cisco

TRex

Realistic Traffic Generator

Realistic Traffic Generator

Hanoch Haim – Principal Engineer

FOSDEM

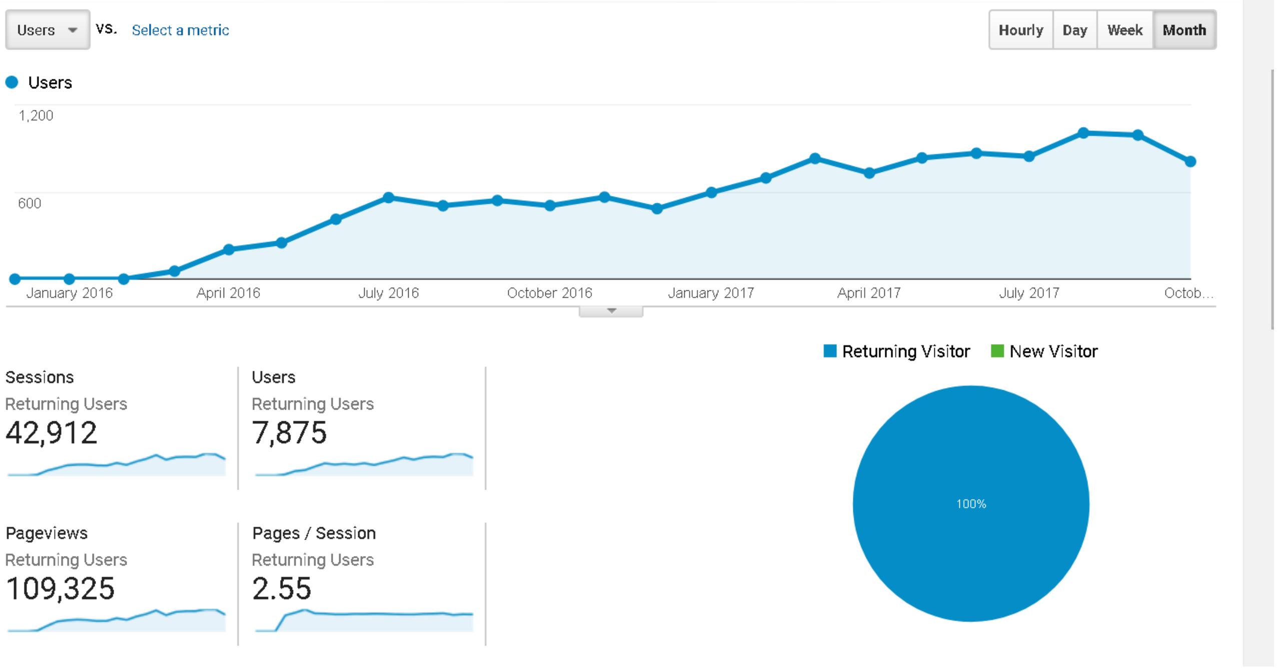
Agenda

- 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

Advance Stateful
(ASTF)

Stateful
(STF)

Stateless
(STL)

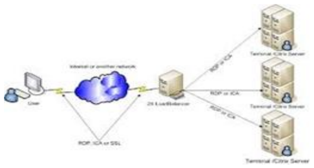
- L7, DUT terminate TCP/SSL, flow based

- DUT inspect L7. does not change TCP. Flow based

- DUT L2/L3 Switch , packet based

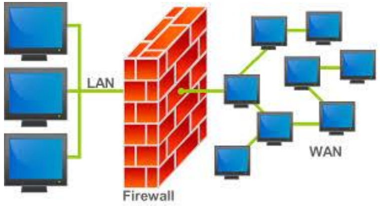
What Problem is Being Solved?

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



LB

DPI/AVC



Firewall, NAT

- Requires testing with stateful and realistic traffic mix

What Problem is Being Solved?

- Traffic generators for realistic traffic are
  - Expensive ~$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

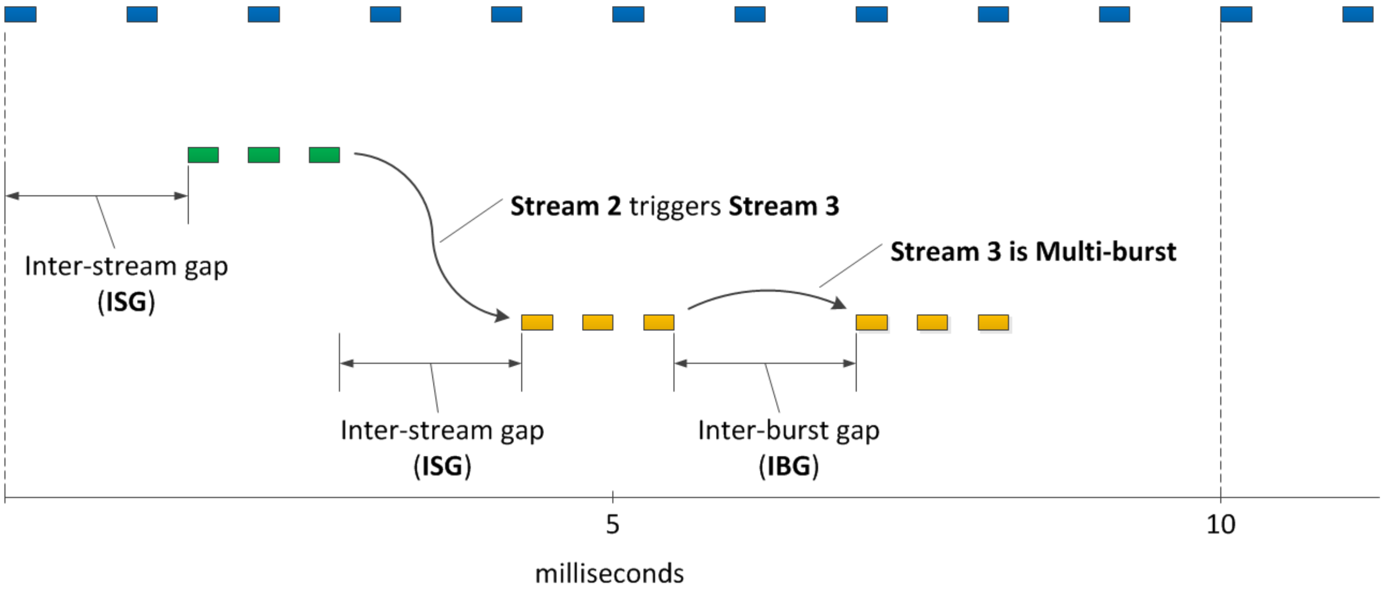
- High scale – ~10M-35MPPS/core
- Profile can support multiple streams, scalable to 20K parallel streams
- Interactive support – GUI/TUI
- Statistic per port/ stream (e.g. latency/ Jitter)
- Python automation support
- Multi-user support
- Capture to Wireshark
- Scalable services using plugins (e.g.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

Traffic Profile Example

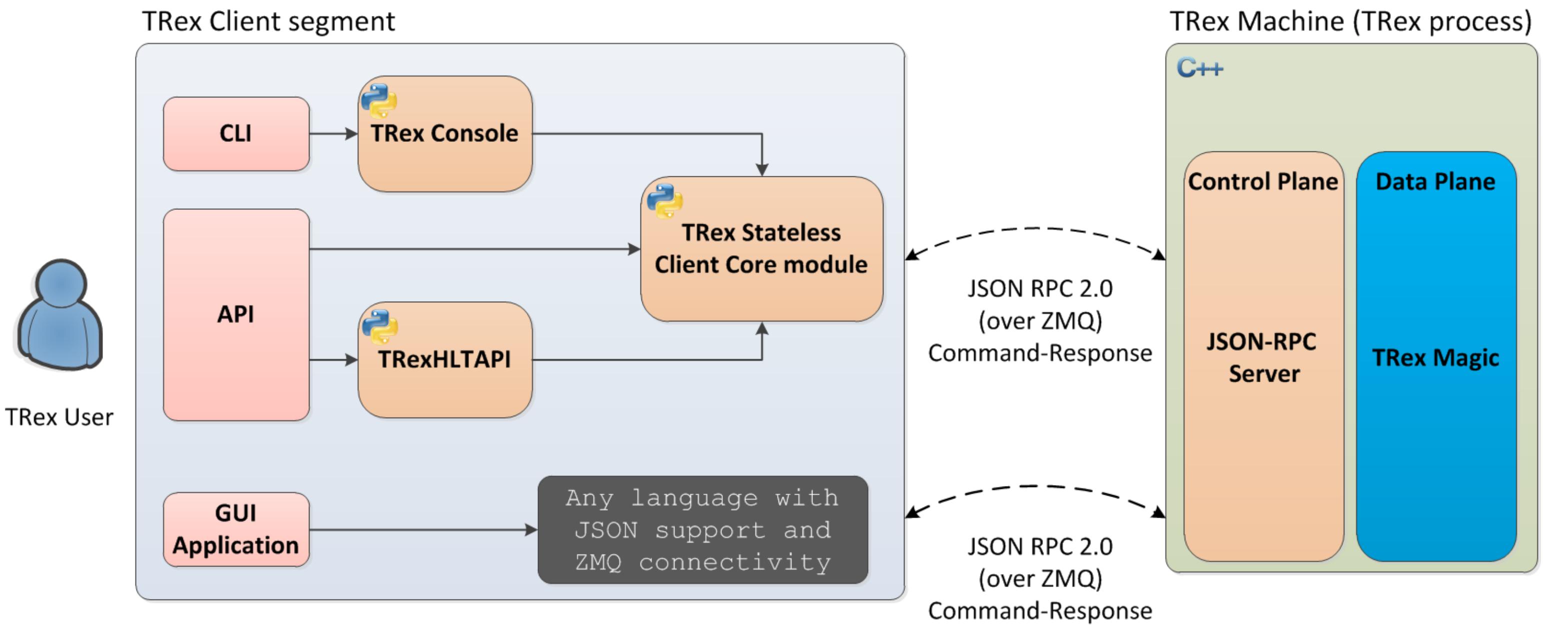
Stream 1
IP/TCP
Continuous 1 KPPS
10.0.0.1 – 10.0.0.255

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



Control plane High level



One stream with two directions

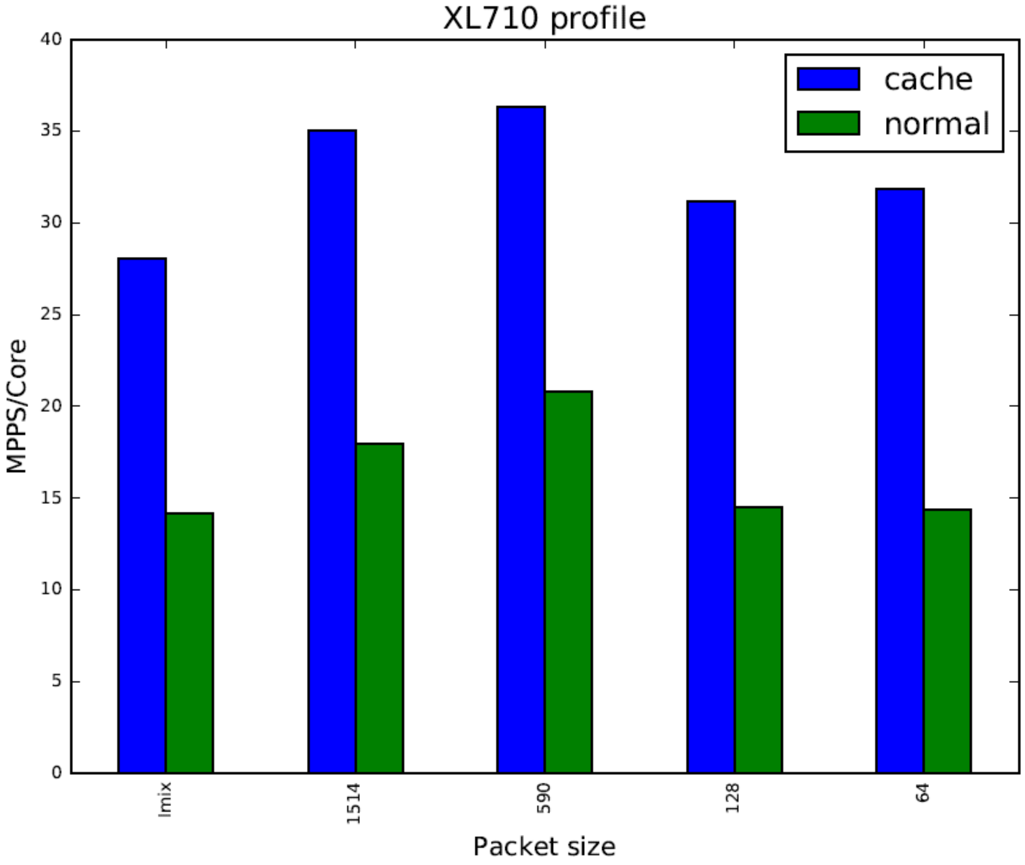
```
def get_streams (self, direction = 0, **kwargs):
    # create 1 stream
    if direction==0:
        src_ip="16.0.0.1"
        dst_ip="48.0.0.1"
    else:
        src_ip="48.0.0.1"
        dst_ip="16.0.0.1"

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

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

❶

Python Automation example

```python
c = STLClient(username = "itay",server = "10.0.0.10", verbose_level = LoggerApi.VERBOSE_HIGH)

try:
    # 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()
```

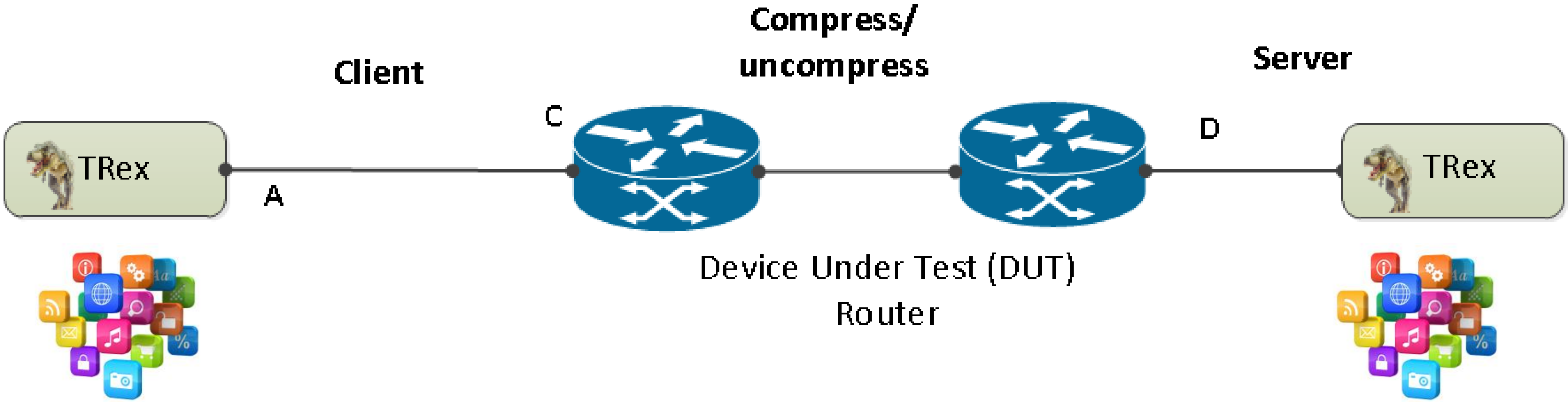
Performance XL710 MPPS/Core



link

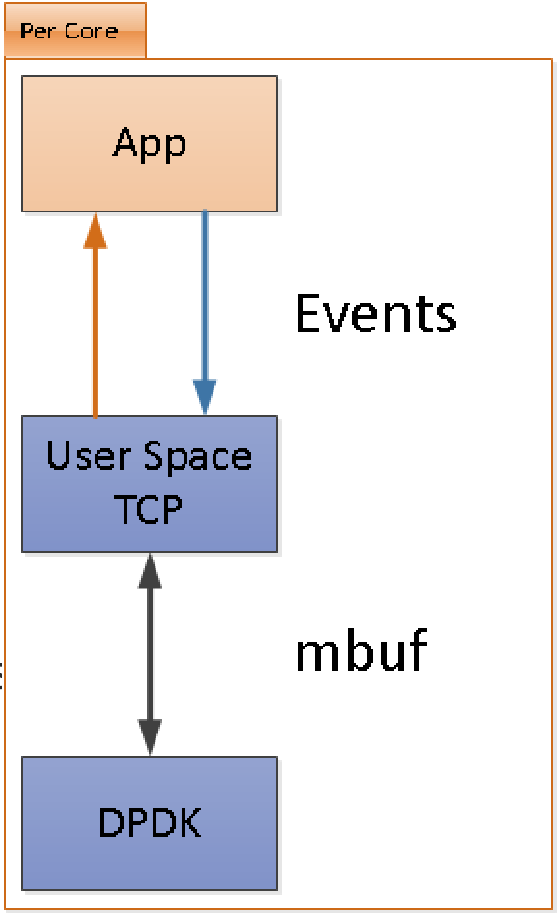
Advanced Stateful

User space TCP stack



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()
```

HTTP simple profile

```
from trex_astf_lib.api import *


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

    def get_profile(self):
        # ip generator
        ip_gen_c = 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"),          ❶
                           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)
                                    ])                                         ❷


def register():
    return Prof1()
```

Client side pseudo code

Client side pseudo code

```
template = choose_template()                                  ❶

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

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)         ❷

s.connect(dest_ip,dst_port)                                   ❸

# program                                                     ❹
s.write(template.request)      #write the following taken from the pcap file

                               # 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.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

```
# 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,dest_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)                ❶

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

    s.listen(1)

    #program of the template                                              ❷
    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: 32000
                                      # body ..

    s.close()
```

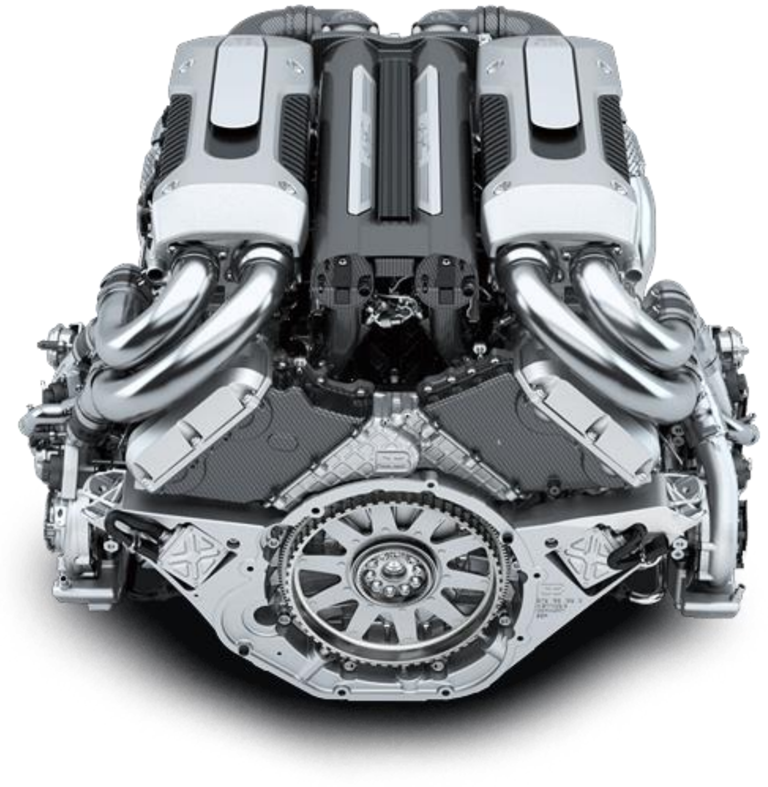
Profile with two template

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

        def get_profile(self):
        # ip generator
        ip_gen_c = ASTFIPGenDist(ip_range=["16.0.0.0", "16.0.0.255"],
                                 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),      ❶
                             ASTFCapInfo(file="avl/delay_10_https_0.pcap",
                                         cps=2)       ❷
                                    ])


def register():
    return Prof1()
```

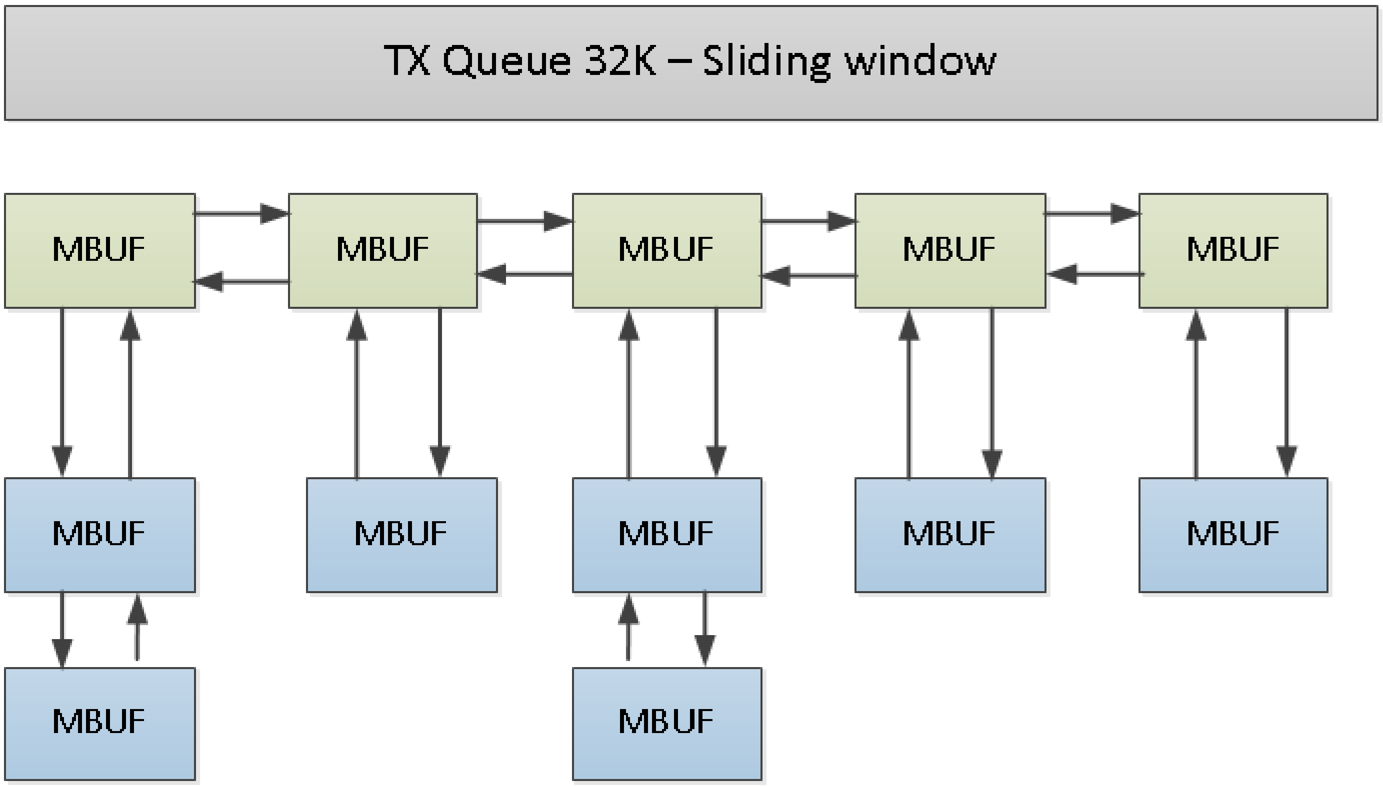
Statistic

```
                 |      client   |        server  |
 --------------------------------------------------------------------------------
  m_active_flows |      39965    |       39966    |  active flows
     m_est_flows |      39950    |       39952    |  active est flows
    m_tx_bw_l7_r | 31.14 Mbps    |   4.09 Gbps    |  tx bw
    m_rx_bw_l7_r |  4.09 Gbps    |  31.14 Mbps    |  rx bw
      m_tx_pps_r |140.36 Kpps    | 124.82 Kpps    |  tx pps
      m_rx_pps_r |156.05 Kpps    | 155.87 Kpps    |  rx pps
      m_avg_size |    1.74 KB    |    1.84 KB     |  average pkt size
               - |        ---    |         ---    |
             TCP |        ---    |         ---    |
               - |        ---    |         ---    |
tcps_connattempt |      73936    |           0    |  connections initiated
    tcps_accepts |          0    |       73924    |  connections accepted
   tcps_connects |      73921    |       73910    |  connections established
     tcps_closed |      33971    |       33958    |  conn. closed (includes drops)
  tcps_segstimed |     213451    |      558085    |  segs where we tried to get rtt
 tcps_rttupdated |     213416    |      549736    |  times we succeeded
     tcps_delack |     344742    |           0    |  delayed acks sent
   tcps_sndtotal |     623780    |      558085    |  total packets sent
    tcps_sndpack |      73921    |      418569    |  data packets sent
    tcps_sndbyte |   18406329    |  2270136936    |  data bytes sent
    tcps_sndctrl |      73936    |           0    |  control (SYN,FIN,RST) packets sent
   tcps_sndacks  |     475923    |      139516    |  ack-only packets sent
    tcps_rcvpack |     550465    |      139502    |  packets received in sequence
    tcps_rcvbyte | 2269941776    |    18403590    |  bytes received in sequence
 tcps_rcvackpack |     139495    |      549736    |  rcvd ack packets
 tcps_rcvackbyte |   18468679    |  2222057965    |  tx bytes acked by rcvd acks
    tcps_preddat |     410970    |           0    |  times hdr predict ok for data pkts
  tcps_rcvoopack |          0    |           0    |  *out-of-order packets received   #1
               - |        ---    |         ---    |
      Flow Table |        ---    |         ---    |
               - |        ---    |         ---    |
  redirect_rx_ok |          0    |           1    |  redirect to rx OK
```

Under the hood



TCP stack – Flow Scale -TX



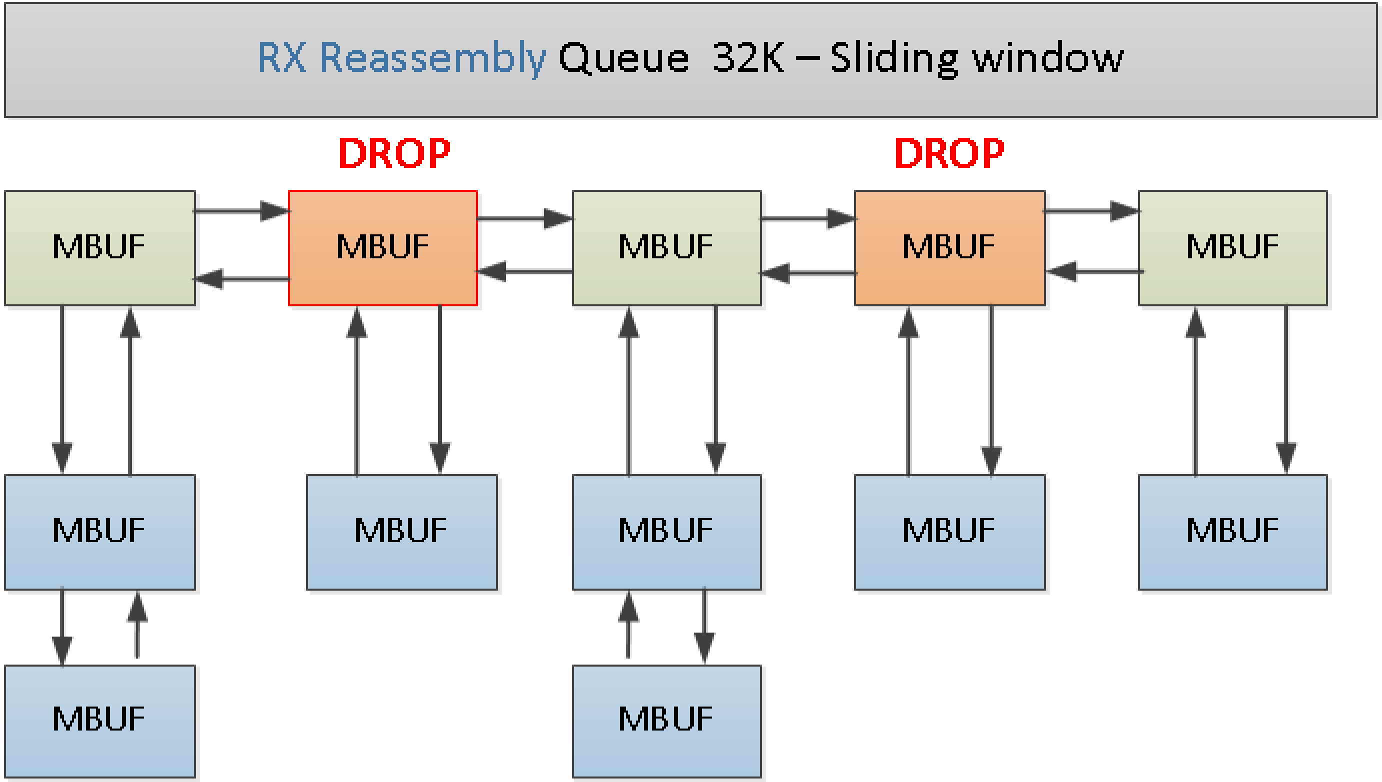
10M flows

320GByte

320M mbuf =10GB

0.01GB

TCP stack – Flow Scale issue – RX



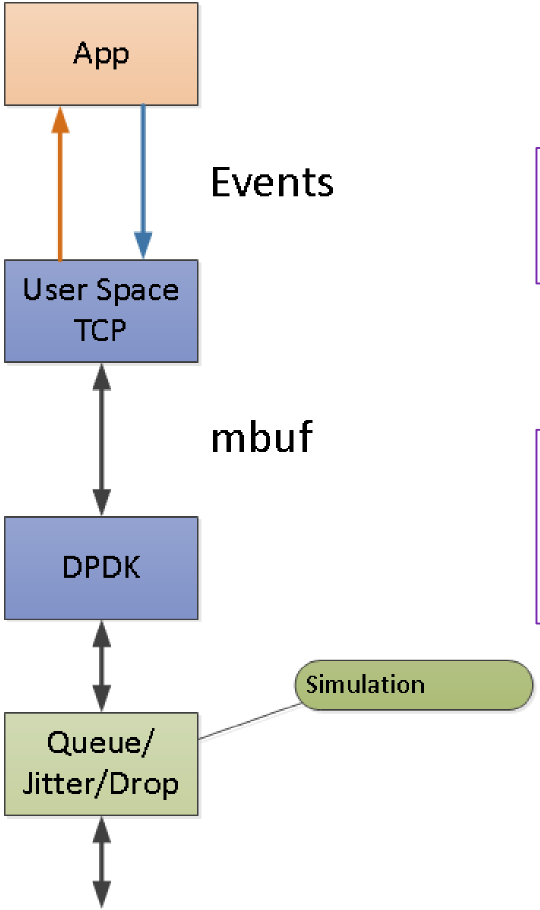
10M flows

320GByte

3-50GB for 1% drop rate

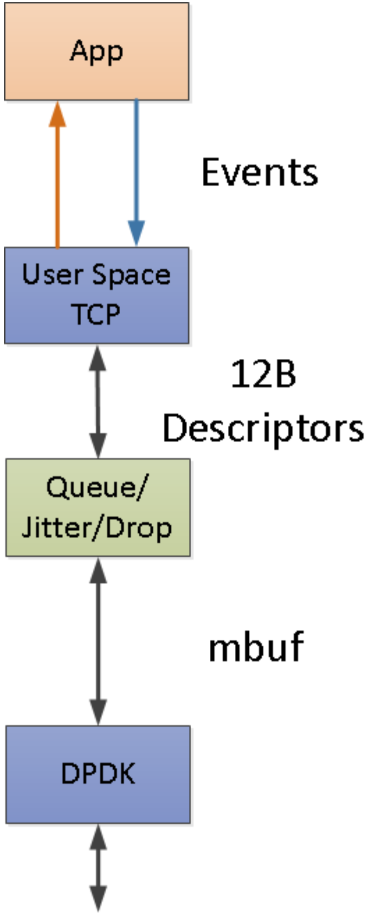
0.01GB

TCP stack – Delay/Jitter/Drop simulation

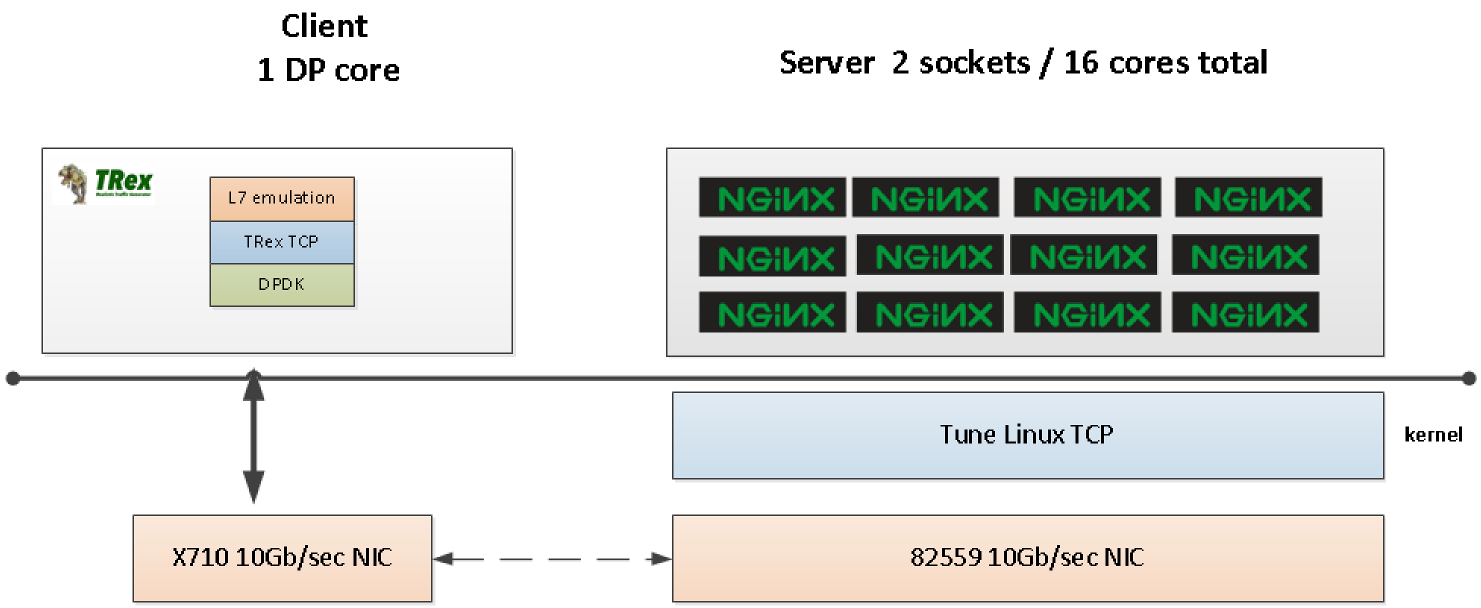


100MPPS *100msec =10MPPS in Queue

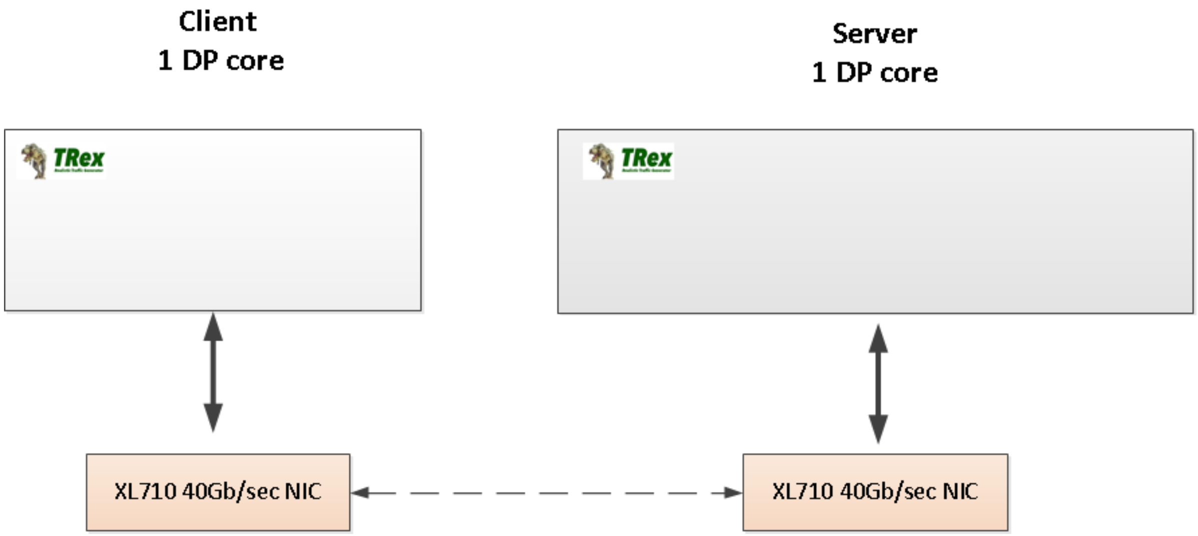
100MPPS *100msec =10*16MPPS=0.16GB



TRex vs NGINX



Performance setup #2



Performance numbers

m	TRex one DP core							
m	CPU (1DP) %%	cps	rps	rx (mb/sec)	pps(tx+rx)	active flows	drop	Memory TCP/MB
1000	0.6	1000	1000	265	34444	600	0	0.3
5000	2.7	5000	5000	1320	172222	3000	0	1.4
10000	5.7	10000	10000	2210	344444	6000	0	2.9
20000	13.5	20000	20000	5410	688889	12000	0	5.7
50000	40.8	50000	50000	13480	1722222	29870	0	14.2
87500	79.0	87500	87500	23670	3013889	55329	0	26.4
90000	86.1	90000	90000	24271	3100000	56217	0.10%	26.8

Figure 3. TRex with 1 DP core

					Linux 16 cores			
m	cps	rps	rx (mb/sec)	active flows	16xCPU	drop	Kernel memory SLAB (MB)	Total Memory used (free-h) MB
1000	1000	1000	265	600		no		21000
5000	5000	5000	1320	3000		no		22000
10000	10000	10000	2210	6000		no		25000
20000	20000	20000	5410	12000	20%/25% 8x cores IRQ break	yes	800	31000
50000	50000	50000	13480	29870				
87500	87500	87500	23670	55329				
90000	90000	90000	24271	56217				

Figure 4. NGINX 16 cores

x80 faster

x2000 less memory

https://trex-tgn.cisco.com/trex/doc/trex_astf_vs_nginx.html