Network CI with Open Traffic Generator API

Alex Bortok, Keysight

19-OCT-2022

Agenda

- When to use a Traffic Generator
- Open Traffic Generator API
- How to use OTG API
- OTG and OpenConfig
- NetOps CI with OTG

When to use a Traffic Generator?

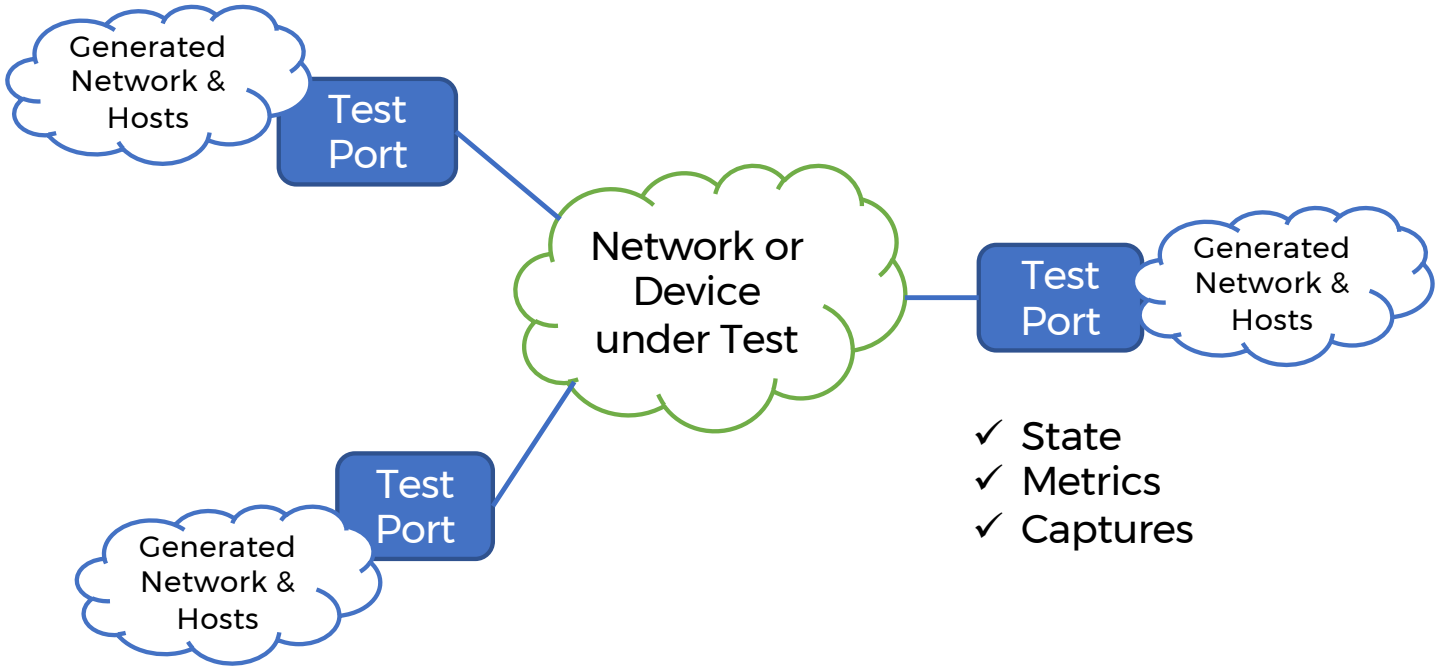
19-OCT-2022

NANOG™

What is a Traffic Generator?

- ✓ Packet frame constructor
- ✓ Flow scheduler
- ✓ L2-3 protocol emulator

- ➢ Stateful connections
- ➢ Application payloads



- ✓ State
- ✓ Metrics
- ✓ Captures

Traffic Generator creates “clouds” of network and hosts behind its Test Ports with complete configuration of OSI layers 2-4, with optional L4-7 realism.

Why use a Traffic Generator?

✓ Mature
❖ More established
❑ Less established

Build Product	*Deploy Network*	*Operate System*
Control	Validate	Maintain
✓ Quality	✓ Components	❖ Interoperability
✓ Specs	❖ SLOs	❖ RCAs
✓ Conformance	❑ Design	❑ Availability

Opportunities to Enable

Opportunities with Openness

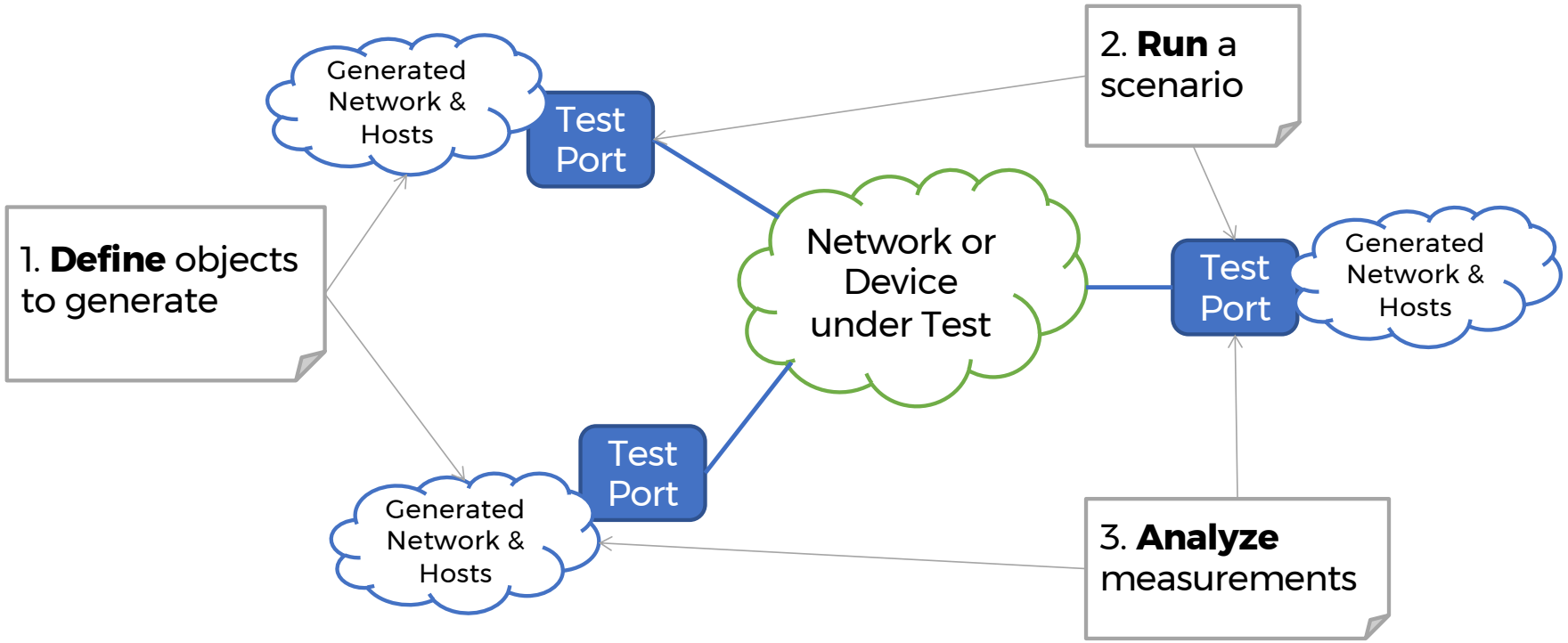
- Shared Vendor/Operator test workflow
  - Without Test Vendor Lock-in
- Test content for open-source NOS projects
  - Accelerate testing for scale by corporate users
- Lower cost of 3rd party integration
  - Multiple parties can contribute more easily
- Enable Continuous Integration for Network Operators
  - Reuse & contribute through community

Open Traffic Generator API

19-OCT-2022

NANOG™

API Surface


Generated Network & Hosts
Test Port
2. **Run** a scenario
1. **Define** objects to generate
Network or Device under Test
Test Port
Generated Network & Hosts
Test Port
Generated Network & Hosts
3. **Analyze** measurements

OTG Model

Define

Configuration

- Layer1
- Ports/LAGs
- Flows
- Devices
- Events

Run

Control

- Link
- Protocol
- Route
- Transmit
- Flow
- Capture

Analyze

States

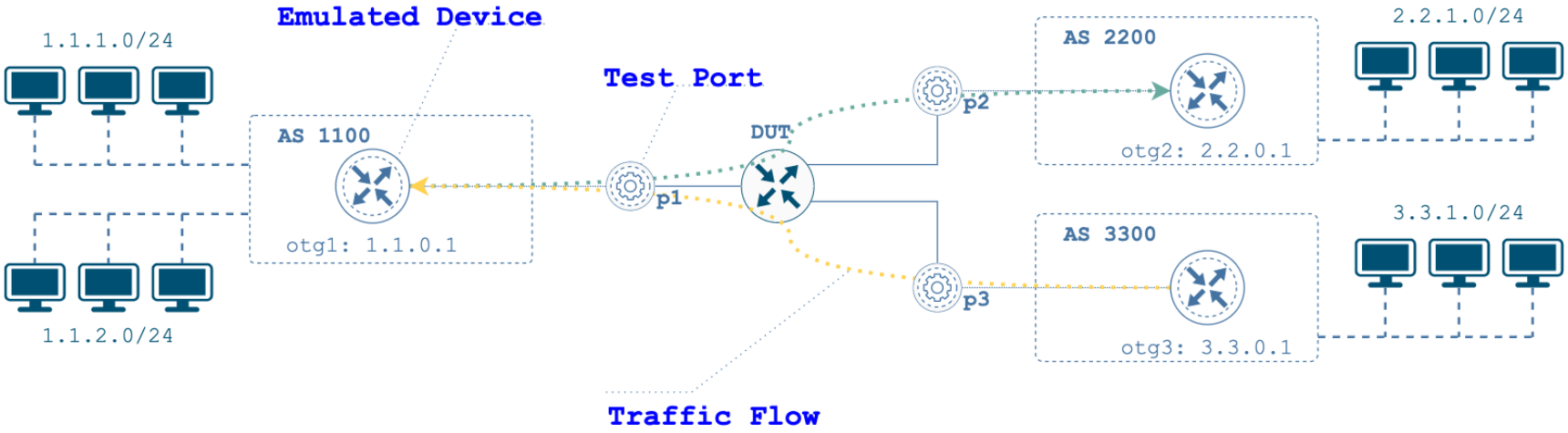
- ARP/NDISC
- Protocols

Metrics

- Port/LAG
- Flow
- Protocols

https://otg.dev/model/

OTG Configuration Elements



Visualization of OTG example configuration

Actual source: OpenAPI YAML/JSON/protobuf

https://otg.dev/model/

NANOG™

OTG Implementations

Clients

- REST/gRPC API
- snappi library
- otgen CLI tool

Engines

- Keysight Ixia-c
- OpenConfig magna
- Cisco TRex
- Keysight Elastic Network Generator

Test Content

- otg.dev/examples
- OpenConfig Feature Profiles
- SONiC Testbed extensions
- SONiC-DASH CI Pipeline

https://otg.dev/implementations

How to use OTG API

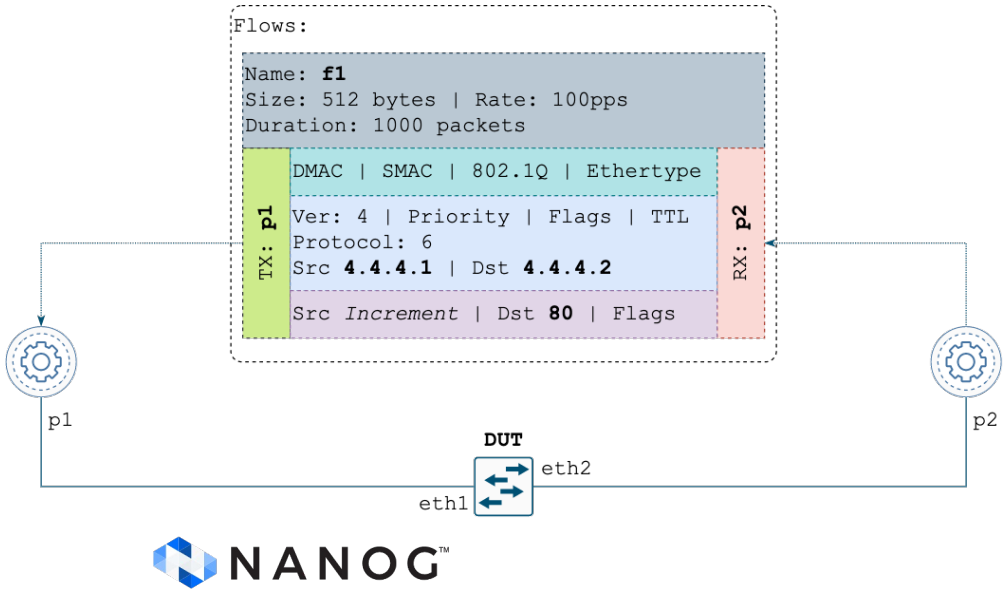
19-OCT-2022

NANOG™

otgen: raw traffic

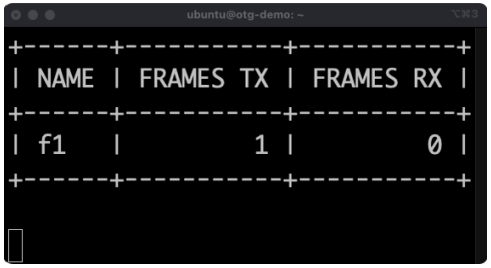
```
otgen create flow -P tcp -s 4.4.4.1 -d 4.4.4.2 -p 80 -r 100 |
otgen run --metrics flow | otgen report --metrics flow
```

OTG configuration



Run

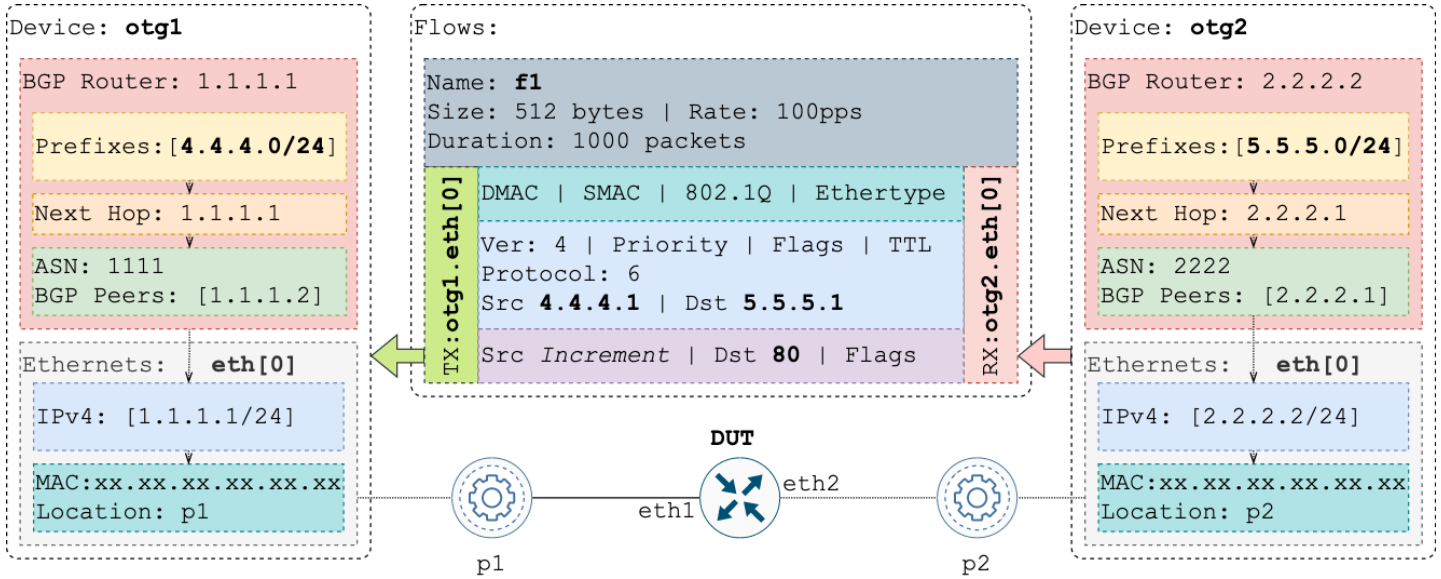
Report



NANOG™

otgen: traffic between BGP routers

```
otgen create device --name   otg1 --ip 1.1.1.1 --gw 1.1.1.2 --port p1 |
otgen add    device --name   otg2 --ip 2.2.2.2 --gw 2.2.2.1 --port p2 |
otgen add    bgp    --device otg1 --asn 1111 --peer 1.1.1.2 --prefix 4.4.4.0/24 |
otgen add    bgp    --device otg2 --asn 2222 --peer 2.2.2.1 --prefix 5.5.5.0/24 |
otgen add    flow   --tx otg1 --rx otg2 -s 4.4.4.1 -d 5.5.5.1
```



NANOG™

Test program: gosnappi

DEFINE
1. Import or create OTG config with snappi
2. Configure a DUT as needed

```
// Configure the header stack
pkt := flow.Packet()
eth := pkt.Add().Ethernet()
eth.Src().SetValue(flowSrcMac)
```

RUN
1. Start protocols and wait for convergence
2. Start traffic, periodically pull metrics
3. Stop when conditions are met

```
// push traffic configuration
res, err := api.SetConfig(config)
checkResponse(res, err)

// start transmitting configured flows
```

ANALYZE options
A. Export metric snapshots
B. Analyze metrics in test code
C. Consume metrics by external systems

```
// print metrics snapshots
for trafficRunning() {
  time.Sleep(otgPullInterval)
  metrics, err = api.GetMetrics(req)
  checkResponse(metrics, err)
}
```

Common Pitfalls

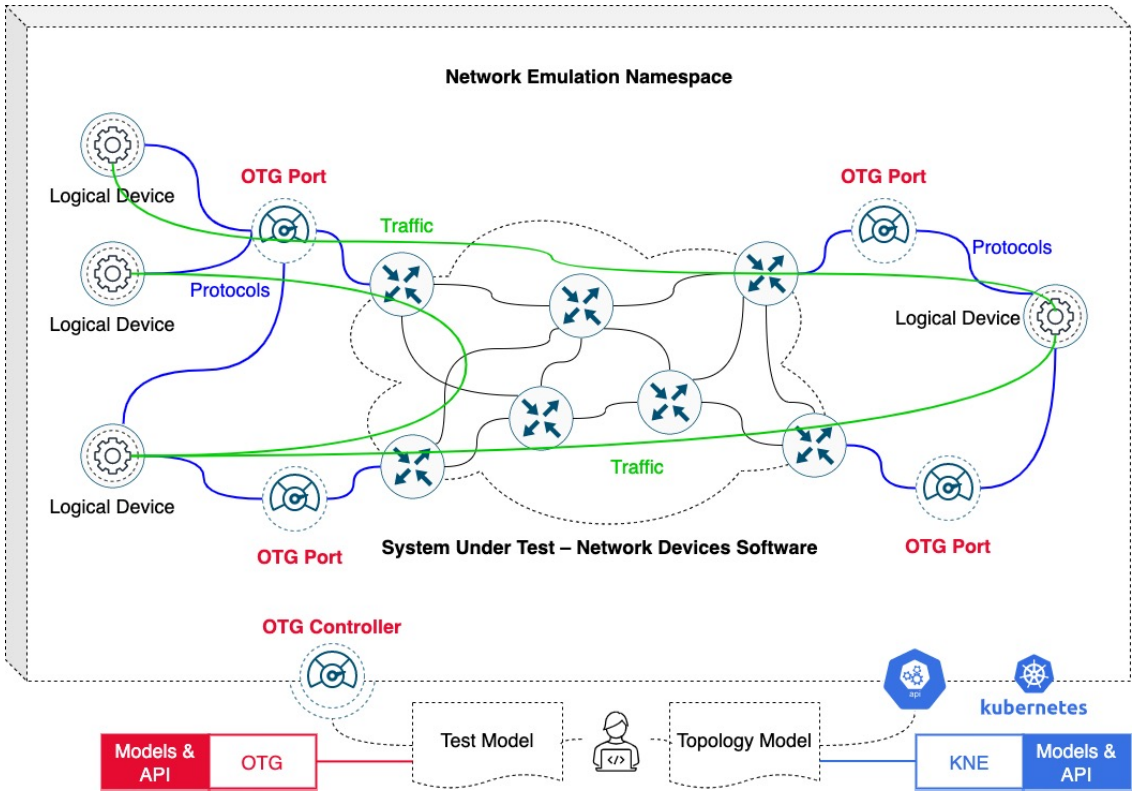
- Use of pre-existing VMs – broken dependencies
  - Start with clean Linux VM
- Going after realistic topologies – complicated OTG models
  - Start with two back-2-back test ports
  - Then one DUT – two test ports
- Writing your own snappi tests prematurely – delayed success
  - Test setup with otgen
  - Use otg-examples
- Non-declarative configurations – hard to reproduce
  - Use docker compose instead of docker run
  - Leverage network emulation: KNE or Containerlab

OTG and OpenConfig

19-OCT-2022

NANOG™

OTG with KNE



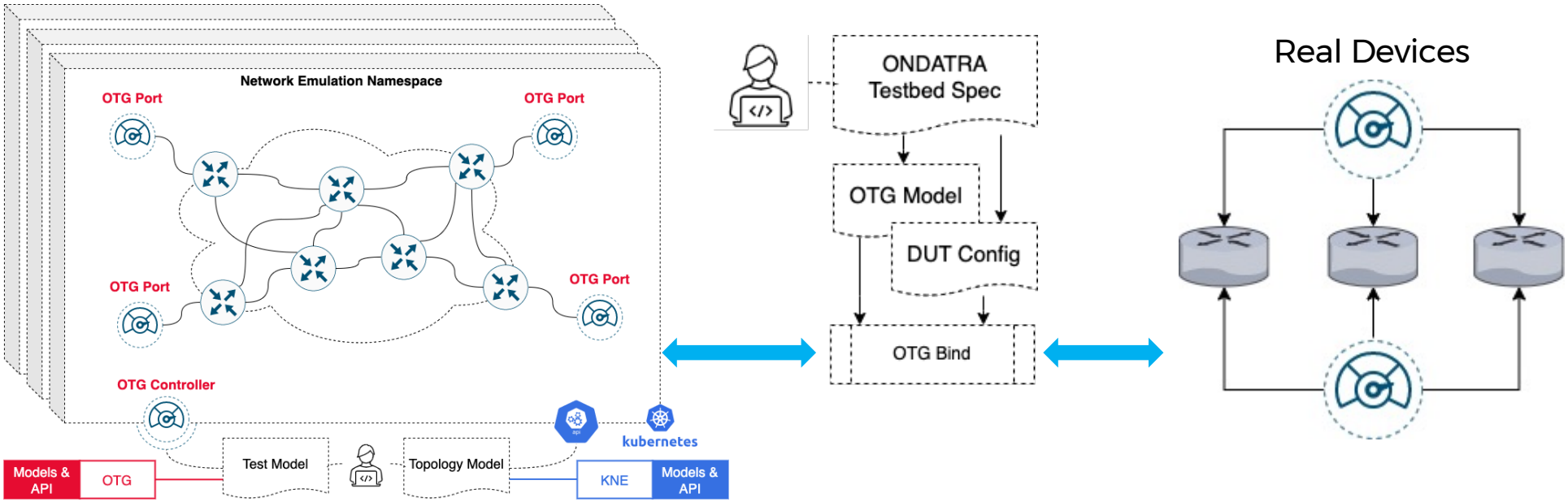
- Network topology with NOS containers
- OTG ports as Edge nodes
- Logical devices & networks behind OTG ports
- Routing protocols between logical devices and NOS containers
- OTG traffic flows originating from behind logical devices

NANOG™

https://github.com/openconfig/kne

OPENCONFIG Feature Profiles

Common Test Framework



NANOG™

https://github.com/openconfig/featureprofiles

OTG & NetOps CI

19-OCT-2022

NANOG™

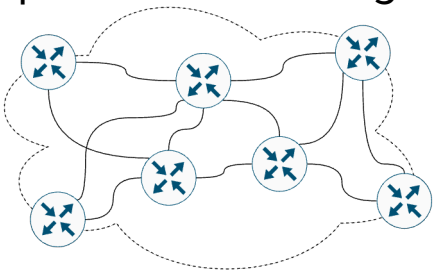
Would the latest NOS work?

- What if you would have to upgrade tonight?
- Would automated test of every new version help?

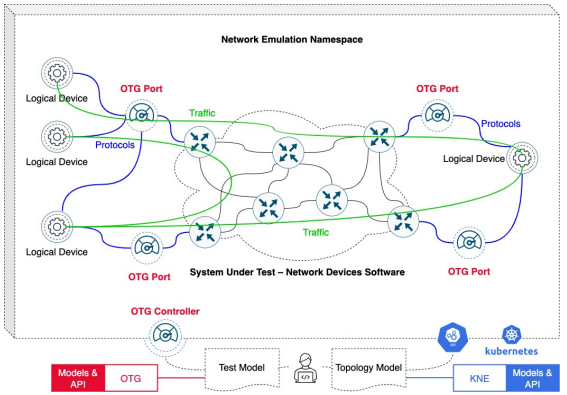
New NOS version

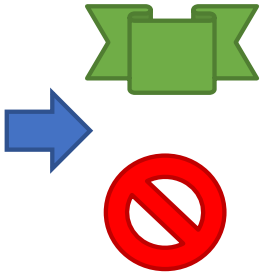
Service objectives

Impacted network segments



any @ outside	1.2.3.4 UDP/53	Should pass
any @ outside	1.2.3.4 TCP/22	Should fail



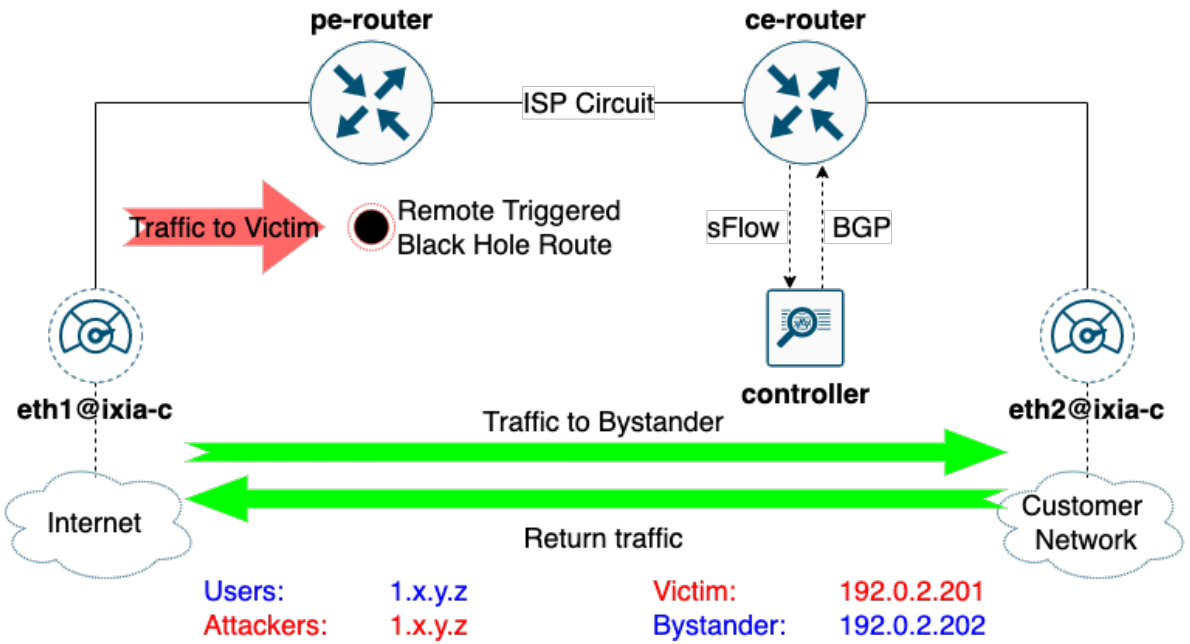


NANOG™

snappi

CI Example

RTBH DDoS Mitigation Validation



Any > Servers	Below threshold	Should pass
Any > Servers	Above threshold	Should be blocked
Servers > Any	N/A	Should pass

https://github.com/open-traffic-generator/otg-examples/tree/main/clab/rtbh

CI Example

Catching breaking changes in FRR v7.4.0 with RFC 8212 implementation

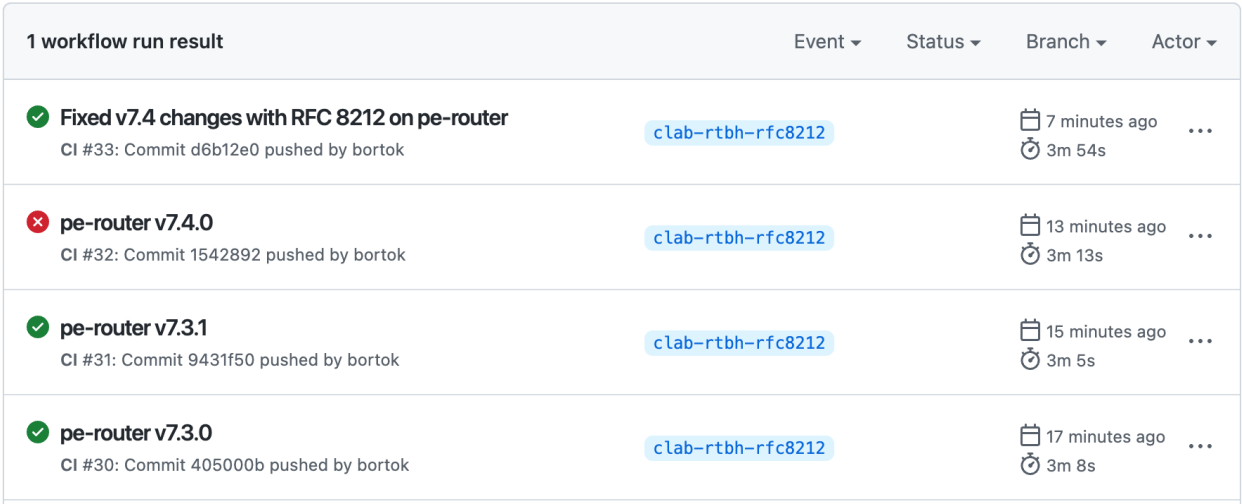
https://github.com/open-traffic-generator/otg-examples/actions?query=branch%3Aclab-rtbh-rfc8212

Makefile

```
deploy:
  sudo -E containerlab deploy --reconfigure -t topo.yml

test:
  go test ...
```

CI runs



References

Open Traffic Generator	https://otg.dev
Ixia-c engine free version	https://ixia-c.dev
Slack channel for support	https://otg.dev/#community
Compatible engines	https://otg.dev/implementations/

Thank you

19-OCT-2022

NANOG™