Network stack specialization for performance

Ilia Marinos§, Robert N.M. Watson§, Mark Handley*

§ University of Cambridge, * University College London
Motivation

Providers are scaling out rapidly. Key aspects:

- 1 machine: N functions → N machines: 1 function
- Performance is critical
- Scalability on multicore systems
- Cost & energy concerns
Motivation

Providers are scaling out rapidly. Key aspects:

- 1 machine: N functions → N machines: 1 function
- Performance is critical
- Scalability on multicore systems
- Cost & energy concerns

Are general-purpose stacks the right solution for that kind of role?
The Problem

- Conventional stacks are great for bulk transfers, but what about short ones?
The Problem

Network Throughput (Gbps)

Throughput (Gbps)

HTTP object size (KB)

Throughput (Gbps):

- 8 KB: 8 Gbps
- 16 KB: 6 Gbps
- 24 KB: 8 Gbps
- 32 KB: 8 Gbps
- 64 KB: 8 Gbps
- 128 KB: 8 Gbps
The Problem

[Graph showing network throughput and CPU utilization across different HTTP object sizes (KB)].
The Problem

- Network Throughput (Gbps)
- CPU utilization (%)

HTTP object size (KB)

NIC saturation, Low CPU-usage
The Problem

Throughput/CPU ratio is low

NIC saturation, Low CPU-usage

Network Throughput (Gbps)

CPU utilization (%)

Throughput (Gbps)

HTTP object size (KB)

Temperature

CPU utilization (%)

0 200

0 80 120 160

0 40 80 120

0 10

8 16 24 32 64 128

8 6 4 2 0

8 16 24

32

64 128

The graph shows the relationship between network throughput, CPU utilization, and HTTP object size. It indicates that the throughput/CPU ratio is low and there is NIC saturation with low CPU usage.
The Problem

Throughput/CPU ratio is low

NIC saturation, Low CPU-usage

Short-lived HTTP flows are a problem!
Why is this important?
Why is this important?

Distribution based on traces from Yahoo! CDN [Al-Fares et’al 2011]
Why is this important?

- 90% of the HTTP requested object sizes ≤ 25K
- 95% of the HTTP requested object sizes ≤ 50K

Distribution based on traces from Yahoo! CDN [Al-Fares et’al 2011]
Design Goals

Design a network stack that:

• Allows transparent flow of memory from NIC to the application and vice versa

• Reduces system costs (e.g., batching, cache-locality, lock- and sharing-free, CPU-affinity)

• Exploits application-specific knowledge to reduce repetitive processing costs (e.g. TCP segmentation of web objects, checksums)
Sandstorm: A specialized webserver stack

Prototyped on top of FreeBSD’s netmap framework:

- **libnmio**: abstracting netmap-related I/O
- **libeth**: lightweight ethernet layer
- **libtcpip**: optimized TCP/IP layer
- **application**: simple HTTP server that serves static content
Sandstorm: A specialized webserver stack

Key decisions (some of them):

• Application & stack are merged into the same process address space

• Static content is pre-segmented into network packets and \textit{a-priori} loaded to DRAM

• Received packet frames are processed in-place on the RX rings, w/o memory copying/buffering

• RX/TX packet batching greatly amortizes the system call overhead

• Bufferless, synchronous model (no socket layer)
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

netmap_input()
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

kernel space

user space

netmap_input()
Sandstorm Architecture
(10,000ft view)

app

tcip

eth

ether_input()

netmap_input()

nmio

POLLIN

ix0:RX

Content

A

B

.. 

A

B

.. 

ix0:TX

NIC driver

kernel space

user space

nmio

tcip

app

Sandstorm Architecture
(10,000ft view)
Sandstorm Architecture
(10,000ft view)

app

tcpi

eth

nmio

kernel space

user space

NIC driver

netmap_input()
tcpi_input()
ether_input()
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

kernel space

user space

netmap_input()
ether_input()
tcip_input()
TCP FSM
Sandstorm Architecture
(10,000ft view)

app

netmap_input()
ether_input()
tcip_input()

TCP FSM

websrv_accept()
webrsv_receive()
tcip_output()

inet

POLLIN

ix0:RX
ix0:TX

content

A
B

NIC driver

user space

kernel space
Sandstorm Architecture
(10,000ft view)

- App
  - tcpip
    - ether_input()
    - tcpipl_input()
  - eth
    - ether_input()
    - netmap_input()
  - nmio
    - POLLIN
    - lx0:RX
      - NIC driver
    - CONTENT
      - A
      - B
      - ..
      - A
      - B
      -..
    - lx0:TX

- User space
  - tcpip_output()
  - websrv_accept()
  - websrv_receive()
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

user space

kernel space

webserv_accept()
webserv_receive()
tcip_output()
ether_output()
tcip_input()
ether_input()
netmap_input()

POLLIN

TCP FSM

TCP

content

ix0:RX

ix0:TX
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

user space

kernel space
Sandstorm Architecture
(10,000ft view)

app

tcpip

eth

nmio

NIC driver

user space

kernel space

TCP

FSM

webserv_accept()

webserv_receive()
tcip_output()
ether_output()
nmap_output()
tcip_input()
ether_input()
nmap_input()

app

TCP

content

A

B

A

B

..
Sandstorm Architecture
(10,000ft view)

- **app**
  - websrv_accept()
  - websrv_receive()

- **tcpip**
  - tcpip_input()
  - tcpip_output()

- **eth**
  - ether_input()
  - ether_output()

- **nmio**
  - netmap_input()
  - netmap_output()

- **NIC driver**
  - POLLIN
  - ix0:RX
  - ix0:TX

- **TCP FSM**

- **user space**

- **kernel space**
Sandstorm Architecture
(10,000ft view)

app
- websrv_accept()
- websrv_receive()

tcpip
- tcpip_input()
- tcpip_output()

eth
- ether_input()
- ether_output()

nmio
- netmap_input()
- netmap_output()

NIC driver

kernel space

user space

POLLIN

content

A

B

..
Sandstorm Architecture
(10,000ft view)
Evaluation

Throughput - 6NICs (Gbps)

HTTP Object Size (KB)

nginx+FreeBSD
nginx+Linux
Sandstorm

Throughput - 6NICs (Gbps)

HTTP Object Size (KB)
Evaluation

Throughput - 6NICs (Gbps)

HTTP Object Size (KB)

~9.8x

~3.6x

~1.8x

nginx+FreeBSD
nginx+Linux
Sandstorm
Evaluation

Throughput - 6NICs (Gbps)

HTTP Object Size (KB)

- nginx+FreeBSD
- nginx+Linux
- Sandstorm

~9.8x
~3.6x
~1.8x

Start converging for sizes ≥ 256K
To copy or not to copy?

/* Get src and destination slots */
struct netmap_slot *bf = &ppool->slot[slotindex];
struct netmap_slot *tx = &txring->slot[cur];

/* zero-copy packet */
tx->buf_idx = bf->buf_idx;
tx->len = bf->len;
tx->flags = NS_BUF_CHANGED;

OR

/* Get source and destination bufs */
char *srcp = NETMAP_BUF(ppool, bf->buf_idx);
char *dstp = NETMAP_BUF(txring, tx->buf_idx);

/* memcpy packet */
memcpy(dstp, srcp, bf->len);
tx->len = bf->len;
To copy or not to copy?

Throughput (Gbps)

<table>
<thead>
<tr>
<th></th>
<th>Sandstorm “zerocopy”</th>
<th>Sandstorm “memcpy”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core 2 (2006)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serving a 24KB HTTP object</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
To copy or not to copy?

Throughput (Gbps)

-33%

Serving a 24KB HTTP object
To copy or not to copy?

Throughput (Gbps)

- Sandstorm “zerocopy”
- Sandstorm “memcpy”

Intel Sandybridge (2013)
Serving a 24KB HTTP object
CPU microarchitecture
~2006
CPU microarchitecture
~2006
CPU microarchitecture
~2006
CPU microarchitecture
~2006
CPU microarchitecture
~2006

Bottleneck
Extra detour to RAM

Raise interrupt
CPU microarchitecture
~2013
CPU microarchitecture
~2013
CPU microarchitecture
~2013

Raise interrupt
CPU microarchitecture
~2013

- C
- C
- LLC
- MC

Raise interrupt

PCIe

PCIe
CPU microarchitecture
~2013

Eventual eviction from LLC

Raise interrupt
CPU microarchitecture
~2013

- Eventual eviction from LLC

- Raise interrupt

- No extra detours to DRAM

- No FSB bottleneck
CPU microarchitecture
~2013

Eventual eviction from LLC

- ✔ No extra detours to DRAM
- ✔ No FSB bottleneck
- ?? LLC utilization (thrashing?)

Raise interrupt
HW/SW Intersection

- Should HW architecture evolution be considered a "black box" for networked systems development?

![Graph showing Mem Read Throughput vs Object Size (KB) for Sandstorm "zerocopy" and "memcpy" methods. Lower is better.](image)
Generality of Specialization

**Natural fit for:**

- Web & DNS servers (Sandstorm, Namestorm — check our paper)
- In-memory Key-Value stores
- RPC-based services
- Rate-adaptive video streaming applications (with MPEG-DASH or Apple HLS)
Generality of Specialization

Natural fit for:

• Web & DNS servers (Sandstorm, Namestorm — check our paper)

• In-memory Key-Value stores

• RPC-based services

• Rate-adaptive video streaming applications (with MPEG-DASH or Apple HLS)

Limitations:

• Possibly not a good fit for CPU- and/or filesystem-intensive applications

• Blocking in application-layer cannot be tolerated
Conclusions

**General-purpose stacks:**

- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don’t need it for application-specific clusters)
- Hard to tune/profile/debug
Conclusions

**General-purpose stacks:**

- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don’t need it for application-specific clusters)
- Hard to tune/profile/debug

**Specialized stacks:**

- 2-10x throughput improvement for web, 9x for DNS
- Linear scaling on multicore systems
- Low CPU utilization
Conclusions

**General-purpose stacks:**
- Great for bulk transfers, bad for short ones (but web is dominated by small-sized objects!)
- Picked a lot of generality in favor of flexibility (we don’t need it for application-specific clusters)
- Hard to tune/profile/debug

**Specialized stacks:**
- 2-10x throughput improvement for web, 9x for DNS
- Linear scaling on multicore systems
- Low CPU utilization

**Specialized network stacks not only viable, but necessary!**
Backup Slides
Supported TCP features

- Follows RFC 793, with Reno congestion control

**Limitations:**

- Support of the required TCP subset to serve incoming connections (not initiating them)
- TCP reordering not supported (not needed with typical HTTP requests)
# Latency

<table>
<thead>
<tr>
<th>Avg. Latency (μs)</th>
<th>0</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
<th>1200</th>
<th>1400</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Sandstorm**
- **Linux+nginx**
- **FreeBSD+nginx**

Serving a 24KB object

- # Concurrent Connections
- 4
- 16
- 32
- 48
- 64
- 80
Overview

Problems with general-purpose stacks:

• System-call overhead
• Shared accept-queue, PCB locks
• Cache-unfriendly due to async. design
• Memory-related overhead (e.g., mbuf alloc./copying)

Solutions with specialized stacks:

• Packet batching
• Share- & Lock-free design, per-core state
• Process-to-completion, cache-friendly, incr. cksum
• Pre-packetization, no memory copying/buffering