OS Kernel Support for a Low-Overhead Container Overlay Network

Danyang Zhuo, Kaiyuan Zhang, Yibo Zhu, Hongqiang Harry Liu, Matthew Rockett, Arvind Krishnamurthy, Thomas Anderson
Containers are ubiquitous

Cache

Web server

Database

Big data

Deep learning

Microservice
Hardware

VM

- App
- OS
- Hypervisor
- Hardware

Container

- App
- OS
- Hardware

- docker
- kubernetes
How do containers communicate?

- Host mode
  - Use the host network interface to communicate

- Macvlan mode (or SR-IOV)
  - Make container’s IP address routable on the host network

- Overlay mode
  - Container network virtualization

Two containers cannot bind to the same port.

Complicates host network routing

Are the network virtualization overheads fundamental?

High overheads
In this talk...

• Existing approach: Packet-based network virtualization results in high overheads.

• Slim: connection-based network virtualization that is compatible with existing Linux applications.

• Saving up to 56% CPU cycles on popular cloud applications (e.g., Memcached, Nginx, PostgreSQL, Apache Kafka).
Give a set of containers an illusion of owning a dedicated network.
Container network virtualization

Container A
1.2.3.4

Container B
1.2.3.5

Container C
1.2.3.6

Container D
1.2.3.7

vSwitch

Host
10.1.2.3

Host
10.1.2.4

Data
1.2.3.7
Container network virtualization

- Container A (1.2.3.4) connected to vSwitch (10.1.2.3)
- Container B (1.2.3.5) connected to vSwitch (10.1.2.3)
- Container C (1.2.3.6) connected to vSwitch (10.1.2.3)
- Container D (1.2.3.7) connected to vSwitch (10.1.2.3)

Host 10.1.2.3

Red box: Container A
Blue box: Container B
Blue box: Container C
Blue box: Container D

Host 10.1.2.4

Red box: Host 10.1.2.4
Blue box: Data 1.2.3.7
Container network virtualization

Packet-based network virtualization
Why packet-based virtualization?

Container A 1.2.3.4

Container B 1.2.3.5

Container C 1.2.3.6

Container D 1.2.3.7

vSwitch

Host 10.1.2.3

vSwitch

Host 10.1.2.4
## Performance overheads

<table>
<thead>
<tr>
<th>Setup</th>
<th>Throughput</th>
<th>Latency (RTT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra, Host</td>
<td>48.4 Gbps</td>
<td>5.9 us</td>
</tr>
<tr>
<td>Intra, Container</td>
<td>37.4 Gbps (23%)</td>
<td>7.9 us (34%)</td>
</tr>
<tr>
<td>Inter, Host</td>
<td>26.8 Gbps</td>
<td>11.3 us</td>
</tr>
<tr>
<td>Inter, Container</td>
<td>14.0 Gbps (48%)</td>
<td>20.9 us (85%)</td>
</tr>
</tbody>
</table>

Intel Xeon E5-2680 (2.5 GHz), Linux 4.4, Intel XL710 NIC (40G).
## Performance overheads

<table>
<thead>
<tr>
<th>Setup</th>
<th>Throughput</th>
<th>Latency (RTT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>14.0 Gbps (48%)</td>
<td>20.9 us (85%)</td>
</tr>
<tr>
<td>Improved</td>
<td>24.5 Gbps (9%)</td>
<td>21.2 us (88%)</td>
</tr>
<tr>
<td>Host</td>
<td>26.8 Gbps</td>
<td>11.3 us</td>
</tr>
</tbody>
</table>

*Packet steering*
CPU overheads, 10 Gbps

- Vanilla: 93%
- Improved: 60%
- Host:
Packet-based virtualization

Container A 1.2.3.4
Container B 1.2.3.5
Container C 1.2.3.6
Container D 1.2.3.7

vSwitch

Host 10.1.2.3
Host 10.1.2.4
Packet-based virtualization

Container A
1.2.3.4

vSwitch

Host
10.1.2.3

Container D
1.2.3.7

Host 10.1.2.4
Packet-based virtualization

Container A

Application
- Socket
- Accept
- Connect
- Send
-Recv
- Close
- ...
- ...

POSIX Socket interface

vNIC 1.2.3.4

Container D

Host 10.1.2.4

vSwitch

NIC 10.1.2.3

Host

10.1.2.3

vNIC 1.2.3.4

10.1.2.7
Packet-based virtualization

**Container A**
- Application
  - Start (Target IP)
  - Send (Buffer)
  - Recv (Buffer)
  - End
- vNIC
  - 1.2.3.4

**Host**
- vSwitch
- NIC
  - 10.1.2.3

**Container D**
- 1.2.3.7

**Host**
- 10.1.2.4
Packet-based virtualization

Container A

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

vNIC
1.2.3.4

Host
NIC
10.1.2.3

vSwitch

A capability to send/receive packets to/from 1.2.3.7 through vNIC.

Container D
1.2.3.7

Host 10.1.2.4
Packet-based virtualization

Container A

- Application
  - Con = Start (1.2.3.7)
  - Con.Send("ABC")
  - Con.End()

Network stack (vNIC)
- Connections:
  - Con: 1.2.3.4 <-> 1.2.3.7
- Packet generation, Device driver

File descriptor

Host

- NIC 10.1.2.3
- vSwitch
- Container D 1.2.3.7
- Host 10.1.2.4
Packet-based virtualization

**Application**
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

**Network stack (vNIC)**
- Connections:
  - Con: 1.2.3.4 <-> 1.2.3.7
- Packet generation, Device driver

**vNIC 1.2.3.4**

**vSwitch**

**Host**
- NIC 10.1.2.3

**Container A**
- 10.1.2.3

**Container D**
- Container D 1.2.3.7
- Host 10.1.2.4

Packet: "ABC"
Packet-based virtualization

Container A
- **Application**
  - Con = Start (1.2.3.7)
  - Con.Send(“ABC”)
  - Con.End()
- **vNIC**
  - 1.2.3.4

Host
- **NIC**
  - 10.1.2.3
- **vSwitch**

Network stack (vNIC)
- **Connections**
  - Con: 1.2.3.4 <-> 1.2.3.7
- **Packet generation, Device driver**

Container D
- 1.2.3.7
- Host 10.1.2.4

Connections:
- 10.1.2.4 1.2.3.7 A
- 10.1.2.4 1.2.3.7 B
- 10.1.2.4 1.2.3.7 C
Packet-based virtualization

Container A

Application
- Con = Start (1.2.3.7)
- Con.Send(“ABC”)
- Con.End()

vNIC
1.2.3.4

Network stack (vNIC)
Connections:
- Con: 1.2.3.4 <-> 1.2.3.7

Packet generation, Device driver

Host

vSwitch

NIC
10.1.2.3

Network stack (NIC)
Connections:
Packet generation, Device driver

Container D

1.2.3.7

Host 10.1.2.4

Application
- Con = Start (1.2.3.7)
- Con.Send(“ABC”)
- Con.End()
In this talk...

- Existing approach: Packet-based network virtualization results in high overheads.

- Slim: connection-based network virtualization that is compatible with existing Linux applications.

- Saving up to 56% CPU cycles on popular cloud applications (e.g., Memcached, Nginx, PostgreSQL, Apache Kafka).
Slim: connection-based virtualization

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Network stack (NIC)
Connections:
- Con: 1.2.3.4 <--> 1.2.3.7

Packet generation, Device driver

Host
10.1.2.3

vSwitch

Network stack (vNIC)
Connections:

"ABC"

Packet generation, Device driver

Container A
10.1.2.4

Container D
1.2.3.7

Host 10.1.2.4
Slim: connection-based virtualization

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Network stack (NIC)
Connections:
- Con: 1.2.3.4 <--> 1.2.3.7
Packet generation, Device driver

Network stack (NIC)
Connections:
Packet generation, Device driver

Host
NIC 10.1.2.3

Container A

Container D

Host 10.1.2.4
Slim: connection-based virtualization

Container A

Application
- Con = Start (1.2.3.7)
- Con.Send(“ABC”)  
- Con.End()

Host

NIC 10.1.2.3

Network stack (NIC)

Connections:
Packet generation, Device driver

“ABC”
Challenge #1: Network virtualization

How to give the container an illusion of a dedicated network?

Container A

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Host

NIC
10.1.2.3

What's 1.2.3.7?

Container D

1.2.3.7

Host 10.1.2.4

Network stack (NIC)
Packet generation, Device driver
Challenge #2: Compatibility

How to work with unmodified applications?

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Where’s my NIC?

Network stack (NIC)
- Connections:
- Packet generation, Device driver

Container A

Host
- NIC 10.1.2.3

Container D 1.2.3.7

Host 10.1.2.4
Challenge #3: Network Policies

How do we enforce network policies?

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Network stack (NIC)
- Connections:
- Packet generation, Device driver

Container A

Host

NIC 10.1.2.3

Container D 1.2.3.7

Host 10.1.2.4

"ABC"
Challenge #4: Security

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

Tell me my IP address

How do we enforce security?
Slim: connection-based virtualization

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

SlimRouter

Host

NIC 10.1.2.3

Network stack (NIC)

Connections:
Packet generation, Device driver

Container D 1.2.3.7
Host 10.1.2.4

Application

• Con = Start (1.2.3.7)
• Con.Send("ABC")
• Con.End()
Slim: connection-based virtualization

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send(“ABC”)
- Con.End()

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

Network stack (NIC)
- Connections:
  - Con: 10.1.2.3 <-> 10.1.2.4
- Packet generation, Device driver

Host 10.1.2.4

NIC 10.1.2.3

Container D 1.2.3.7

Application
- Con = Start (1.2.3.7)
Slim: connection-based virtualization

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send(“ABC”)
- Con.End()

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

Host 10.1.2.4

NIC 10.1.2.3

Network stack (NIC)
- Connections:
  - Con: 10.1.2.3 <-> 10.1.2.4
- Packet generation, Device driver

Container D 1.2.3.7
Slim: compatibility

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

SlimSocket

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

Host

NIC 10.1.2.3

Network stack (NIC)
- Connections:
  - Con: 10.1.2.3 <-> 10.1.2.4
- Packet generation, Device driver

Container D 1.2.3.7

Host 10.1.2.4
sock = socket()
bind(sock, 1.2.3.4, 80)
con = accept(sock, addr)
send(con, buf)
recv(con, buf)

h_s = socket()
bind(h_s, 10.1.2.3, 1234)
h_c = accept(h_s, addr)
connect(h_s, 10.1.2.3, 1234)
dup2(h_c, con)
dup2(h_s, con)
listen(sock)
listen(h_s)

NIC
IP = 10.1.2.3
recv(con, buf)
send(con, buf)

NIC
IP = 10.1.2.4
recv(con, buf)
send(con, buf)

Host connection created.
sock = socket()
bind(sock, 1.2.3.4, 80)
con = accept(sock, addr)
send(con, buf)
recv(con, buf)

h_s = socket()
bind(h_s, 10.1.2.3, 1234)
h_c = accept(h_s, addr)
connect(h_s, 10.1.2.3, 1234)
dup2(h_c, con)
send(con, buf)
recv(con, buf)

NIC
IP = 10.1.2.3

Router

NIC
IP = 10.1.2.4

(a) Web server
(b) Web client
Slim: Support network policies

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

SlimSocket

SlimRouter

Reject a connection if destination IP = 1.2.3.7

Host

NIC
10.1.2.3

Network stack (NIC)

Connections:

Packet generation, Device driver
Slim: Security enforcement

Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

SlimSocket

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

Host

Network stack (NIC)

Connections:
- Con: 10.1.2.3 <-> 10.1.2.4

Packet generation, Device driver

A capability to access NIC.
Container A: 1.2.3.4

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()

SlimSocket

Host

NIC
10.1.2.3

Network stack (NIC)
- Connections:
  - Con: 10.1.2.3 <-> 10.1.2.4

Packet generation, Device driver

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

Slim: Security enforcement

Your IP is 1.2.3.4
Con.GetMyIP()
Container A: 1.2.3.4

- SlimSocket

SlimRouter
- 1.2.3.4 <-> 1.2.3.7 is mapped to 10.1.2.3 <-> 10.1.2.4

SlimKernModule

Network stack (NIC)
- Con: 10.1.2.3 <-> 10.1.2.4
- Packet generation, Device driver

Host
- NIC 10.1.2.3

Container 1.2.3.4 is malicious. Request rejected.

Application
- Con = Start (1.2.3.7)
- Con.Send("ABC")
- Con.End()
In this talk…

• Existing approach: Packet-based network virtualization results in high overheads.

• Slim: connection-based network virtualization that is compatible with existing Linux applications.

• Saving up to 56% CPU cycles on popular cloud applications (e.g., Memcached, Nginx, PostgreSQL, Apache Kafka).
## Microbenchmark: performance

<table>
<thead>
<tr>
<th>Setup</th>
<th>Throughput</th>
<th>Latency (RTT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>14.0 Gbps (48%)</td>
<td>20.9 us (85%)</td>
</tr>
<tr>
<td>Improved</td>
<td>24.5 Gbps (9%)</td>
<td>21.2 us (88%)</td>
</tr>
<tr>
<td>Slim</td>
<td>26.8 Gbps</td>
<td>11.4 us</td>
</tr>
<tr>
<td>Host</td>
<td>26.8 Gbps</td>
<td>11.3 us</td>
</tr>
</tbody>
</table>
Microbenchmark: CPU

TCP Throughput (Gbps) vs. Virtual Cores

- Improved
- Host
- Slim

40% improvement
Evaluation: applications

• In-memory key-value store
  • Memcached 1.5.6

• Web server
  • Nginx 1.10.3

• Database
  • PostgreSQL 9.5

• Stream-processing framework
  • Kafka 2.0.0
Evaluation: Memcached performance

Throughput (K):
- Vanilla: 0
- Improved: 2
- Slim: 4
- Host: 8

Latency (ms):
- Vanilla: 1
- Improved: 0.8
- Slim: 0.6
- Host: 0.4

71% increase in throughput
42% decrease in latency
Evaluation: Memcached CPU

CPU reduction: 56%
Evaluations: CPU

<table>
<thead>
<tr>
<th>Application</th>
<th>CPU utilization reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>56%</td>
</tr>
<tr>
<td>Nginx</td>
<td>24%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>22%</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td>10%</td>
</tr>
</tbody>
</table>
Limitation

• Connection setup time is longer
  • 270 us -> 556 us

• Limited support for packet-based network policies

• Cannot support unmodified low-level network tools

• Cannot speed up datagram sockets (i.e., UDP)
Summary

• Packet-based network virtualization results in high overheads in terms of throughput, latency, and CPU utilization.

• Slim integrates efficient connection-based network virtualization support natively into the OS network stack.
  • Saving up to 56% CPU cycles on popular cloud applications.

• Open sourced at https://github.com/danyangz/slim.