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
What is a Traffic Generator?

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

- Stateful connections
- Application payloads

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?

**Build Product**
- Control
- ✓ Quality
- ✓ Specs
- ✓ Conformance

**Deploy Network**
- Validate
- ✓ Components
- ❖ SLOs
- ❏ Design

**Operate System**
- Maintain
- ❖ Interoperability
- ❖ RCAs
- ❏ Availability

- Proprietary CI
- H/W Certification Lab
- Ad-hoc Labs

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
1. **Define** objects to generate

2. **Run** a scenario

3. **Analyze** measurements

API Surface

Network or Device under Test

Generated Network & Hosts

Test Port

Test Port

Test Port

Generated Network & Hosts

Generated Network & Hosts
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/](https://otg.dev/model/)
OTG Configuration Elements

Visualization of OTG example configuration

Actual source: OpenAPI YAML/JSON/protobuf

https://otg.dev/model/
## 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](https://otg.dev/implementations)
How to use OTG API

19-OCT-2022
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
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
```
Test program: gosnappi

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

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

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

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

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

// start transmitting configured flows
res, err = api.SetTransmitState(ts)
checkResponse(res, err)

// 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 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

https://github.com/openconfig/kne
Would the latest NOS work?

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

<table>
<thead>
<tr>
<th>Service objectives</th>
<th>any @ outside</th>
<th>1.2.3.4 UDP/53</th>
<th>Should pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>any @ outside</td>
<td>1.2.3.4 TCP/22</td>
<td>Should fail</td>
<td></td>
</tr>
</tbody>
</table>
CI Example

RTBH DDoS Mitigation Validation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Traffic State</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any &gt; Servers</td>
<td>Below threshold</td>
<td>Should pass</td>
</tr>
<tr>
<td>Any &gt; Servers</td>
<td>Above threshold</td>
<td>Should be blocked</td>
</tr>
<tr>
<td>Servers &gt; Any</td>
<td>N/A</td>
<td>Should pass</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Event</th>
<th>Status</th>
<th>Branch</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed v7.4 changes with RFC 8212 on pe-router</td>
<td></td>
<td>clab-rtbh-rfc8212</td>
<td>7 minutes ago 3m 54s</td>
</tr>
<tr>
<td>pe-router v7.4.0</td>
<td></td>
<td>clab-rtbh-rfc8212</td>
<td>13 minutes ago 3m 13s</td>
</tr>
<tr>
<td>pe-router v7.3.1</td>
<td></td>
<td>clab-rtbh-rfc8212</td>
<td>15 minutes ago 3m 5s</td>
</tr>
<tr>
<td>pe-router v7.3.0</td>
<td></td>
<td>clab-rtbh-rfc8212</td>
<td>17 minutes ago 3m 8s</td>
</tr>
<tr>
<td>References</td>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Traffic Generator</td>
<td><a href="https://otg.dev">https://otg.dev</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ixia-c</strong> engine free version</td>
<td><a href="https://ixia-c.dev">https://ixia-c.dev</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slack channel for support</td>
<td><a href="https://otg.dev/#community">https://otg.dev/#community</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatible engines</td>
<td><a href="https://otg.dev/implementations/">https://otg.dev/implementations/</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>