DPDK

TRex

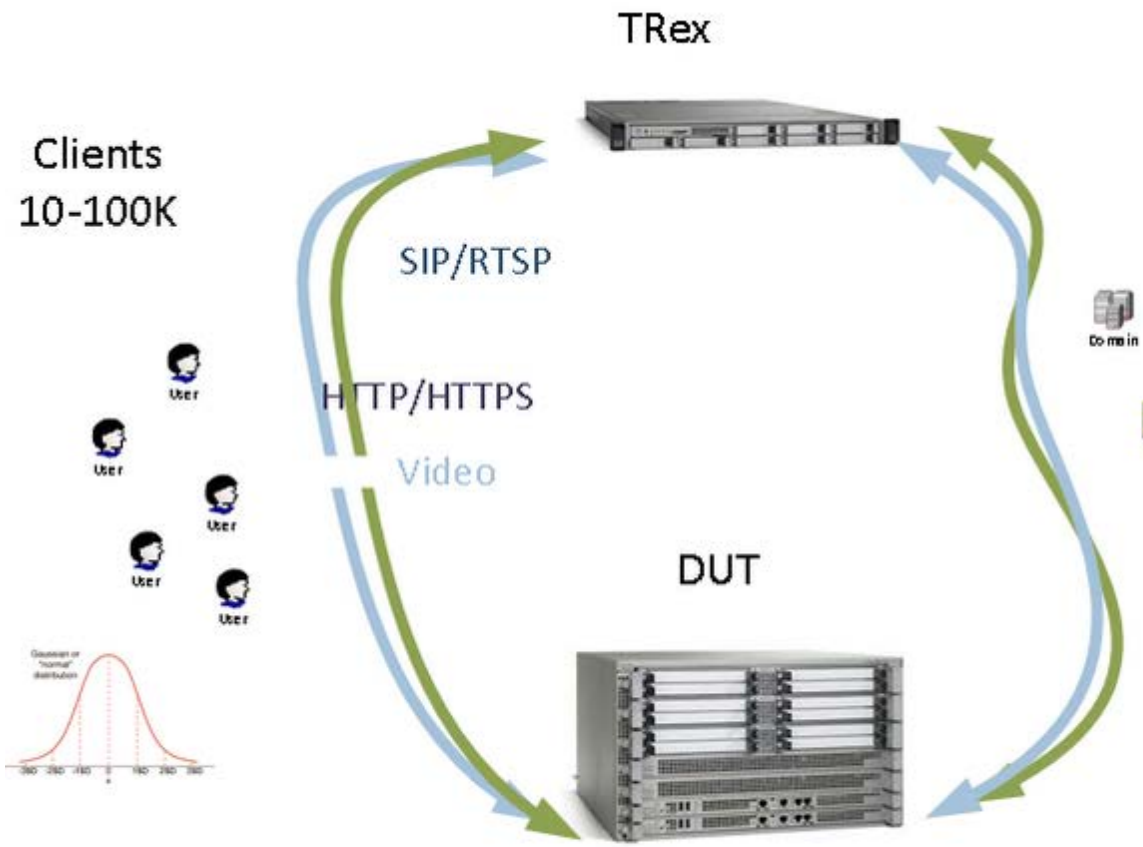
Realistic Traffic Generator

Stateless support

Hanoch Haim

DPDK Summit Userspace - Dublin- 2016

TRex started as Stateful TGN

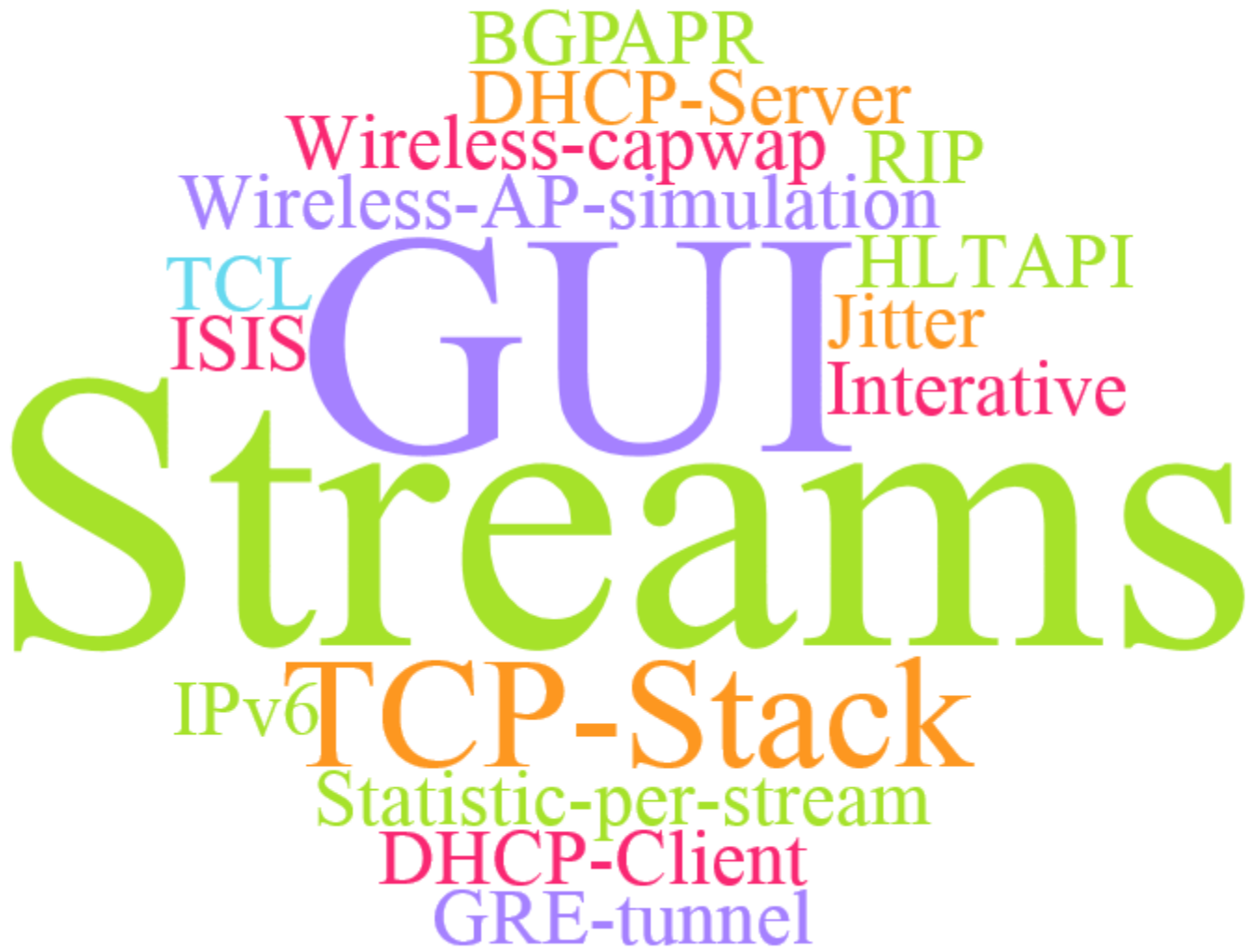



```
- duration : 1.0
  generator :
          distribution : "seq"
          clients_start : "16.0.0.1"
          clients_end   : "16.0.0.10"
          servers_start : "48.0.0.1"
          servers_end   : "48.0.0.3"
          dual_port_mask : "1.0.0.0"
          tcp_aging      : 1
          udp_aging      : 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>
```

TRex
Realistic Traffic Generator

Cisco Pioneer Award winner - 2015

Requirements



TRex
Realistic Traffic Generator

Requirement fd.io CSIT project

https://wiki.fd.io/view/CSIT

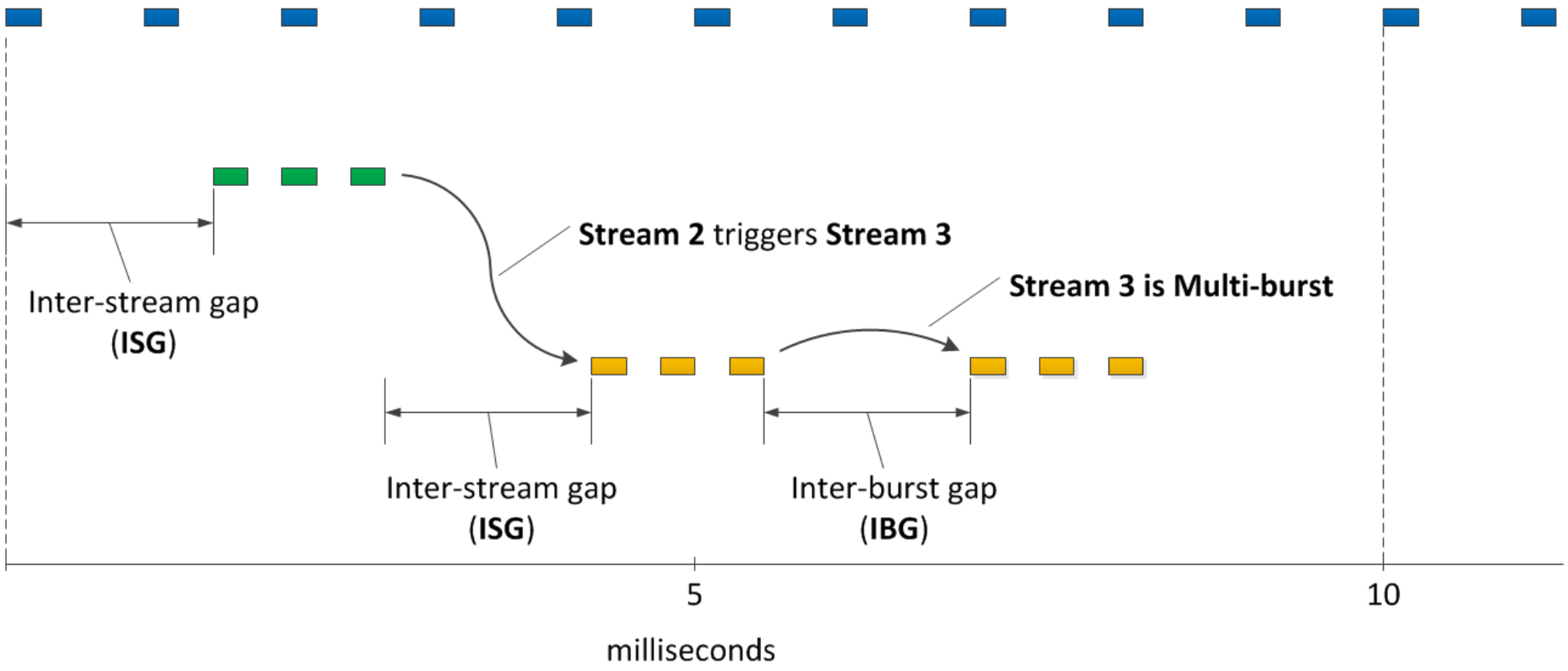
- **RFC 2544**
  - High performance /scale
  - Multi streams
  - Latency/Jitter measurement
  - Statistic per stream
  - Automation

Traffic Profile Example

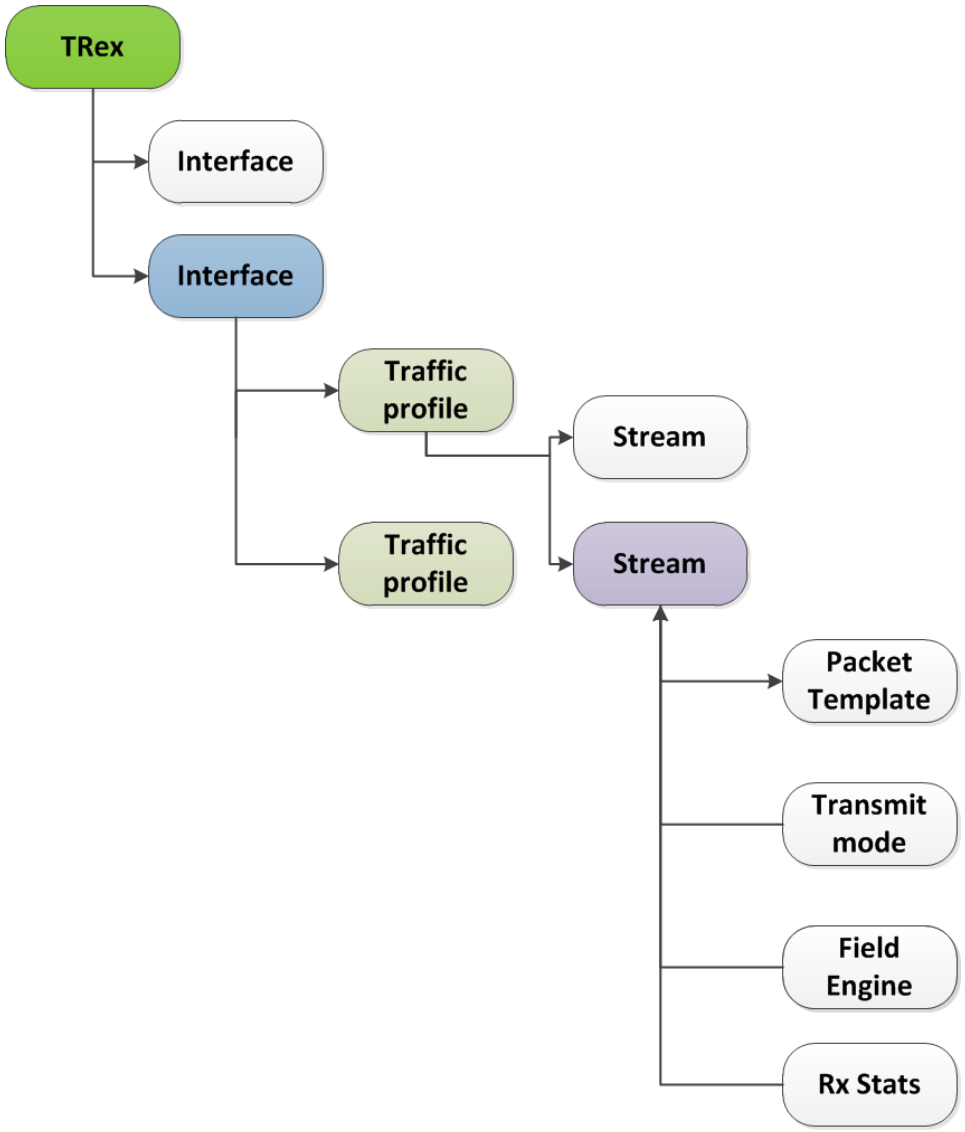
Stream 1
IP/TCP
Continuous 1 KPPS
10.0.0.1 – 10.0.0.255

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

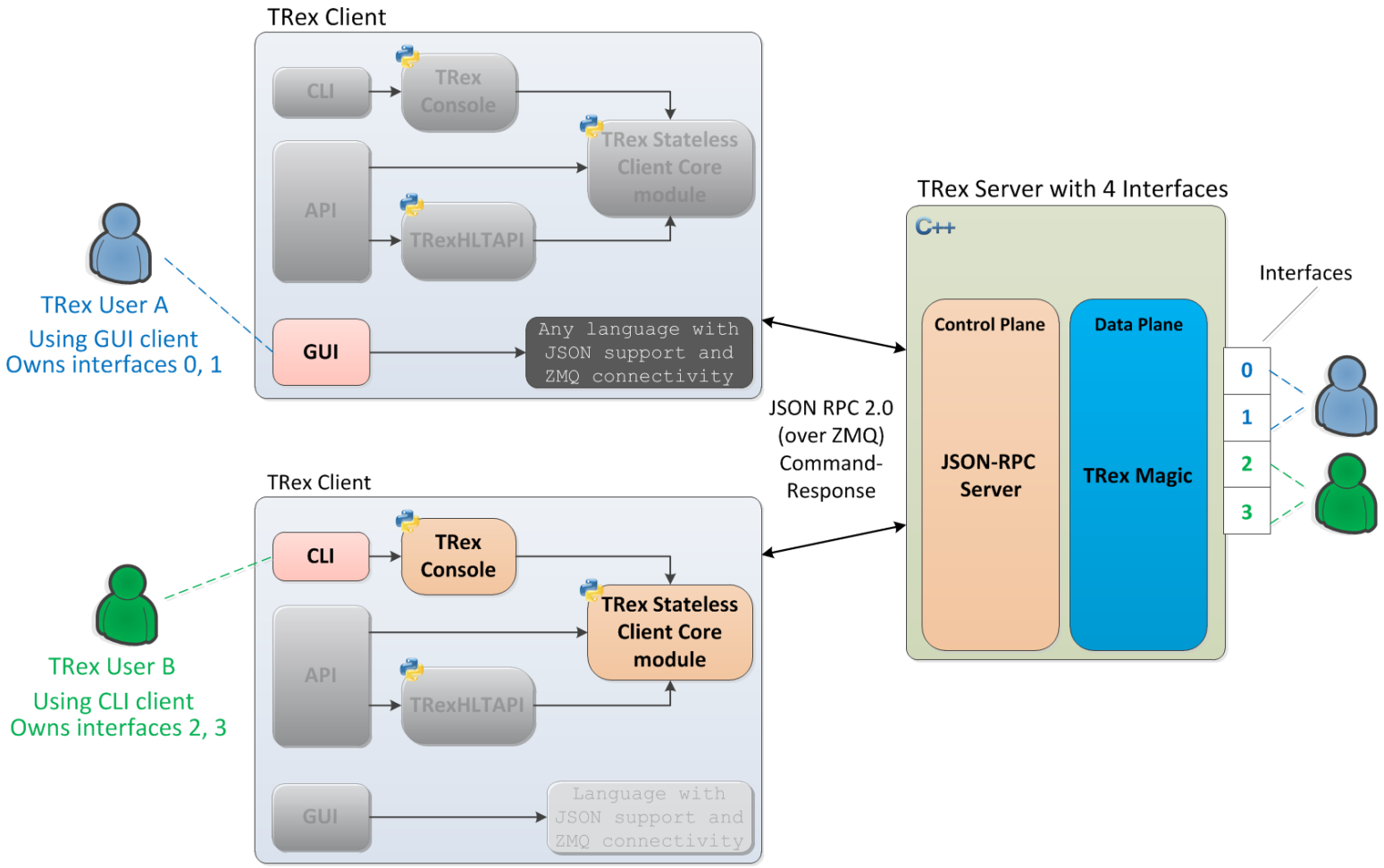


TRex Objects relations

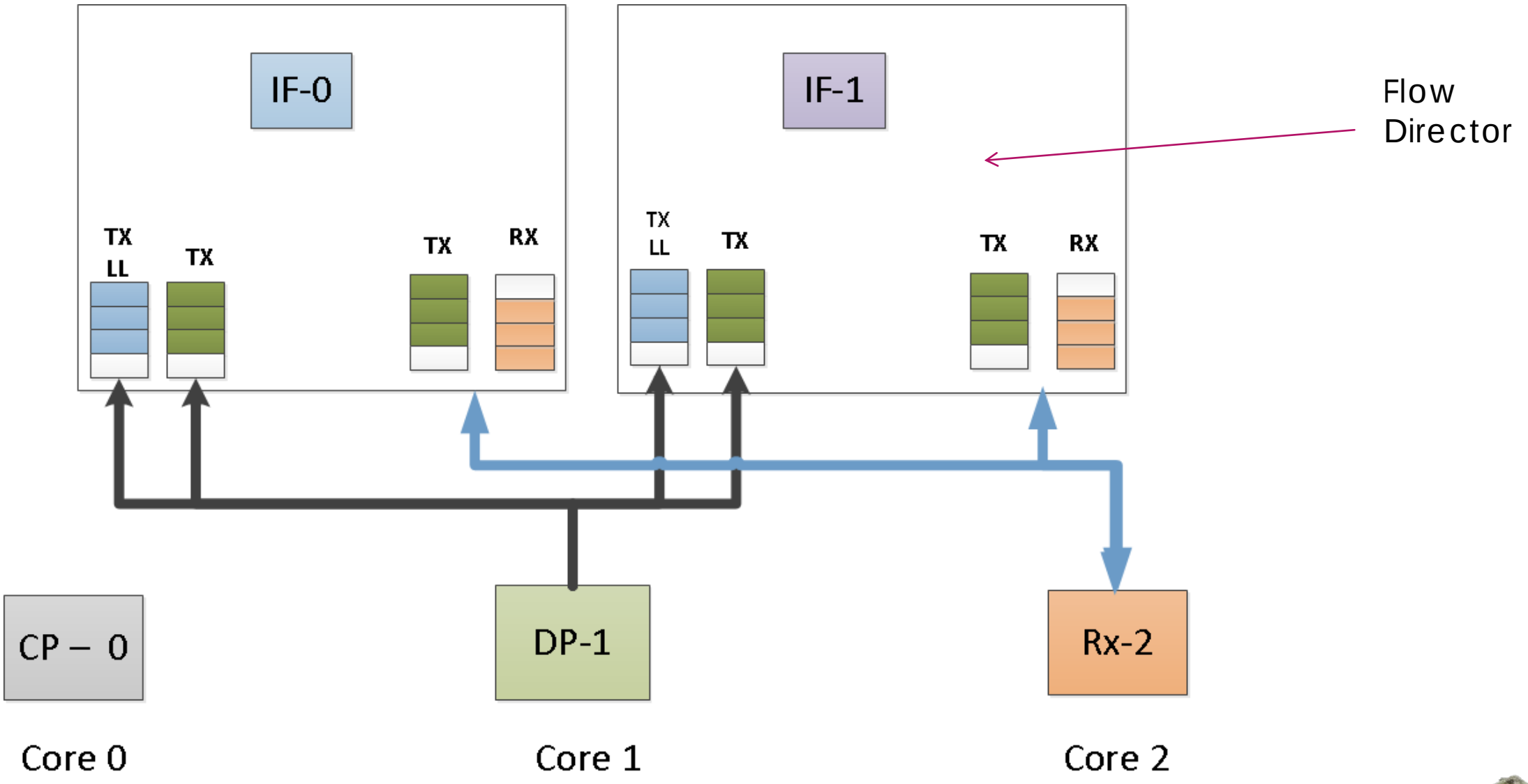


High level CP/DP

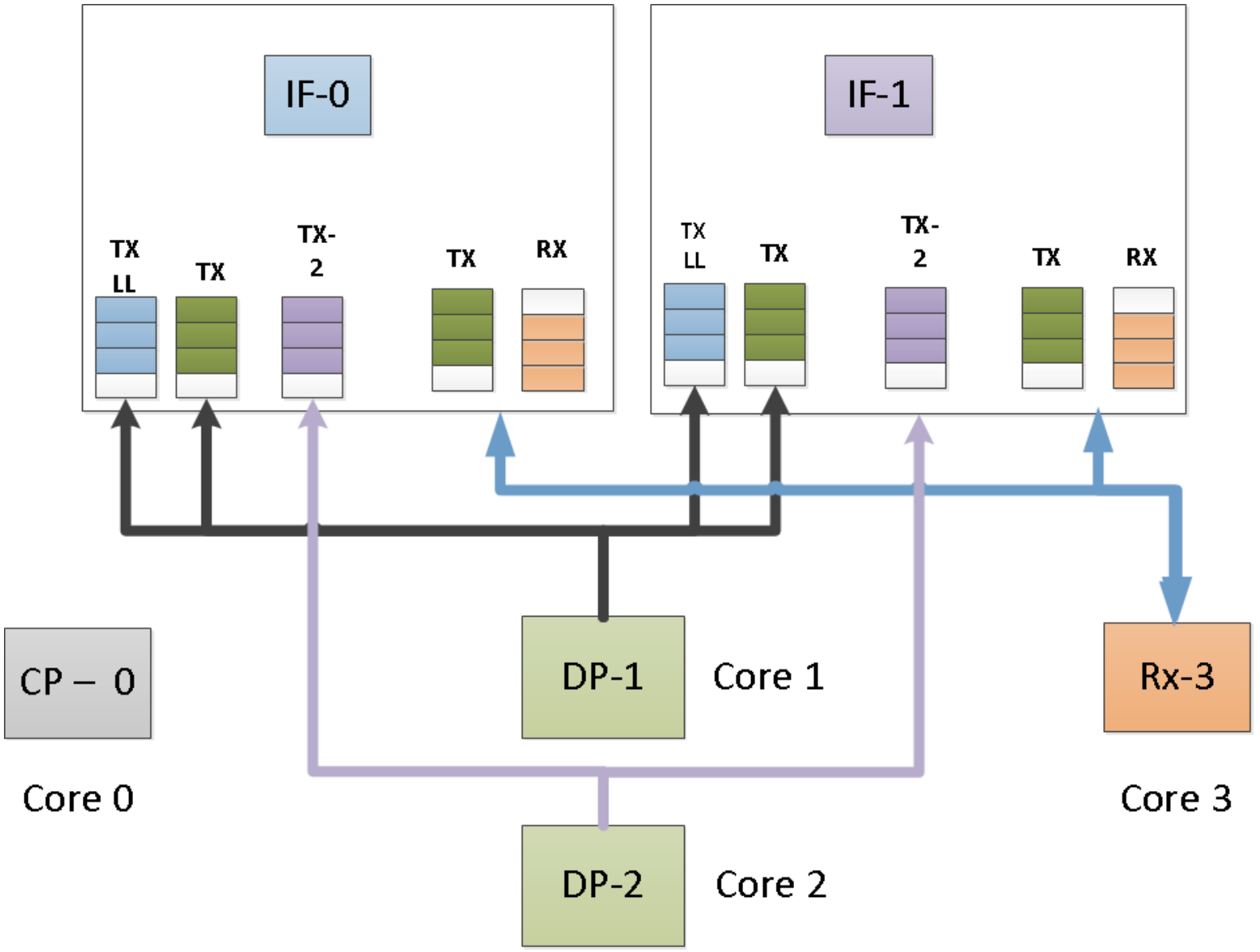
Control plane High level



TRex DP queues/cores #1 DP



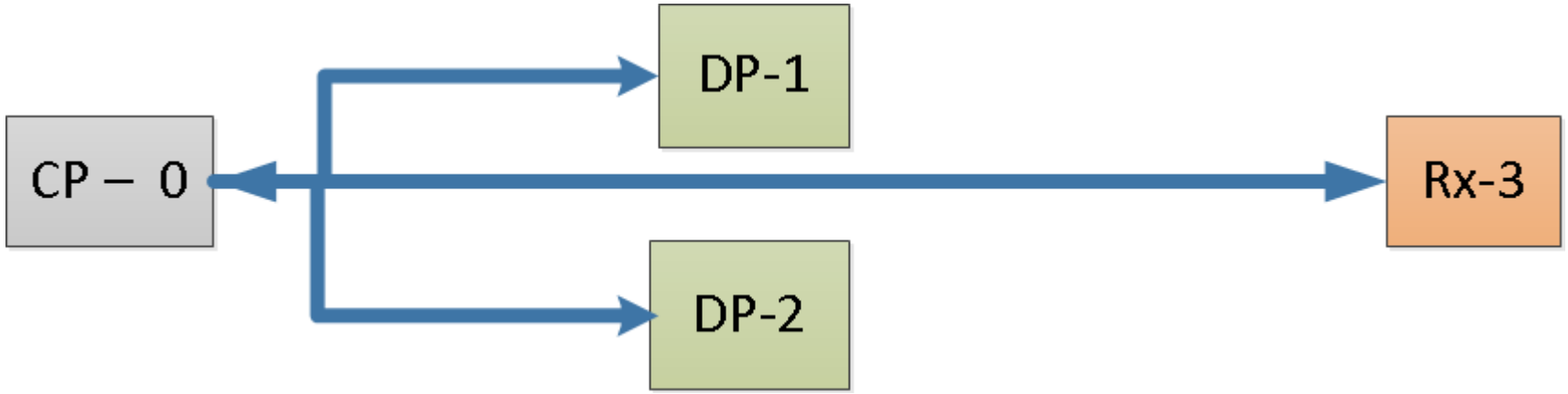
TRex DP queues/cores #2 DP



TRex
Realistic Traffic Generator

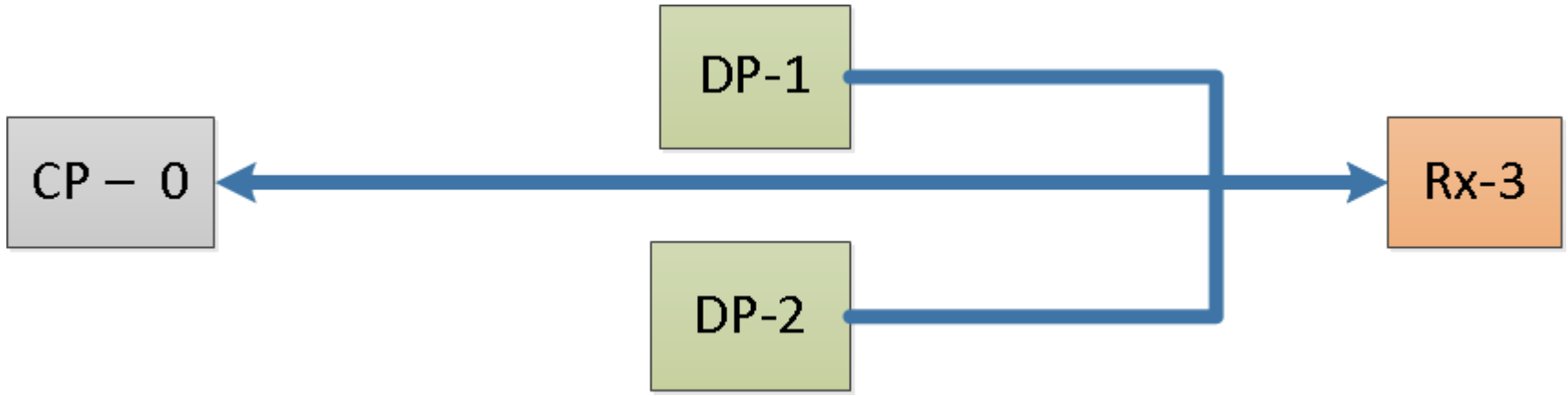
TRex CP-DP messaging

- One to many
- No locks



TRex RX-DP messaging

- One to many
- No locks

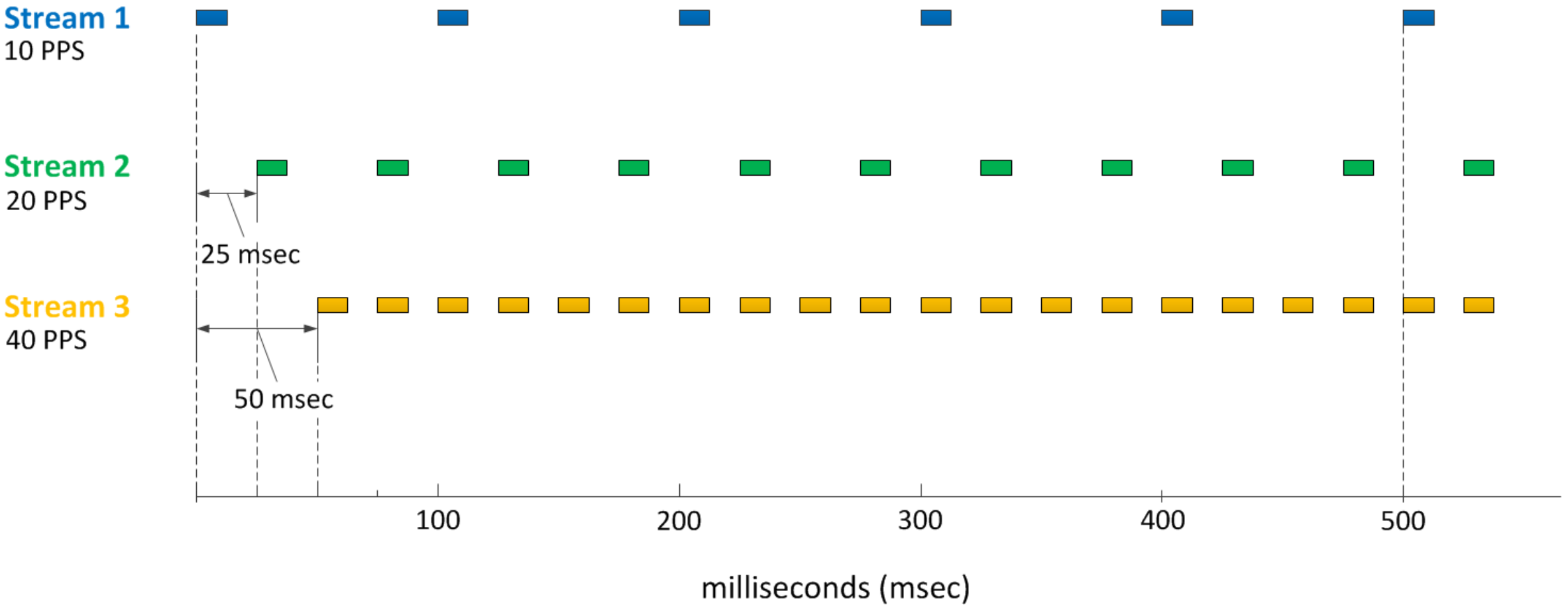


Traffic profile

One stream with two directions

```
def get_streams (self, direction = 0, **kwargs):
    # create 1 stream
    if direction==0:                                                    ❶
        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()) ]
```

Simple Interleaving streams



TRex
Realistic Traffic Generator

Simple Interleaving streams

```
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)
    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),
                                    ),

                         STLStream( isg = 25000.0, #defined in usec, 25 msec
                                    packet  = STLPktBuilder(pkt = base_pkt1/pad),
                                    mode    = STLTXCont( pps = 20),
                                    ),

                         STLStream(  isg = 50000.0,#defined in usec, 50 msec
                                     packet = STLPktBuilder(pkt = base_pkt2/pad),
                                     mode    = STLTXCont( pps = 40)

                                    )
                       ]).get_streams()
```

❶ ❷ ❸ ❹

TRex
Realistic Traffic Generator

Multi burst

Stream 0

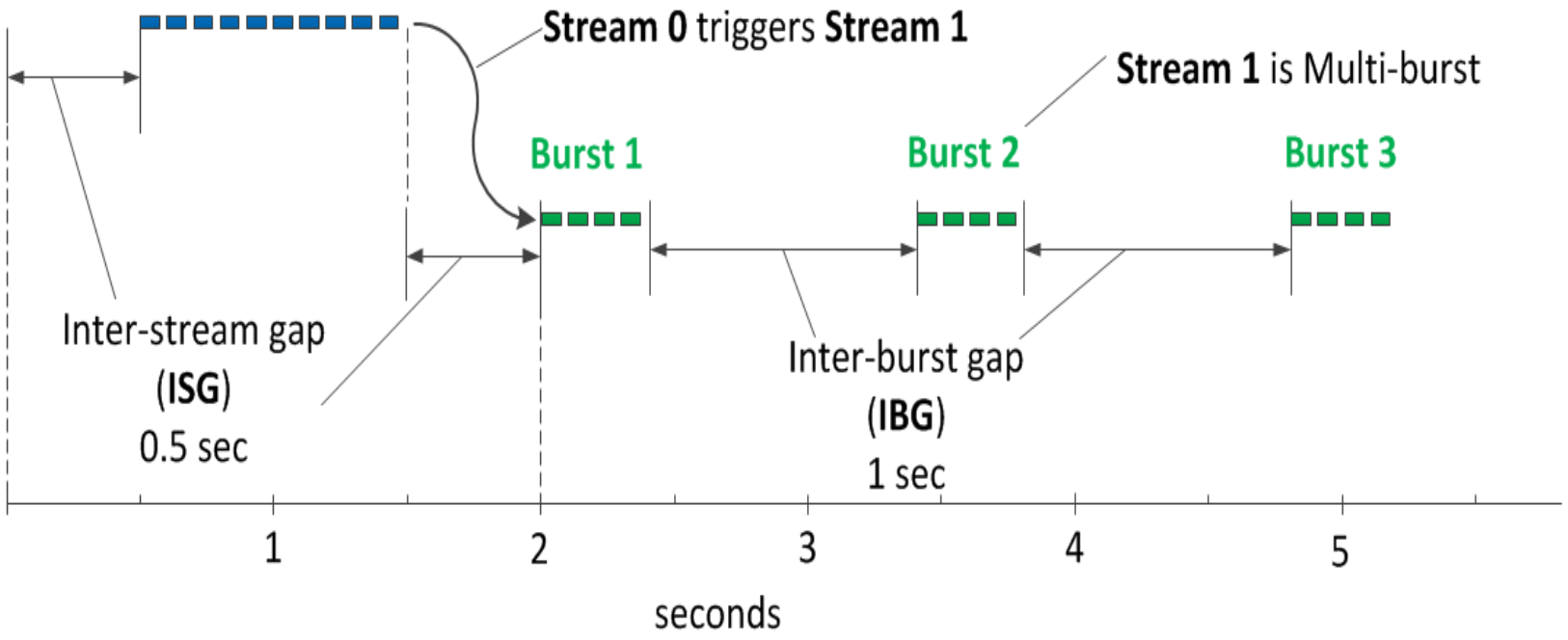
At start, inter-stream gap (**ISG**) of 0.5 sec

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 = STLTXSingleBurst( 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     = STLTXMultiBurst( pps = 1000,
                                                                    pkts_per_burst = 4,
                                                                    ibg = 1000000.0,
                                                                    count = 5)
                                        )

                            ]).get_streams()
```

Field Engine

- Flexible engine to change any field inside the packet
- Examples
  - Change TOS 1-20
  - Range/random of client IP 10.0.0.1-10.0.0.254
  - Fix TCP/UDP checksum
  - Repeatable random range
  - Packet size change 64-9k (e.g. random size)
  - Write bits

```
    def create_stream (self):

        # 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

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

                           STLVmFixIpv4(offset = "IP"), # fix checksum      ❺

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

                         ]
                       )

        pkt = STLPktBuilder(pkt = base_pkt,
                            vm = vm)

        return STLStream(packet = pkt,
                         random_seed = 0x1234,# can be removed. will give the same random value
any run
                         mode = STLTXCont())
```

Field Engine, SYN attack

TRex
Realistic Traffic Generator

Instructions

https://trex-tgn.cisco.com/trex/doc/cp_stl_docs/api/field_engine.html

Instruction type	Description	API
STLVmFlowVar	Define a variable	`STLVmFlowVar(name, init_value=None, min_value=0, max_value=255, size=4, step=1, op='inc')`
STLVmFlowVarRepetableRandom	Repeatable random var	`STLVmFlowVarRepetableRandom(name, size=4, limit=100, seed=None, min_value=0, max_value=None)`
STLVmTupleGen	Tuple generator struct	`STLVmTupleGen(name, ip_min='0.0.0.1', ip_max='0.0.0.10', port_min=1025, port_max=65535, limit_flows=100000, flags=0)`
STLVmTrimPktSize	Trim the packet size by var name	`STLVmTrimPktSize(fv_name)`
STLVmFixIpv4	Fix IPv4 header checkssum	`STLVmFixIpv4(offset)`
STLVmFixChecksumHw	Fix TCP/UDP checksum	`STLVmFixChecksumHw(l3_offset, l4_offset, l4_type)`
STLVmWrMaskFlowVar	Write a variable to bits	`STLVmWrMaskFlowVar(fv_name, pkt_offset, pkt_cast_size=1, mask=255, shift=0, add_value=0, offset_fixup=0, is_big=True)`
STLVmWrFlowVar	Write a variable	`STLVmWrFlowVar(fv_name, pkt_offset, offset_fixup=0, add_val=0, is_big=True)`

TRex
Realistic Traffic Generator

Per stream statistics

- Implemented using hardware assist with Intel X710/XL710 NIC flow director rules
- With other NICs (Intel I350, 82599), implemented in software.

```
class STLS1(object):

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

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

TRex
Realistic Traffic Generator

Per stream statistics -TUI

Streams Statistics

PG ID	12	7	
Tx pps	5.00 Kpps	999.29 pps	#❶
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	#❷
Rx bps	N/A	N/A	#❸
----			
opackets	222496	44500	
ipackets	222496	44500	
obytes	131272640	2670000	
ibytes	N/A	N/A	#❸
-----			
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	#❸

TRex
Realistic Traffic Generator

Per stream latency/jitter

- Base on per stream stats hardware assist
- Forward specific type of packets
- Filter is based on IPV4.ID and IPv6.flow_id
- 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))
                ]
```

TRex
Realistic Traffic Generator

Latency/Jitter-TUI

```
Connection   : csi-kiwi-02, Port 4501
Version      : v2.09, UUID: N/A
Cpu Util.    : 0.01% @ 8 cores (4 per port)
Rx Cpu Util. : 0.03%
Async Util.  : 0.89% / 2.58 KB/sec
             :
```

Latency Statistics (**usec**)

PG ID	12	13	14	15
TX pkts	97922	97922	97922	97922
RX pkts	97922	97922	97922	97922
Max latency	**37**	**35**	**38**	**43**
Avg latency	**22**	**22**	**22**	**22**
-- Window --				
Last (max)	28	24	30	29
Last-1	28	24	30	30
Last-2	29	24	30	30
Last-3	29	24	30	30
Last-4	35	24	24	30
Last-5	24	23	24	30
Last-6	24	23	23	34
Last-7	28	24	29	30
Last-8	29	24	30	30
Last-9	29	31	30	30

Jitter	0	0	0	0

Errors	0	0	0	0

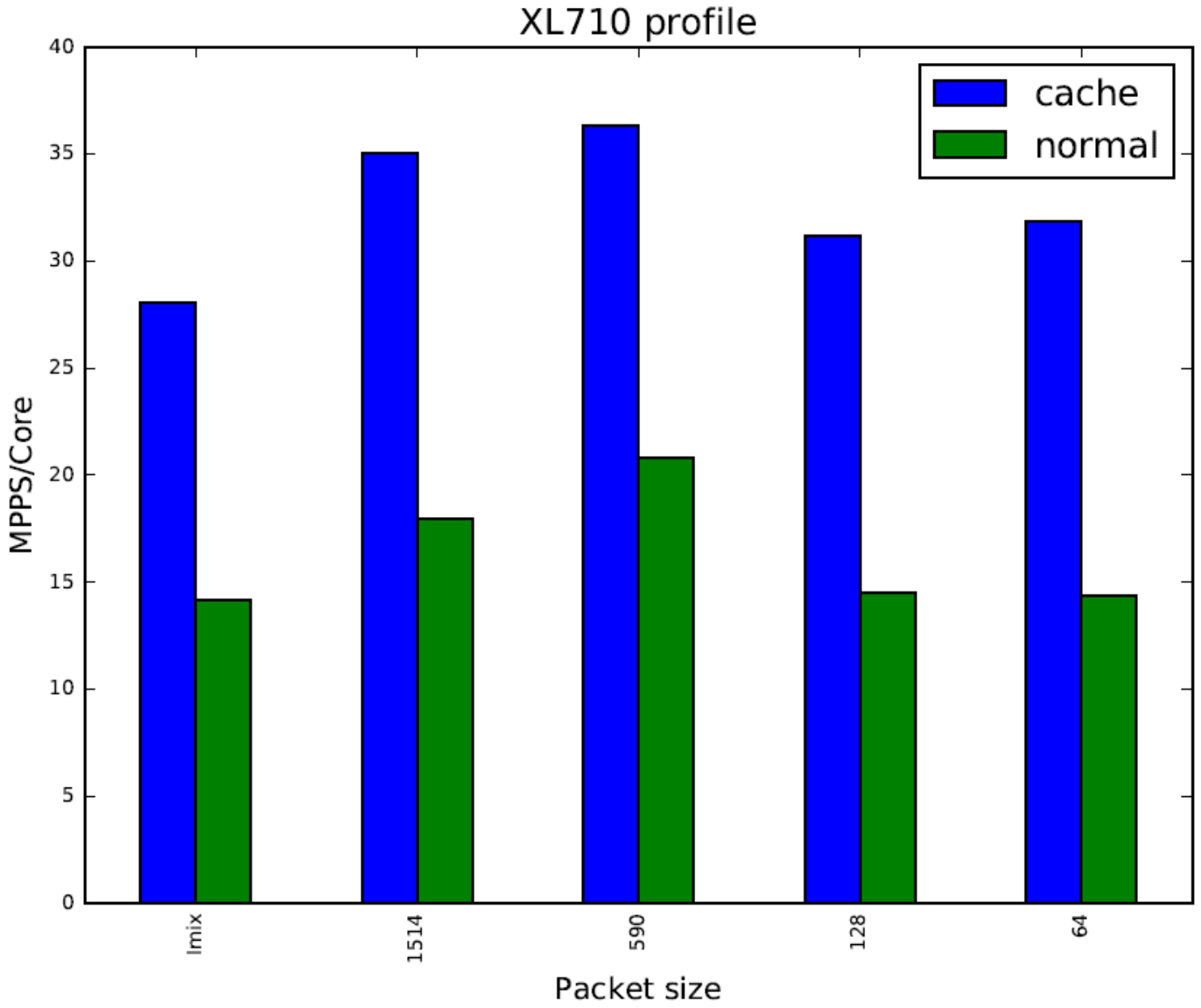
TRex
Realistic Traffic Generator

Performance

Performance setup

Server:	UCSC-C240-M4SX
CPU:	2 x Intel® Xeon® CPU E5-2667 v3 @ 3.20GHz
RAM:	65536 @ 2133 MHz
NICs:	2 x Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev 01)
QSFP:	Cisco QSFP-H40G-AOC1M
OS:	Fedora 18
Switch:	Cisco Nexus 3172 Chassis, System version: 6.0(2)U5(2).
TRex:	v2.09 using 7 cores per dual interface.

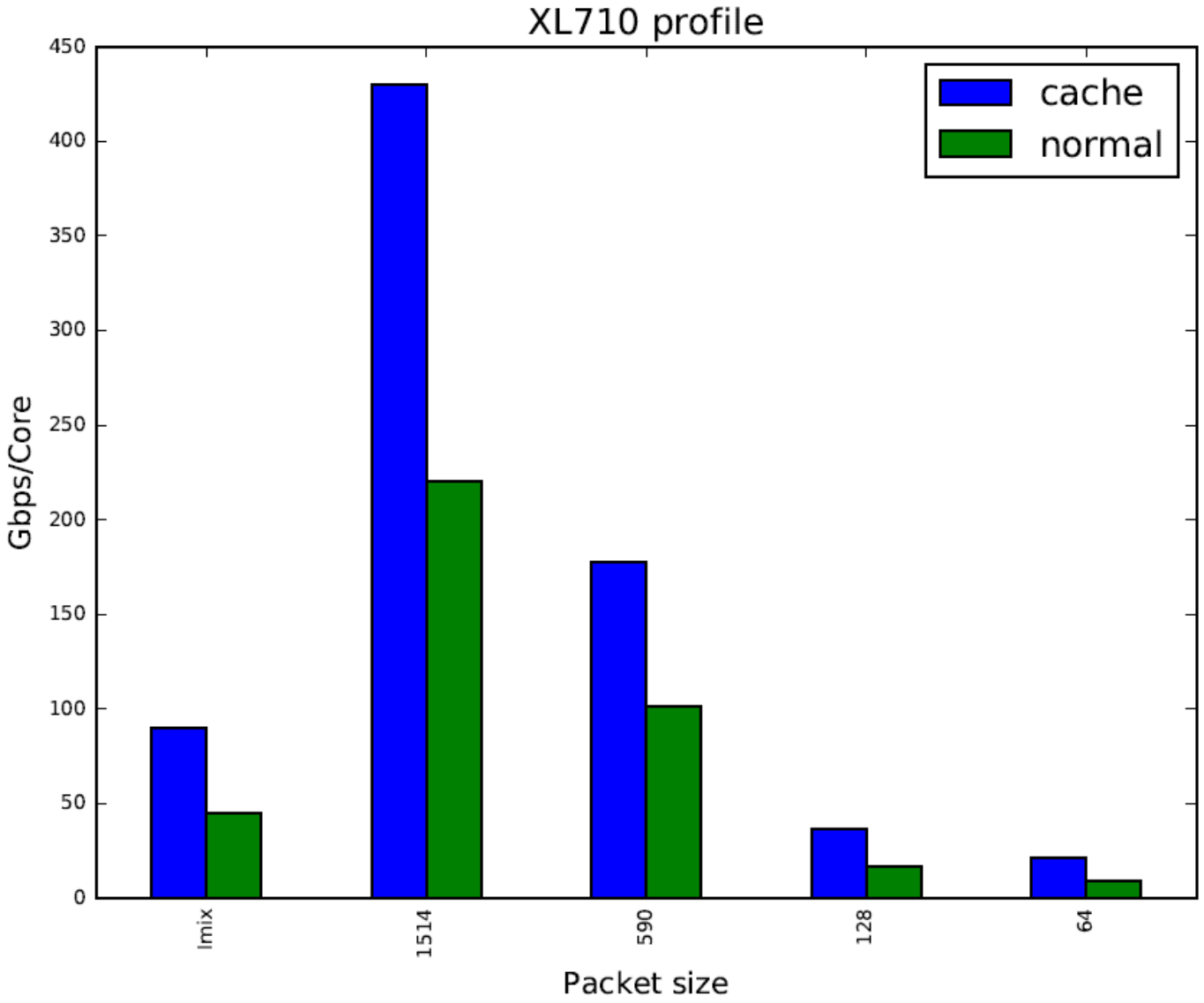
Performance XL710 MPPS/Core



link

TRex
Realistic Traffic Generator

Performance XL710 gbps/core



link

TRex
Realistic Traffic Generator

Automation

Simple 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()
```

TRex
Realistic Traffic Generator

```
hltapi = CTRexHltApi()
res = hltapi.connect(device = args.ip, port_list = [0, 1],
                     reset = True, break_locks = True)

res = hltapi.traffic_config(mode = 'create',
                            bidirectional = True,
                            port_handle = ports[0],
                            port_handle2 = ports[1],
                            frame_size = args.frame_size,
                            mac_src = args.src_mac, mac_dst = args.dst_mac,
                            mac_src2 = args.dst_mac, mac_dst2 = args.src_mac,
                            l3_protocol = 'ipv4',
                            ip_src_addr = '10.0.0.1',
                            ip_src_mode = 'increment',
                            ip_src_count = 254,

                            ip_dst_addr = '8.0.0.1',
                            ip_dst_mode = 'increment',
                            ip_dst_count = 254,

                            l4_protocol = 'udp',
                            udp_dst_port = 12,
                            udp_src_port = 1025,

                            stream_id = 1,
                            rate_pps = args.rate_pps,
                            )
res = hltapi.traffic_control(action = 'run', port_handle = ports[:2])
wait_with_progress(args.duration)
res = hltapi.traffic_control(action = 'stop', port_handle = ports[:2])
res = hltapi.traffic_stats(mode = 'aggregate', port_handle = ports[:2])
res = hltapi.cleanup_session(port_handle = 'all')
```

Python HLTAPI Automation

TRex
Realistic Traffic Generator

Interaction

Console

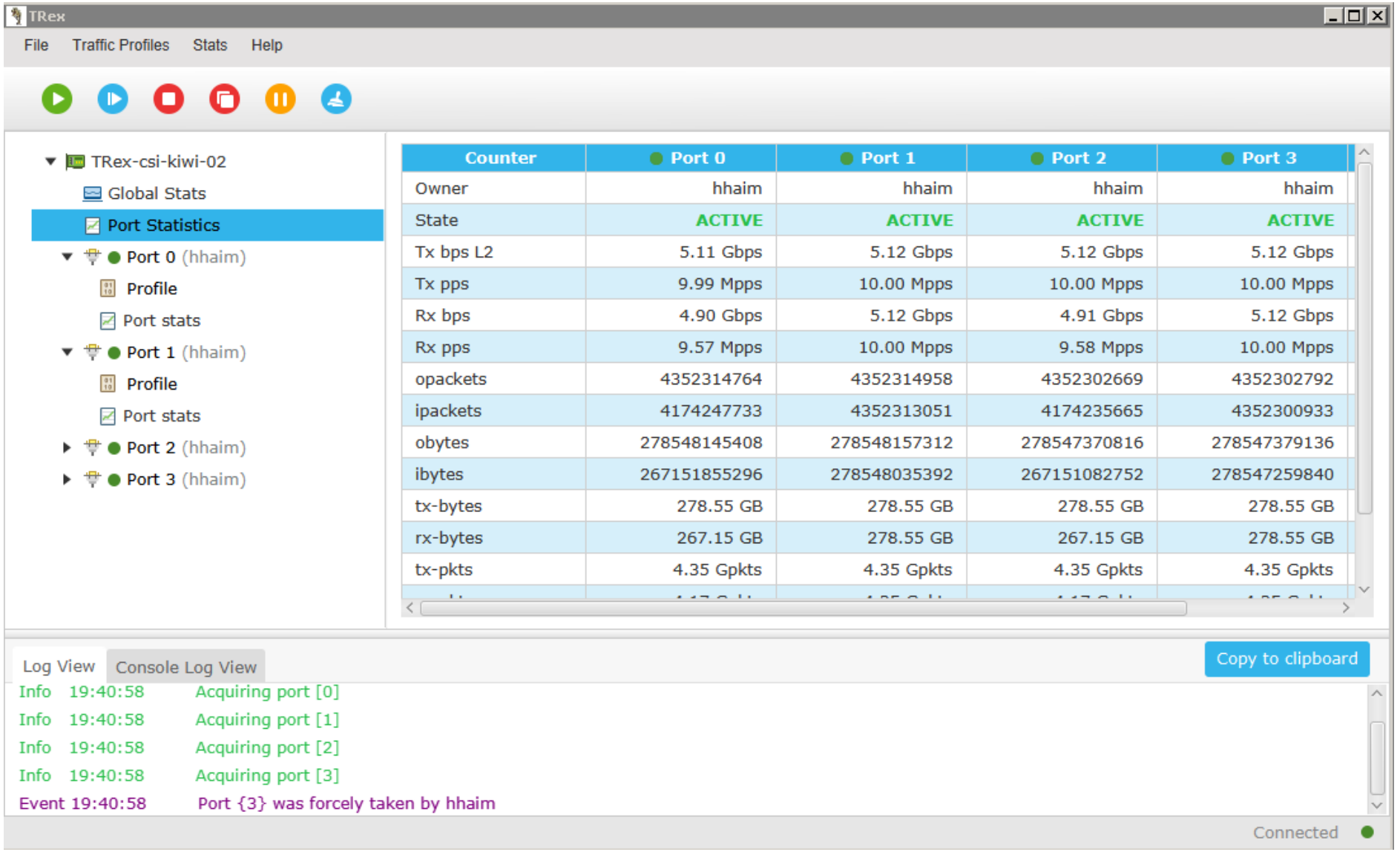
```
#load the trex as a server for interactive mode
$sudo ./t-rex-64 –i

#connect to the server from any server ( Python 2/3.4)
$./trex-console
```

Shell

```
#start traffic on all port
>start -a -m 1 -f stl/imix_1pkt.py

#pause traffic on all port
>pause -a

#resume traffic on all port
>resume -a

#stop traffic on all port
>stop -a

#show dynamic statistic
>tui

#show port statistic
>stats –p

#clear statistic
>clear

#show stream statistic
>streams
```

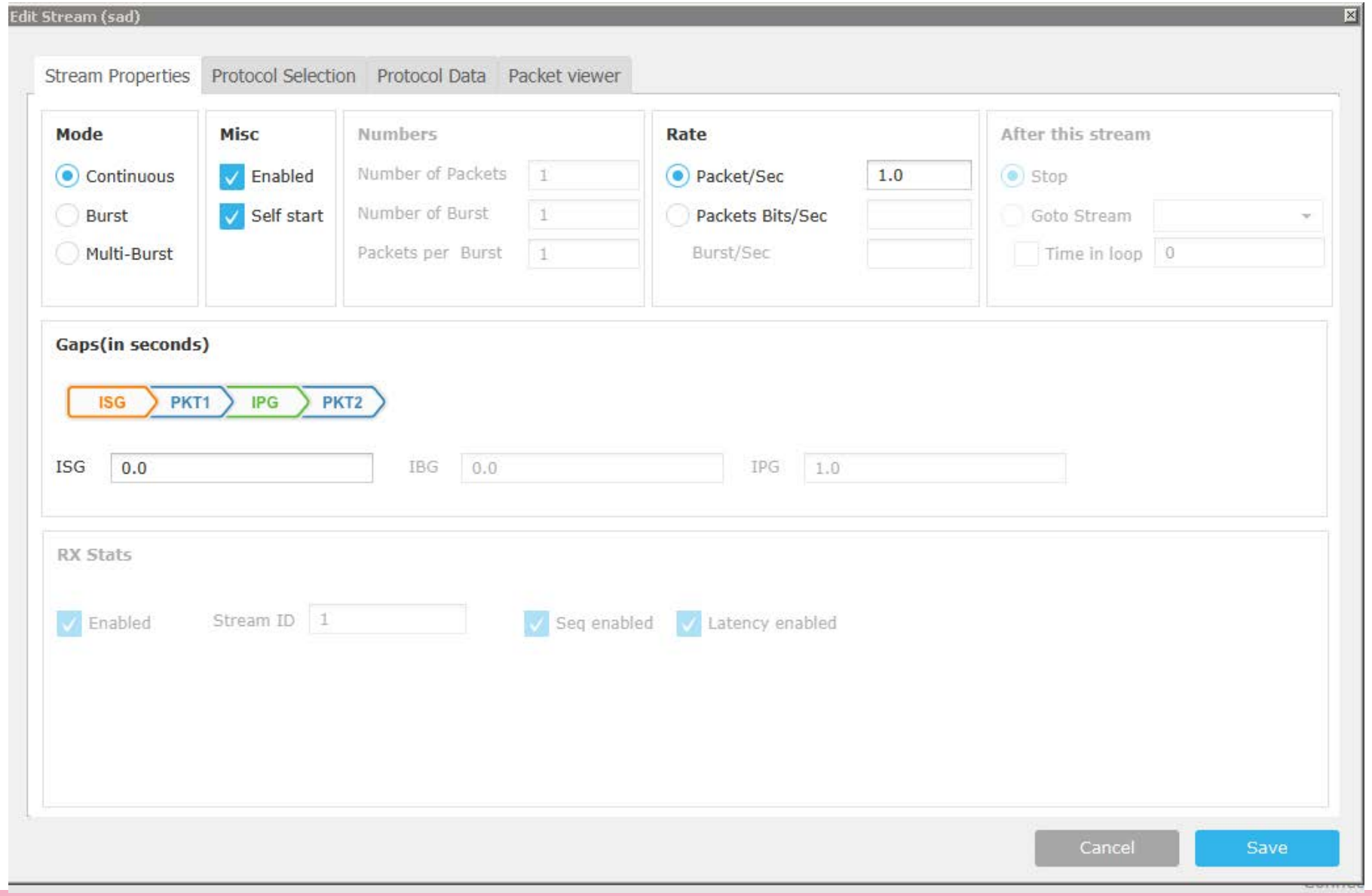
Console

TRex Stateless GUI



TRex
Realistic Traffic Generator

TRex Stateless GUI – Stream builder



More info

- Stateless manual
- TRex documents Index
- GitHub

TRex
Realistic Traffic Generator

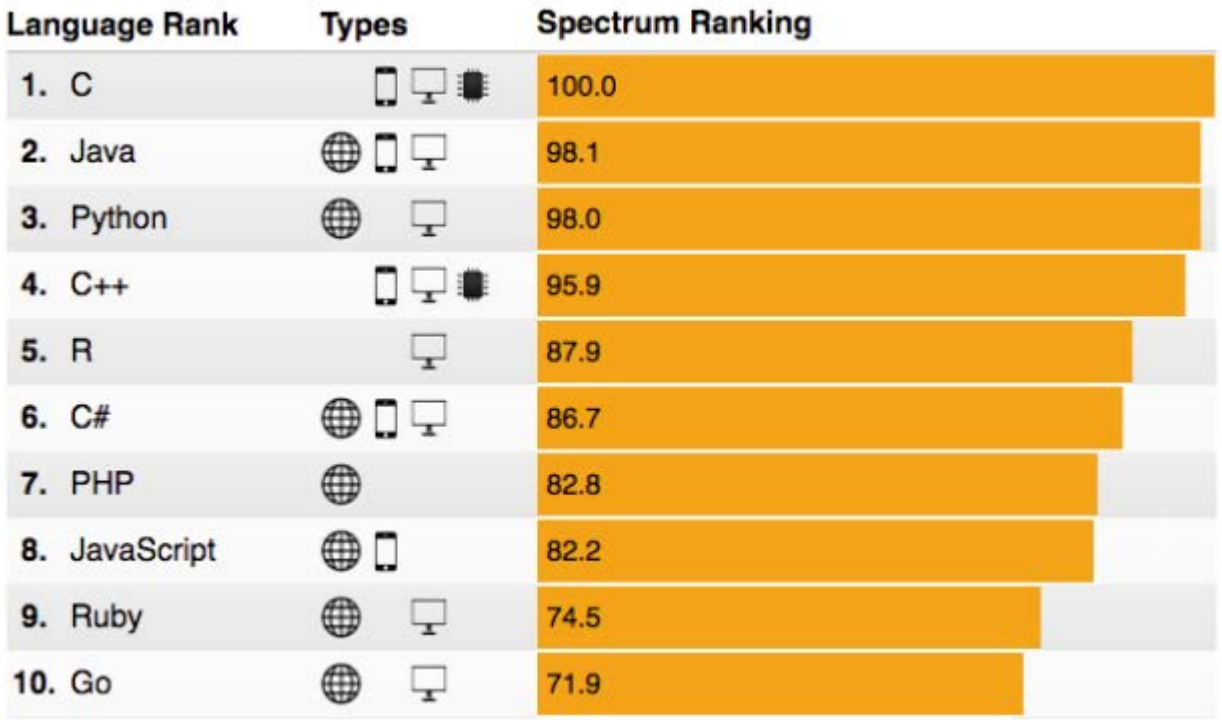
Questions?

Hanoch Haim

hhaim@cisco.com

Requirements - HLPAPI /TCL

By IEEE Spectrum's TCL is not part of the top 10 language



Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		98.0
4. C++		95.9
5. R		87.9
6. C#		86.7
7. PHP		82.8
8. JavaScript		82.2
9. Ruby		74.5
10. Go		71.9

TRex
Realistic Traffic Generator

HLTAPI

Should be common API for IXIA/Spirent

- There are a lot of exceptions
- Mixing packet crafting and mode of sending
- IXIA has 2000 pages of HLTAPI implementation details

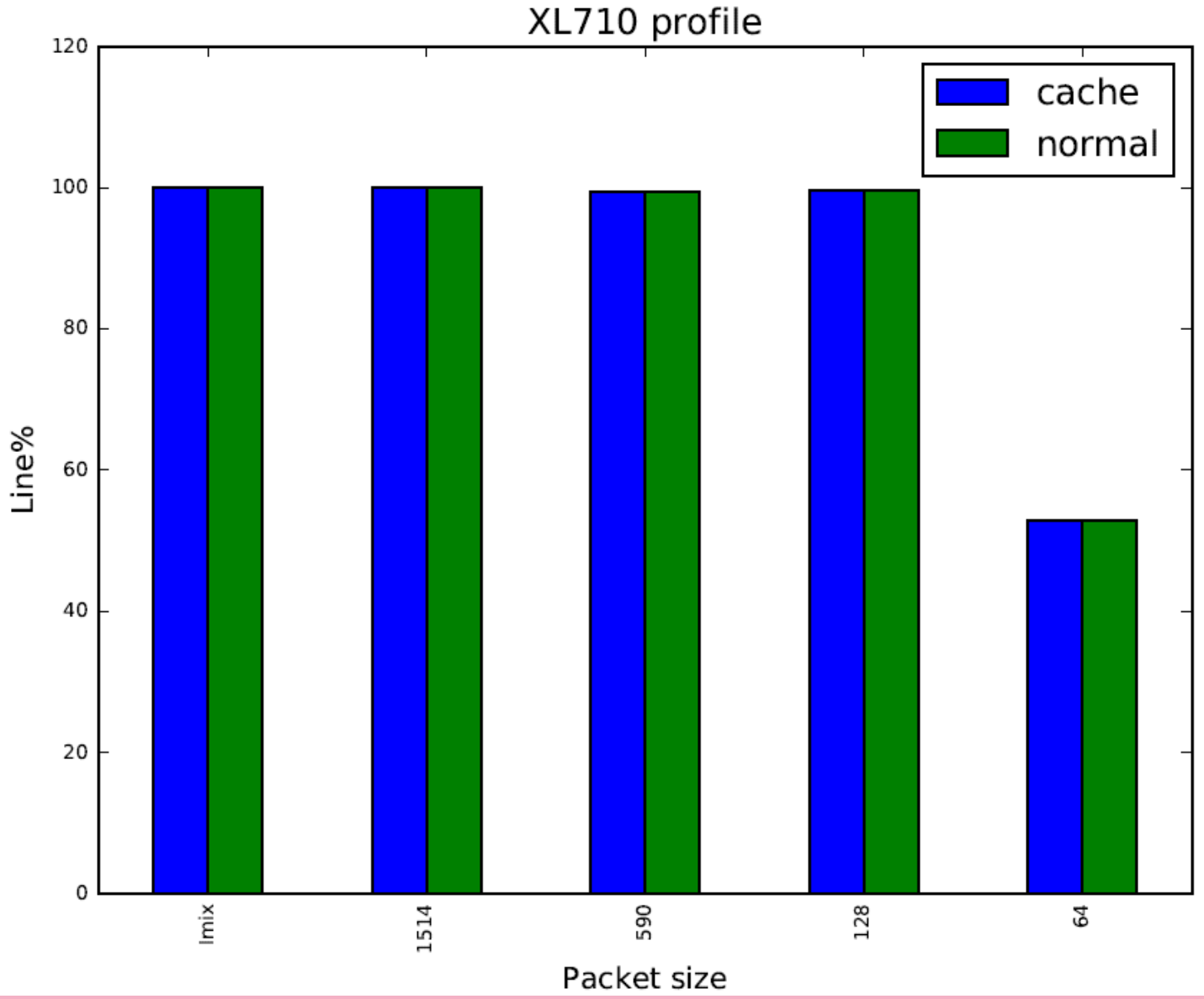
HLTAPI

```
res = hltapi.traffic_config(mode = 'create',
                            bidirectional = True,
                            port_handle = ports[0],
                            port_handle2 = ports[1],
                            frame_size = args.frame_size,
                            mac_src = args.src_mac, mac_dst = args.dst_mac,
                            mac_src2 = args.dst_mac, mac_dst2 = args.src_mac,
                            l3_protocol = 'ipv4',
                            ip_src_addr = '10.0.0.1',
                            ip_src_mode = 'increment',
                            ip_src_count = 254,

                            ip_dst_addr = '8.0.0.1',
                            ip_dst_mode = 'increment',
                            ip_dst_count = 254,

                            l4_protocol = 'udp',
                            udp_dst_port = 12,
                            udp_src_port = 1025,

                            stream_id = 1,
                            rate_pps = args.rate_pps,
                            )
```

- Packet definition
- Field Engine
- Tx Mode

- Bloat of API arguments

Performance XL710 Line%



link

- XL710 support line rate from 128B

Simple load profile

```
try:
    # connect to server
    c.connect()

    my_ports=[0,1]

    # prepare our ports
    c.reset(ports = my_ports)

    profile_file =   "../../../../stl/udp_1pkt_simple.py"   # a traffic profile file

    try:                                                    # load a profile
        profile = STLProfile.load(profile_file)
    except STLError as e:
        print format_text("\nError while loading profile '{0}'\n".format(profile_file), 'bold')
        print e.brief() + "\n"
        return

    print profile.dump_to_yaml()                            # print it as YAML

    c.remove_all_streams(my_ports)                          # remove all streams


    c.add_streams(profile.get_streams(), ports = my_ports)  # add them

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

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

TRex
Realistic Traffic Generator