P4 2022 Workshop
May 24-26th

P4 as a single source of truth for SONiC DASH use cases on both SoftSwitch and Hardware

Reshma Sudarshan, Dir. Applications Engineering, Intel Corp.

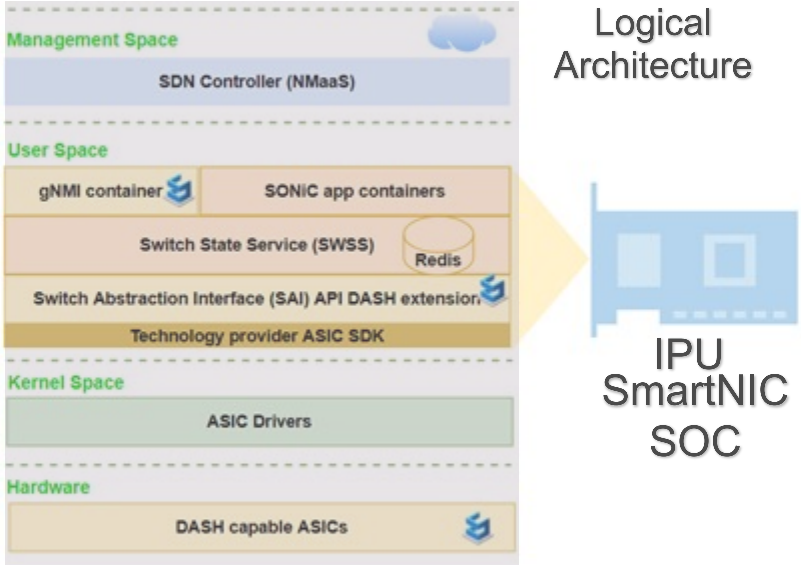
Chris Sommers, Sr. SW Architect, Keysight Technologies

DASH SONiC

DASH extends SONiC APIs to Edge Use cases
- Stateless Underlay Route, LPM, ACL Support
- Stateful Connections for L4 Load Balancing
- SDN managed Overlay Services

Use case Scenarios
- VNET to VNET Service - Optimal TCP flow management
  Optimize high CPS, add-on miss HW tables
  Connection Tracking, optimize Flow-close state machine
- Load Balancer Service
  Customized algorithms with fine grained criteria for LB
- Encryption Gateway Service
  Crypto Offload IPsec infrastructure tenant crypto

- VNET to VNET service
- VNET peering service
- Service tunnel & Private link service
- Load balancer service
- Encryption gateway service
- Express route gateway service



Management Space

SDN Controller (NMaaS)

User Space

gNMI container

SONiC app containers

Switch State Service (SWSS)

Redis

Switch Abstraction Interface (SAI) API DASH extension

Technology provider ASIC SDK

Kernel Space

ASIC Drivers

Hardware

DASH capable ASICs

Logical Architecture

IPU SmartNIC SOC

SONiC ✦ Switch | TOR | Spine | Border-Leaf

DASH ✦ SmartNIC | Appliance | Smart-Switch

P4 to Describe all Network Elements

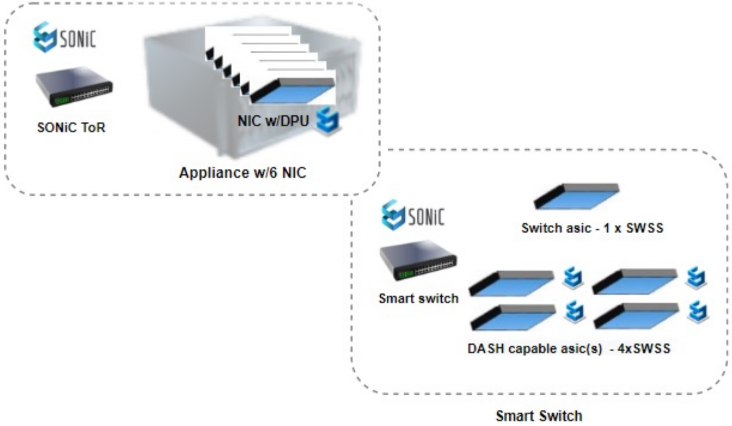
Rich data plane representation

- Precise and comprehensive definition of life of a packet
- Visibility and control to the Network operator
- Allows manageability
- Network function specific mechanism – PSA | PNA

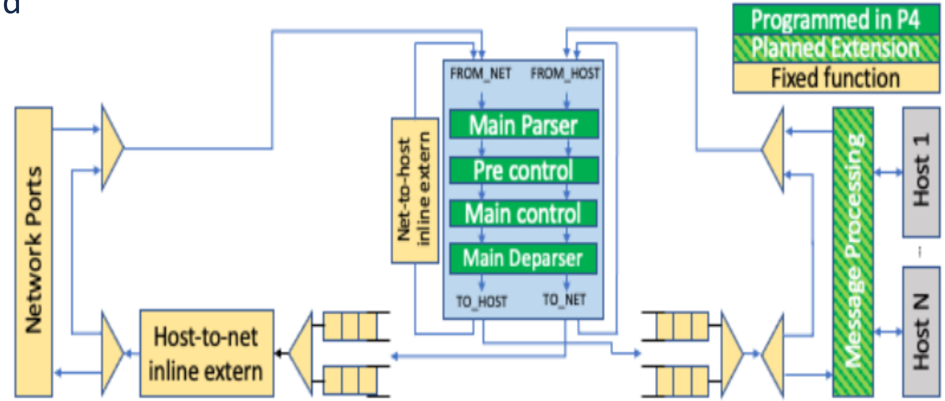
PNA for DASH scenarios -

New P4 properties and extern functions defined in P4 standard Portable NIC Architecture.

Uniform standards based way to describe all network functions in P4



Programmable NIC Architecture



P4 for DASH Overlay - Stateful Connection Tracking

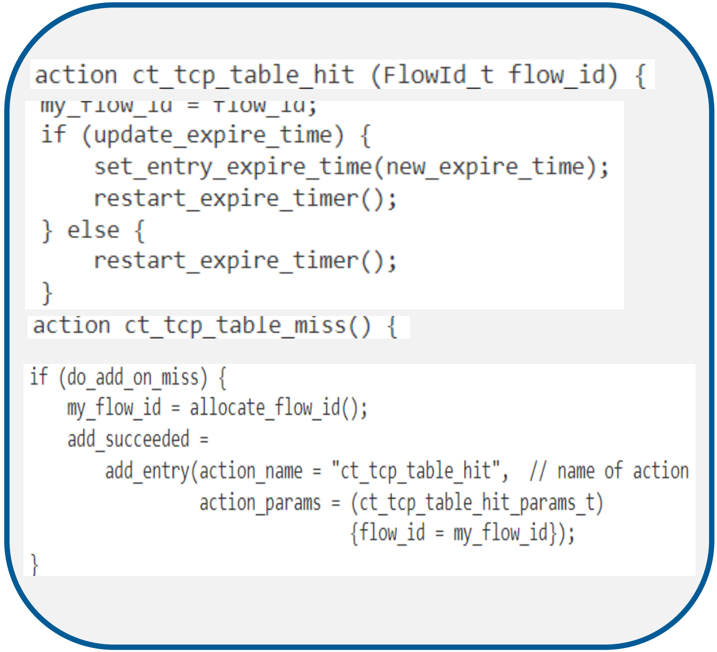
Optimal management of large number of Flows

High Connections Per Second rules are programmed in HW to add flow entry in HW without Control Plane involvement

P4 to program flow Creation – Termination Timers and State Machine

- First SYN triggers add-on-miss adds to flow cache, start absolute timer
- In New state SYN+ACK flags trigger Established state timer
- In Established state TCP FIN or RST flags transition to teardown state and start timers
- In Tear Down state machine FIN+ACK triggers flow deletion

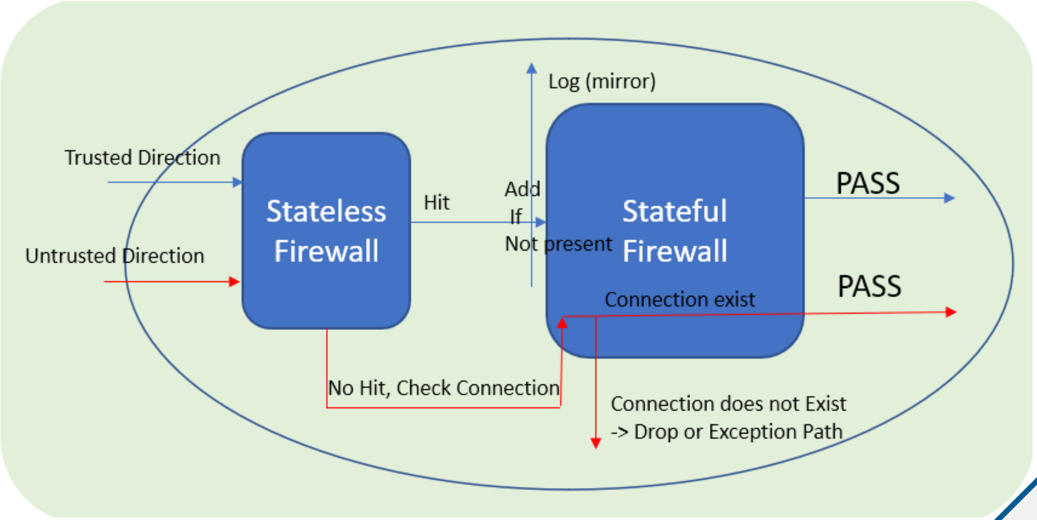
A public version of this example program can be found here:
https://github.com/p4lang/pna/blob/main/examples/pna-example-tcp-connection-tracking.p4



```
action ct_tcp_table_hit (FlowId_t flow_id) {
 my_flow_id = flow_id;
 if (update_expire_time) {
     set_entry_expire_time(new_expire_time);
     restart_expire_timer();
 } else {
     restart_expire_timer();
 }
action ct_tcp_table_miss() {

if (do_add_on_miss) {
    my_flow_id = allocate_flow_id();
    add_succeeded =
        add_entry(action_name = "ct_tcp_table_hit",  // name of action
                  action_params = (ct_tcp_table_hit_params_t)
                                  {flow_id = my_flow_id});
}
```

Stateful Firewall and Load Balancer

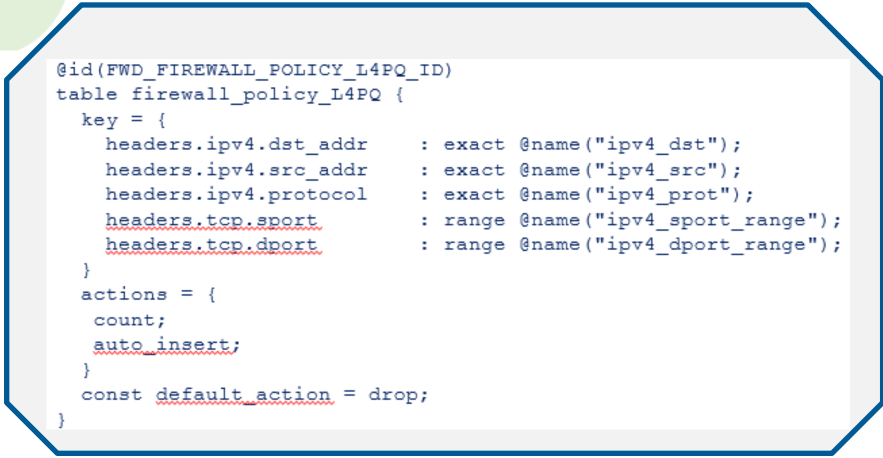


Stateful Firewalls

- Stateless Firewall : Match = Remote IP, IP Protocol, Dest L4 port)
- Stateful Firewall : Match = 5 tuple + CT Zone (Unique connection)
- Packet Permitted by Stateless firewall rule OR is part of existing connection.

SDN based Centralized load balancing

Incoming request to Service VIP load balanced to service end point

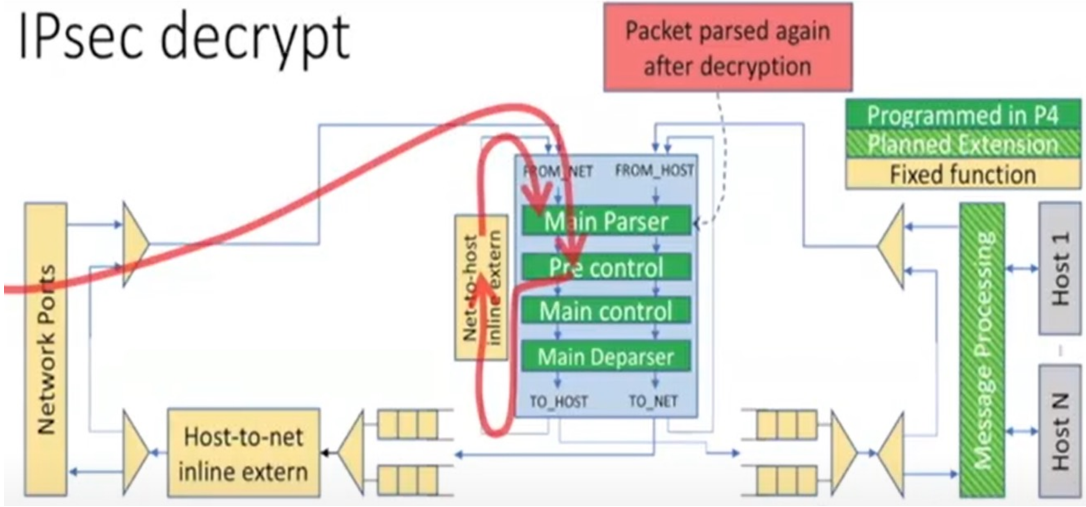


```
@id(FWD_FIREWALL_POLICY_L4PQ_ID)
table firewall_policy_L4PQ {
  key = {
    headers.ipv4.dst_addr     : exact @name("ipv4_dst");
    headers.ipv4.src_addr     : exact @name("ipv4_src");
    headers.ipv4.protocol     : exact @name("ipv4_prot");
    headers.tcp.sport         : range @name("ipv4_sport_range");
    headers.tcp.dport         : range @name("ipv4_dport_range");
  }
  actions = {
   count;
   auto_insert;
  }
  const default_action = drop;
}
```

Security - IPsec Crypto offload

P4 implementation for IPsec

- Table lookup check ESP header in IPsec packet
- Parser SPI + Src-IP lookup
- - Security association index
- Parse decrypted packet
- IPSEC and is wrapped in a tunnel like GRE



IPsec feature in SONiC (Roadmap)

- New IPsec container in SONiC
- StrongSwan application for IKE exchange
- Plugin for Security Association index programming

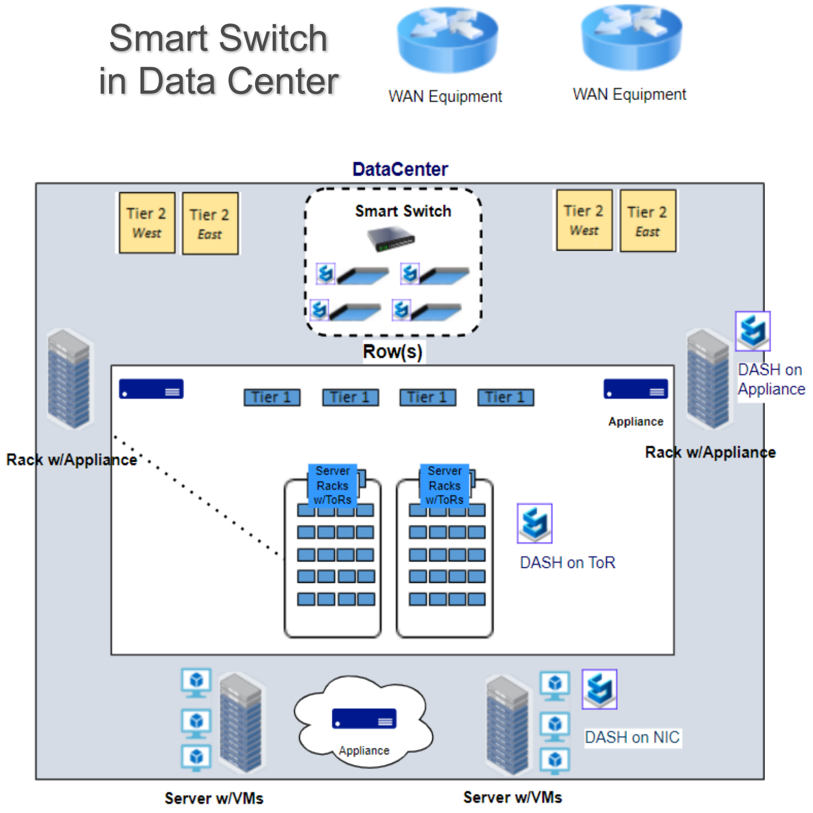
P4 Implementation for DASH Smart Switch

Smart Switch assumes role of Switch and offloads server functionality via SmartNIC

Flow table lookups in NPU and IPUs are described in P4

- 5 tuple lookups for flow programming
  - IPU add-on-miss
- IPSEC cache with entropy
  - 3-tuple hash with Src-IP+Dest-IP+SP
- VxLAN lookup
  - UDP src port 5-tuple hash in VxLAN header

```
table l1_cache {
   key = {
      ig_md.lkp.ip_src_addr[95:64] : exact;
      ig_md.lkp.ip_dst_addr[95:64] : exact;
      ig_md.lkp.ip_proto : exact;
      ig_md.lkp.l4_src_port: exact;
      ig_md.lkp.l4_dst_port: exact;
   }
```


Smart Switch in Data Center

SONiC DASH SoftSwitch with P4DPDK

Same software stack to manage

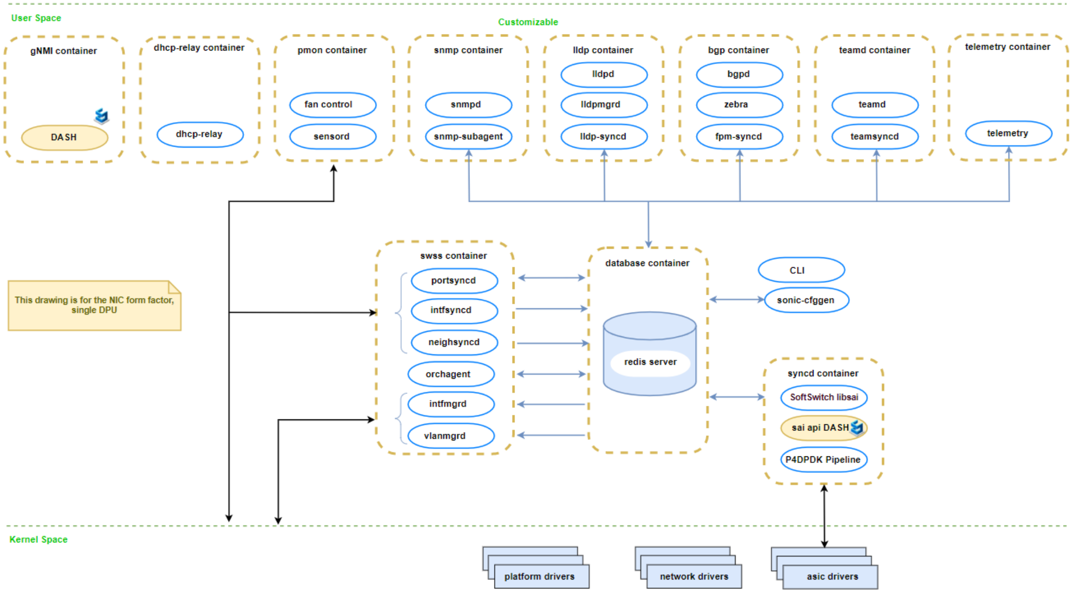
DASH HW and SW

SONiC runs in Host / VM

- OS de-coupled from customer’s environment
- Separate software lifecycle

P4-DPDK Dataplane

P4 Compiled Dataplane



SONiC DASH Softswitch Architecture

```
action set_nexthop_id(
nexthop_id_t nexthop_id)
    nexthop_id_valid =
true;
    nexthop_id_value =
nexthop_id;
```

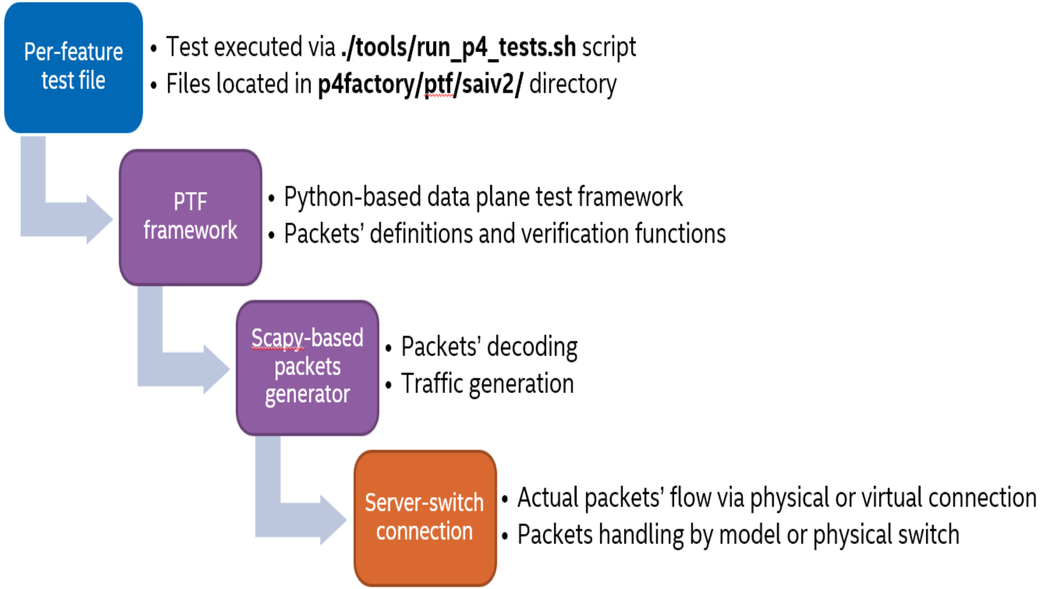
```
table ipv4_table {
  key = {
     hdr.ipv4.dst_addr : lpm;}
   actions = {
     set_nexthop_id;
     @defaultonly NoAction;}
   const default_action = NoAction;
```

SAI PTF Test Harness for DASH

- Auto-generated Python based dataplane testing framework
- Thrift wrapper functions to call C-based SAI functions
- Generated wrapper functions for SAI which can instead be generated based on SAI headers

DASH related enhancements:

- Auto-generation framework and Underlay Test cases adopt to platforms with fewer ports
- Auto-generation framework for DASH SAI APIs meta infrastructure
- DASH and new Overlay test cases using this new framework



Per-feature test file

- Test executed via **./tools/run_p4_tests.sh** script
- Files located in **p4factory/ptf/saiv2/** directory

PTF framework

- Python-based data plane test framework
- Packets' definitions and verification functions

Scapy-based packets generator

- Packets' decoding
- Traffic generation

Server-switch connection

- Actual packets' flow via physical or virtual connection
- Packets handling by model or physical switch

Keysight's Role in the SONiC & DASH Communities

KEYSIGHT
TECHNOLOGIES

- Global electronic test and measurement company, multi-$B revenue. If it has electrons, radio waves or qubits, we can test it!
- Known in the SONiC community for our testing expertise; HW & SW solutions such as Testbed-in-a-box; Plug-fests; WG presence; and GitHub contributions.
- Our IXIA-brand Traffic Generators are a fixture in the networking industry.

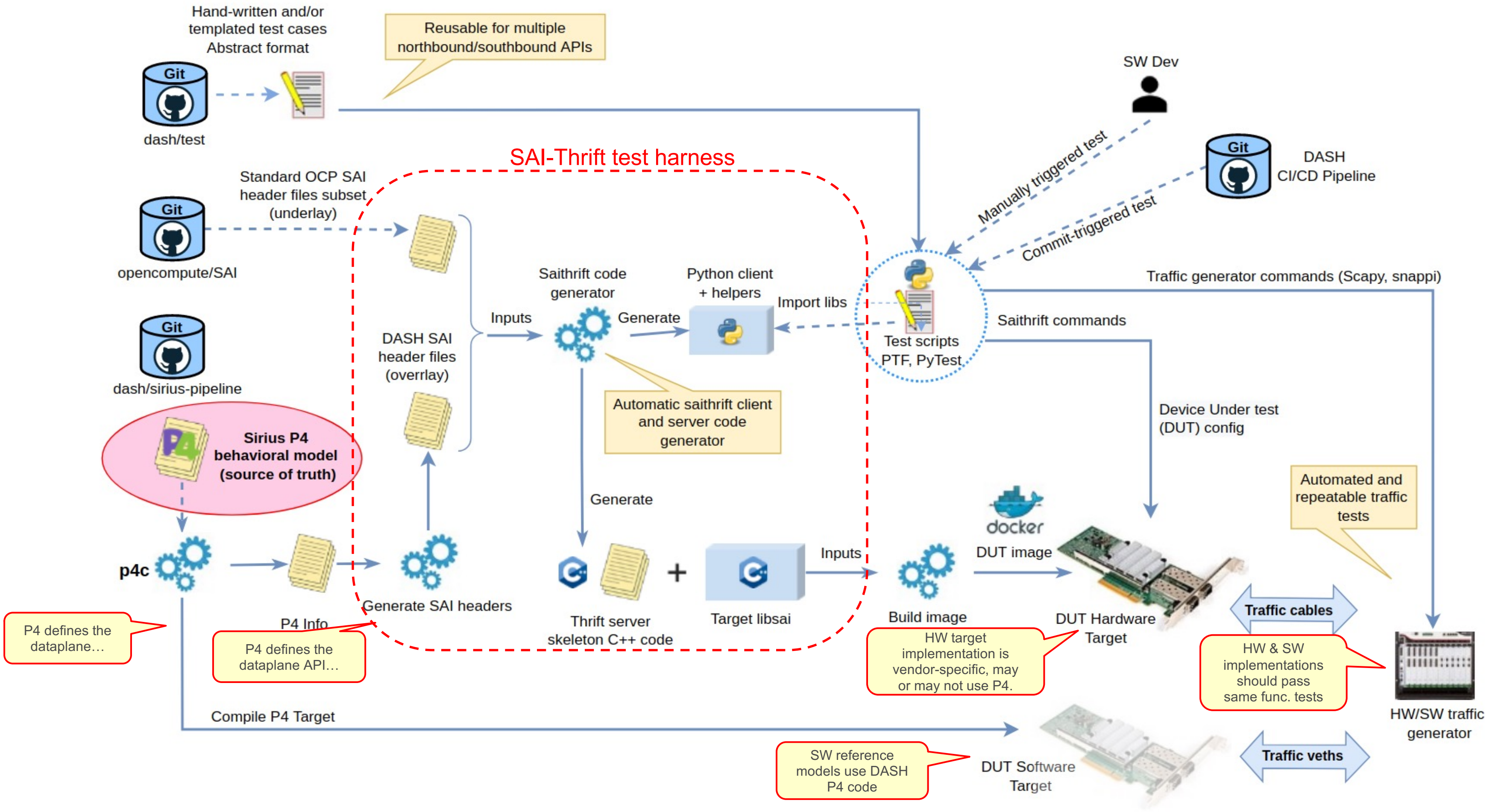
ixia
A Keysight Business

- Leveraging our SONiC expertise in the SONiC-DASH project.
- Keysight is a trusted, neutral partner, to help define and deliver test infrastructure, automation, and test cases.
- Community engagement: Contributions to GitHub and working groups.
- Customer engagement: confidential testing and evaluations.

DASH Testing – Objectives

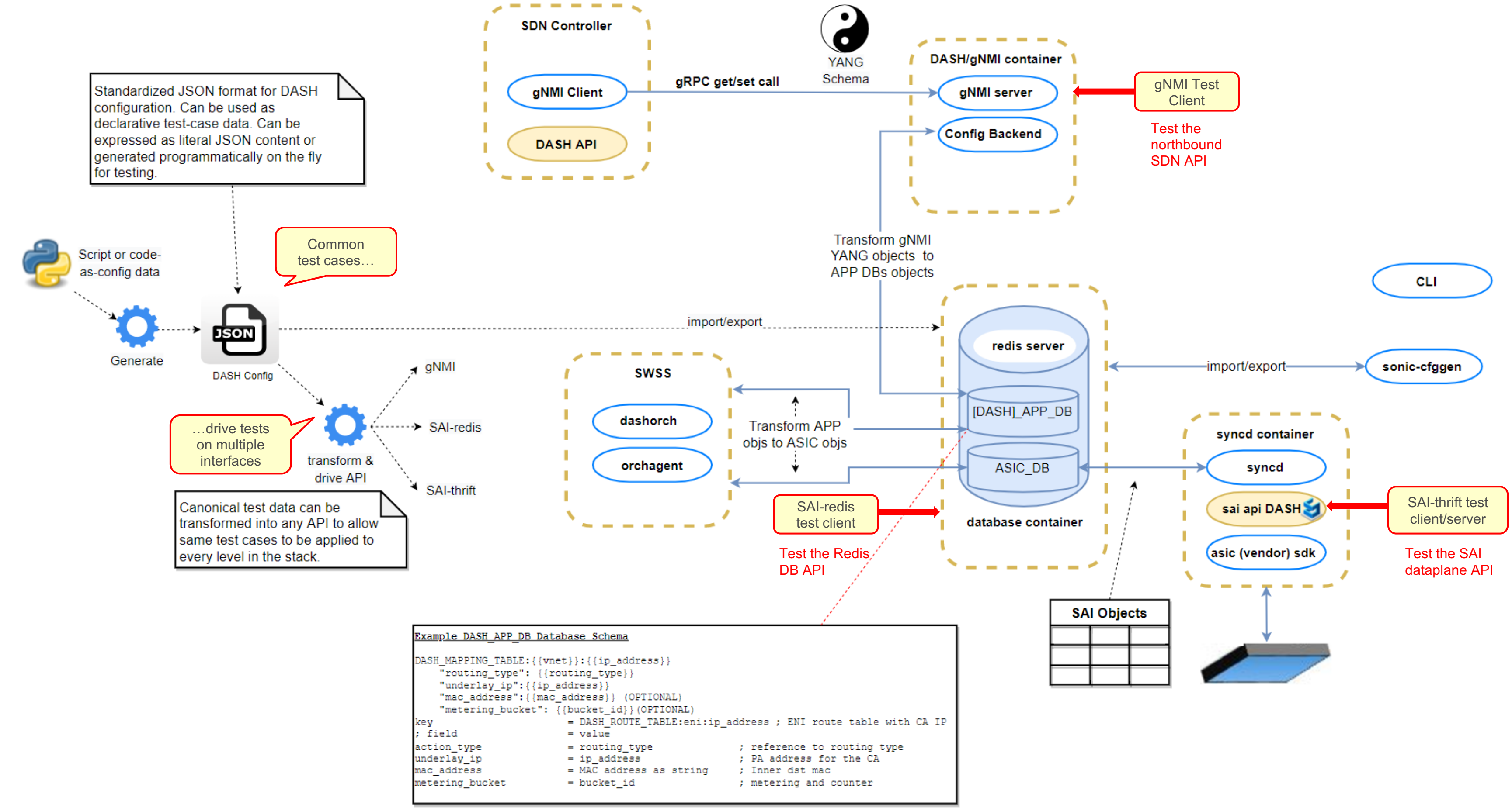
- Stateless (Layer2/Layer3) and stateful (Layer4) traffic tests
- Performance (e.g. 3M+ connections per sec) and Conformance (thorough API/functional)
- P4 models the dataplane traffic path; SAI configures the dataplane.
- Test multiple API layers: dataplane (SAI); SONiC (Redis); SDN (gNMI)
- Same functional tests used for multiple targets (scale/performance varies):
  - Pure SW implementations (P4), can run on a server w/ SW traffic generators
  - Line-rate, HW implementations – DPU/IPU/SmartNIC + HW Traffic Generators
- Automated CI/CD regression testing, in the cloud and the lab (GitHub actions)
- Provide a framework where everyone can contribute test cases

DASH Testing – Workflows & auto-generated artifacts



Hand-written and/or templated test cases Abstract format
Reusable for multiple northbound/southbound APIs
Git
dash/test
SW Dev
SAI-Thrift test harness
Standard OCP SAI header files subset (underlay)
Manually triggered test
Git
DASH CI/CD Pipeline
Commit-triggered test
Git
opencompute/SAI
Saithrift code generator
Python client + helpers
Import libs
Traffic generator commands (Scapy, snappi)
Inputs
Generate
DASH SAI header files (overlay)
Test scripts PTF, PyTest
Saithrift commands
Git
dash/sirius-pipeline
Automatic saithrift client and server code generator
Device Under test (DUT) config
Sirius P4 behavioral model (source of truth)
Generate
Automated and repeatable traffic tests
docker
DUT image
p4c
Inputs
+
Generate SAI headers
P4 Info
Thrift server skeleton C++ code
Target libsai
Build image
DUT Hardware Target
Traffic cables
P4 defines the dataplane…
P4 defines the dataplane API…
HW target implementation is vendor-specific, may or may not use P4.
HW & SW implementations should pass same func. tests
HW/SW traffic generator
Compile P4 Target
SW reference models use DASH P4 code
DUT Software Target
Traffic veths

DASH Testing – API/Schema layers, common test cases



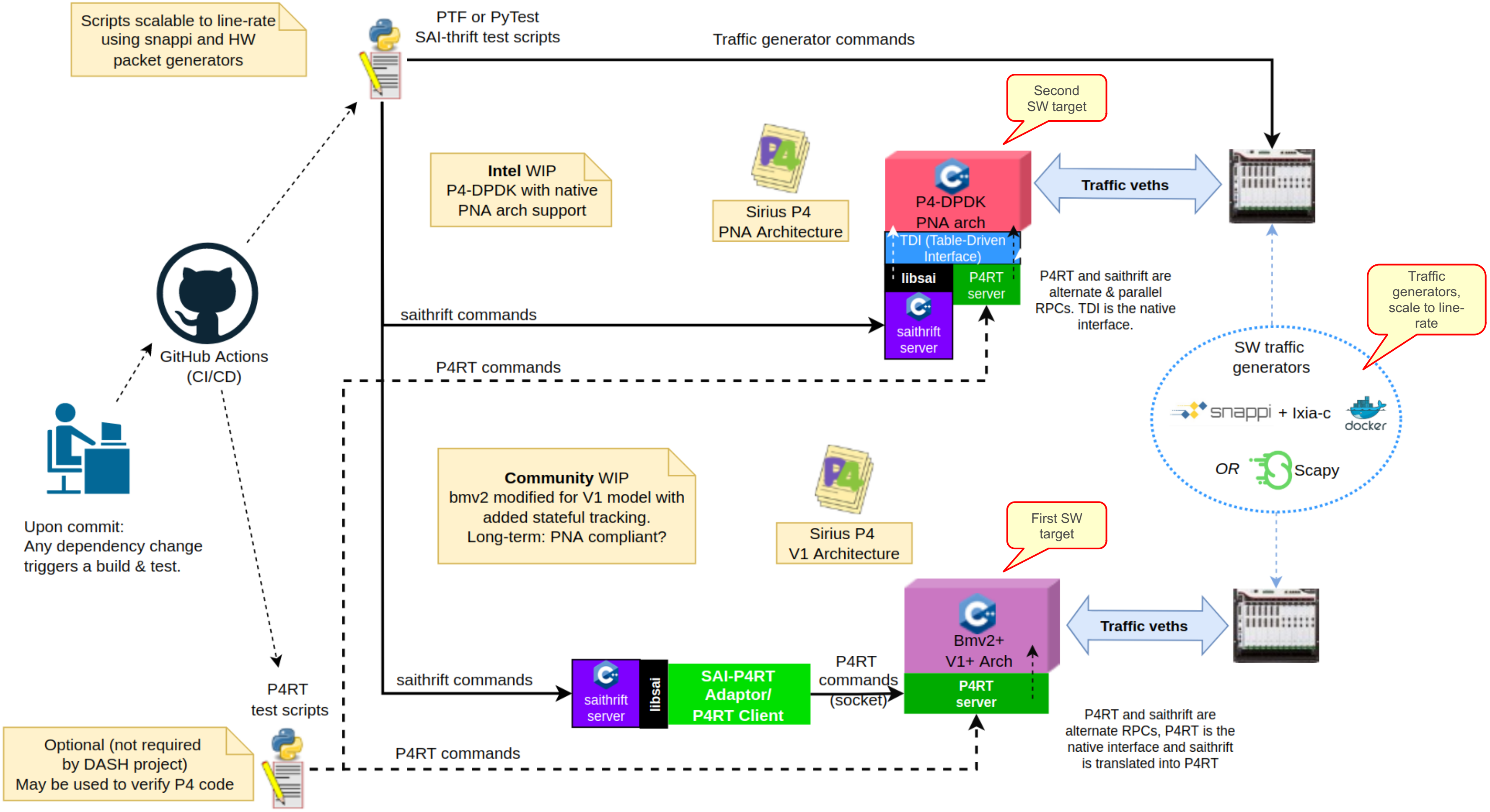
Standardized JSON format for DASH configuration. Can be used as declarative test-case data. Can be expressed as literal JSON content or generated programmatically on the fly for testing.

Canonical test data can be transformed into any API to allow same test cases to be applied to every level in the stack.

```
Example DASH_APP_DB Database Schema

DASH_MAPPING_TABLE:{{vnet}}:{{ip_address}}
    "routing_type": {{routing_type}}
    "underlay_ip":{{ip_address}}
    "mac_address":{{mac_address}} (OPTIONAL)
    "metering_bucket": {{bucket_id}}(OPTIONAL)
key                         = DASH_ROUTE_TABLE:eni:ip_address ; ENI route table with CA IP
; field                     = value
action_type                 = routing_type                ; reference to routing type
underlay_ip                 = ip_address                  ; PA address for the CA
mac_address                 = MAC address as string       ; Inner dst mac
metering_bucket             = bucket_id                   ; metering and counter
```

DASH Testing – P4 Model, multiple SW targets



Scripts scalable to line-rate using snappi and HW packet generators

PTF or PyTest SAI-thrift test scripts

Traffic generator commands

Second SW target

Intel WIP P4-DPDK with native PNA arch support

Sirius P4 PNA Architecture

P4-DPDK PNA arch

TDI (Table-Driven Interface)

libsai

P4RT server

saithrift server

Traffic veths

P4RT and saithrift are alternate & parallel RPCs. TDI is the native interface.

Traffic generators, scale to line-rate

GitHub Actions (CI/CD)

saithrift commands

P4RT commands

SW traffic generators

snappi + Ixia-c docker

OR Scapy

Upon commit:
Any dependency change triggers a build & test.

Community WIP bmv2 modified for V1 model with added stateful tracking. Long-term: PNA compliant?

Sirius P4 V1 Architecture

First SW target

Bmv2+ V1+ Arch

P4RT server

Traffic veths

P4RT test scripts

saithrift commands

saithrift server

libsai

SAI-P4RT Adaptor/ P4RT Client

P4RT commands (socket)

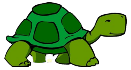
P4RT and saithrift are alternate RPCs, P4RT is the native interface and saithrift is translated into P4RT

Optional (not required by DASH project) May be used to verify P4 code

P4RT commands

DASH Testing Framework:Traffic Generators

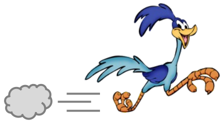
Traditional PTF: Python Framework + Scapy



- Very popular; a large body of test cases
- SW Traffic generator, great for packet-at-a-time functional tests
- No line-rate support

Each approach has merits and DASH will embrace both

Enhancement: Python Framework + snappi



ixia-c

- SW or HW Traffic Generators
- Agnostic data model & API
- Advanced features – latency, flow stats, etc.
- Scale to line rate w/ same scripts

Golang
Also available

https://github.com/open-traffic-generator • https://github.com/open-traffic-generator/snappi

* See “snappi” video links at end of document

Conclusions

- P4 is being used to model stateful DASH overlay services, as a *single source of truth for dataplane behavior.*
- DASH P4 can be run in pure SW switches, or HW/SoC-based devices
- DASH P4 can model behavior on both non-P4 devices and P4-based devices
- DASH P4 code generates APIs, e.g., DASH-SAI header files + SAI-Thrift test harness.
- Declarative, data-driven test cases can exercise multiple API layers in the SONiC stack: SAI, SAI-Redis, gNMI.
- Classic SAI-thrift tests are being extended to support new DASH services
- Adding snappi-based tests to handle slow or fast traffic testing with same scripts, HW or SW

Call to Action

Join the DASH Project:

- https://github.com/Azure/DASH
- https://groups.google.com/g/sonic-dash • https://groups.google.com/g/sonic-dash-test-workgroup

Join the IPDK Project:

- https://ipdk.io

Join the Portable NIC Architecture Working Group:

- https://p4.org/working-groups • https://github.com/p4lang/pna

Contribute to the Open Traffic Generator data models and implementations:

- https://github.com/open-traffic-generator • https://github.com/open-traffic-generator/snappi

Additional Links

- https://github.com/Azure/DASH/tree/main/sirius-pipeline
- https://github.com/opencomputeproject/SAI/tree/master/test/saithriftv2

- Goodbye Scapy hello snappi – YouTube (https://www.youtube.com/watch?v=Db7Cx1hngVY)
- Open Traffic Generator snappi Ixia-c – YouTube (https://www.youtube.com/watch?v=3p72YnLFZVQ)

Notices and disclaimers

- Intel technologies may require enabled hardware, software or service activation.
- No product or component can be absolutely secure.
- Your costs and results may vary.
- Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.
-

Thank You

Sudarshan, Reshma reshma.sudarshan@intel.com
Chris Sommers chris.sommers@keysight.com