Offloading Online MapReduce tasks with Stateful Programmable Data Planes

Valerio Bruschi, Marco Faltelli, Angelo Tulumello, Salvatore Pontarelli, Francesco Quaglia, Giuseppe Bianchi

Scenario

- CPUs are at a standstill
  - Moore's law, Dennard Scaling…
- Now more than ever, we need acceleration!
- A new architectural approach is on the rise
- Domain Specific Architectures:
  - Tailored to a specific domain of applications
  - Programmable and power efficient!
  - Examples: Google's TPUs, GPUs, FPGAs

The networking perspective

- **From:** The network is «just plumbing»*
  - We still teach grad students the end to end principle [Saltzer, Redd & Clark, 1981]

- **To:** New classes of (smart!) switches
  - Fast (12.8 Tbps!)
  - Programmable
  - Power efficient

* Source: R. Soulè, in SIGCOMM’18

New trends

- An opportunity to co-design data centers applications with modern HW! [Caulfield, Costa and Ghobadi, HPSR’18]
- Some tasks could be offloaded to dedicated HW…
- … while keeping the most complex logics in general purpose CPUs
- Does this vision work? Recent works say so!
  - 10.000x improvement in throughput [NetPaxos, SOSR’15]
  - 5x gain in power consumption [FlowBlaze, NSDI’19]

This work in short

- We investigate the opportunity to offload MapReduce tasks to stateful data planes
- We find the common requirements for MR tasks to perform well on programmable HW
- We find out programmable data planes can achieve low latency, low congestion processing
- We validate our approach through a HTTP traffic use case

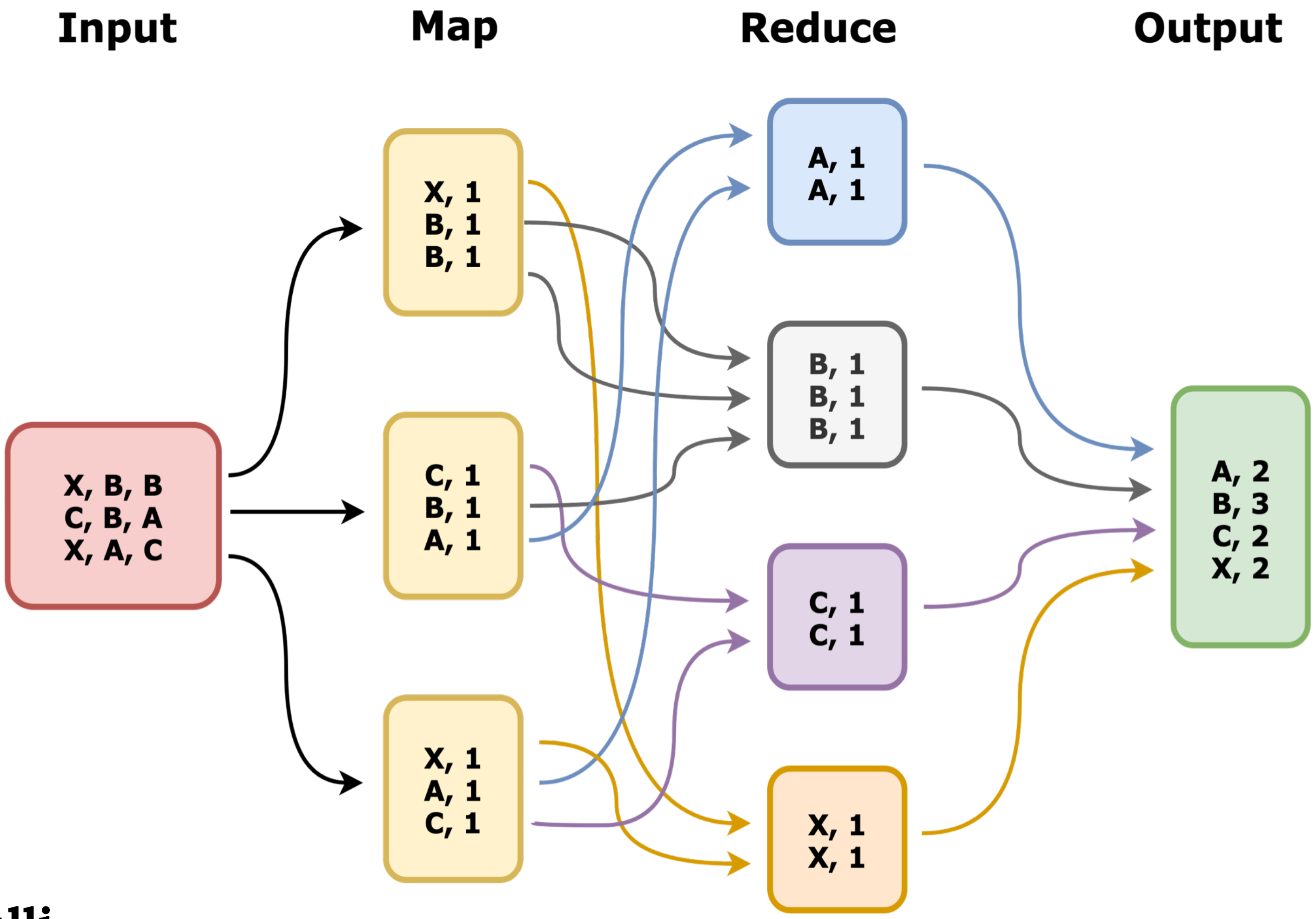
Background: MapReduce

- A programming model proposed by Google [OSDI '04]
- Users define Map() and Reduce() functions
- Goals:
  - process huge amounts of data
  - in a distributed fashion (divide-and-conquer style)
- Newer programming models…
- … Are no more than a superset of the MapReduce one!

Background: MapReduce(2)

- Map():
  - processes a generic input and generates intermediate <*key, value*> pairs
  - Multiple Map() instances, each receiving a split of the incoming data as input
- Reduce():
  - merges intermediate values associated within the same key
  - Multiple Reduce() instances, each receives a partition of the key space

A toy example: WordCount



MapReduce on programmable HW

- Can we port any MapReduce task to networking HW?
  - No way!
- We can rather identify a subset of simple (yet meaningful!) offloading-amenable tasks for data plane HW
- What for?
  - Low latency processing (very few ns), low variability
  - In-network aggregation reduces congestion
  - Free CPU cycles
- Let’s get into details…

HW-MapReduce: requirements

- **Map():**
  - We need to restrict the possible <key, value> pairs
  - Programmable HW handles well packet headers
  - Solution: we use a programmable parser

Stateless!

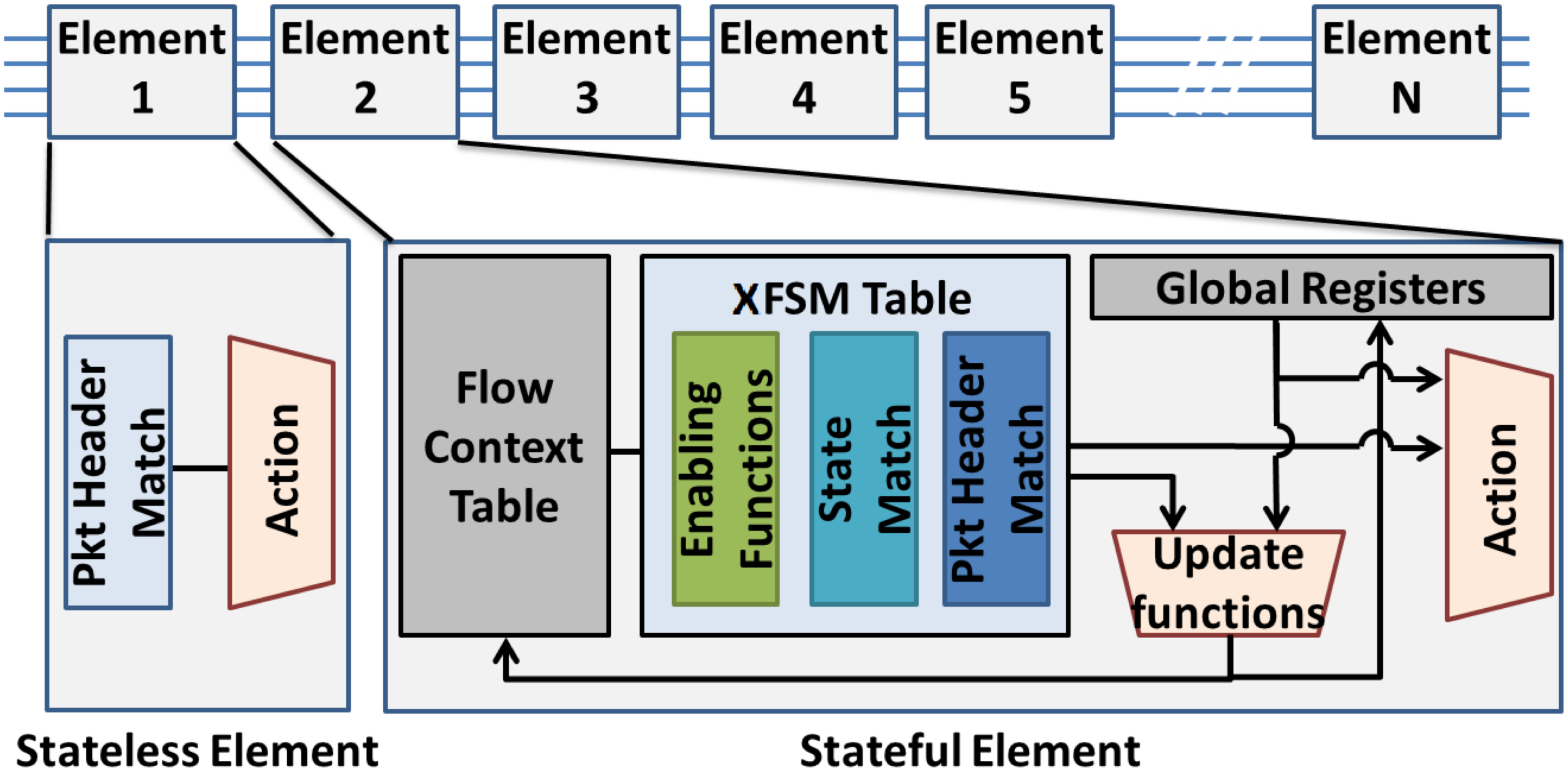
- **Reduce():**
  - Devices must perform at line rate, few operations allowed!
  - No loops allowed
  - Small per-flow memory footprint (very few registers)
    - Associative & commutative operations (mean, sum, max)… OK!

Stateful!

Is there a HW-MapReduce executor?

- Yes! FlowBlaze [NSDI'19], a stateful programmable data plane
- Developed as a NF accelerator for both SW and SmartNICs
- A pipeline of stages:
  - Stateless (match-action table)
  - Stateful (Per-flow EFSM functionality)
- Processing restricted to a few clock cycles (i.e. nanoseconds!)
  - Corresponding SW executors are bounded to milliseconds
  - Many applications need strict real-time requirements (e.g. in High Frequency Trading, every microsecond can make the difference!)

FlowBlaze overview



Why not P4?

- 2 ways proposed to manage stateful functionalities in P4:

1. New flow insertion driven by the control plane:
    - When a new flow arrives, the packet is forwarded to the control plane
    - Increased latency
    - Consistency issues between packet arrival and rule insertion
2. Hash-based selection:
    - Register array index is selected through a hash function
    - No easy way to resolve collisions! (use case depending)

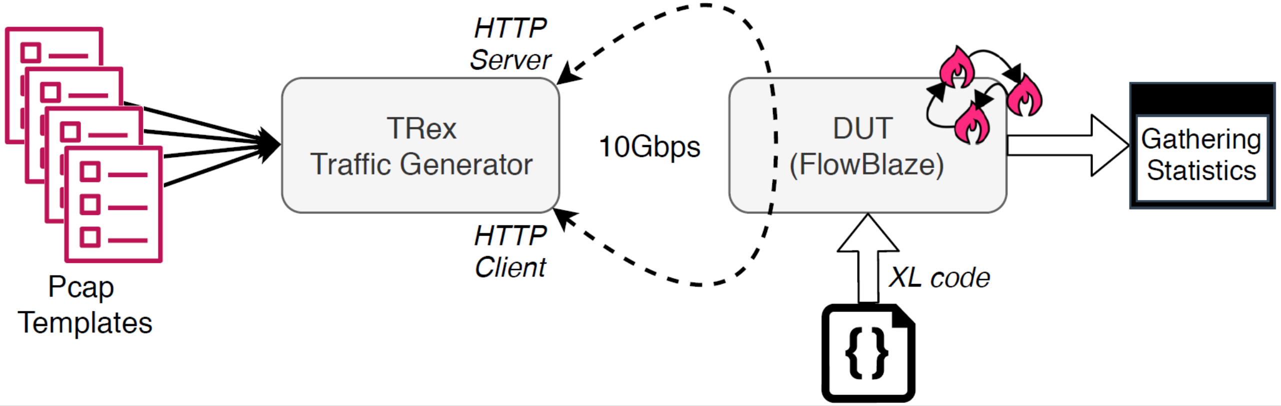
- FlowBlaze manages collisions transparently for the user!

Network Placement

- MapReduce massively expoits parallelism on many nodes
- We propose the same architecture distributing the FlowBlaze nodes in the network
  - E.g. in a fat-tree data center topology
- What if a few HW devices are available?
  - We can route traffic to the FlowBlaze instance
  - Or, we could use FlowBlaze as a SmartNIC endpoint

Preliminary results

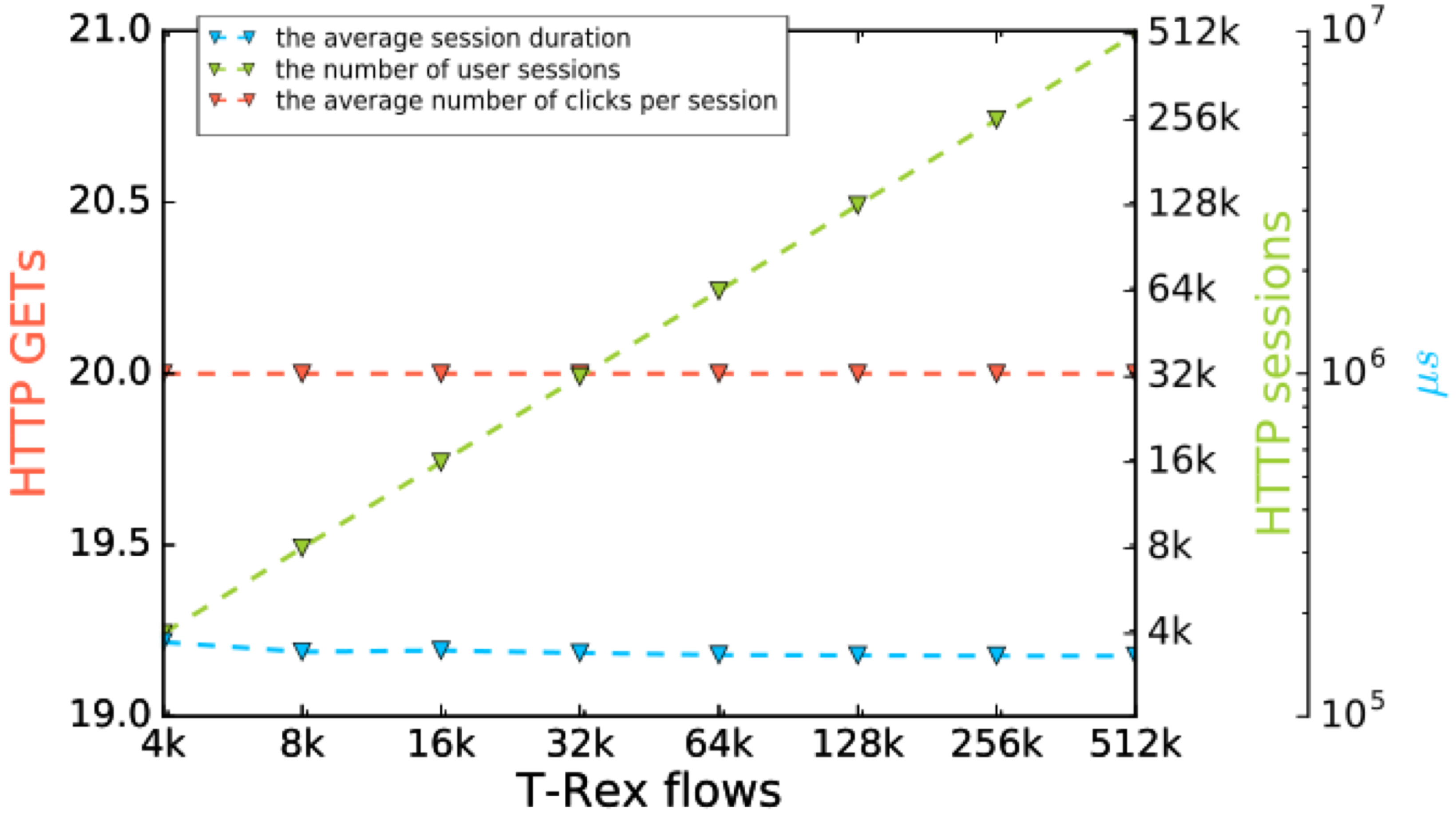
- A click-stream HTTP traffic analysis [Yu @SOSP'09]
- The MapReduce task snoops packets and computes three different metrics:
  - The number of user sessions (group by TCP 5-tuple & count)
  - Average clicks per session (group by 5-tuple & HTTP.GET count)
  - Average session duration (group by 5-tuple & avg session time)
- We used the FlowBlaze SW implementation and the Trex traffic generator



Preliminary results: workload scaling

Trex parameters:
- 20 HTTP.GET requests per session
- 140 ms average session time

Saturating a 10Gb link, no losses.
Single CPU @2.1GHz

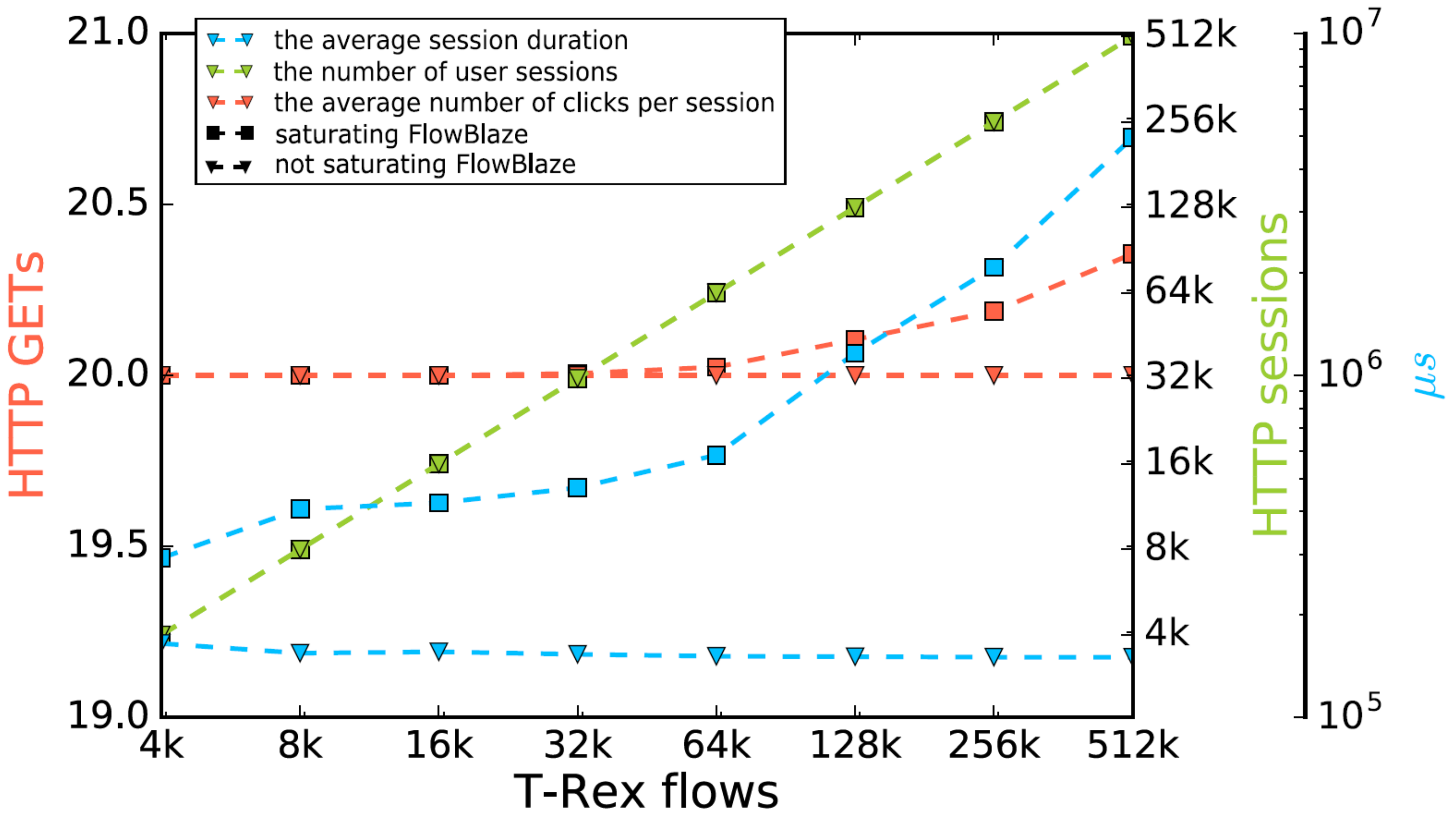


Preliminary results: workload scaling

Trex parameters:
- 20 HTTP.GET requests per session
- 140 ms average session time

Saturating a 10Gb link

Single CPU @1.8GHz



Future work

- Integrate the XL toolchain in a MapReduce environment
- Implement a wider range of partition/aggregation applications
- Execute multiple MapReduce tasks on the same HW concurrently
    - FlowBlaze as a multitenancy Function-as-a-service (FaaS) device
- Compare FlowBlaze and P4 through the P4→NetFPGA workflow

Thank you for your attention!