Agenda

• Overview
• Stateless
• Advance Stateful
TREx – Results

• Open Source

• Cisco Customers
TRex Usage Analytics monthly report (*)

(*) ~1200 distinct returning users,

(**) Users are distinct
TReX models of operation

- L7, DUT terminate TCP/SSL, flow based
- DUT inspect L7. does not change TCP. Flow based
- DUT L2/L3 Switch , packet based
What Problem is Being Solved?

• Networks include complex L4–7 features, such as
  • Load Balancer, DPI/AVC, Firewall, NAT

• Requires testing with stateful and realistic traffic mix
What Problem is Being Solved?

- Traffic generators for realistic traffic are
  - Expensive $\sim$100-500K
  - Not scalable for high rates
  - Not flexible

- Implication
  - Limited and late testing
  - Different benchmarks and test methodologies
  - Real life bottlenecks and design issues
What is TRex?

• Linux user-space application uses DPDK library
• **Stateless**: Stream based uses Scapy
• **Stateful**: flow based
  Generates, manipulates and amplifies based on templates of real, captured flows (W/O TCP stack)
• **High performance**: up to 200 Gb/sec
• **Low cost**: Standard server hardware
• Flexible and Open Software
• Virtualization
• Easy installation and deployment
Stateless
Stateless High level functionality

- High scale – ~10M-35MPPS/core
- Profile can support multiple streams, scalable to 20K parallel streams
- Interactive support – GUI/TUI
- Statistic per port/ stream (e.g. latency/ Jitter)
- Python automation support
- Multi-user support
- Capture to Wireshark
- Scalable services using plugins (e.g. DHCP, IPv6)
Traffic Profile Example

**Stream 1**
IP/TCP
Continuous 1 KPPS
10.0.0.1 – 10.0.0.255

**Stream 2**
IP/UDP
Burst 2 KPPS, 3 packets
16.0.0.1 – 10.0.0.255

**Stream 3**
MPLS/IP/GRE/VXLAN
Multi-burst 2 KPPS, 3 packets
200.0.0.1 – 200.0.0.255
Enabled by Stream 2

Stream 2 triggers Stream 3
Stream 3 is Multi-burst

Inter-stream gap (ISG)
Inter-burst gap (IBG)

milliseconds
Control plane High level

TReX Client segment

- CLI
- API
- GUI Application

TReX Machine (TReX process)

- Control Plane
- Data Plane
- TReX Magic

- TReX Console
- TReX Stateless Client Core module
- TReX HLTAPI

Any language with JSON support and ZMQ connectivity

JSON RPC 2.0 (over ZMQ) Command-Response

JSON RPC 2.0 (over ZMQ) Command-Response
One stream with two directions

```python
def get_streams(self, direction = 0, **kwargs):
    # create 1 stream
    if direction==0:
        src_ip="16.0.0.1"
        dst_ip="48.0.0.1"
    else:
        src_ip="48.0.0.1"
        dst_ip="16.0.0.1"

    pkt = STLPktBuilder(
        pkt = Ether()/IP(src=src_ip,dst=dst_ip)/
        UDP(dport=12,sport=1025)/(10*'x') )

    return [ STLStream( packet = pkt,mode = STLTXCont() ) ]
```
Python Automation example

c = STLClient(username = "itya", server = "10.0.0.10", verbose_level = LoggerApi.VERBOSE_HIGH)
	ry:
   # connect to server
   c.connect()

   # prepare our ports (my machine has 0 <-> 1 with static route)
   c.reset(ports = [0, 1])

   # add both streams to ports
   c.add_streams(s1, ports = [0])

   # clear the stats before injecting
   c.clear_stats()

   c.start(ports = [0, 1], mult = "5mpps", duration = 10)

   # block until done
   c.wait_on_traffic(ports = [0, 1])

   # check for any warnings
   if c.get_warnings():
      # handle warnings here
      pass

finally:
   c.disconnect()
Performance XL710 MPPS/Core

XL710 profile

- cache
- normal

Packet size

MPPS/Core

- Lmix
- 1514
- 590
- 128
- 64
Advanced Stateful
User space TCP stack

Device Under Test (DUT)
Router

Client

Compress/
uncompress

Server

Trex

A

C

D

Trex

18
TRex ASTF features

- High scale
- TCP is the core component
  - Can be tuned MSS/initwnd/delay-ack
- TCP is based on BSD with acceleration
- Interactive
- Accurate latency measurement – usec
- Simulation of latency/jitter/drop in high rate
- OpenSSL integration
- L7 emulation layer
  - Emulate application using “micro-instructions”
  - Field engine
L7 Emulation layer

Client

send(request)
wait_for_response(len<=1000)
delay(random(min=100,max=1000)) // in usec
send(request2)
wait_for_response(len<=2000)
close()

Server side

wait_for_request(len<=100)
send_response(response)
wait_for_request(len<=200)
send_response(response)
wait_for_peer_close()
from trex_astf_lib.api import *


class Prof1():
    def __init__(self):
        pass

    def get_profile(self):
        ip_gen = ASTFIPGenDist(ip_range=["16.0.0.0", "16.0.0.255"],
                               distribution="seq")

        ip_gen_s = ASTFIPGenDist(ip_range=["48.0.0.0", "48.0.255.255"],
                               distribution="seq")

        ip_gen = ASTFIPGen(glob=ASTFIPGenGlobal(ip_offset="1.0.0.0"),
                          dist_client=ip_gen_c,
                          dist_server=ip_gen_s)

        return ASTFProfile(default_ip_gen=ip_gen,
                            cap_list=[ASTFCapInfo(
                                         file="../avl/delay_10_http_browsing_0.pcap",
                                         cps=1)]
                            )

    def register():
        return Prof1()
Client side pseudo code

```plaintext
template = choose_template()

src_ip, dest_ip, src_port = generate from pool of client
dest_port = template.get_dest_port()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((dest_ip, dest_port))

# program
s.write(template.request)  # write the following taken from the poop file
    # GET /3394 HTTP/1.1
    # Host: 22.0.0.3
    # Connection: Keep-Alive
    # User-Agent: Mozilla/4.0
    # Accept: */*
    # Accept-Language: en-us
    # Accept-Encoding: gzip, deflate, compress

s.read(template.request_size)  # wait for 32K bytes and compare some of it
    # HTTP/1.1 200 OK
    # Server: Microsoft-IIS/6.0
    # Content-Type: text/html
    # Content-Length: 32000
    # body ..

s.close();
```
Server side pseudo code

```python
# if this is SYN for flow that already exist, let TCP handle it

if (flow_table_lookup(pkt) == False):
    # first SYN in the right direction with no flow
    compare (pkt.src_ip/dst_ip to the generator ranges) # check that it is in the range or valid server IP (src_ip,dst_ip)
    template = lookup_template(pkt.dest_port) # get template for the dest_port

    # create a socket for TCP server
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # bind to the port
    s.bind((pkt.dst_ip, pkt.dst_port))
    s.listen(1)

    # program of the template
    s.read(template.request_size) # just wait for x bytes, don't check them

    # GET /3384 HTTP/1.1
    # Host: 22.0.0.3
    # Connection: Keep-Alive
    # User-Agent: Mozilla/4.0 ..
    # Accept: */*
    # Accept-Language: en-us
    # Accept-Encoding: gzip, deflate, compress

    s.write(template.response) # just wait for x bytes, don't check them (TCP check the seq and checksum)

    #HTTP/1.1 200 OK
    #Server: Microsoft-IIS/6.0
    #Content-Type: text/html
    #Content-Length: 30000
    # body ..

    s.close()
```
Profile with two template

class Prof1():
    def __init__(self):
        pass

    def get_profile(self):
        # ip generator
        ip_gen_c = ASTFIPGenDist(ip_range=["16.0.0.0", "16.0.255"],
                                distribution="seq")
        ip_gen_g = ASTFIPGenDist(ip_range=["48.0.0.0", "48.0.255.255"],
                                distribution="seq")
        ip_gen = ASTFIPGen(glob=ASTFIPGenGlobal(ip_offset="1.0.0.0"),
                           dist_client=ip_gen_c,
                           dist_server=ip_gen_g)

        return ASTFProfile(default_ip_gen=ip_gen,
                            cap_list=[
                                ASTFCapInfo(file="../avl/delay_10_http_browsing_0.pcap",
                                           cps=1),
                                ASTFCapInfo(file="../avl/delay_10_https_0.pcap",
                                           cps=2)
                            ])

    def register():
        return Prof1()
<table>
<thead>
<tr>
<th>Statistic</th>
<th>client</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_active_flows</td>
<td>39965</td>
<td>39966</td>
</tr>
<tr>
<td>m_est_flows</td>
<td>39950</td>
<td>39951</td>
</tr>
<tr>
<td>m_tx_bw_17_r</td>
<td>31.14 Mpps</td>
<td>4.09 Mpps</td>
</tr>
<tr>
<td>m_rx_bw_17_r</td>
<td>4.09 Mpps</td>
<td>31.14 Mpps</td>
</tr>
<tr>
<td>m_tx_pps_r</td>
<td>140.36 Kpps</td>
<td>124.82 Kpps</td>
</tr>
<tr>
<td>m_rx_pps_r</td>
<td>156.05 Kpps</td>
<td>155.87 Kpps</td>
</tr>
<tr>
<td>m_avg_size</td>
<td>1.74 KB</td>
<td>1.84 KB</td>
</tr>
<tr>
<td>TCP</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>tcp_connattempt</td>
<td>73916</td>
<td>0</td>
</tr>
<tr>
<td>tcp_synsent</td>
<td>73911</td>
<td>73910</td>
</tr>
<tr>
<td>tcp_synrecv</td>
<td>33971</td>
<td>33958</td>
</tr>
<tr>
<td>tcp_seqsent</td>
<td>213491</td>
<td>554085</td>
</tr>
<tr>
<td>tcp_rttupdated</td>
<td>213491</td>
<td>549716</td>
</tr>
<tr>
<td>tcp_synack</td>
<td>344142</td>
<td>0</td>
</tr>
<tr>
<td>tcp_smtotal</td>
<td>633750</td>
<td>554085</td>
</tr>
<tr>
<td>tcp_sndbytes</td>
<td>14066329</td>
<td>2770789136</td>
</tr>
<tr>
<td>tcp_sndctrl</td>
<td>73916</td>
<td>0</td>
</tr>
<tr>
<td>tcp_sndack</td>
<td>475923</td>
<td>1395216</td>
</tr>
<tr>
<td>tcp_rcvpackets</td>
<td>550465</td>
<td>139502</td>
</tr>
<tr>
<td>tcp_rcvbytes</td>
<td>2269441776</td>
<td>18401530</td>
</tr>
<tr>
<td>tcp_rcvackpackets</td>
<td>1339495</td>
<td>549716</td>
</tr>
<tr>
<td>tcp_rcvackbytes</td>
<td>256886792</td>
<td>2222057965</td>
</tr>
<tr>
<td>tcp_pread</td>
<td>410970</td>
<td>0</td>
</tr>
<tr>
<td>tcp_rcvpackets</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>redirect_rx_ok</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Under the hood
TCP stack – Flow Scale -TX

TX Queue 32K – Sliding window

10M flows
320GByte
320M mbuf =10GB

0.01GB
TCP stack – Flow Scale issue – RX

RX Reassembly Queue 32K – Sliding window

10M flows
320GByte
3-50GB for 1% drop rate
0.01GB
TCP stack – Delay/Jitter/Drop simulation

App

Events

User Space TCP

mbuf

DPDK

Simulation

Queue/Jitter/Drop

100MPPS * 100msec = 10 MPPS in Queue

100 MPPS * 100 msec = 10 * 16 MPPS = 0.16 GB

App

User Space TCP

Events

12B Descriptors

Queue/Jitter/Drop

mbuf

DPDK
TRex vs NGINX

Client
1 DP core

Server
2 sockets / 16 cores total

Tune Linux TCP

X710 10Gb/sec NIC

82559 10Gb/sec NIC
Performance setup #2

Client
1 DP core

Server
1 DP core

XL710 40Gb/sec NIC

XL710 40Gb/sec NIC
## Performance numbers

### TRex one DP core

<table>
<thead>
<tr>
<th>m</th>
<th>CPU (1DP)</th>
<th>cps</th>
<th>rps</th>
<th>rx (mb/sec)</th>
<th>pps (tx+rx)</th>
<th>active flows</th>
<th>drop</th>
<th>Memory TCP/MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.6</td>
<td>1000</td>
<td>1000</td>
<td>265</td>
<td>34444</td>
<td>960</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>5000</td>
<td>2.7</td>
<td>5000</td>
<td>5000</td>
<td>1320</td>
<td>172222</td>
<td>3000</td>
<td>0</td>
<td>1.4</td>
</tr>
<tr>
<td>10000</td>
<td>5.7</td>
<td>10000</td>
<td>10000</td>
<td>2210</td>
<td>344444</td>
<td>6000</td>
<td>0</td>
<td>2.8</td>
</tr>
<tr>
<td>20000</td>
<td>13.5</td>
<td>20000</td>
<td>20000</td>
<td>5410</td>
<td>660969</td>
<td>12000</td>
<td>0</td>
<td>5.7</td>
</tr>
<tr>
<td>50000</td>
<td>40.0</td>
<td>50000</td>
<td>50000</td>
<td>13460</td>
<td>172222</td>
<td>29070</td>
<td>0</td>
<td>14.2</td>
</tr>
<tr>
<td>67500</td>
<td>79.0</td>
<td>87500</td>
<td>87500</td>
<td>23670</td>
<td>301389</td>
<td>55229</td>
<td>0</td>
<td>26.4</td>
</tr>
<tr>
<td>50000</td>
<td>66.1</td>
<td>90000</td>
<td>90000</td>
<td>24271</td>
<td>310000</td>
<td>56217</td>
<td>0.10</td>
<td>26.6</td>
</tr>
</tbody>
</table>

Figure 3. TRex with 1 DP core

### Linux 16 cores

<table>
<thead>
<tr>
<th>m</th>
<th>cps</th>
<th>rps</th>
<th>rx (mb/sec)</th>
<th>active flows</th>
<th>16xCPU</th>
<th>drop</th>
<th>Kernel memory SLAB (MB)</th>
<th>Total Memory used (free-h) MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>265</td>
<td>600</td>
<td>no</td>
<td>21000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>1320</td>
<td>3000</td>
<td>no</td>
<td>22000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>2210</td>
<td>6000</td>
<td>no</td>
<td>29000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>20000</td>
<td>20000</td>
<td>5410</td>
<td>12000</td>
<td>no</td>
<td>26%/25% 8x cores IRQ break</td>
<td>yes</td>
<td>800</td>
</tr>
<tr>
<td>50000</td>
<td>50000</td>
<td>50000</td>
<td>13480</td>
<td>29870</td>
<td>no</td>
<td>33600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87500</td>
<td>87500</td>
<td>87500</td>
<td>23670</td>
<td>55329</td>
<td>no</td>
<td>44500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90000</td>
<td>90000</td>
<td>90000</td>
<td>24271</td>
<td>56217</td>
<td>no</td>
<td>56470</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. NGINX 16 cores

x80 faster
x2000 less memory