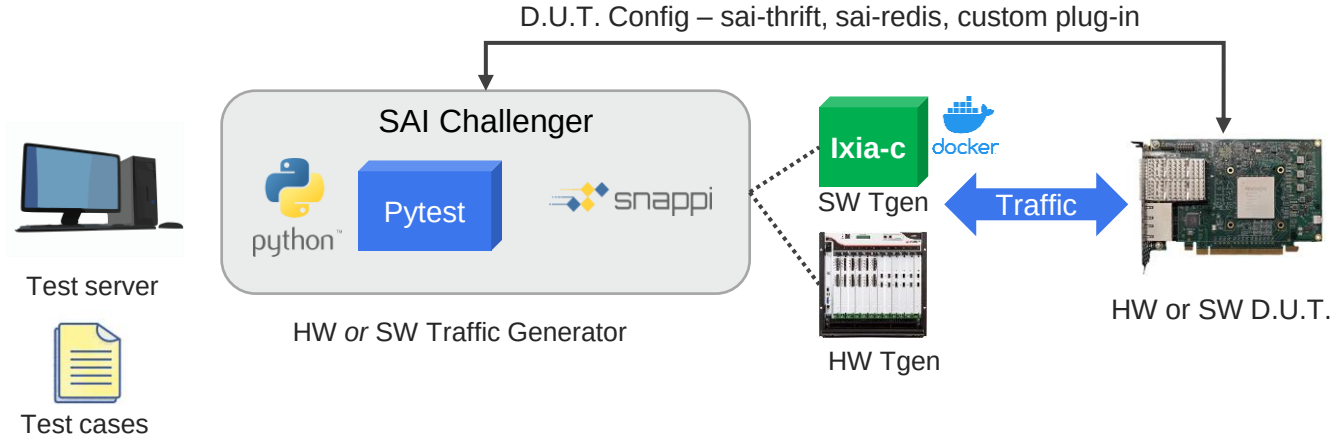
SAI-PTF (Packet Test Framework) - Legacy

- Combines Python unittest + scapy + packet testing utilities
- Lots of existing test-cases in SAI community
- Popular, but somewhat limited
- Does ***not*** support **line-rate** or **flow-based testing**

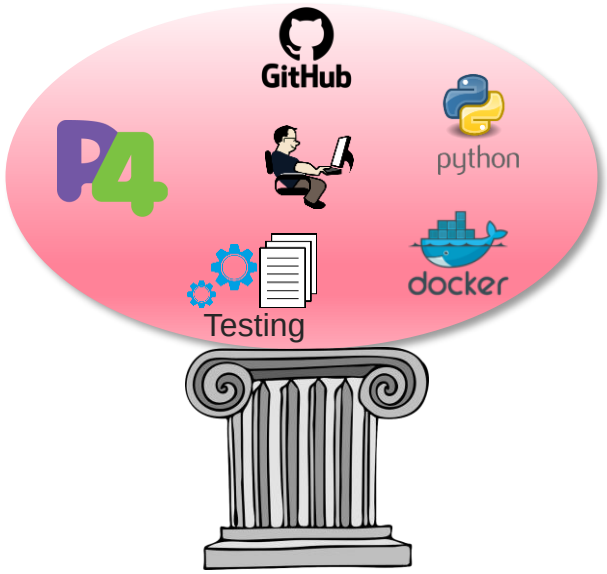


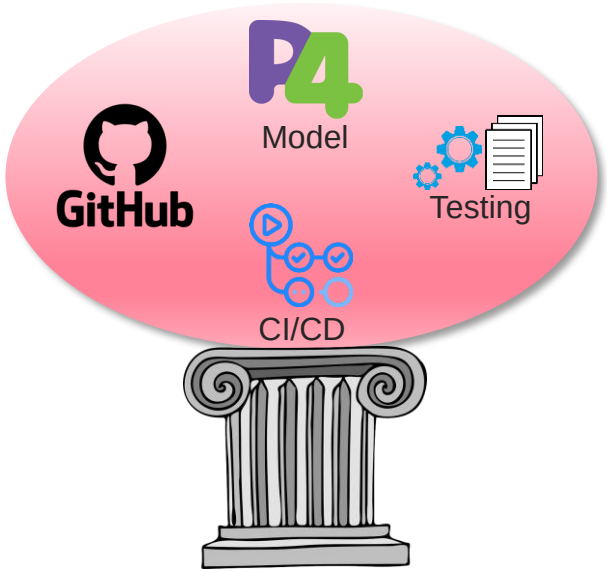
SAI Challenger – the next generation

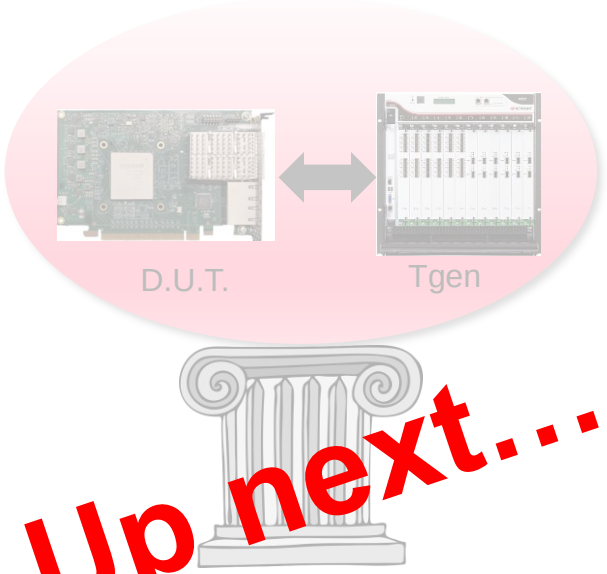
- Support for *Open Traffic Generator* via the snappi API
- Works with SW & HW traffic generators, same scripts!
- Flow-based traffic, high scale, line-rate capable
- Pluggable DUT configuration APIs
- Data-driven, more scalable, multi-DUT topologies, ….
- Can run all existing PTF tests as umbrella framework
- A modern framework for *all* SAI testing needs.



Recap: The four pillars of SONiC-DASH Community DevTest







Up next…

1. P4 Behavioral Model

- Community-developed, functional behavioral model.
- Single source of truth.
- Human-readable, machine executable
- Generates many artifacts: SAI headers, BMv2 libsai and test framework.

2. Dev environment, Functional Tests

- Standardized build & test environment.
- Easy on-ramp: “git clone” & “make all.” Up and running in minutes!
- Functional tests which can be run on behavioral model or real hardware.

3. Continuous Integration

- New P4 code is immediately tested in GitHub using CI/CD pipelines.
- Regressions are caught, new features are validated.

4. Performance / Scale Tests

- “Hero” test specification – minimum performance criteria
- Hardware test-bed and test-cases verify performance and scale of vendor implementations.
- Easy to compare figure-of-merit

What Keysight has worked on

KEYSIGHT

SONiC-DASH performance benchmark testing using “Hero Test”

June 13, 2023
Mircea Dan Gheorghe, Director System Test
Mircea-dan.gheorghe@keysight.com

Agenda

1. What is hero test
2. Why benchmark
3. ACLs. Routing, Mapped vs routed IPs
4. Config Generator
5. Testbed
6. Other metrics
7. What's next: Smart Switch
8. Post deployment

https://github.com/sonic-net/DASH/blob/main/documentation/general/program-scale-testing-requirements-draft.md

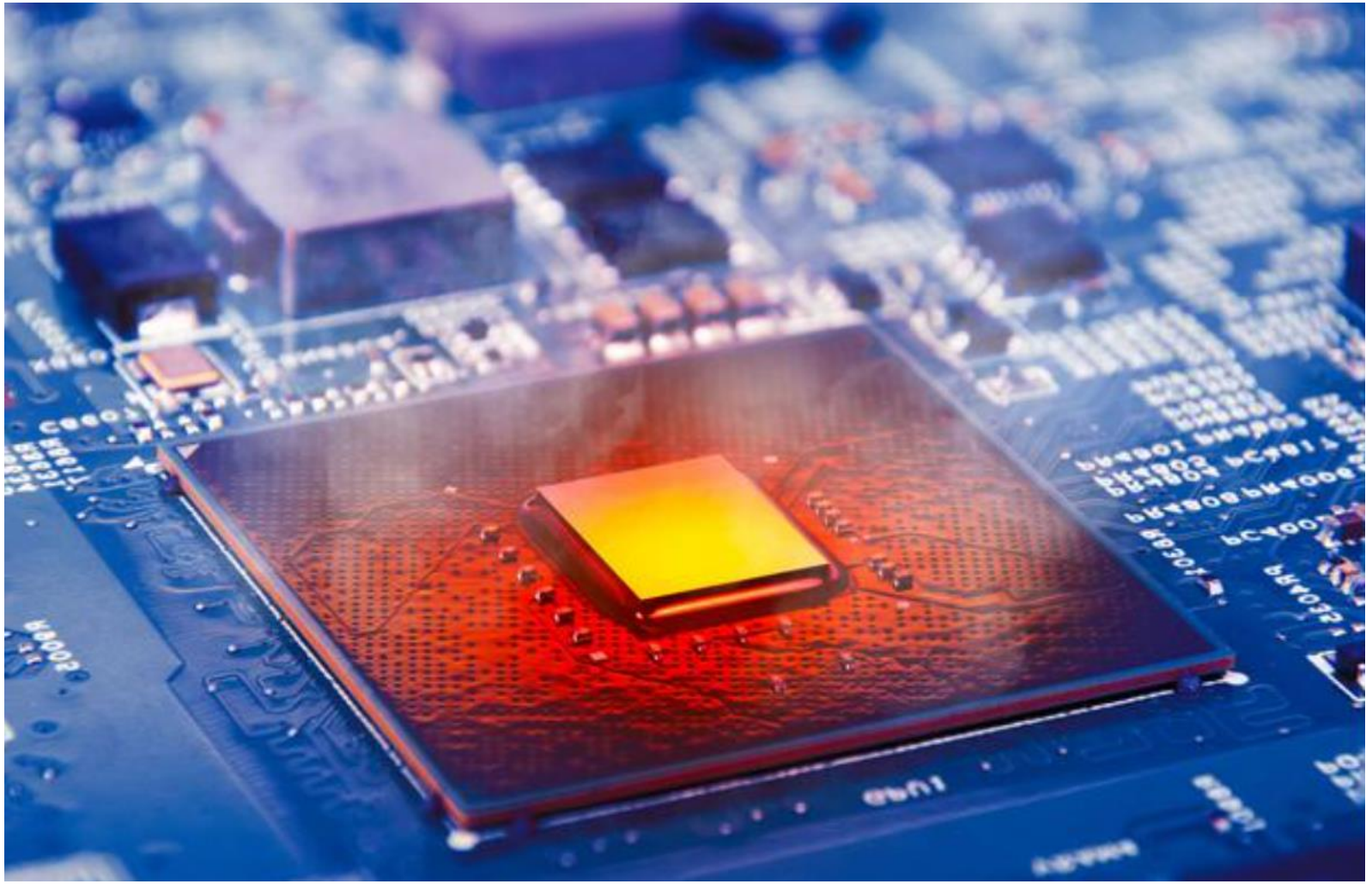
Hero Test

"Hero Test" establishes minimum performance requirements and the testing methodology.

- Test Methodology
- Definitions
- Feature Requirements
- Policy Requirements
- Route Requirements
- MSFT LAB with Keysight

Why?

Stress the device in order to define the behavior under the worst-case scenario



Scale

	per ENI	200G (DPU)	400G	800G	1.6T (smart switch)
VNETs		1024	2048	4096	8192
ENIs		64	128	256	512
Routes	100K	6.4M	12.8M	25.6M	51.2M
NSGs	5in + 5out	640	1280	2560	5120
ACLs prefixes	10x100K	64M	128M	256M	512M
ACLs Ports	10x10K	6.4M	12.8M	25.6M	51.2M
Mappings (CA to PA)	160K	10M	20M	40M	80M
Act Con	500K (bidir w/ connection pool capable of oversubscription)	32M	64M	128M	256M
CPS		3.75M	7.5M	15M	30M
bg flows TCP		15M (bidir w/ connection pool capable of oversubscription)	30M	60M	120M
bg flows UDP		15M (bidir w/ connection pool capable of oversubscription)	30M	60M	120M

https://github.com/sonic-net/DASH/blob/main/documentation/general/program-scale-testing-requirements-draft.md

ACLs, Routing, Mapped vs routed IPs

IP assignment considerations

- Stress the system (prevent summarization)
- Human readable (understand immediately what that value represents)
- Able to replicate the values across all devices DPU/IPU, Test HW.....

ACLs

- Odd IP values are allows (x.y.z.1 and .3 and .5)
- Even IP values are denies (x.y.z.2 and .4 and .6)

Routes

- x.y.z.0/30, x.y.z.4/31, x.y.z.7/32 and then we start over x.y.z.8/30, x.y.z.12/31, x.y.z.15/32...
- x.y.z.6 and x.y.z.14 is missing (map a deny)

Mapped/Routed

- Not all IPs have a VXLAN IP/MAC mapping

Config example

- ACLs
- 1.128.0.1 allow
- 1.128.0.2 deny
- 1.128.0.3 allow
- 1.128.0.4 deny
- 1.128.0.5 allow
- 1.128.0.6 deny
- 1.128.0.7 allow
- 1.128.0.8 deny
- …...

- Routes
- 1.128.0.0 / 27
- 1.128.0.32 / 28
- 1.128.0.48 / 29
- 1.128.0.56 / 30
- 1.128.0.60 / 31
- 1.28.0.62 / 32 is missing and overlaps with a deny
- 1.128.0.63 / 32

- 1.128.0.64 / 27
- ……..

- 25% IPs mapped, 75% routed

CA:2.128.0.1

PA:221.0.2.102

MAC:00:1B:6E:18:00:01

VNI:102

VTEP:221.0.2.102

MAC:00:1B:6E:18:00:01

PFX:2.128.0.0 / 27

VNI:102

Other metrics

PPS:
- Blast stateless UDP traffic
- PPS / TCP packets ~= CPS

CPS:
- Validates pps/tcp = CPS assumption
- Can be done before full hero test

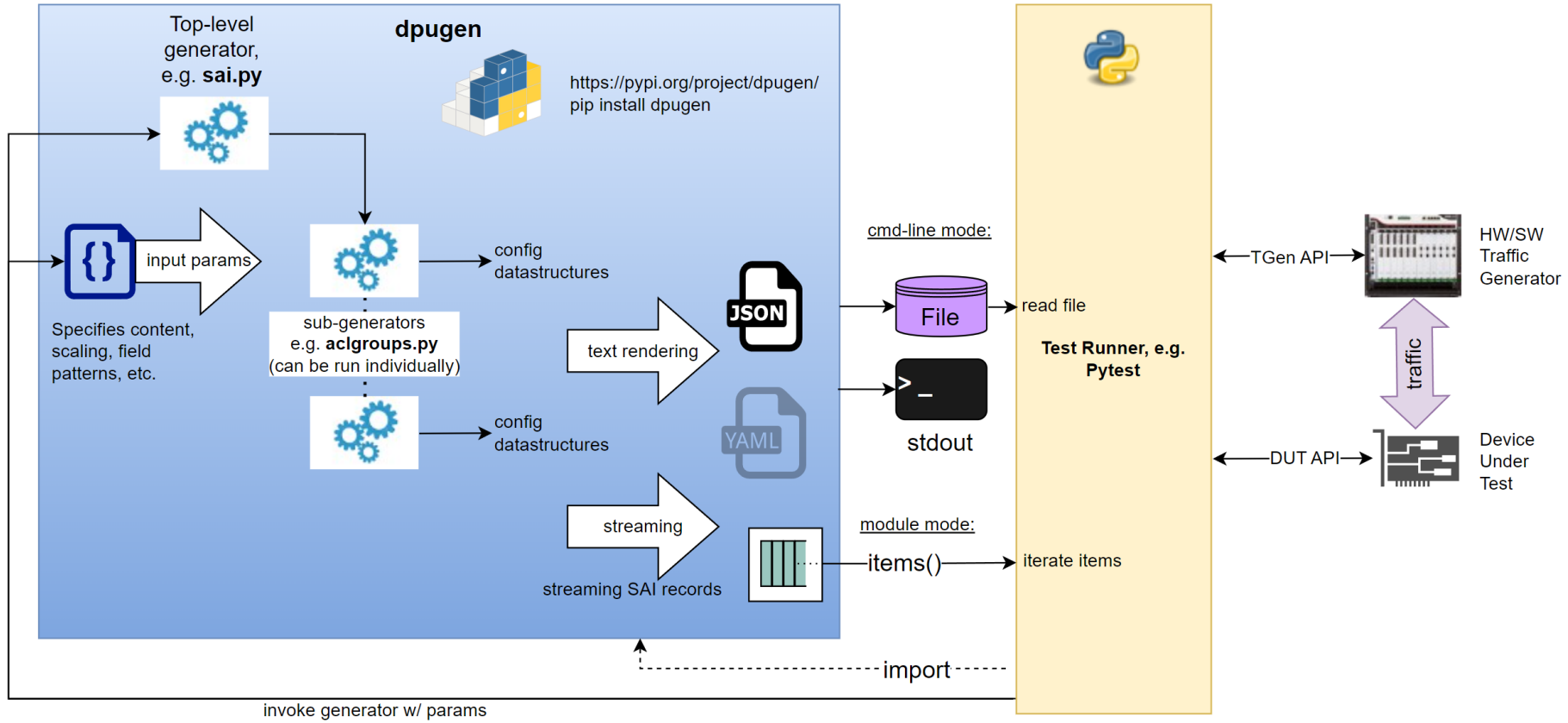
Flow capacity:
- Observe behavior once flow table is full
- 600s+ for flow expiration timer

Graph the metrics at different flow scales

Config Generator

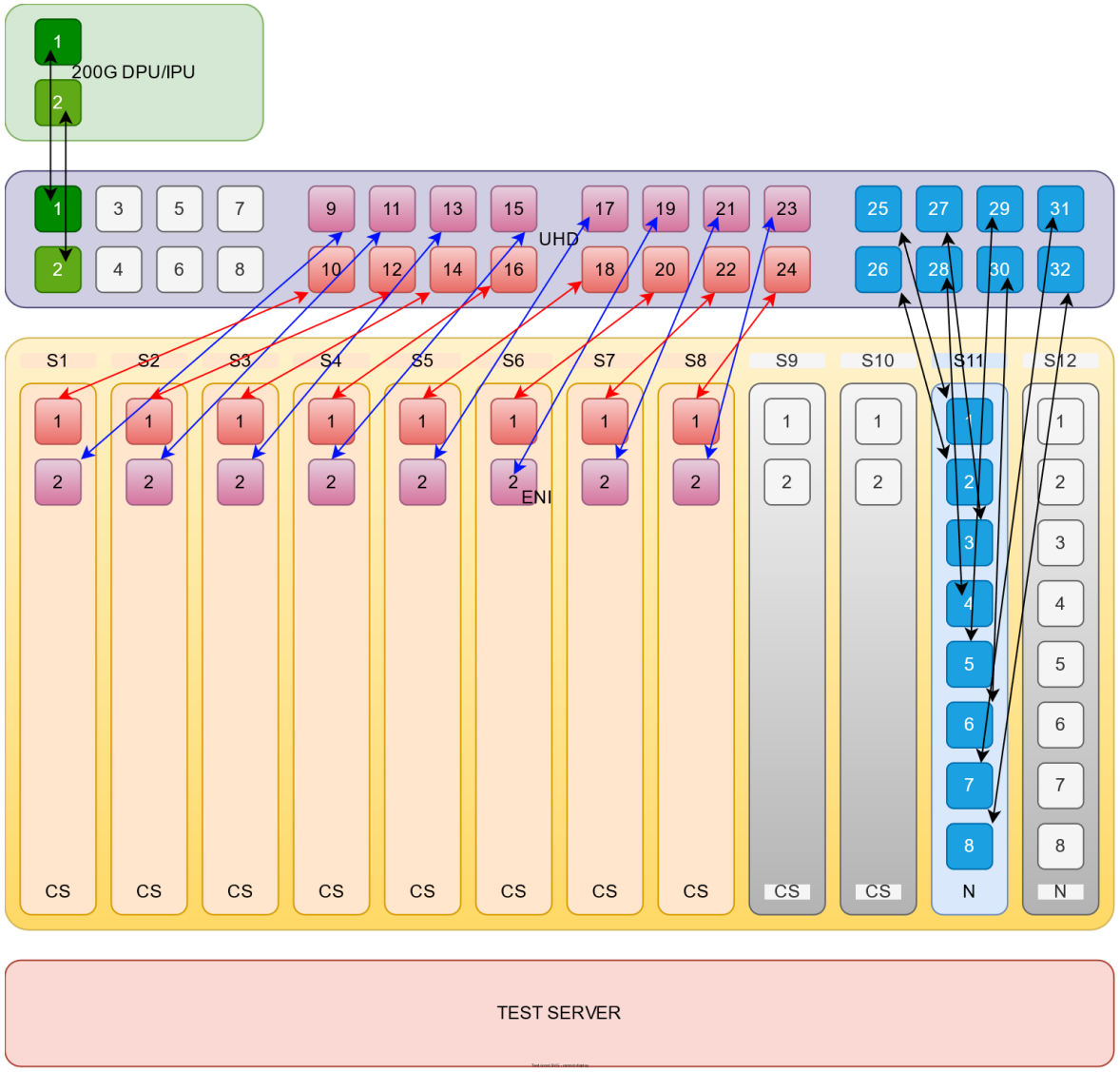
- SAI format (saigen)
- DASH format (dashgen)
- Private or other project formats available on request

batching is work in progress



https://github.com/mgheorghe/dpugen

Testbed diagram



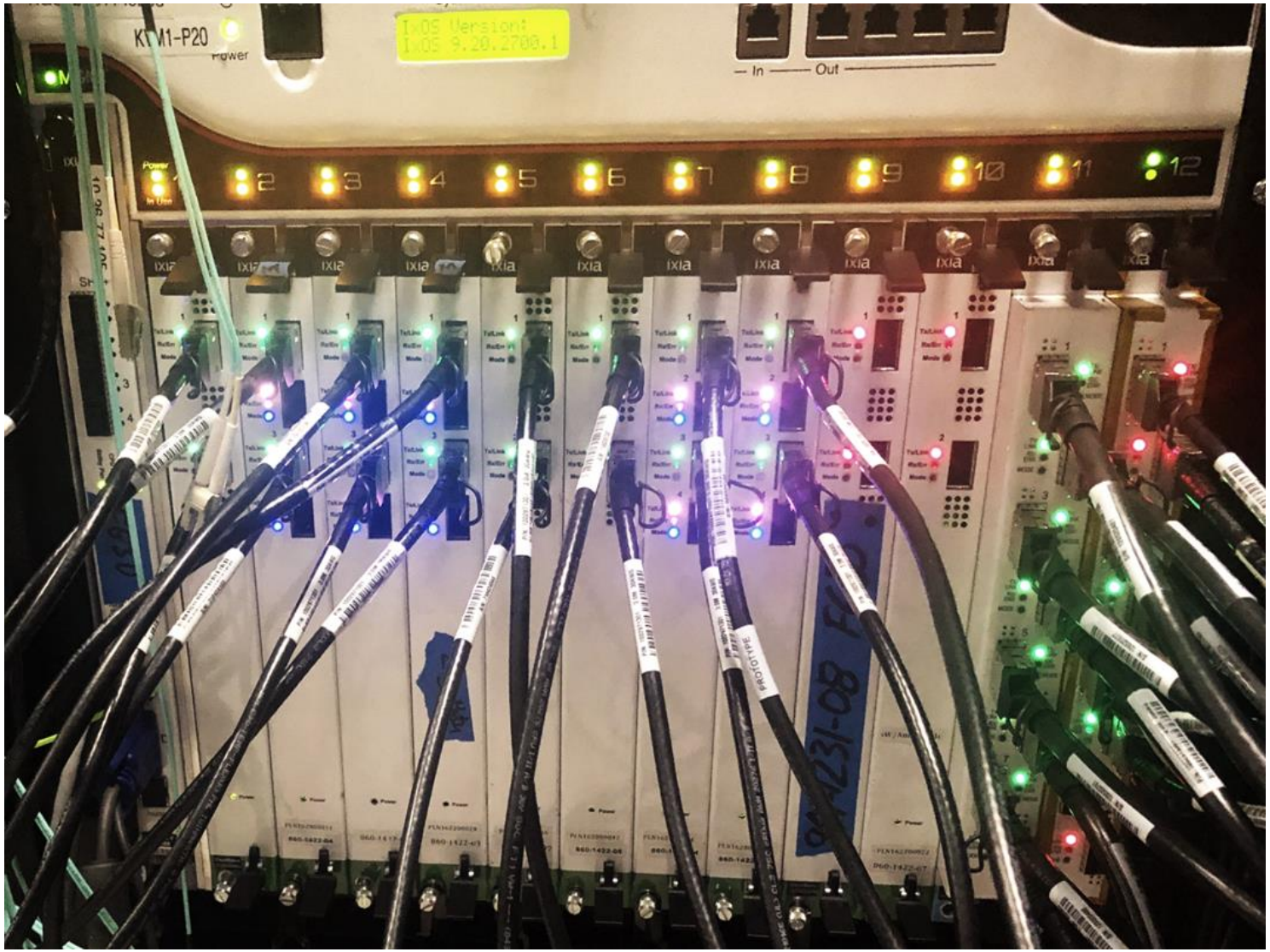
Up to 4 2x100G DPU/IPUs could be connected

UHD Connect for vlan to vxlan and split/merge of UDP/TCP traffic

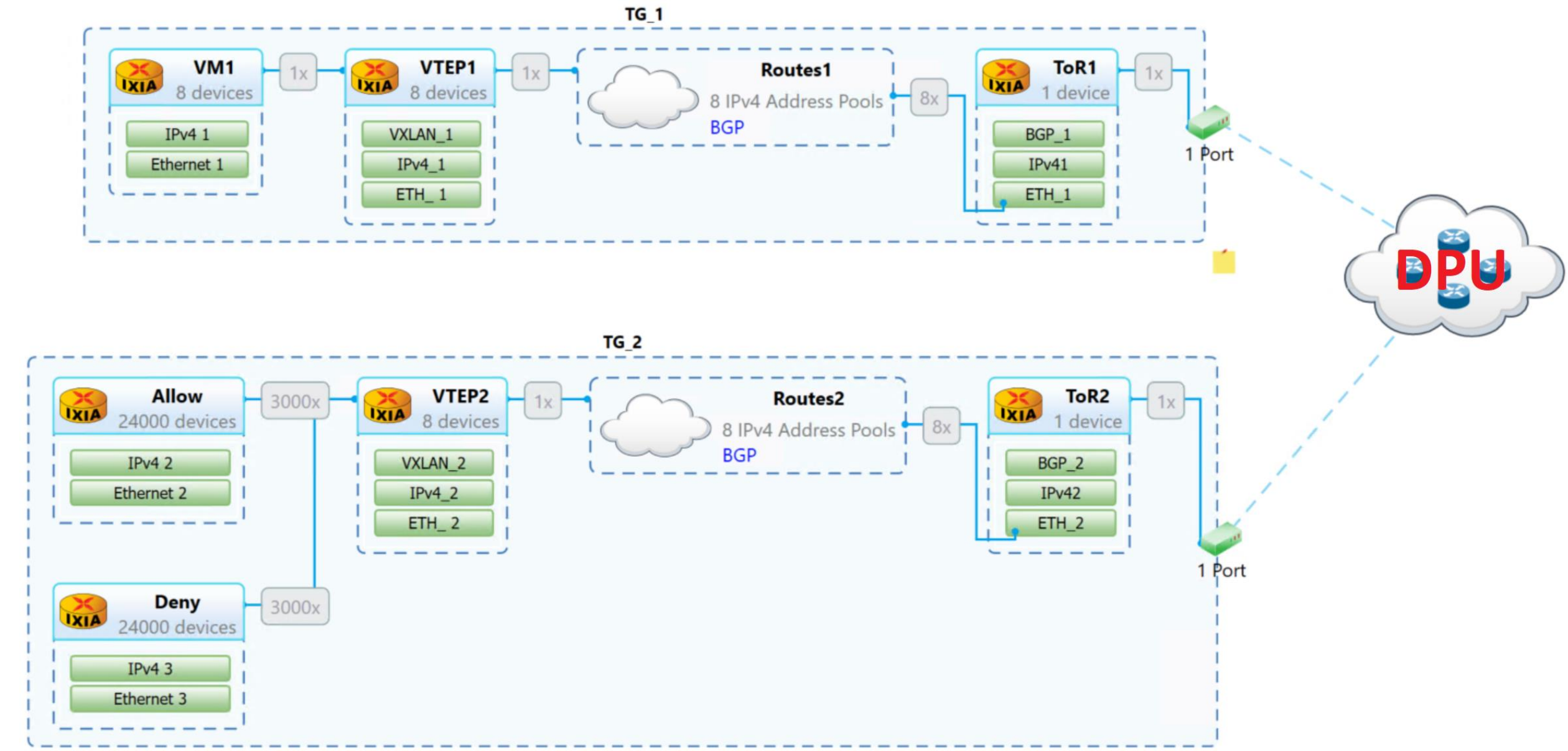
Keysight XGS12 chassis with up to 8 Cloud Storm cards and a Novus load module

Server to run the scripts and host any api server that may be needed for the test

Testbed


KTM1-P20
Power
In
Out
2
3
4
5
6
7
8
9
10
11
12
ixia


P/N: 1002971301
3.0M 30AWG

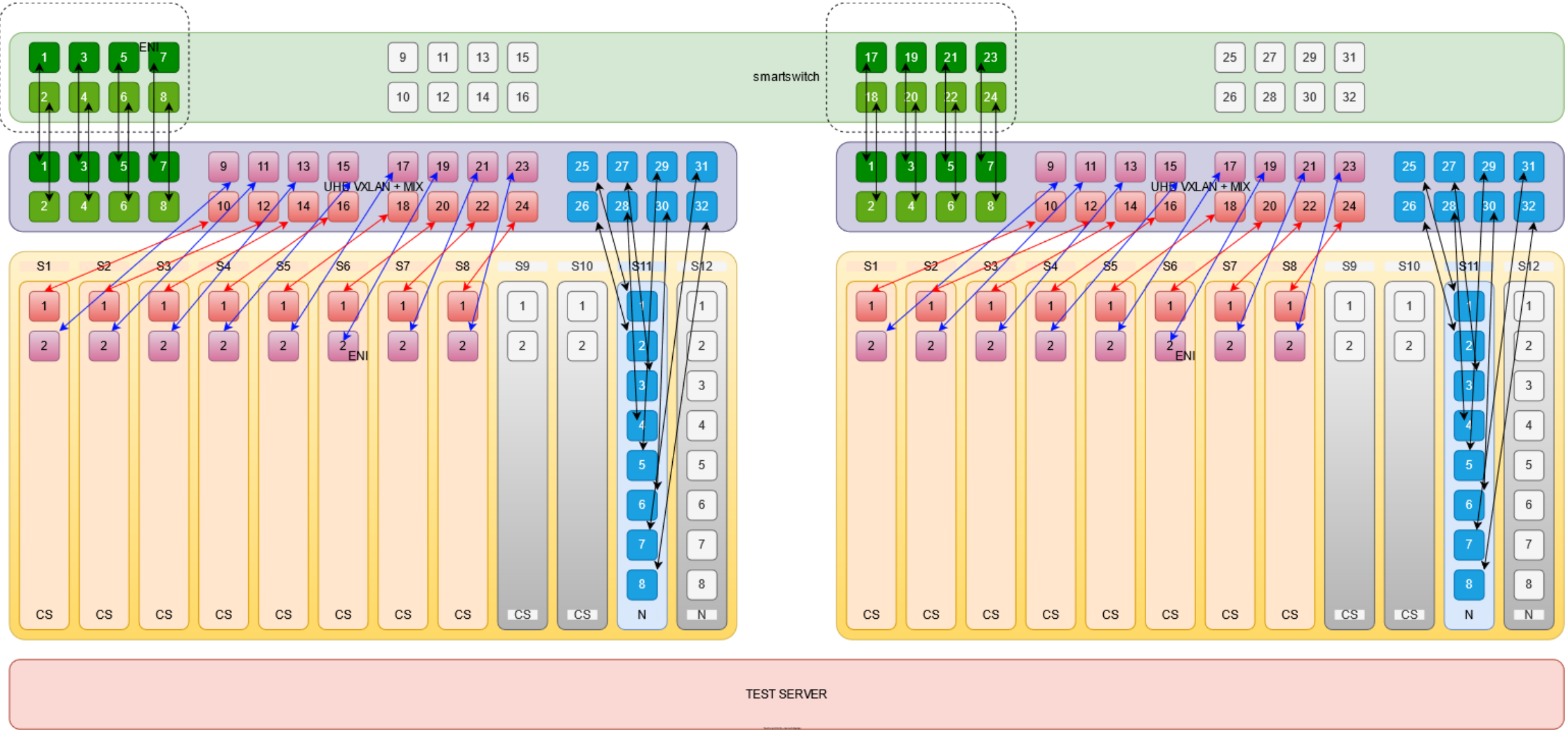


	Traffic Item	Tx Frames	Rx Frames	Frames Delta	Loss %	Tx Frame Rate	Rx Frame Rate	Tx L1 Rate (bps)	Rx L1 Rate (bps)	Rx Bytes	Tx Rate (Bps)	Rx Rate (Bps)	Tx Rate (bps)	Rx Rate (bps)	Tx Rate (Kbps)	Rx Rate (Kbps)	Tx Rate (Mbps)	Rx Rate (Mbps)	Store-F
1	Allow	30,527,839	30,527,832	7	0.000	2,000,000.000	2,000,000.000	2,368,000,000.000	2,368,000,000.000	3,907,562,496	256,000,000.000	256,000,000.000	2,048,000,000.000	2,048,000,000.000	2,048,000.000	2,048,000.000	2,048.000	2,048.000	
2	Deny	15,264	0	15,264	100.000	1,000.000	0.000	1,184,000.000	0.000	0	128,000.000	0.000	1,024,000.000	0.000	1,024.000	0.000	1.024	0.000	

Scale

	per ENI	200G (DPU)	400G	800G	1.6T (smart switch)
VNETs		1024	2048	4096	8192
ENIs		64	128	256	512
Routes	100K	6.4M	12.8M	25.6M	51.2M
NSGs	5in + 5out	640	1280	2560	5120
ACLs prefixes	10x100K	64M	128M	256M	512M
ACLs Ports	10x10K	6.4M	12.8M	25.6M	51.2M
Mappings (CA to PA)	160K	10M	20M	40M	80M
Act Con	500K (bidir w/ connection pool capable of oversubscription)	32M	64M	128M	256M
CPS		3.75M	7.5M	15M	30M
bg flows TCP		15M (bidir w/ connection pool capable of oversubscription)	30M	60M	120M
bg flows UDP		15M (bidir w/ connection pool capable of oversubscription)	30M	60M	120M

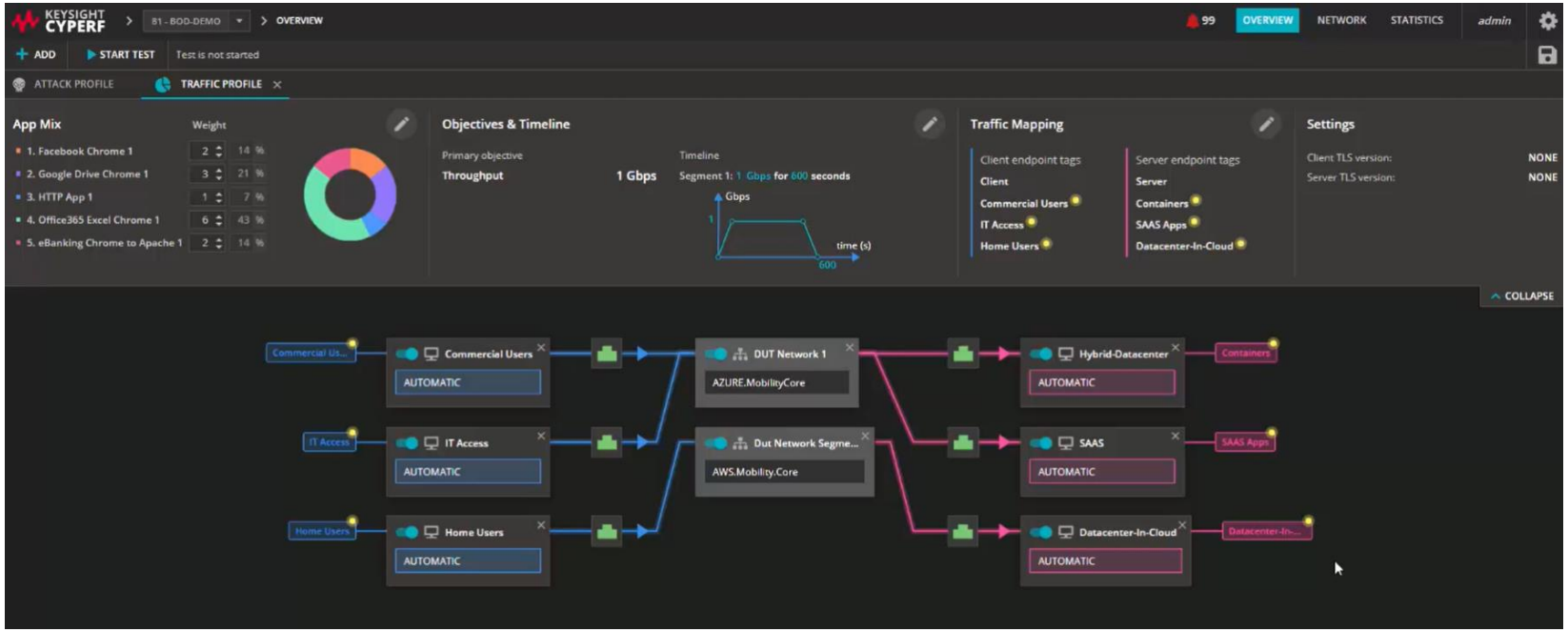
Smart Switch



Post deployment

Changes to hero test:

- Replace the ACLs to make use of many UDP/TCP ports instead of IPs
- New config generator that will use azure cli to configure the testbed and rules
- Use CyPerf Azure marketplace images to generate the flows



https://azuremarketplace.microsoft.com/en-US/marketplace/apps/keysighttechnologies_cyperf.keysight-cyperf?ocid=GTMRewards_WhatsNewBlog_keysight-cyperf_120722

Call to action

- Great ***partnership*** so far in SONiC and DASH projects
- We will continue the ***contribution***
- Bring us in for new projects so we can contribute from initiation
- We are happy to work as an ***extended team*** to jointly make open-source projects like SONiC a success

Let's

CODESIGN

the

Networks of Future

KEYSIGHT

Thank you