



DEVNET

Welcome to Webinar with Cisco DevNet



Hanoch Haim
Principal Engineer
@cisco

@CiscoDevNet
#DevNet

About the Webinar

- Ask a question in the Q&A panel, send to All Panelists
- Join the WebEx Audio > Select Communicate > Join Audio
- For a WebEx call back > Click allow phone button at the bottom of participants side panel
- Recording will be sent after the webinar
- Please complete the post-event survey
- Join us for upcoming webinars: <http://bit.ly/DevNetWebinarWed>
- Join Cisco DevNet, go here: <http://developer.cisco.com/join>

About the Speaker and Panelist



Hanoch Haim

@cisco

Principal Engineer
Focus: DPI/ TRex
Cisco DevNet



Itay Marom

@cisco

Engineer
Focus: DPI/ TRex
Cisco DevNet

Agenda

- Overview
- Stateful
- Advanced Stateful
- Stateless



TRex –results



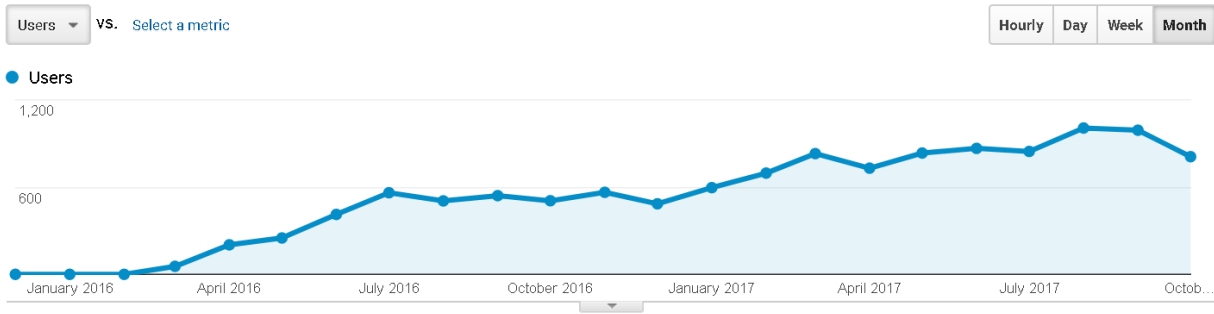
- Open Source



- Cisco Customers



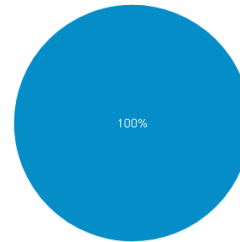
TRex Usage Analytics monthly report (*)



(*) ~1200 distinct **returning** users,

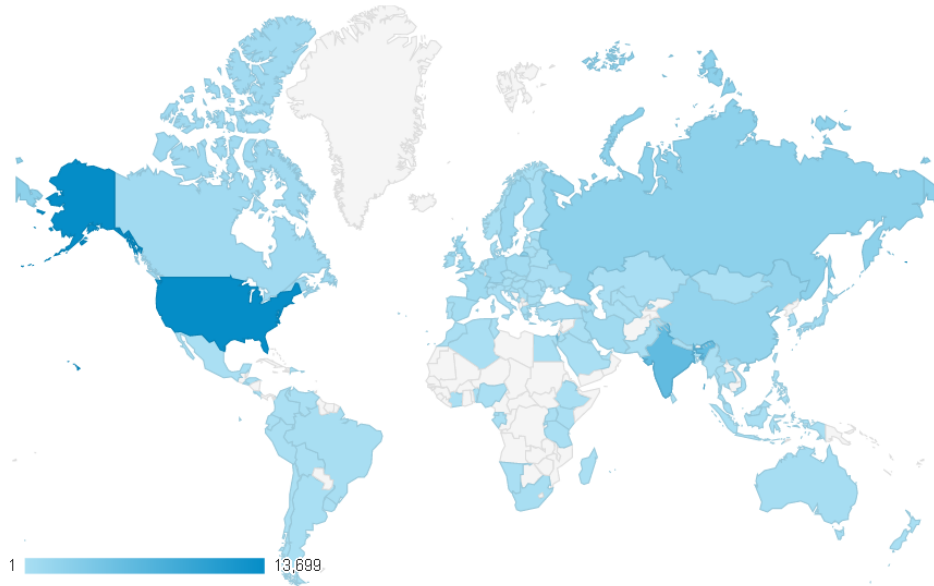


■ Returning Visitor ■ New Visitor



(**) Users are distinct

TRex Usage Analytics Location



Returning Users		42,912 % of Total: 60.37% (71,082)
1.	 United States	13,699 (31.92%)
2.	 India	6,061 (14.12%)
3.	 Israel	4,169 (9.72%)
4.	 Russia	2,408 (5.61%)
5.	 China	1,800 (4.19%)
6.	 Sweden	1,255 (2.92%)
7.	 United Kingdom	1,051 (2.45%)
8.	 Germany	938 (2.19%)
9.	 France	907 (2.11%)
0.	 Iran	818 (1.91%)

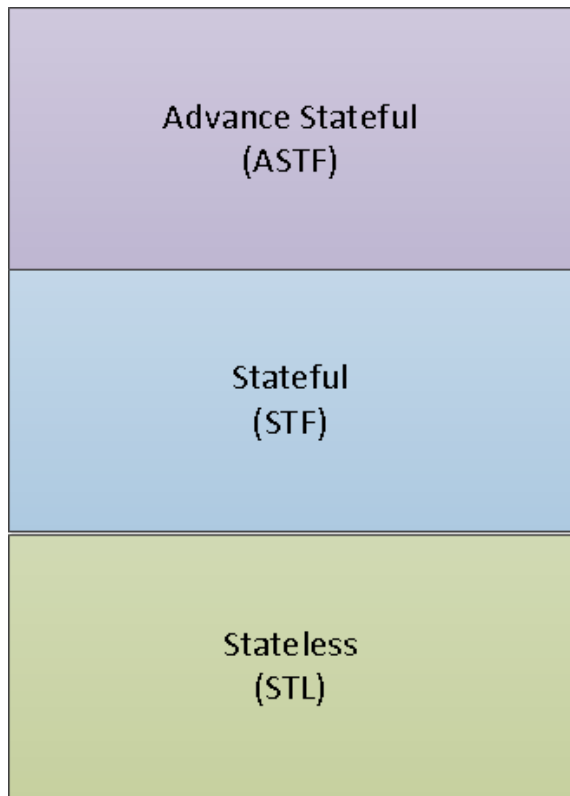
(*) Source is Google analytics of TRex documentation

(**) Users are distinct

Open and Free Bugatti Chiron for all

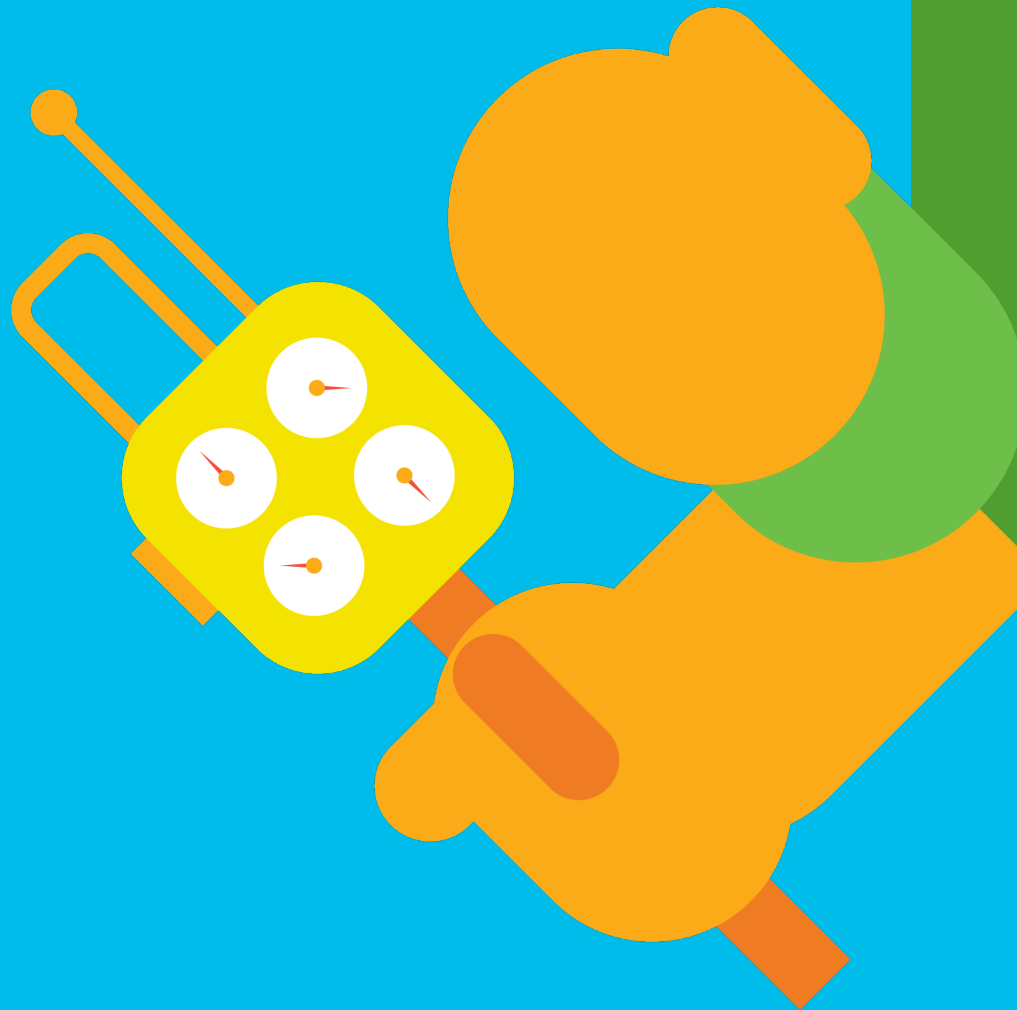


TRex models of operation



- L7, DUT terminate TCP/SSL, flow based
- DUT inspect L7. does not change TCP. Flow based
- DUT L2/L3 Switch , packet based

Stateful



Current Situation

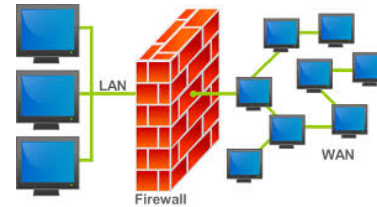
- Networks include complex L4-7 features, such as
 - Load Balancer, DPI/AVC, Firewall, NAT



LB



DPI/AVC



Firewall, NAT

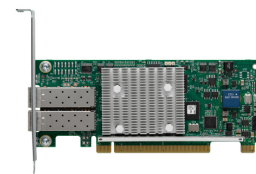
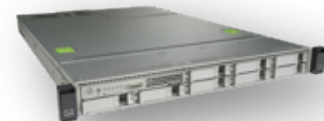
- Requires testing with stateful and realistic traffic mix

What Problem is Being Solved?

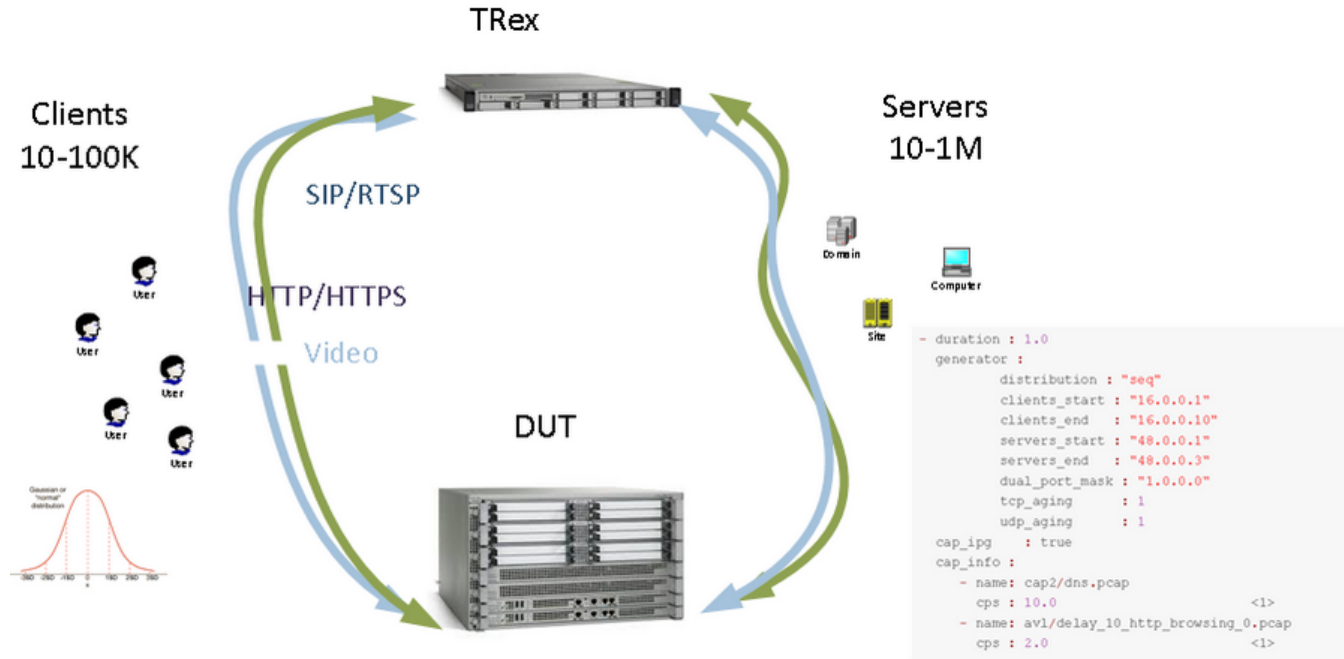
- Traffic generators for realistic traffic are
 - Expensive ~\$100-500K
 - Not scalable for high rates
 - Not flexible
- Implication
 - Limited and late testing
 - Different benchmarks and test methodologies
 - Real life bottlenecks and design issues

What is TRex?

- Stateful traffic generator – smart replay
- Generates, manipulates and amplifies based on templates of real, captured flows
- **High performance:** up to 200 Gb/sec
- **Low cost:** Standard server hardware
- Flexible and Open Software
- Virtualization
- Easy installation and deployment

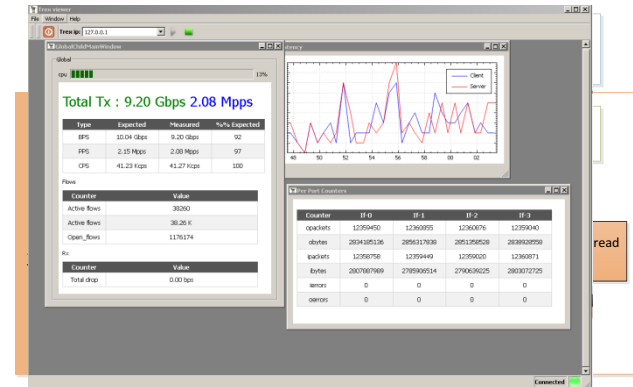


Stateful Traffic Generation Model



High level software architecture

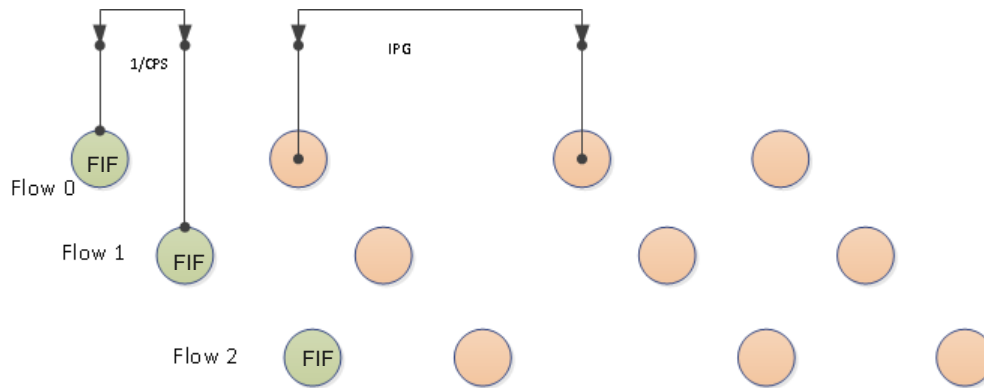
- **DPDK/Multi-Threaded**
 - Scales linearly
 - ~5MPPS/~20 Gb/sec per core
 - Supports 1/10/25/50/40/100 Gb
- **Flow-based**
 - Fast event scheduler
 - Generates flow templates
 - Can support 1K templates
 - Scales up to 100K clients, 1M servers
- **Flexible**
 - Client/server generation models
 - Measures jitter/latency/flow order
 - NAT translation/IPv6/Tunnels



- **User Interface**
 - Python API
 - Benchmark automation
 - GUI

Flow Generation

- Example of one flow with four packets



$$\text{Total PPS} = \sum_{k=0}^n \text{CPS}_k \times \text{flow_pkts}_k$$

$$\text{Total CPS} = \sum_{k=0}^n \text{CPS}_k$$

Concurrent flows

$$= \sum_{k=0}^n \text{CPS}_k \times \text{flow_duration}_k$$

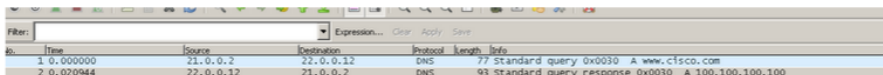
DNS simple profile example

```

$more cap2/dns_test.yaml
- duration : 10.0
  generator :
    distribution : "seq"
    clients_start : "16.0.0.1"
    clients_end : "16.0.0.255"
    servers_start : "48.0.0.1"
    servers_end : "48.0.0.255"
    dual_port_mask : "1.0.0.0"
    tcp_aging : 1
    udp_aging : 1
  cap_info :
    - name: cap2/dns.pcap          <1>
      cps : 1.0                    <2>

```

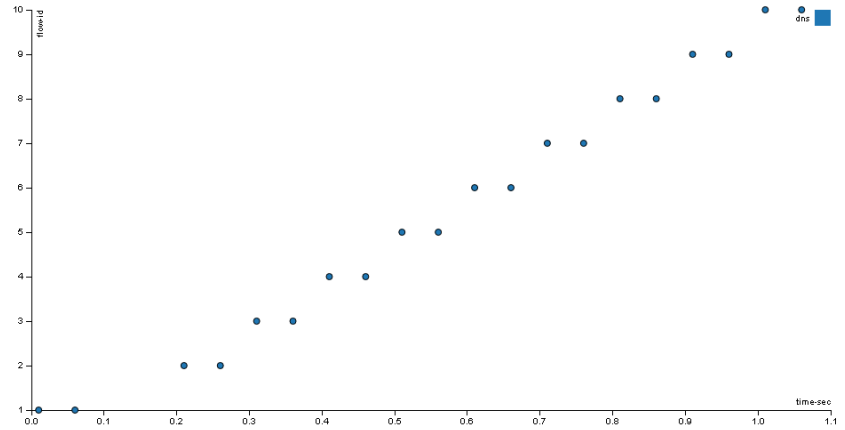
1. the pcap file that include DNS cap file that will be replicate
2. how many connection per second to generate, 1.0 means 1 connection per second



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	21.0.0.2	22.0.0.12	DNS	77	Standard query 0x0030 A www.cisco.com
2	0.020944	22.0.0.12	21.0.0.2	DNS	93	Standard query response 0x0030 A 100.100.100.100

DNS output

pkt	time sec	fid	flow-pkt-id	client_ip	client_port	server_ip	direction
1	0.010000	1	1	16.0.0.1	1024	48.0.0.1	→
2	0.020000	1	2	16.0.0.1	1024	48.0.0.1	←
3	2.010000	2	1	16.0.0.2	1024	48.0.0.2	→
4	2.020000	2	2	16.0.0.2	1024	48.0.0.2	←
5	3.010000	3	1	16.0.0.3	1024	48.0.0.3	→
6	3.020000	3	2	16.0.0.3	1024	48.0.0.3	←
7	4.010000	4	1	16.0.0.4	1024	48.0.0.4	→
8	4.020000	4	2	16.0.0.4	1024	48.0.0.4	←
9	5.010000	5	1	16.0.0.5	1024	48.0.0.5	→
10	5.020000	5	2	16.0.0.5	1024	48.0.0.5	←
11	6.010000	6	1	16.0.0.6	1024	48.0.0.6	→



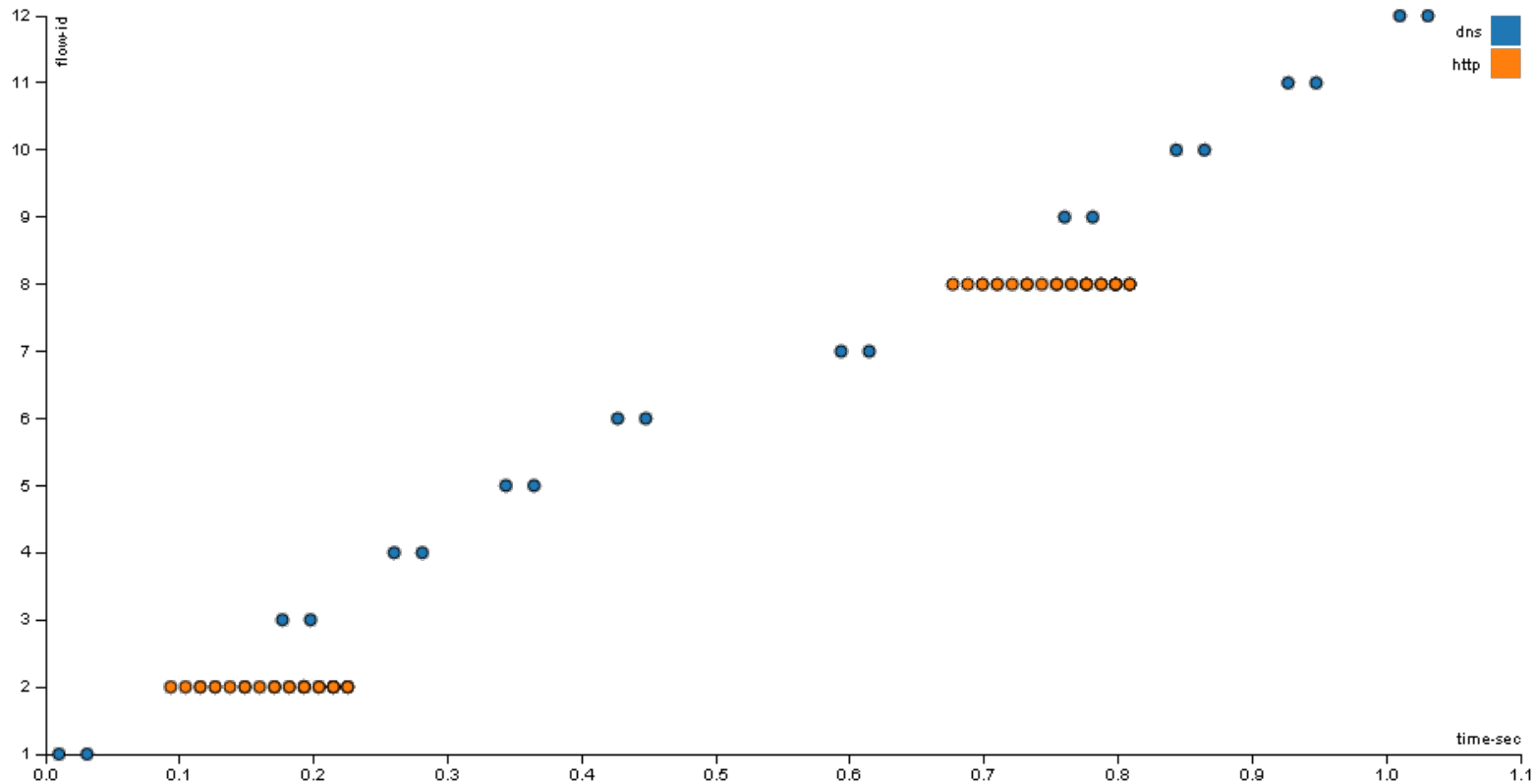
HTTP & DNS

```

- duration : 1.0
  generator :
    distribution : "seq"
    clients_start : "16.0.0.1"
    clients_end : "16.0.0.10"
    servers_start : "48.0.0.1"
    servers_end : "48.0.0.3"
    dual_port_mask : "1.0.0.0"
    tcp_aging : 1
    udp_aging : 1
  cap_ipg : true
  cap_info :
    - name: cap2/dns.pcap
      cps : 10.0 <1>
    - name: avl/delay_10_http_browsing_0.pcap
      cps : 2.0 <1>

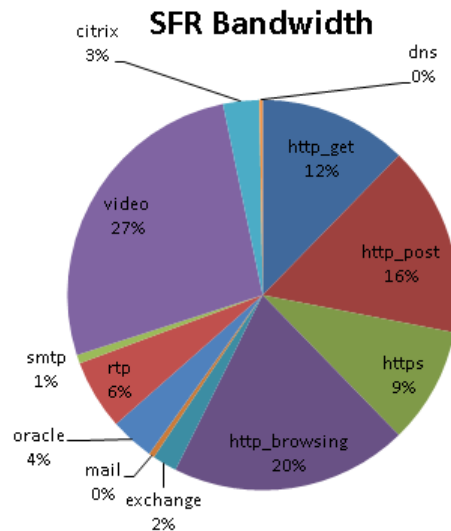
```

HTTP & DNS



Enterprise traffic profile

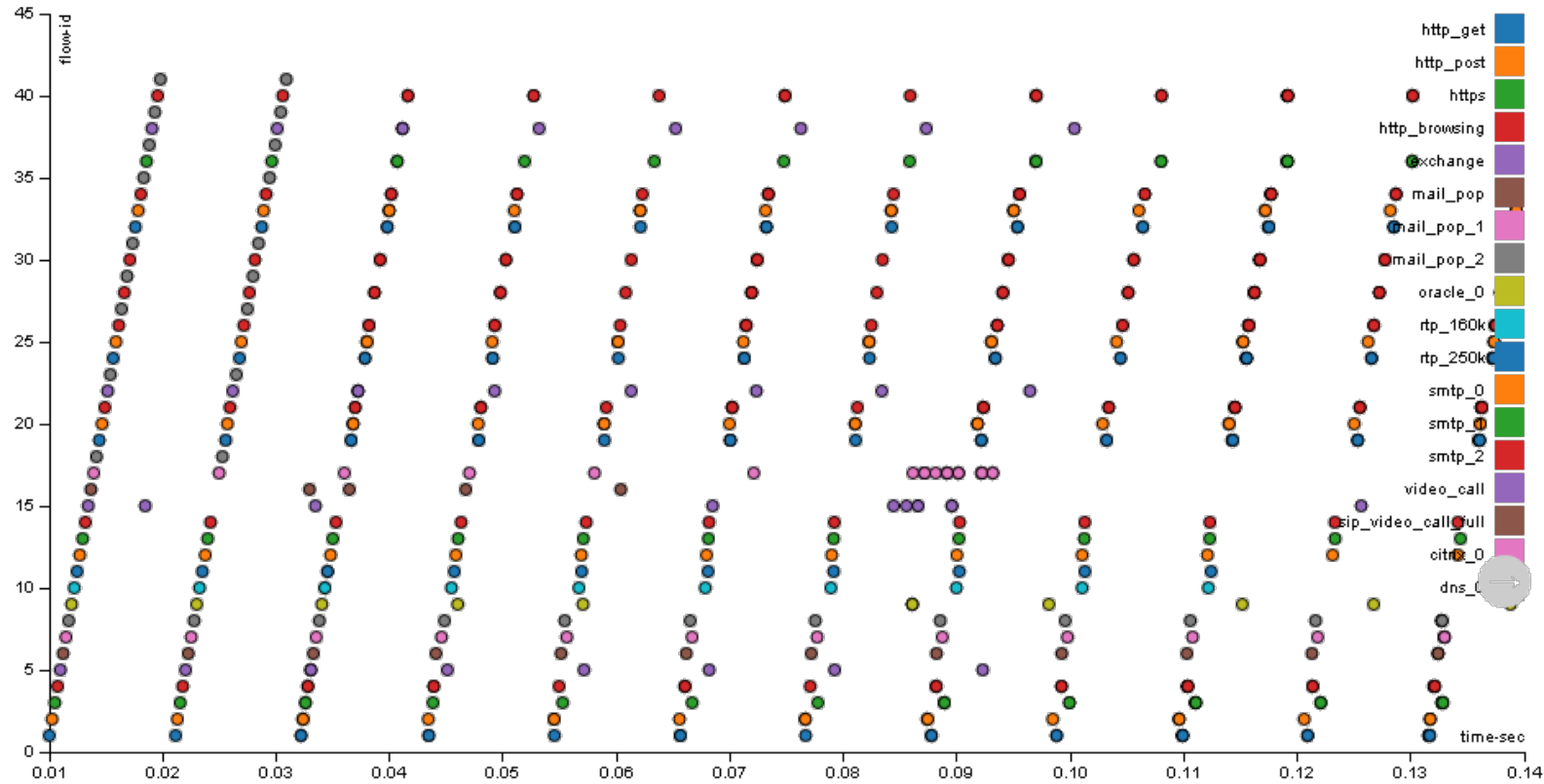
- Includes protocols with Control/Data dependency
 - SIP
 - RTSP



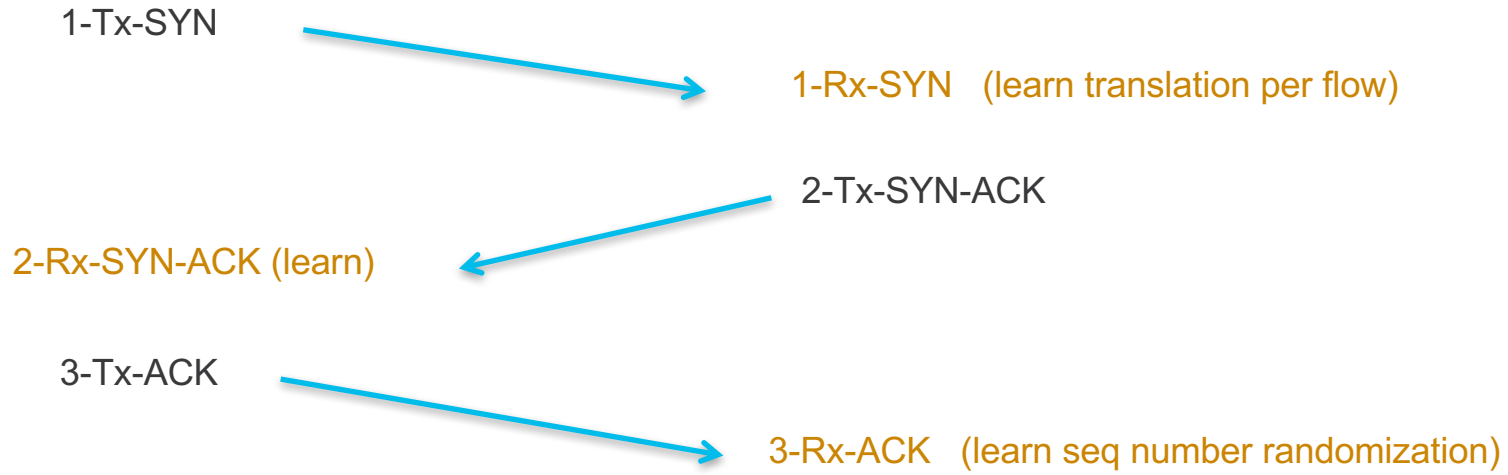
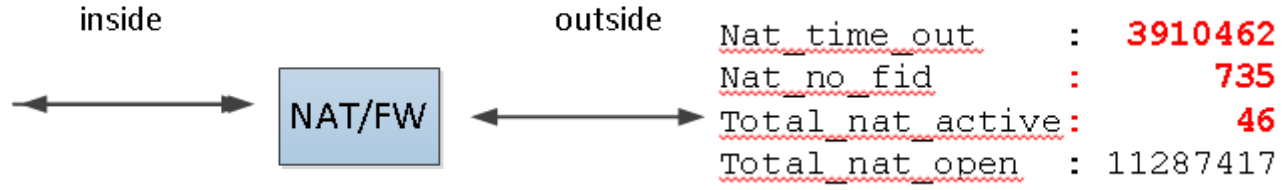
EMIX - YAML

```
- duration : 0.1
  generator :
    distribution : "seq"
    clients_start : "16.0.0.1"
    clients_end : "16.0.1.255"
    servers_start : "48.0.0.1"
    servers_end : "48.0.20.255"
    dual_port_mask : "1.0.0.0"
  cap_ipg : true
  cap_info :
    - name: avl/delay_10_http_get_0.pcap
      cps : 404.52
    - name: avl/delay_10_http_post_0.pcap
      cps : 404.52
    - name: avl/delay_10_https_0.pcap
      cps : 130.8745
    - name: avl/delay_10_http_browsing_0.pcap
      cps : 709.89
    - name: avl/delay_10_exchange_0.pcap
      cps : 253.81
    - name: avl/delay_10_mail_pop_2.pcap
      cps : 4.759
    - name: avl/delay_10_oracle_0.pcap
      cps : 79.3178
    - name: avl/delay_10_rtp_160k_full.pcap
      cps : 2.776
    - name: avl/delay_10_smtp_0.pcap
      cps : 7.3369
    - name: avl/delay_10_sip_video_call_full.pcap
      cps : 29.347
    - name: avl/delay_10_citrix_0.pcap
      cps : 43.6248
    - name: avl/delay_10_dns_0.pcap
      cps : 1975.015
```

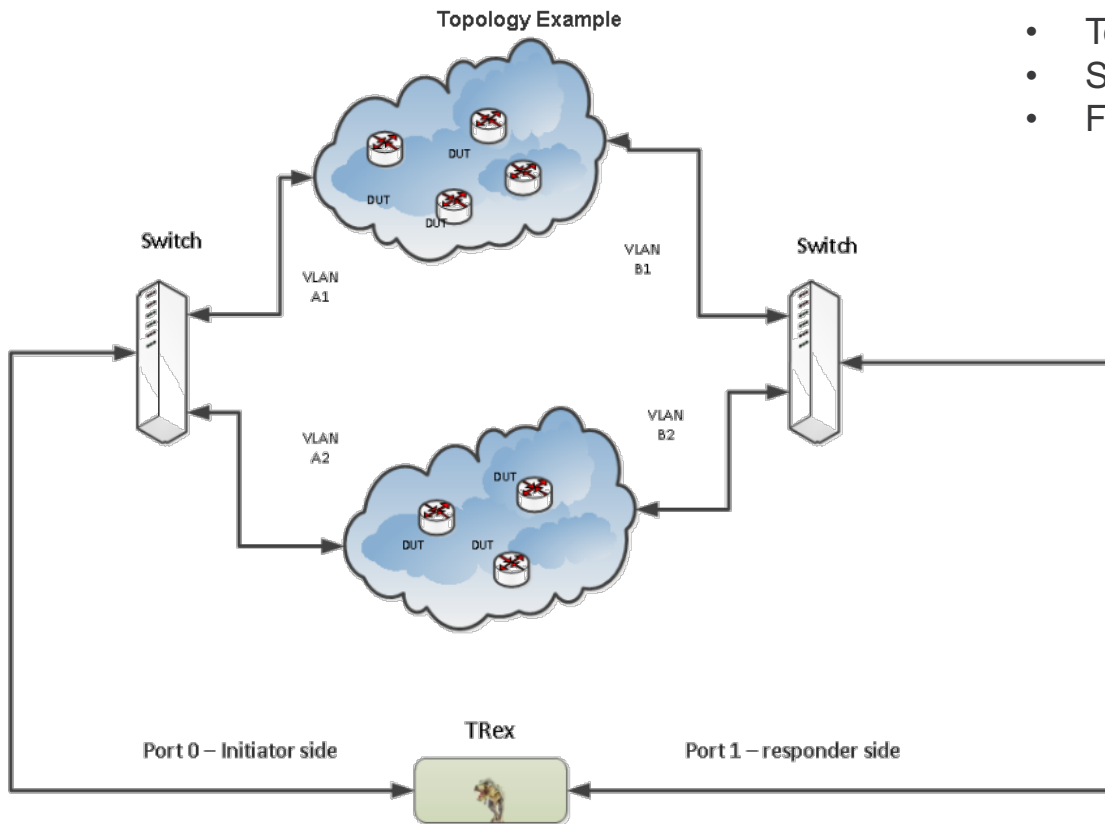
EMIX



NAT/FW learning translation/randomization

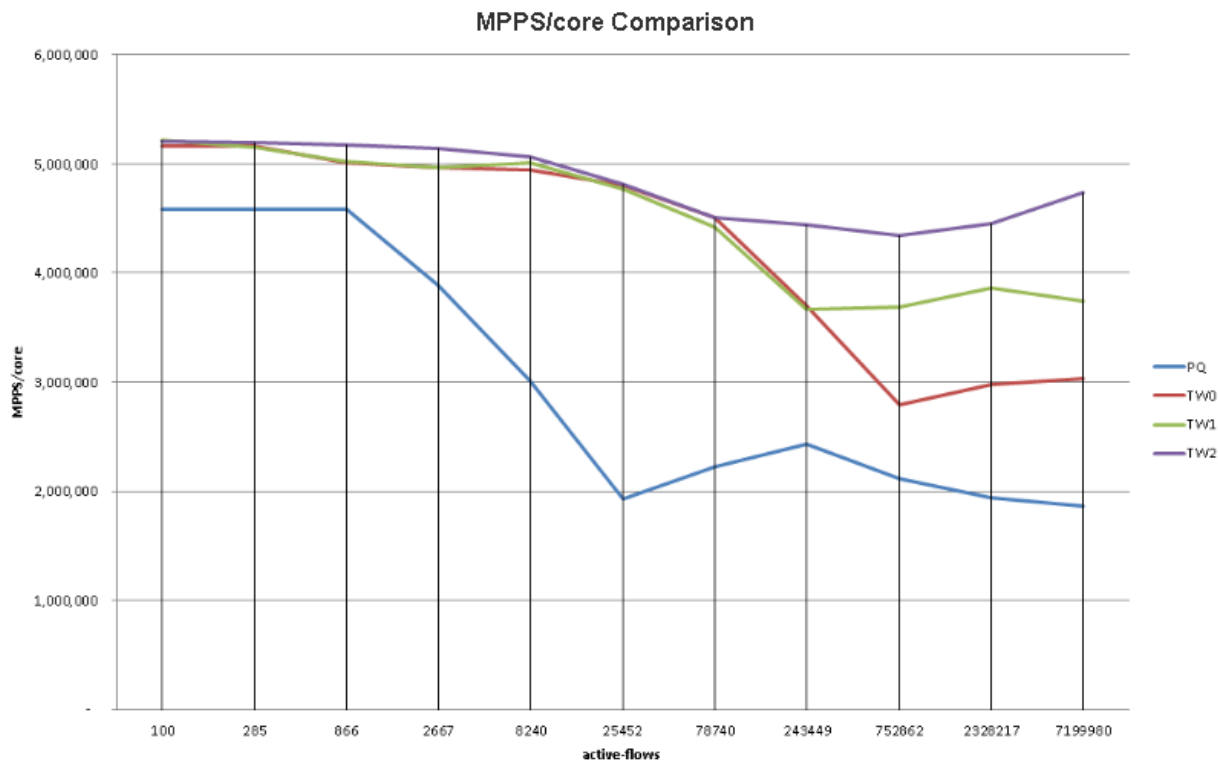


Client Clustering

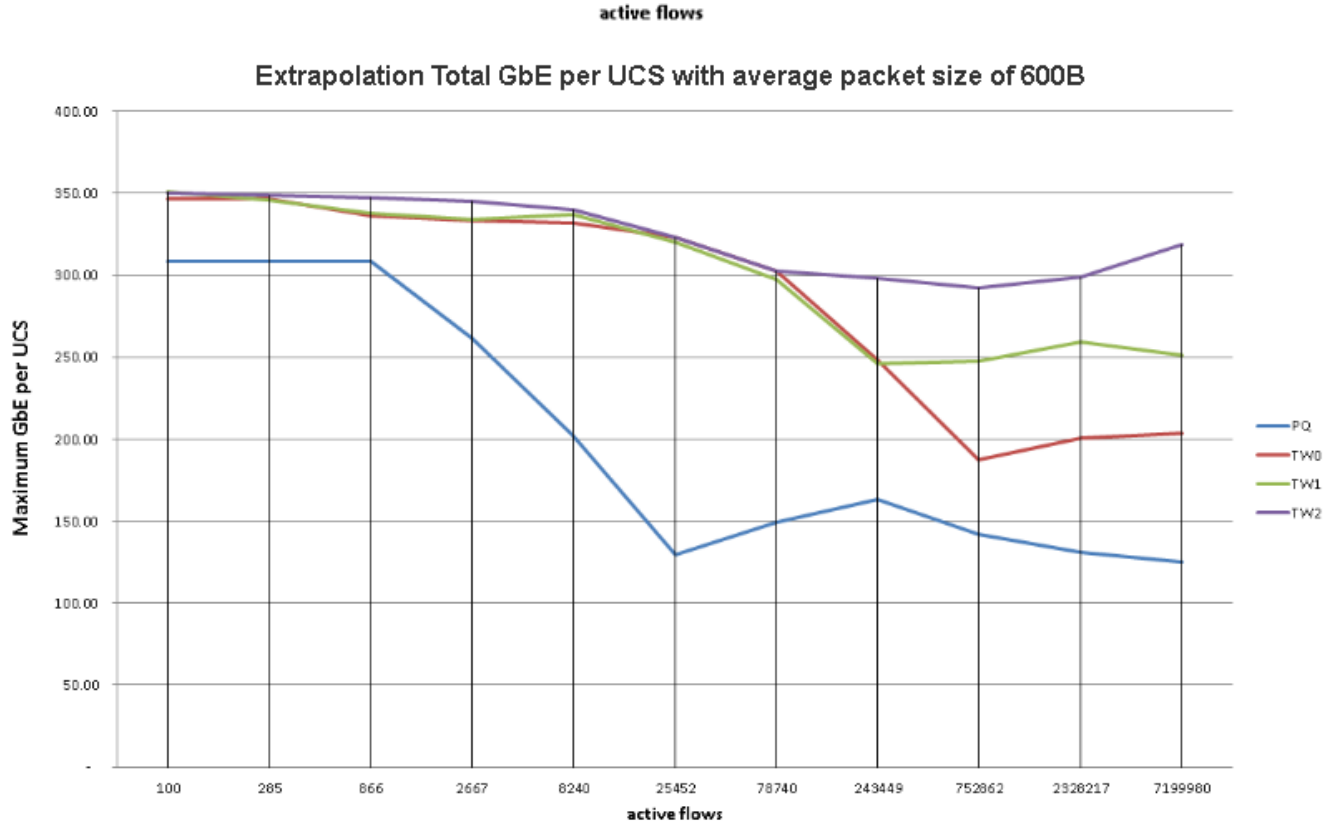


- To simulate a big network
- Scale of the number of clients
- For Controller testing

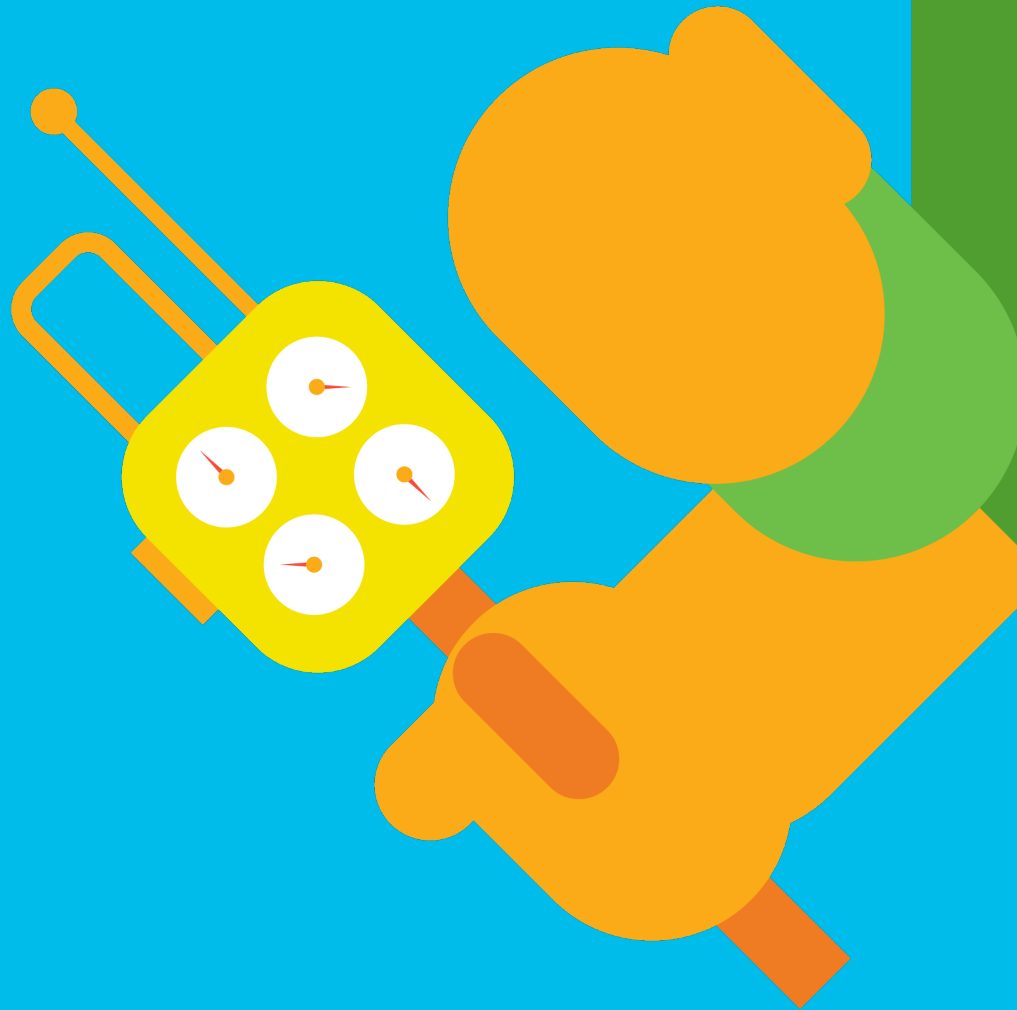
Performance MPPS/Core @ 8M flows



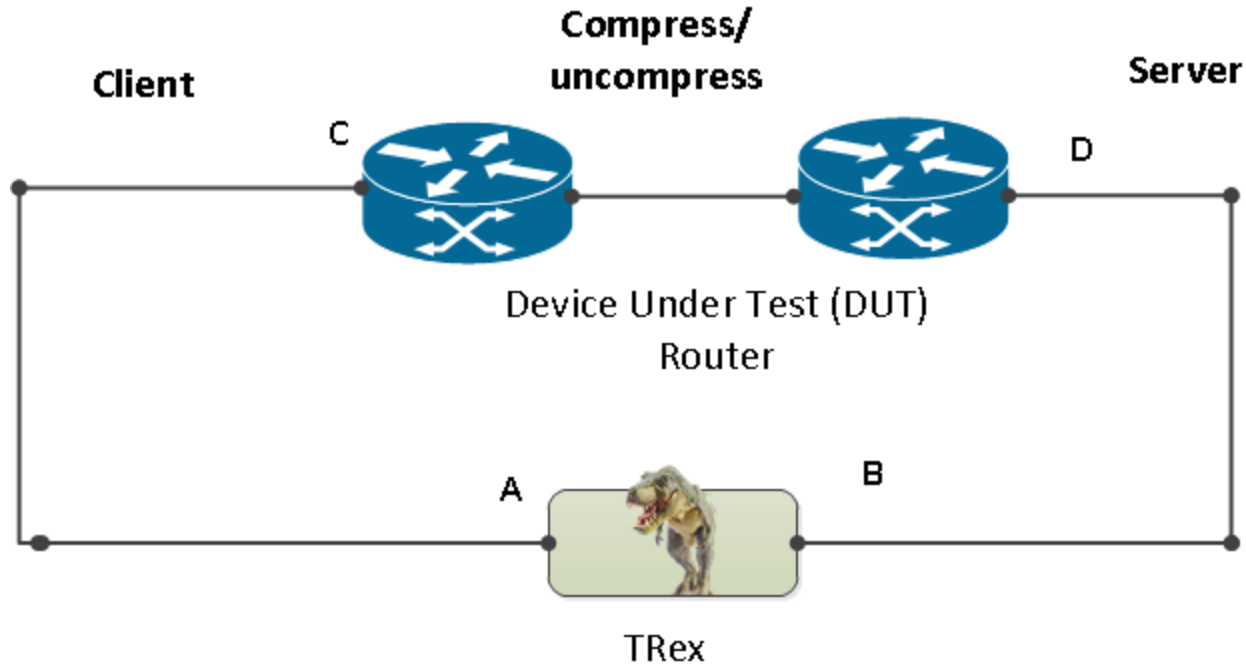
Gbps extrapolation, average packet size 600B



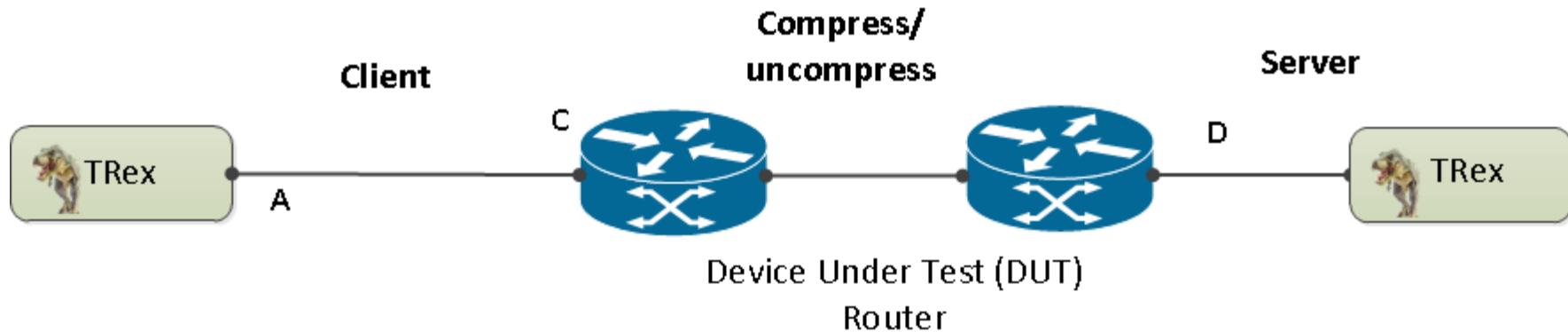
Advanced Stateful



User space TCP stack – Why ?

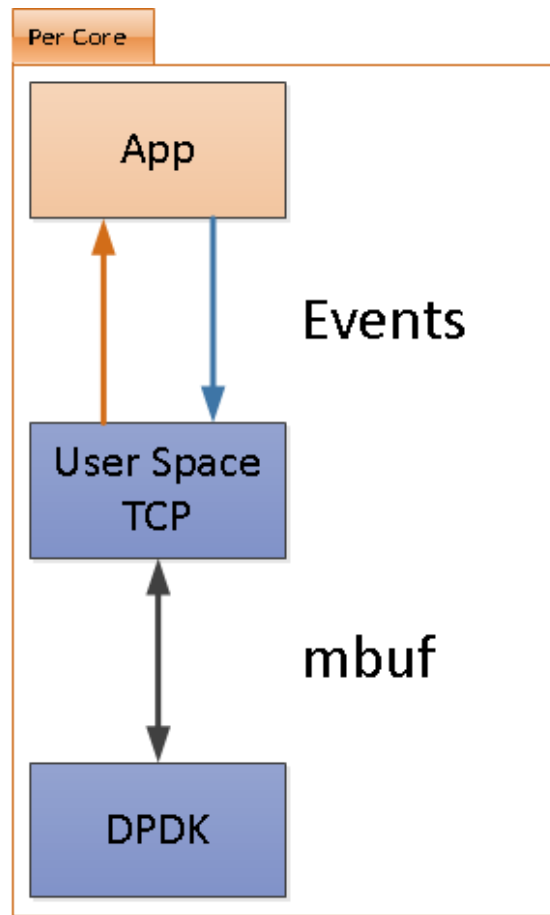


User space TCP stack – Why ?



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



TRex ASTF features status

- 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

HTTP Client

```

send(request, len=100)
wait_for_response(len<=1000)
delay(random(100-1000) *usec)
send(request2, len=200)
wait_for_response(len<=2000)
close()

```

HTTP Server

```

wait_for_request(len<=100)
send_response(data, len=1000)
wait_for_request(len<=200)
send_response(data, len=2000)
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,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) ❶

# 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: 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()

```

Different IP pool for each template

```

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

    def create_profile(self):
        ip_gen_c1 = ASTFIPGenDist(ip_range=["16.0.0.1", "16.0.0.255"],
                                distribution="seq")           ❶
        ip_gen_s1 = ASTFIPGenDist(ip_range=["48.0.0.1", "48.0.255.255"],
                                distribution="seq")
        ip_gen1 = ASTFIPGen(glob=ASTFIPGenGlobal(ip_offset="1.0.0.0"),
                           dist_client=ip_gen_c1,
                           dist_server=ip_gen_s1)

        ip_gen_c2 = ASTFIPGenDist(ip_range=["10.0.0.1", "10.0.0.255"],
                                distribution="seq")           ❷
        ip_gen_s2 = ASTFIPGenDist(ip_range=["20.0.0.1", "20.255.255"],
                                distribution="seq")
        ip_gen2 = ASTFIPGen(glob=ASTFIPGenGlobal(ip_offset="1.0.0.0"),
                           dist_client=ip_gen_c2,
                           dist_server=ip_gen_s2)

        profile = ASTFProfile(cap_list=[
            ASTFCapInfo(file="../cap2/http_get.pcap",
                       ip_gen=ip_gen1),                       ❸
            ASTFCapInfo(file="../cap2/http_get.pcap",
                       ip_gen=ip_gen2, port=8080)             ❹
        ])

    return profile

```

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 #0
-	---	---	
Flow Table	---	---	
-	---	---	
redirect_rx_ok	0	1	redirect to rx OK

Client/Server only

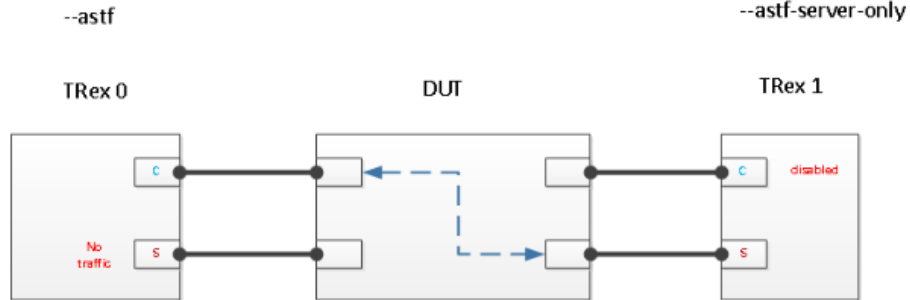
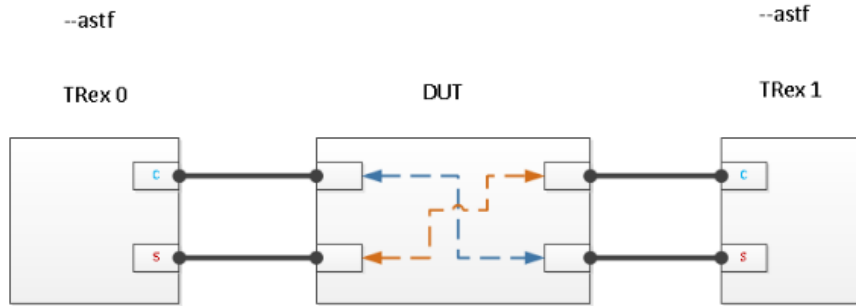
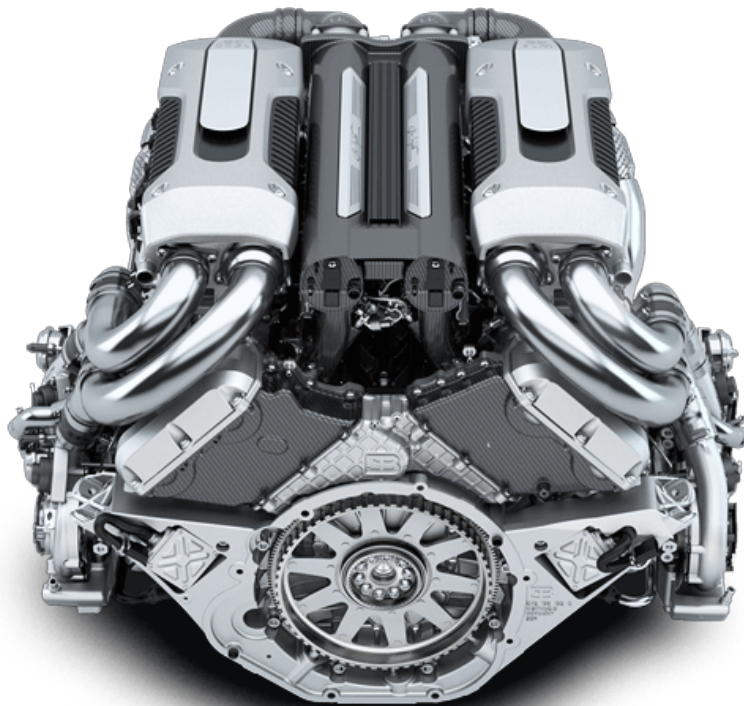


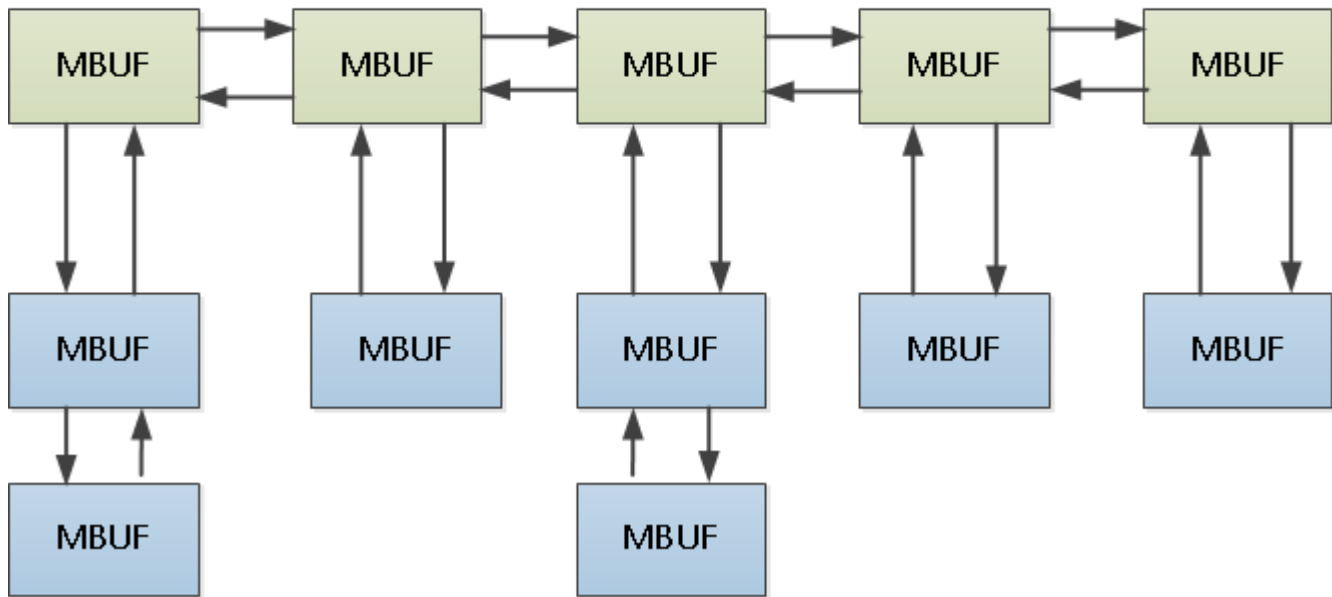
Figure 8. C/S modes examples

Under the hood



TCP stack – Flow Scale -TX

TX Queue 32K – Sliding window



10M flows



320GByte

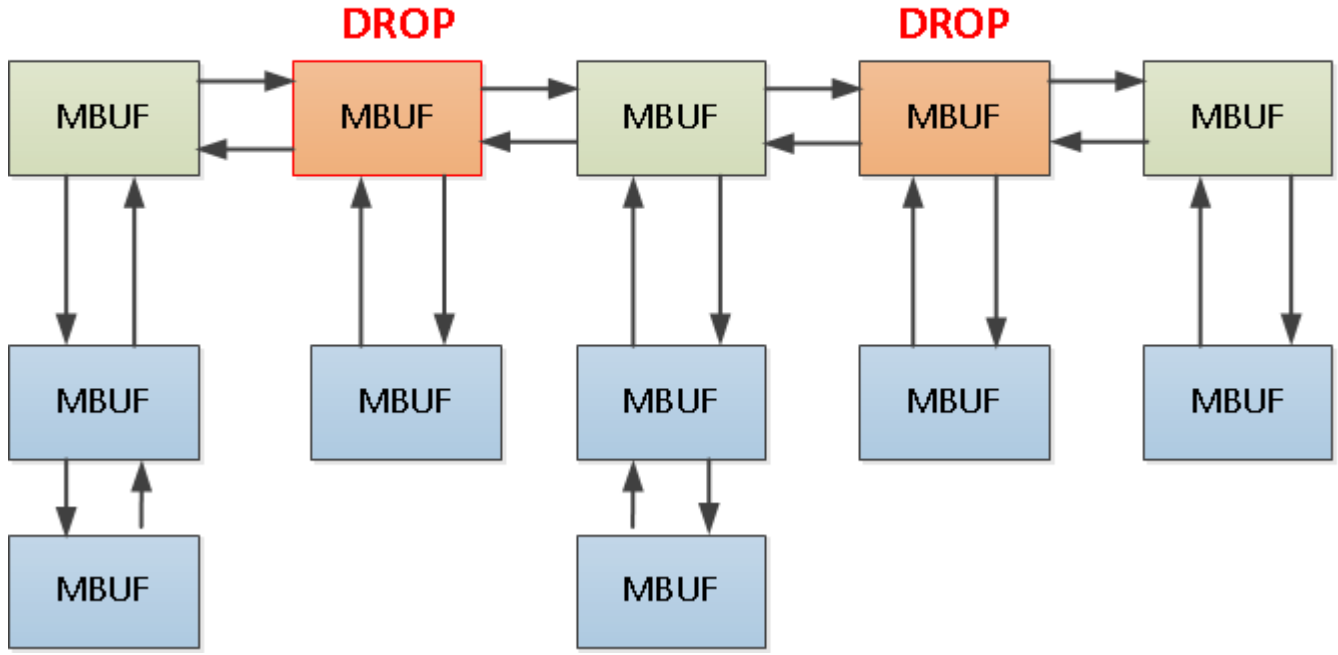


320M mbuf
=10GB

0.01GB

TCP stack – Flow Scale issue - RX

RX Reassembly Queue 32K – Sliding window



10M flows



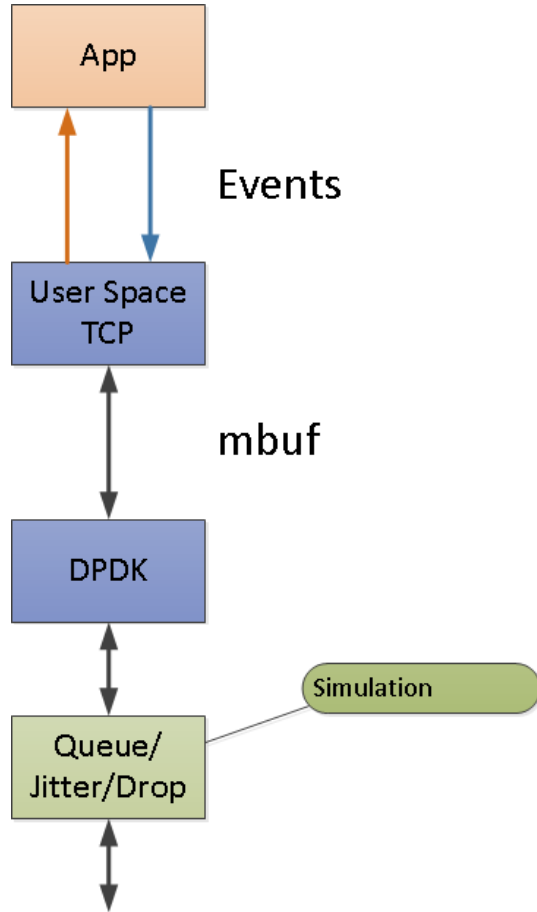
320GByte



3-50GB for
1% drop
rate

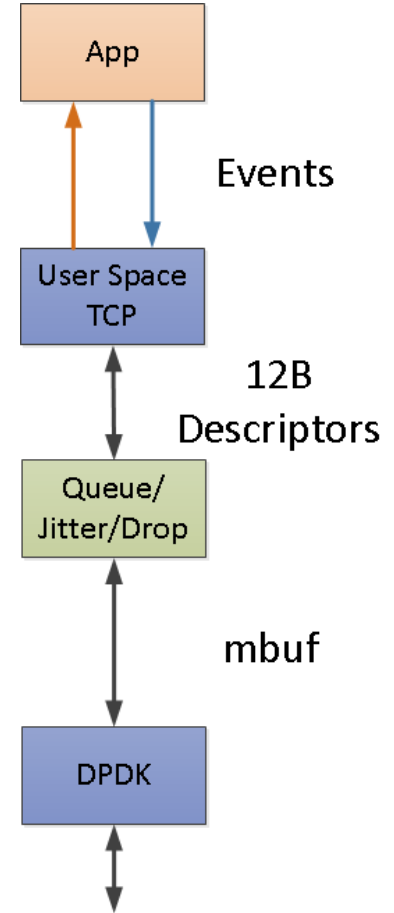
0.01GB

TCP stack – Delay/Jitter/Drop simulation



100MPPS * 100msec
= 10MPPS in Queue

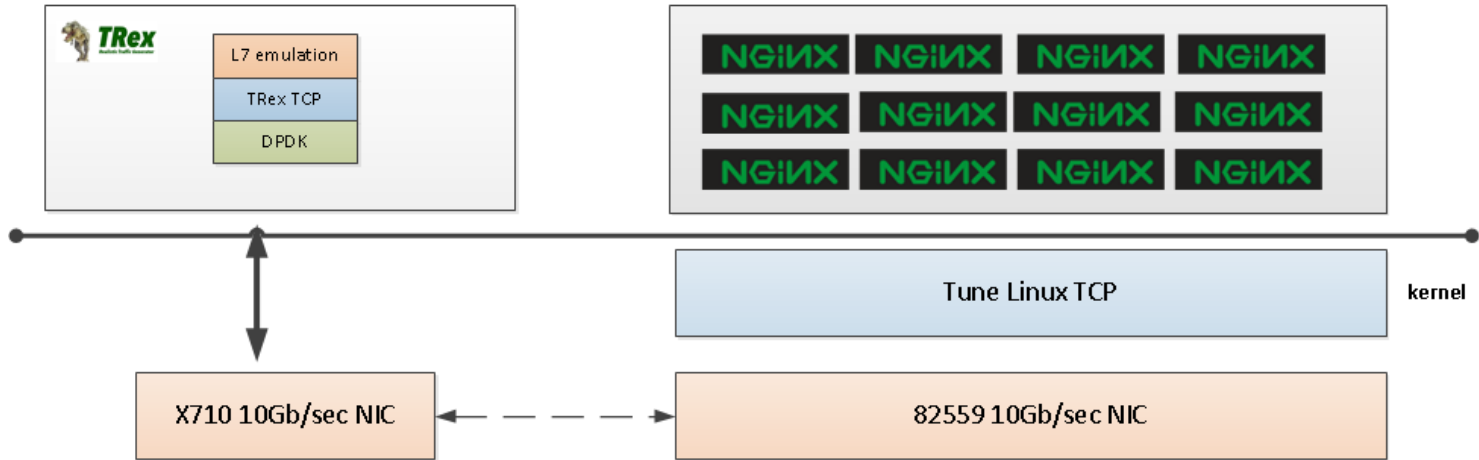
100MPPS * 100msec
= 10 * 16MPPS = 0.16GB



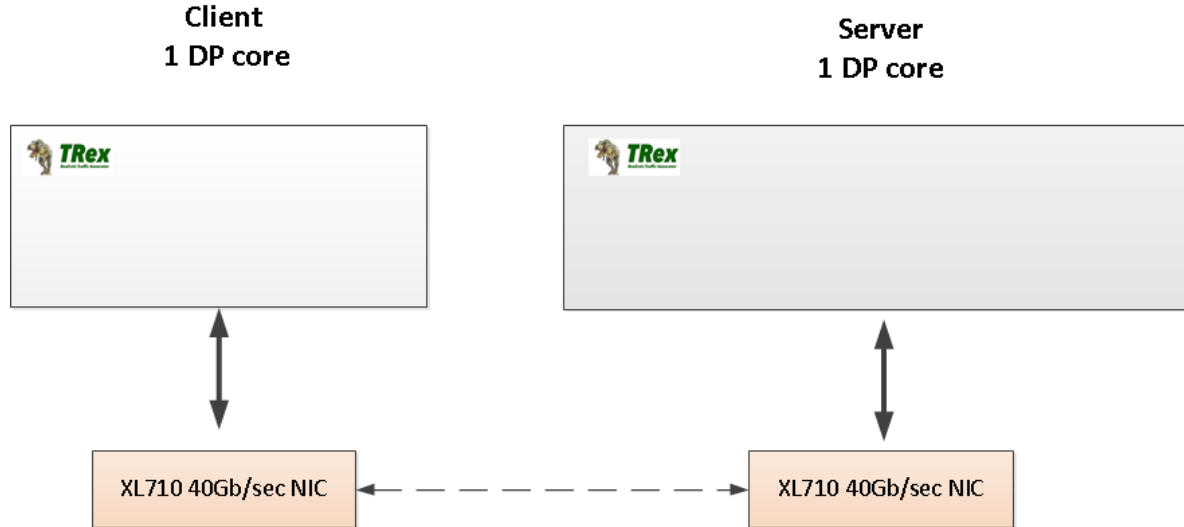
TRex vs NGINX

Client
1 DP core

Server 2 sockets / 16 cores total



Performance setup #2



Performance numbers

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

x80 faster
x2000 less memory


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

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

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
- Statistic per stream (e.g. latency/ Jitter) usec
- Python automation support 
- Multi-user support



Traffic Profile Example

Stream 1

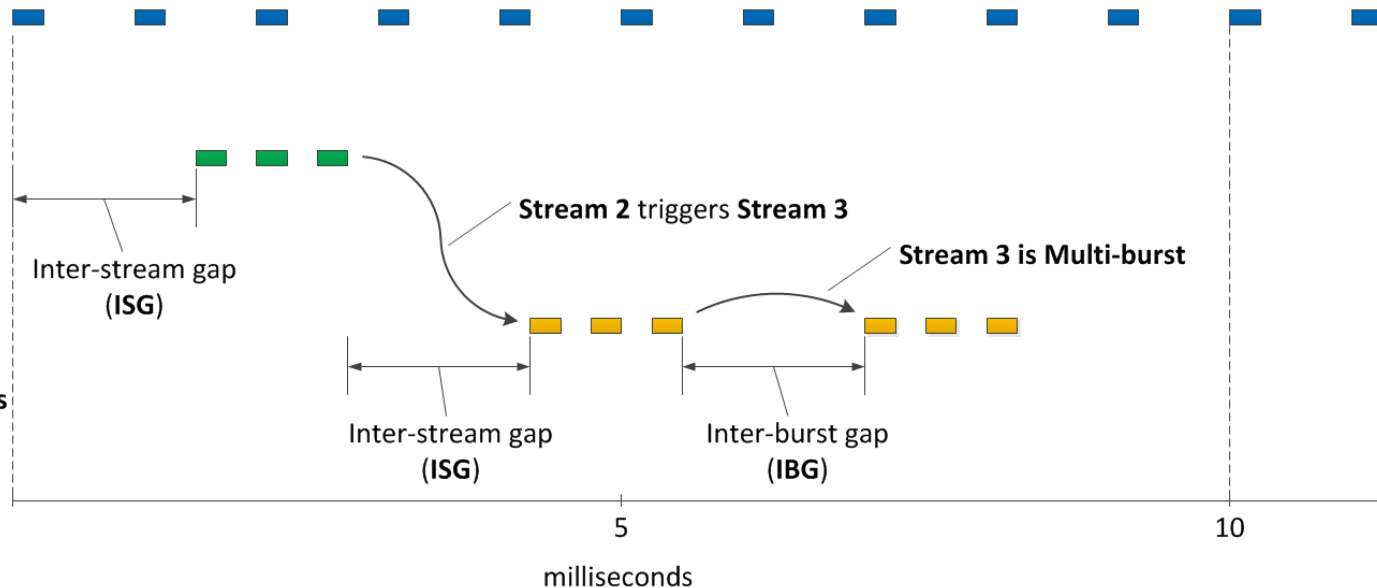
IP/TCP
 Continuous 1 KPPS
 10.0.0.1 – 10.0.0.255

Stream 2

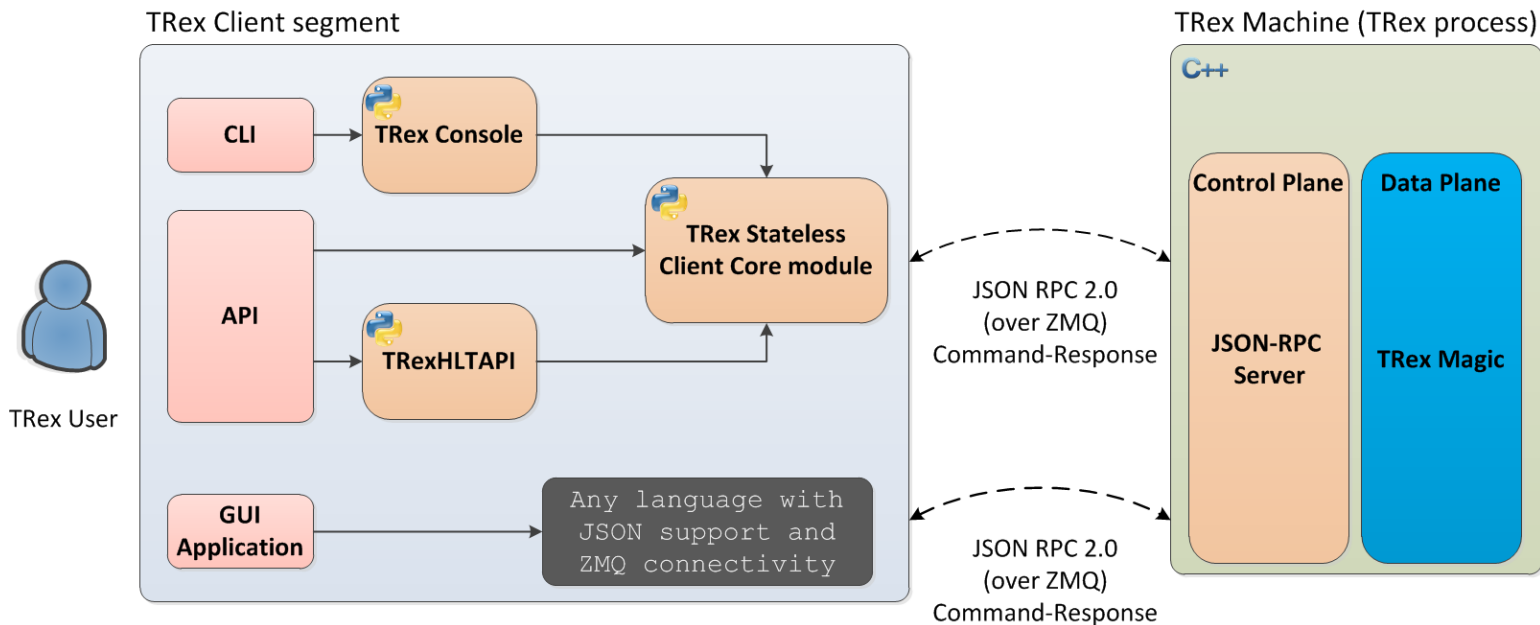
IP/UDP
 Burst 2 KPPS, 3 packets
 16.0.0.1 – 10.0.0.255

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() ) ]

```

1



Python Automation example

```
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)

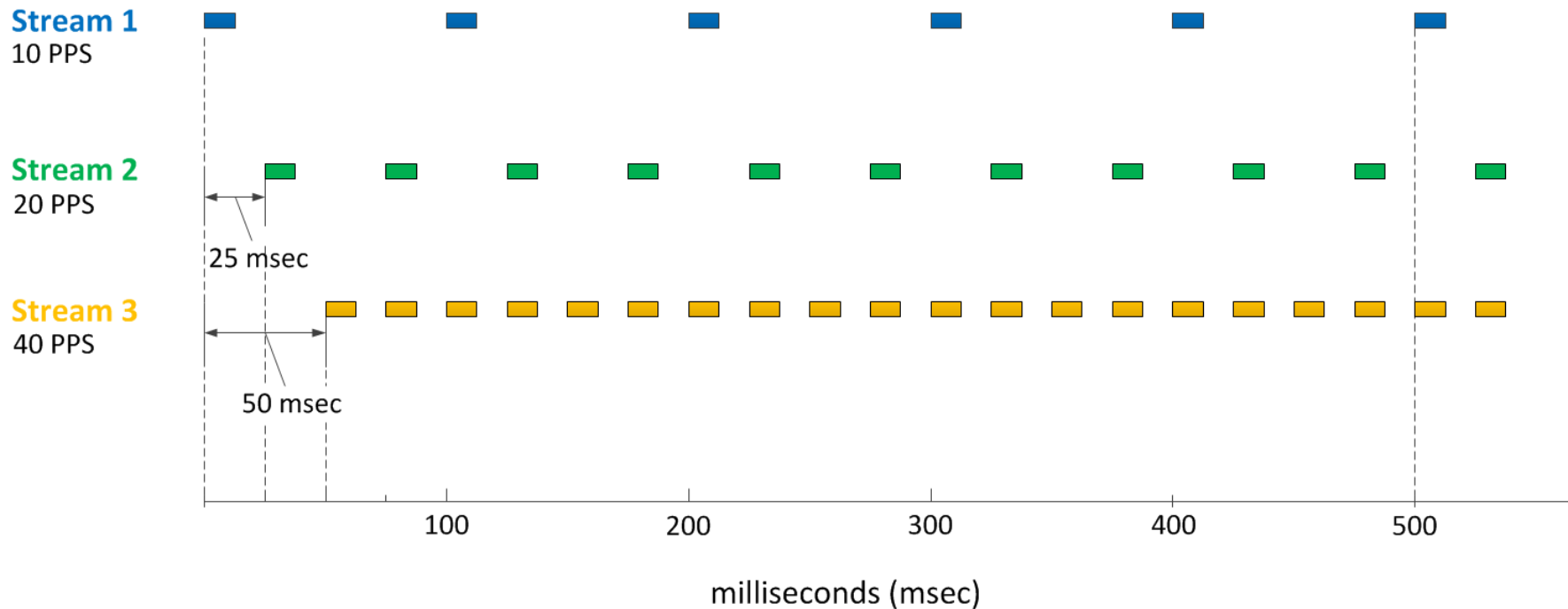
    # block until done
    c.wait_on_traffic(ports = [0, 1])

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

finally:
    c.disconnect()
```

Stateless – Traffic profile

Simple Interleaving streams



Simple Interleaving streams -profile

```
def create_stream (self):  
  
    # create a base packet and pad it to size  
    size = self.fsize - 4 # no FCS  
    base_pkt = Ether()/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=12,sport=1025) ❶  
    base_pkt1 = Ether()/IP(src="16.0.0.2",dst="48.0.0.1")/UDP(dport=12,sport=1025)  
    base_pkt2 = Ether()/IP(src="16.0.0.3",dst="48.0.0.1")/UDP(dport=12,sport=1025)  
    pad = max(0, size - len(base_pkt)) * 'x'  
  
    return STLProfile( [ STLStream( isg = 0.0,  
                                   packet = STLPktBuilder(pkt = base_pkt/pad),  
                                   mode = STLTXCont( pps = 10), ❷  
                                   ),  
  
                        STLStream( isg = 25000.0, #defined in usec, 25 msec  
                                   packet = STLPktBuilder(pkt = base_pkt1/pad),  
                                   mode = STLTXCont( pps = 20), ❸  
                                   ),  
  
                        STLStream( isg = 50000.0, #defined in usec, 50 msec  
                                   packet = STLPktBuilder(pkt = base_pkt2/pad),  
                                   mode = STLTXCont( pps = 40) ❹  
                                   )  
                        ] ).get_streams()
```


Multi burst

Stream 0

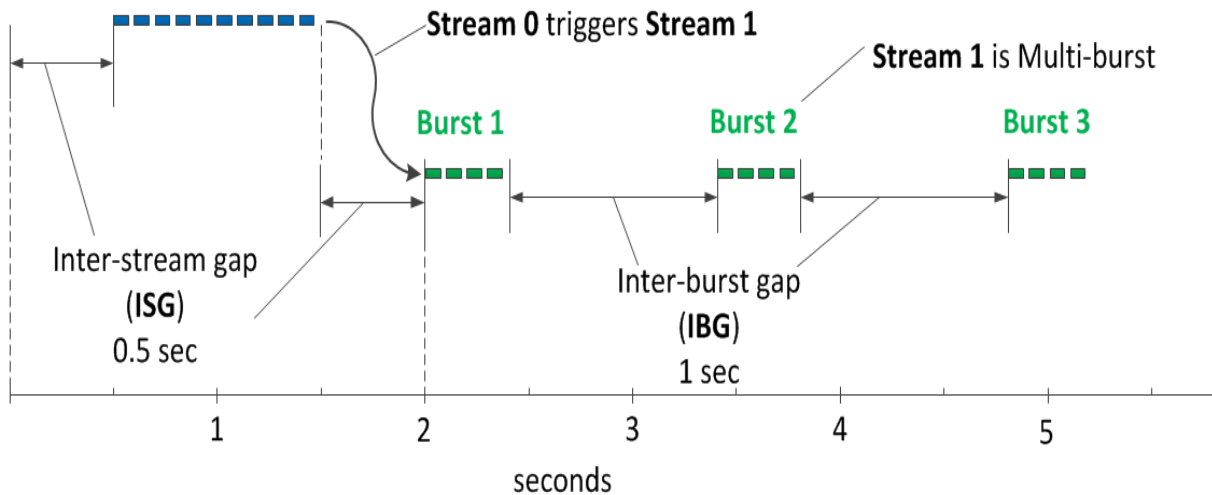
At start, inter-stream gap (ISG) of 0.5 sec

1 burst of 10 packets at 10 PPS

Stream 1

Triggered by Stream 0, inter-burst gap (IBG) of 1 sec

3 bursts of 4 packets at 10 PPS



Multi burst profile

```

def create_stream (self):

    # create a base packet and pad it to size
    size = self.fsize - 4 # no FCS
    base_pkt = Ether()/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=12,sport=1025)
    base_pkt1 = Ether()/IP(src="16.0.0.2",dst="48.0.0.1")/UDP(dport=12,sport=1025)
    pad = max(0, size - len(base_pkt)) * 'x'

    return STLProfile( [ STLStream( isg = 10.0, # start in delay

        ❶
        name      = 'S0',
        packet    = STLPktBuilder(pkt = base_pkt/pad),
        mode      = STLTXTSingleBurst( pps = 10, total_pkts = 10),
        next      = 'S1'), # point to next stream

        STLStream( self_start = False, # stream is disabled. Enabled by S0

        ❷
        name      = 'S1',
        packet    = STLPktBuilder(pkt = base_pkt1/pad),
        mode      = STLTXTMultiBurst( pps = 1000,
                                     pkts_per_burst = 4,
                                     ibg = 1000000.0,
                                     count = 5)

        )

    ]).get_streams()

```

Field Engine

- Flexible engine for changing packet fields
- Examples
 - Change TOS 1-20
 - Range of client IPv4/IPv6 e.g. 10.0.0.1-10.0.0.254
 - Random packet size 64-9k
 - Random destination IPv4/IPv6
 - Support any tunnel even not valid packet like QinQ/GRE/MPLS/Ipv6/UDP/Ipv4/HTTP

Field Engine, Syn attack

```

# TCP SYN
base_pkt = Ether()/IP(dst="48.0.0.1")/TCP(dport=80,flags="S") ❶

# VM
vm = STLScVmRaw( [ STLVmFlowVar(name="ip_src",
                               min_value="16.0.0.0",
                               max_value="18.0.0.254",
                               size=4, op="random"), ❷

                  STLVmFlowVar(name="src_port",
                               min_value=1025,
                               max_value=65000,
                               size=2, op="random"), ❸

                  STLVmHdrFlowVar(fv_name="ip_src", pkt_offset= "IP.src" ), ❹

                  STLVmFixIPv4(offset = "IP"), # fix checksum ❺

                  STLVmHdrFlowVar(fv_name="src_port",
                                  pkt_offset= "TCP.sport") # U ❻

                ]
            )

```

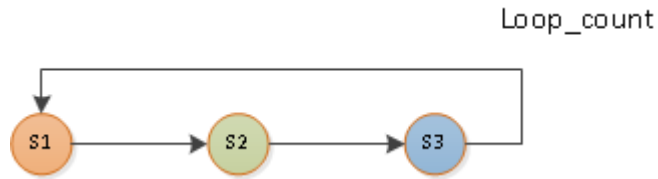
Pcap File Conversion to Streams

```

def get_streams (self,
                 ipg_usec = 10.0,           ❶
                 loop_count = 1):          ❷

    profile = STLProfile.load_pcap(self.pcap_file,  ❸
                                   ipg_usec = ipg_usec,
                                   loop_count = loop_count)

```



Per Stream Statistics

```

class STLS1(object):

    def get_streams (self, direction = 0):
        return [STLStream(packet = STLPktBuilder(pkt = "st1/yaml/udp_64B_no_crc.pcap"),
                        mode = STLTXCont(pps = 1000),
                        flow_stats = STLFlowStats(pg_id = 7)), ❶

                STLStream(packet = STLPktBuilder(pkt = "st1/yaml/udp_594B_no_crc.pcap"),
                        mode = STLTXCont(pps = 5000),
                        flow_stats = STLFlowStats(pg_id = 12)) ❷

        ]
    
```

Per Stream Statistics - TUI

Streams Statistics

PG ID	12	7	
Tx pps	5.00 Kpps	999.29 pps	#1
Tx bps L2	23.60 Mbps	479.66 Kbps	
Tx bps L1	24.40 Mbps	639.55 Kbps	

Rx pps	5.00 Kpps	999.29 pps	#2
Rx bps	N/A	N/A	#3

opackets	222496	44500	
ipackets	222496	44500	
obytes	131272640	2670000	
ibytes	N/A	N/A	#3

tx_pkts	222.50 Kpkts	44.50 Kpkts	
rx_pkts	222.50 Kpkts	44.50 Kpkts	
tx_bytes	131.27 MB	2.67 MB	
rx_bytes	N/A	N/A	#3

Per Stream Latency/Jitter

- Base on per stream stats hardware assist
- Software measures latency and jitter resolution is ~usec (not nsec)

```
def get_streams (self, direction = 0, **kwargs):
    return [STLStream(packet = STLPktBuilder(pkt = "yaml/udp_64B_no_crc.pcap"),
                    mode = STLTxCont(pps=1000),
                    flow_stats = STLFlowLatencyStats(pg_id = 7)),

           STLStream(packet = STLPktBuilder(pkt = "yaml/udp_594B_no_crc.pcap"),
                    mode = STLTxCont(pps=5000),
                    flow_stats = STLFlowLatencyStats(pg_id = 12))
    ]
```

|

Per Stream Statistics - TUI

```

trex>start -f stl/flow_stats.py --port 0

trex>tui

Latency Statistics (usec)

  PG ID      |      7      |      12
-----|-----|-----
Max latency  |           0 |           0 #1
Avg latency  |           5 |           5 #2
-- Window -- |           |
Last (max)   |           3 |           4 #3
Last-1      |           3 |           3
Last-2      |           4 |           4
Last-3      |           4 |           3
Last-4      |           4 |           4
Last-5      |           3 |           4
Last-6      |           4 |           3
Last-7      |           4 |           3
Last-8      |           4 |           4
Last-9      |           4 |           3
---|-----|-----
Jitter       |           0 |           0 #4
----|-----|-----
Errors       |           0 |           0 #5

```

Stateless – Service mode

Service Mode

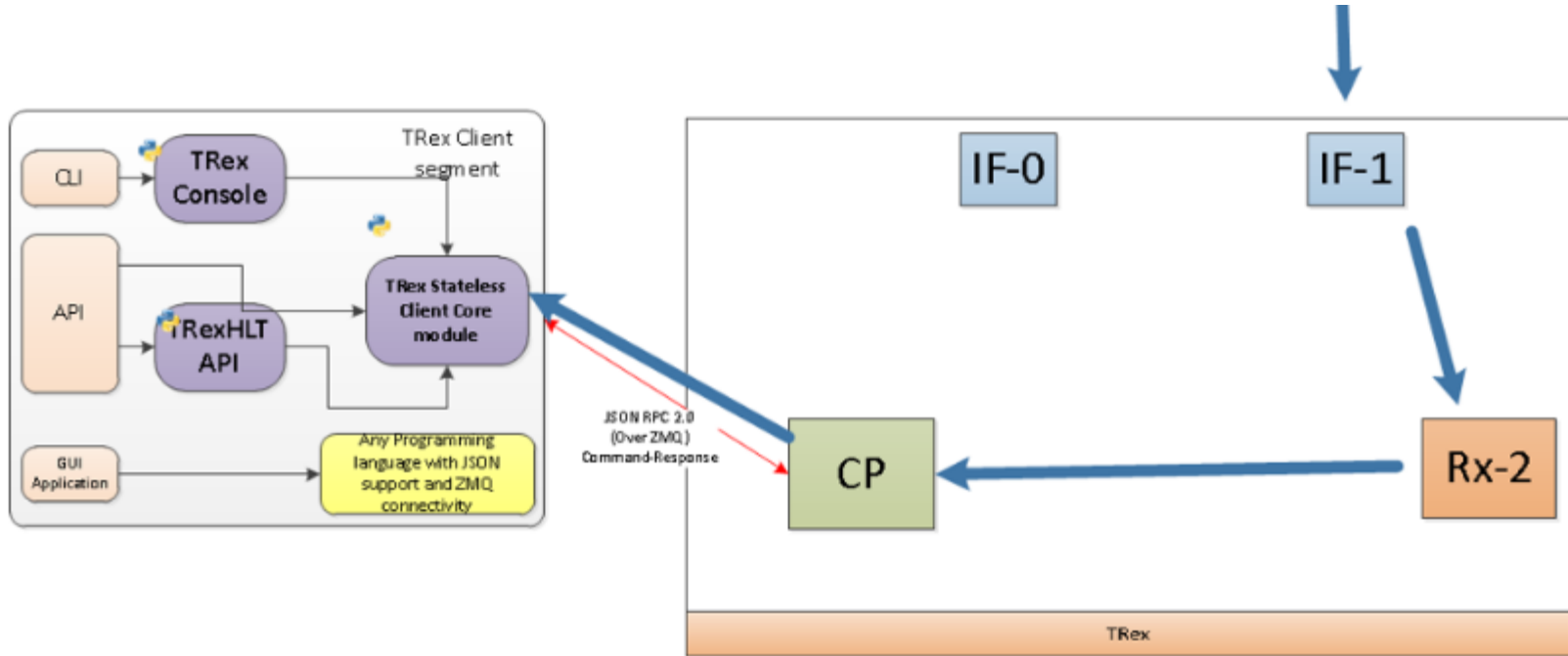


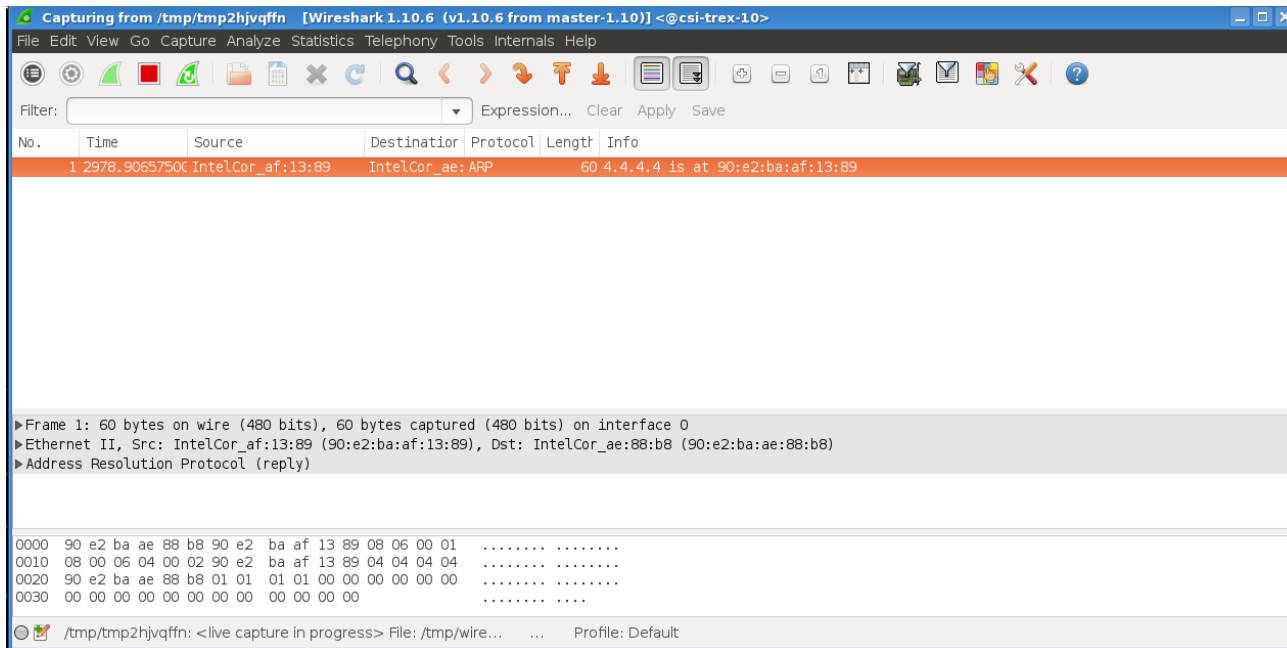
Figure 6. Port Under Service Mode

Service Mode

- Ping / ARP/DHCP Client
- IPv6 ND /Multicast setup
- Scan6 support
- Traffic Capturing
- Functional test
- New protocols multiplex framework

Service Mode

Capture Monitoring – Wireshark Pipe



Capturing from /tmp/tmp2hjqvffn [Wireshark 1.10.6 (v1.10.6 from master-1.10)] <@csi-trex-10>

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	2978.9065750	IntelCor_af:13:89	IntelCor_ae:ARP	60	4.4.4.4	is at 90:e2:ba:af:13:89

▶Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▶Ethernet II, Src: IntelCor_af:13:89 (90:e2:ba:af:13:89), Dst: IntelCor_ae:88:b8 (90:e2:ba:ae:88:b8)
▶Address Resolution Protocol (reply)

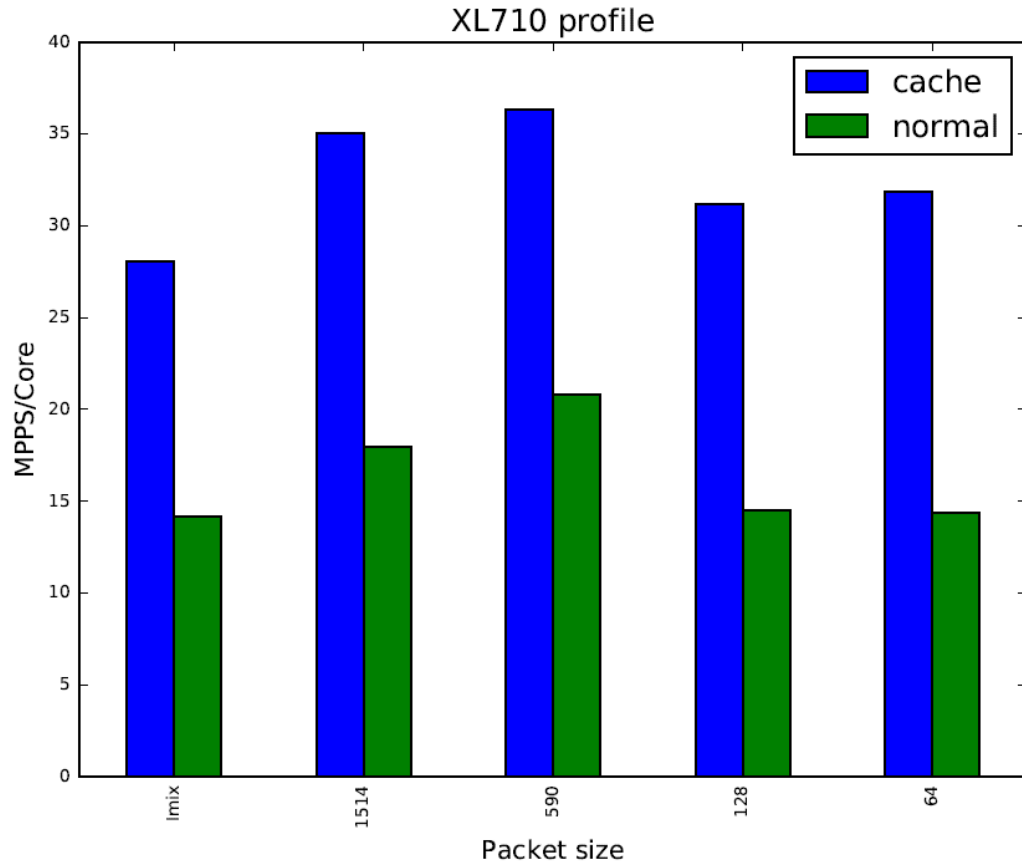
```
0000 90 e2 ba ae 88 b8 90 e2 ba af 13 89 08 06 00 01 .....  
0010 08 00 06 04 00 02 90 e2 ba af 13 89 04 04 04 04 .....  
0020 90 e2 ba ae 88 b8 01 01 01 01 00 00 00 00 00 .....  
0030 00 00 00 00 00 00 00 00 00 00 00 .....  
.....
```

/tmp/tmp2hjqvffn: <live capture in progress> File: /tmp/wire... Profile: Default

```
trex(service)>capture monitor start --rx 0 --pipe  
Starting pipe capture monitor [SUCCESS]  
*** Please run 'wireshark -k -i /tmp/tmp2hjqvffn' ***  
Waiting for Wireshark pipe connection [SUCCESS]  
  
*** Capture monitoring started ***  
trex(service)>arp -p 0  
Resolving destination on port(s) [0]: [SUCCESS]  
Port 0 - Received ARP reply from: 4.4.4.4, hw: 90:e2:ba:af:13:89  
55.18 [ms]
```

Stateless – Performance

Performance XL710 MPPS/Core



[link](#)

Demo



Continue Your Education

- [Stateless manual](#)
- [TRex documents Index](#)
- [GitHub](#)
- [DevNet zone](#)



Thank you

Thanks for Coming!

- Join us for future webinars: <http://bit.ly/DevNetWebinarWed>
- Please complete our post-event survey
- Recording will be sent out shortly
- Join DevNet: ENTER URL here
- Follow us @CiscoDevNet



DEVNET