High-speed stateful packet processing
Network Functions Are Pervasive

Firewall

NAT

Load-Balancer
Network Functions Are Pervasive

Network Functions Virtualization is an essential architectural paradigm of today’s networks
Network speeds have been increasing dramatically.
Talk goal

How to handle stateful network functions for terabit per second of traffic?

- High-speed stateful software packet processing
- Switch-assisted stateful packet processing
- NIC-assisted stateful packet processing
- NFV service chain combined stateful packet processing
Connection tracking is about classifying packets in micro-flows
Connection Tracking

Connection tracking is about classifying packets in micro-flows
Connection Tracking in a (stateful) Load Balancer

Send all the flow’s packets to the same server

<table>
<thead>
<tr>
<th>5-tuple</th>
<th>Flow</th>
<th>Selected server</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0.1.1:8888-&gt;100.100.100.100:80, TCP</td>
<td>1</td>
<td>Server #3</td>
</tr>
<tr>
<td>100.0.1.2:9999-&gt;100.100.100.100:80, TCP</td>
<td>2</td>
<td>Server #1</td>
</tr>
</tbody>
</table>
Challenges in High-speed Connection Tracking

- On a 100 Gbps link, packets arrive every 6.72 ns
- A DRAM access takes ~100 ns

Data structures must leverage CPU caches
How do we build 100GbE+ software stateful Network Functions?
Hash Tables (HT) in a nutshell
Hash Tables (HT) in a nutshell

Perfect Hash Functions are difficult to implement

HT implementations differ for collision handling
Chaining Hash Tables

Lists can grow infinitely!
Cuckoo Hash Tables

Index Data

Under high load, long swapping chain

Constant time lookups

Primary Location

Secondary Location

<table>
<thead>
<tr>
<th>Index</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>11.11.1.1...</td>
</tr>
<tr>
<td>03</td>
<td>33.33.3.3...</td>
</tr>
<tr>
<td>04</td>
<td>44.44.4.4...</td>
</tr>
<tr>
<td>05</td>
<td></td>
</tr>
</tbody>
</table>

22.22.2.2:2222->100.100.100.100:80, TCP

44.44.4.4->100.100.100.100:80, TCP
Testbed

- 2 server machines connected back-to-back
- Mellanox ConnectX-5 @ 100Gbps
- FastClick Stateful Load Balancer configuration
- Traces captured at our campus and CAIDA
- Multiple parallel replays

[HPSR’21] Girondi, M Chiesa, T Barbette
Studied Hash Tables

N. Le Scouarnec, Cuckoo++ Hash Tables in ANCS 2018
M. Herlihy et al., Hopscotch hashing in DISC 2008
P. Celis et al., Robin hood hashing in SFCS 1985
Single core: throughput

- DPDK Cuckoo
- Cuckoo++
- HopScotch
- Chaining (FC)
- Chaining (C++)

4M entries hash table, ≈ 55 Gbps
Single core: throughput

4M entries hash table, ≈ 55 Gbps
Single core: throughput

4M entries hash table, \( \approx \) 55 Gbps
Single core: throughput

4M entries hash table, ≈ 55 Gbps
Single core: throughput

4M entries hash table, ≈ 55 Gbps
Single core: throughput

Study of Insertion and Lookup cycles in the paper.

CAIDA has ≈ twice flows, similar trends.
Core scaling: single table

- DPDK Cuckoo Mutex
- DPDK Cuckoo Spinlock
Core scaling: single table

8M entries hash table, ≈ 100 Gbps
Core scaling: single table

- DPDK Cuckoo Mutex
- DPDK Cuckoo Spinlock

8M entries hash table, ≈ 100 Gbps
Core scaling: single table

8M entries hash table, ≈ 100 Gbps
Lock-Free heavily depends on the workload
Multi core scaling: core sharding

Leverage RSS to spread packets to multiple independent cores
Core scaling: sharding

All implementations scale always linearly
The aging dilemma

When flows terminate, their entries should be removed

- Timer-based approach is needed
- Deletion could be more delicate than insertion: concurrency
- Three implementations studied:
  - Scanning
  - Lazy Deletion
  - Timing Wheels
Flow Table maintenance techniques

- **Scanning**
  Parse the table periodically, deleting expired entries.

- **Lazy Deletion**
  compare last access time upon collisions.

- **Timing Wheels**
  entries are registered in time-based buckets.
Deletion: scaling

At scale, Lock Free is 10% slower than Core Sharding

Timing wheels can be as effective as Lazy Deletion

Caida, 4M hash tables, 32x parallel traces ≈ 100 Gbps
Conclusion: we have spare capacity
Can we design a packet processing pipeline that handles
one terabit per second of traffic
on a single dedicated device?
The bandwidth limit

Dedicated NF Server

Switch

H Payload
The bandwidth limit

Dedicated NF Server

Switch

Payload
The bandwidth limit

Dedicated NF Server

Switch

Need to deploy more servers

Throughput capped by the network bandwidth (100-400 Gbps)
The bandwidth limit

Not cost-effective!

Need to deploy more servers

Throughput capped by the network bandwidth (100-400 Gbps)
The bandwidth limit

Not cost-effective!

Many network functions need only headers!

Send Only Relevant Bits!

Dedicated NF Servers

Need to deploy more servers

Throughput capped by the network bandwidth (100-400 Gbps)
Send Only Relevant Bits!

Free up bandwidth

Higher cache-hit ratio

1.5KB Payload

70B
Send Only Relevant Bits!

Where to store payloads?
Store Payloads on the Switch
PayloadPark [CoNEXT '20]
Send Only Relevant Bits!

Where to store payloads?
Store Payloads on the Switch
PayloadPark [CoNEXT '20]
Store Payloads on the Switch

What is the impact?
Store Payloads on the Switch

What is the impact?

Let’s examine a CAIDA trace
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

Switch-to-NF Link Rate [Gbps]

0 1 2 3 4

Packets sent to the NF
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

<table>
<thead>
<tr>
<th>Switch-to-NF Link Rate [Gbps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>

Packets sent to the NF
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

![Graph showing Switch-to-NF Link Rate vs. Packets sent to the NF]

- **Baseline**
- **PayloadPark-like**

Packets sent to the NF:
- **Payload**
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

Packets sent to the NF

Switch-to-NF Link Rate [Gbps]
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

Switch-to-NF Link Rate [Gbps]

Baseline
PayloadPark-like
Ideal

Packets sent to the NF

Baseline
Payload

PayloadPark-like

Ideal
Store Payloads on the Switch

What is the impact?
Let’s examine a CAIDA trace

Switch-to-NF Link Rate [Gbps]

Packets sent to the NF

Baseline
Payload

PayloadPark-like

Ideal

How to extend the switch memory?
How to extend the switch memory?

💡 Using a dedicated external memory (e.g., HBM)

✔ Simple solution
How to extend the switch memory?

- Not cost-effective

- Using a dedicated external memory (e.g., HBM)
  - Simple solution
  - Higher energy footprint
  - High-cost
  - Wastes some ports on the switch
How to extend the switch memory?

The Ribosome Approach

💡 Exploiting a disaggregated pipeline on **shared** servers

✅ Many spare resources in the datacenter

⬆ Better resources usage

⬆ Low-cost

[NSDI’23] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostić, Marco Chiesa
The Ribosome Approach
The Ribosome Approach
The Ribosome Approach

Dedicated NF Processor

Programmable Switch

- Hdr
- Payload

Shared External Memory

PR TOM BARBETTE
The Ribosome Approach
The Ribosome Approach

Dedicated NF Processor

Programmable Switch

Shared External Memory

Payload

Hdr
The Ribosome Approach

Dedicated NF Processor

Programmable Switch

Shared External Memory
The Ribosome Approach

- Dedicated NF Processor
- Programmable Switch
- Shared External Memory
Implementation

- FastClick
- Intel Tofino
- Shared External Memory
Implementation

How to access the remote memory without affecting the server CPUs?

RDMA!
Challenges

- Dedicated NF processor
- Programmable switch
- Shared external memory
- FastClick
- Intel Tofino
- RDMA servers

Store Headers on the Switch
Where to store headers while retrieving payloads from RDMA servers?
Reconstructing millions of packets per second!

Where to store headers while retrieving payloads from RDMA servers?
How to access the remote memory without affecting the server CPUs?
RDMA!

High speed reliable RDMA Connection on multiple servers for storing payloads
See the paper!
Store Headers on the Switch

- FastClick
- Intel Tofino
- RDMA servers
Store Headers on the Switch

- FastClick
- Intel Tofino
- RDMA servers
- Payload

Dedicated NF processor
Shared external memory
Payload
RDMA servers
FastClick
Intel Tofino
PR TOM BARBETTE
Store Headers on the Switch

- Dedicated NF processor
- Shared external memory
- FastClick
- RDMA servers
  - Hdr
  - RDMA Read
  - Payload
Store Headers on the Switch

- FastClick
- Intel Tofino
- RDMA servers
  - RDMA Read
  - Payload
Store Headers on the Switch
Store Headers on the Switch

- FastClick
- Intel Tofino: Hdr, Payload
- RDMA servers
Store Headers on the Switch

FastClick

Intel Tofino

Hdr Payload

RDMA servers

4µs (max)

\[
\left(\frac{1\times10^{12}\, bps}{8\times10^3\, b}\right) \times 4\mu s = 500 \text{ headers}
\]

500 \times 72B = 36000B
Evaluation
Testbed and Workload Generation

![Diagram showing network components: Traffic Generator, Multicast Switch, Ribosome Switch, RDMA Server 1, RDMA Server 2, RDMA Server 3, RDMA Server 4, and NF.]
Testbed and Workload Generation
Throughput Gain

How much Ribosome improves the per-packet throughput on the NF server?

Tested NF: Forwarder

With a bandwidth requirement on the NF of ~20 Gbps

300 Gbps

115 Gbps

100 Gbps

Stores only 160B of payloads

Network I/O bottleneck

PR TOM BARBETTE
Throughput Gain

How much Ribosome improves the per-packet throughput on the NF server?

Tested NF: Forwarder

- 300 Gbps: With a bandwidth requirement on the NF of ~20 Gbps
- 115 Gbps: Stores only 160B of payloads
- 100 Gbps: Network I/O bottleneck

Throughput Gain

Throughput Gain

Throughput Gain
Throughput Gain

How much Ribosome improves the per-packet throughput on the NF server?

Ribosome enables multi-100Gbps packet processing!

![Graph showing throughput gain]

- **Ribosome**
- **Payload Park-like**
- **Baseline**

- 300 Gbps: With a bandwidth requirement on the NF of ~20 Gbps
- 115 Gbps: Stores only 160B of payloads
- 100 Gbps: Network I/O bottleneck
Throughput Gain

How much Ribosome improves the per-packet throughput on the NF server?

Ribosome enables multi-100Gbps packet processing!

~75Gbps for RDMA server
Due to RDMA overheads

300 Gbps
With a bandwidth requirement on the NF of
~20 Gbps

115 Gbps
Stores only 160B of payloads

100 Gbps
Network I/O bottleneck
Throughput Gain

How much Ribosome improves the per-packet throughput on the NF server?
Ribosome enables multi-100Gbps packet processing!

How many servers to potentially process 1Tbps?
14 shared RDMA servers

~75Gbps for RDMA server
Due to RDMA overheads

300 Gbps
With a bandwidth requirement on the NF of ~20 Gbps

115 Gbps
Stores only 160B of payloads

100 Gbps
Network I/O bottleneck

PR TOM BARBETTE
How much Ribosome improves the per-packet throughput on the NF server?

Ribosome enables multi-100Gbps packet processing!

A datacenter has thousands of servers!

How many servers to potentially process 1Tbps?

14 shared RDMA servers

How much Ribosome improves the per-packet throughput on the NF server?

With a bandwidth requirement on the NF of ~20 Gbps

300 Gbps

3x

35

0

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

Input Packet Rate (Mpps)

Output throughput (Gbps)

~75Gbps for RDMA server

Due to RDMA overheads

~115 Gbps

Stores only 160B of payloads

100 Gbps

Network I/O bottleneck

100 Gbps

PR TOM BARBETTE
Latency Gain

How much Ribosome improves the latency gain on the NF server?

- Due to congested queues and the high transmission rate:
  - ~110 µs
  - Almost constant ~25 µs

Tested NF: Forwarder
Latency Gain

How much Ribosome improves the latency gain on the NF server?

![Graph showing the improvement of Ribosome on latency gain]

- Baseline ~ 110 µs
- Due to congested queues and the high transmission rate
- Almost constant
- ~25 µs

Tested NF: Forwarder
Latency Gain

How much Ribosome improves the latency gain on the NF server?

And the tail latency? Similar trend!

Due to congested queues and the high transmission rate

~110 μs

Almost constant

~25 μs
Latency Gain

How much Ribosome improves the latency gain on the NF server?

And the tail latency? Similar trend!

Due to congested queues and the high transmission rate

~500 µs

~60 µs

Almost constant
Latency Gain

How much Ribosome improves the latency gain on the NF server?

Reducing queue sizes and the input throughput on the NF reduce latency!

And the tail latency? Similar trend!
Packet Size Impact

How does the packet size impact the throughput gains?

Tested NF: Forwarder
Packet Size Impact

How does the packet size impact the throughput gains?

![Graph showing packet size impact on throughput gains. The x-axis represents Packet Length (Bytes), and the y-axis represents Output throughput (Gbps). Two lines are shown: one for Ribosome and one for Baseline. The graph highlights the average real packet sizes. Tested NF: Forwarder.]
Packet Size Impact

How does the packet size impact the throughput gains?

Highly effective for relevant real-world scenarios!

![Graph showing the impact of packet size on output throughput. The x-axis represents packet length in bytes, and the y-axis represents output throughput in Gbps. The graph compares 'Ribosome' and 'Baseline' scenarios, with solid and dotted lines, respectively. The graph highlights the average real packet sizes with error bars.](image-url)
Packet Size Impact

How does the packet size impact the throughput gains?
Highly effective for relevant real-world scenarios!

300Gbps
With 400B packets

Average real packet sizes

Output throughput (Gbps) vs. Packet Length (Bytes)
Packet Size Impact

How does the packet size impact the throughput gains?
Highly effective for relevant real-world scenarios!

300Gbps
With 400B packets
Corresponds to ~93Mpps
NF can handle them!

How does the packet size impact the throughput gains?

Average real packet sizes
Packet Size Impact

How does the packet size impact the throughput gains?
Highly effective for relevant real-world scenarios!

300Gbps
With 400B packets
Corresponds to ~93Mpps
NF can handle them!
With 93M packets of 1.5KB
We process 1Tbps!

Average real packet sizes
Packet Size Impact

How does the packet size impact the throughput gains?

Highly effective for relevant real-world scenarios!

We can process 1Tbps on a single dedicated CPU!
Side note: can’t switches handle connection tracking?

After all, that’s the promise of OpenFlow.
Side note: can’t switches handle connection tracking?

After all, that’s the promise of OpenFlow.
Side note: can’t switches handle connection tracking?

After all, that’s the promise of OpenFlow.
Side note: can’t switches handle connection tracking?

After all, that’s the promise of OpenFlow.
Side note: can’t switches handle connection tracking?

After all, that’s the promise of OpenFlow.

TEA [SIGCOMM’20] → Tofino has a maximum of 100k flows/seconds

CPU → Around 7M/core

OpenFlow switches are around 40K at best
Can we build advanced NFs on top of Ribosome?

Output throughput (Gbps) vs. CPU cores on the NF

- Advanced Scheduler
- Per-flow Rate Limiter
- LoadBalancer

- ~300Gbps for stateful NFs
- ~220Gbps, 2.2x speedup

Advanced Scheduler → Reframer

[NSDI'22]
Advanced Network Functions

Can we build advanced NFs on top of Ribosome?
Ribosome supports advanced NFs!
Ribosome moves the NF bottleneck on the CPU! → Back to software!

![Graph showing output throughput (Gbps) vs. CPU cores on the NF]
- ~300Gbps for stateful NFs
- ~220Gbps, 2.2x speedup
Can we improve the performance of connection tracking on the NF server further?
Offloading classification: what are the limits?

What you need to know about (Smart) Network Interface Cards
Open source code and results with NVIDIA ConnectX-4, ConnectX-5, ConnectX-6, and Bluefield NICs (here)
Georgios P. Katsikas, Tom Barbette, Marco Chiesa, Dejan Kostić, and Gerald Q. Maguire Jr.

<table>
<thead>
<tr>
<th>IP Source</th>
<th>IP Destination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1</td>
<td>2.2.2.2</td>
<td>DROP</td>
</tr>
<tr>
<td>3.3.3.3</td>
<td>4.4.4.4</td>
<td>QUEUE=0</td>
</tr>
<tr>
<td>5.5.5.5</td>
<td>6.6.0.0/16</td>
<td>RSS</td>
</tr>
</tbody>
</table>

NIC-Bench
PAM’21
Throughput acc. number of rules

Table 1

# of rules across Tables 1-16 of a 100 GbE Mellanox ConnectX-5 NIC
How do NICs perform processing?

Match-Action Pipelines

Table 0

Match * 
Action 
jumpTable=N

Table N-1

Match udp_port=53 
Action 
dst_ip=1.1.1.1 queue=4

Queue 4
How do NICs perform processing?

Match-Action Pipelines

Table 0

Match

*  

Action

jumpTable=N

Table N-1

Match

udp_port=53  

Action

dst_ip=1.1.1.1

queue=4

Queue 4
How do NICs perform processing?

Match-Action Pipelines

Table 0

Match *
Action
jumpTable=N

Table N-1

Match udp_port=53
Action
dst_ip=1.1.1.1
queue=4

Queue 4
How do NICs perform processing?

Match-Action Pipelines

Table 0
- Match *
- Action jumpTable=N

Table N-1
- Match udp_port=53
- Action dst_ip=1.1.1.1 queue=4

Queue 4
Throughput acc. number of tables

Table 1  Tables 1-2

# of rules across Tables 1-16 of a 100 GbE Mellanox ConnectX-5 NIC
Throughput acc. number of tables

Table 1  Tables 1-2  Tables 1-4

Throughput (Gbps)

# of rules across Tables 1-16 of a 100 GbE Mellanox ConnectX-5 NIC
Throughput acc. number of tables

Table 1  Tables 1-2  Tables 1-4  Tables 1-8

Throughput (Gbps)

0 10k 20k 30k 40k 50k 60k 70k 80k 90k 100k 1M 2M 3M 4M

# of rules across Tables 1-16 of a 100 GbE Mellanox ConnectX-5 NIC
Throughput acc. number of tables

- Table 1
- Tables 1-2
- Tables 1-4
- Tables 1-8
- Tables 1-16

Throughput (Gbps)

# of rules across Tables 1-16 of a 100 GbE Mellanox ConnectX-5 NIC

>7.5x
Scenario 1 - Latency

Table 1

Latency

# of NIC rules in Table 1 of a 100 GbE Mellanox ConnectX-5 NIC
Scenario 1 - Latency

Latency

Table 1

# of NIC rules in Table 1 of a 100 GbE Mellanox ConnectX-5 NIC
Impact of number of flows in the input load

# of rules in Table 1 of a 100 GbE Mellanox ConnectX-5 NIC
Offloading every flow is not going to work

Can we offload some of the flows?
How do we use the NIC to pre-process packets?

- **Rule**
  - Attributes
  - Pattern
  - Actions

- **Server**
  - **Host**
  - **Network Interface Card**

- **Network**

- **Connection states table**
  - Connection 1 state
  - Connection 2 state
  - Connection 3 state
  - ....

Mark packet, tag “3”
WIP: Connection Tracking Offloading

Throughput of different implementations by threshold

~10% Improvement

Done by a master student. Further work needed!

UCLouvain
Can we *combine* the tracking for multiple Network Functions and *offload* some part of the classification?
Service Chaining: Tracking inside each NF
MiddleClick: Combining Classification

MiddleClick: Combining Classification

MiddleClick: Combining Classification

Evaluation: Avoiding re-classification

NAT running between 128 HTTP clients making requests to 4 NGINX servers – 35Gbps is the limit of the testbed
Evaluation: Offloading (HTTP load-balancer)
Takeout

- High-speed **stateful** software packet processing
  - Sharding is the way to go
  - Especially with the trend to many cores
  - Load-balancing problem: use RSS++ [CoNEXT’19]

- **Switch-based** stateful packet processing
  - Ribosome processes 300Gbps worth of traffic with 20Gbps of bandwidth
  - Send what you need where you actually need it

- **NIC-assisted** stateful packet processing
  - Promising approach, still under development

- **Combined** stateful packet processing
  - Do not re-classify the same thing
  - Hardware does help!
Try FastClick, a high speed dataplane based on Click and its PacketMill improvement! While load-balancing with RSS++ and combining sessions with MiddleClick, all included!

Try Retina for high-speed passive traffic analysis!

High-speed packet processing techniques
- NFV with FastClick (+PacketMill + MiddleClick + RSS++)
- Integration for software state (ConnTrack + MiddleClick)

Load-balancing
- Inside (RSS++) and between servers (Cheetah) or both (Metron, CrossRSS)

The network is starting to be programmable, and has per-connection programmability
- Job scheduling, optimization

Build the infrastructure for an efficient, competitive, and local Internet
- Make the network's core programmable by the service provider
- Improve today's network efficiency and enable the agility needed to sustain tomorrow's services