Tackling Parallelization Challenges of Kernel Network Stack for Container Overlay Networks

Jiaxin Lei*, Kun Suo+, Hui Lu*, Jia Rao+

* SUNY at Binghamton
+ University of Texas at Arlington
Containers Are Widely Adopted by Industry

- OS level virtualization
- Lightweight
Containers Are Widely Adopted by Industry

- OS level virtualization
- Lightweight
- Higher consolidation density
- Lower operational cost
Containers Are Widely Adopted by Industry

- OS level virtualization
- Lightweight
- Higher consolidation density
- Lower operational cost
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using *VxLAN* protocol.
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using **VxLAN** protocol.
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using VxLAN protocol.
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using VxLAN protocol.
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using \textbf{VxLAN} protocol.
Overlay Networks Are the Technique For Containers Connectivity

• Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave

• They are generally built upon the tunneling approach like using VxLAN protocol.
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead
Network Packet Processing Path

• **Prolonged** network packet processing path

• **Additional** virtual devices overhead

Receiving Side

**Native**
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

![Diagram of network packet processing path](image-url)

Receiving Side

- NIC

Kernel Space

- IRQ

User Space

Native

Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Receiving Side: NIC → IRQ → SoftIRQ → Network Stack → Kernel Space → User Space → Applications

Native
Network Packet Processing Path

- **Prolonged** network packet processing path

- **Additional** virtual devices overhead

Receiving Side | Kernel Space | User Space
---|---|---
NIC | IRQ | Native
 | SoftIRQ | Network Stack
 |  | Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

![Diagram of network packet processing path]

- Receiving Side
  - NIC
  - IRQ
  - SoftIRQ
- Kernel Space
  - Network Stack
- User Space
  - Container Applications
  - Native
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Diagram:

- Receiving Side
  - NIC
  - IRQ
  - SoftIRQ
  - Network Stack

- Kernel Space
- User Space
  - Container Applications

Native
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Diagram:

- Receiving Side:
  - NIC
  - IRQ
  - SoftIRQ
  - Network Stack

- Kernel Space

- User Space:
  - Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

**Receiving Side**
- NIC

**Overlay**

**Kernel Space**

**User Space**
- Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

![Diagram showing the network packet processing path](image)

- **Receiving Side**
  - NIC
  - IRQ

- **Kernel Space**
  - SoftIRQ
  - Network Stack

- **User Space**
  - Container
  - Applications

Overlay
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

**Receiving Side**

- NIC
  - IRQ

**Kernel Space**

- Network Stack
  - SoftIRQ
  - Packet decapsulation
  - VxLAN

**User Space**

- Container Applications

**Overlay**

- vBridge
- Veth
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Receiving Side

- NIC
- IRQ
- Network Stack
- SoftIRQ

Kernel Space

- Packet decapsulation
- VxLAN
- SoftIRQ

Overlay

- vBridge
- Veth
- SoftIRQ
- Network Stack

User Space

- Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Diagram:

- Receiving Side: NIC → IRQ
- Kernel Space: NIC → Network Stack → SoftIRQ → VxLAN → SoftIRQ → Packet decapsulation → SoftIRQ → Network Stack
- User Space: Network Stack → Container Applications

Overlay:

- vBridge → Veth → Network Stack
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Receiving Side

- NIC
- IRQ

Kernel Space

- Network Stack
- SoftIRQ
- Packet decapsulation
- SoftIRQ
- VxLAN

Overlay

- vBridge
- Veth
- SoftIRQ
- Network Stack

User Space

- Container Applications
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

**Receiving Side**
- NIC
- IRQ

**Kernel Space**
- SoftIRQ
- Network Stack
- SoftIRQ
- VxLAN

**Overlay**
- vBridge
- Veth
- Packet decapsulation

**User Space**
- Container Applications
Existing Optimizations for Packet Processing

Receiving Side

- NIC
- vBridge

Kernel Space

- Network Stack
- Packet decapsulation
- SoftIRQ

User Space

- VxLAN
- Container Applications
- Network Stack
- SoftIRQ

Existing Optimizations for Packet Processing

- Packet decapsulation
- Kernel Space
- User Space
- NIC
- vBridge
- Veth
- Network Stack
- SoftIRQ
Existing Optimizations for Packet Processing

- NIC
- IRQ
- NIC IRQ
- Receiving Side
- vBridge
- Veth
- Kernel Space
- Network Stack
- Packet decapsulation
- SoftIRQ
- Network Stack
- VxLAN
- User Space
- Container Applications
- Existing Optimizations
- Packet decapsulation
- IRQ coalescing
- GRO
- SoftIRQ
- SoftIRQ
- User Space
- Container Applications
Existing Optimizations for Packet Processing

**Receiving Side**
- NIC
- IRQ
- Multi-queue
- IRQ coalescing

**Kernel Space**
- SoftIRQ
- GRO
- RPS
- Network Stack

**User Space**
- SoftIRQ
- VxLAN
- Packet decapsulation
- Container Applications
- vBridge
- Veth
- Network Stack

**Existing Optimizations for Packet Processing**
- Packet decapsulation
- IRQ coalescing
- GRO
- RPS
- Multi-queue
- Kernel Space
## Experimental Settings

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Throughput</td>
</tr>
<tr>
<td>2.2 GHz</td>
<td>iPerf3</td>
</tr>
<tr>
<td>10 cores</td>
<td>Multi-queue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>NIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64GB</td>
<td>40Gbps</td>
</tr>
</tbody>
</table>
Experimental Settings

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Software</td>
</tr>
<tr>
<td>2.2 GHz</td>
<td>iPerf3</td>
</tr>
<tr>
<td>10 cores</td>
<td>mpstat</td>
</tr>
<tr>
<td>Memory</td>
<td>NIC</td>
</tr>
<tr>
<td>64GB</td>
<td>40Gbps</td>
</tr>
<tr>
<td>Throughput</td>
<td>CPU Usage</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>S1 - Native</td>
<td></td>
</tr>
<tr>
<td>S2 - Linux Overlay</td>
<td></td>
</tr>
<tr>
<td>S3 - Docker Overlay</td>
<td></td>
</tr>
</tbody>
</table>
# Experimental Settings

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>iPerf3</td>
</tr>
<tr>
<td>2.2 GHz</td>
<td>mpstat</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Multi-queue</td>
</tr>
<tr>
<td>64GB</td>
<td></td>
</tr>
<tr>
<td><strong>NIC</strong></td>
<td></td>
</tr>
<tr>
<td>40Gbps</td>
<td></td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
</tr>
<tr>
<td>iPerf3</td>
<td></td>
</tr>
<tr>
<td><strong>CPU Usage</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>10 cores</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NIC</strong></td>
<td></td>
</tr>
<tr>
<td>40Gbps</td>
<td></td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
</tr>
<tr>
<td>iPerf3</td>
<td></td>
</tr>
<tr>
<td><strong>CPU Usage</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>10 cores</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**S1 - Native**

**S2 - Linux Overlay**

**S3 - Docker Overlay**
## Experimental Settings

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Memory</td>
</tr>
<tr>
<td>2.2 GHz</td>
<td>64 GB</td>
</tr>
<tr>
<td>10 cores</td>
<td></td>
</tr>
</tbody>
</table>

### S1 - Native
- Host1
  - eth0
  - iPerf3
- Host2
  - eth0
  - iPerf3

### S2 - Linux Overlay
- Host1
  - eth0
  - VxLAN
  - iPerf3
- Host2
  - eth0
  - VxLAN
  - iPerf3

### S3 - Docker Overlay
- Host1
  - eth0
  - iPerf3
- Host2
  - eth0
  - iPerf3
### Experimental Settings

#### S1 - Native

- **Host1**: iPerf3, eth0
- **Host2**: iPerf3, eth0

#### S2 - Linux Overlay

- **Host1**: iPerf3, VxLAN, eth0
- **Host2**: iPerf3, VxLAN, eth0

#### S3 - Docker Overlay

- **Host1**: iPerf3 Container, Veth0, VxLAN, eth0
- **Host2**: iPerf3 Container, Veth0, VxLAN, eth0

---

### Hardware

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Memory</th>
<th>NIC</th>
<th>Throughput</th>
<th>CPU Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S1</strong></td>
<td>2.2 GHz</td>
<td>64GB</td>
<td>40Gbps</td>
<td>iPerf3</td>
<td>mpstat</td>
</tr>
<tr>
<td><strong>S2</strong></td>
<td>10 cores</td>
<td></td>
<td></td>
<td></td>
<td>Multi-queue</td>
</tr>
</tbody>
</table>

---

### Software

- **S1 - Native**: mpstat
- **S2 - Linux Overlay**: mpstat
- **S3 - Docker Overlay**: mpstat
TCP and UDP Throughputs under Three Different Cases

Throughput (Gb/s)

TCP

S1: 23 Gb/s
S2: 6.5 Gb/s
S3: 6.4 Gb/s
• TCP Throughput of Docker Overlay case drops 72% compared with native case.
TCP Throughput of Docker Overlay case drops 72% compared with native case.

UDP Throughput drops 58%.
Single Flow Performance

CPU Usage under Three Different Cases for TCP

* 5% indicates one cpu core is fully saturated.
Single Flow Performance

- Packet processing overhead fully saturates one cpu core in two overlay cases.

* 5% indicates one cpu core is fully saturated.
Single Flow Performance

CPU Usage under Three Different Cases for TCP

- Packet processing overhead **fully saturates** one cpu core in two overlay cases.
- Current solutions **can’t scale** single flow performance.

* 5% indicates one cpu core is fully saturated.
Multiple Flows Performance

Throughput (Gb/s) vs. Pair Number of Iperf Connection

- Native
Multiple Flows Performance

Native case quickly reaches \(~37\) Gbps under TCP with only 2 pairs.
**Multiple Flows Performance**

- Native case quickly reaches \(~37\) Gbps under TCP with only 2 pairs.
- In two overlay cases, TCP throughput grows slowly.
Multiple Flows Performance

- Native case quickly reaches ~37 Gbps under TCP with only 2 pairs.
- In two overlay cases, TCP throughput grows slowly.
Under the same throughput (e.g., 40 Gbps), overlay networks consume **much more** CPU resources (e.g., around **2.5 times**) than the native case.
• Bad scalability is largely due to the *inefficient interplay* of many tasks.
Small Packet Performance

Packet Number /s vs Packet Size of Iperf Connection

- Native
- Linux Overlay
- Docker Overlay
Small Packet Performance

- Docker overlay achieves as low as 50% packet processing rate of that in the native case.
Interrupt Number with Varying Packet Sizes

- IRQ number increases dramatically in the Docker overlay UDP case — 10x of that in the TCP case.
Interrupt Number with Varying Packet Sizes

- IRQ number increases dramatically in the Docker overlay UDP case — 10x of that in the TCP case.
- 3x softIRQ numbers are observed in Docker Overlay case compared with the IRQ numbers.
Insights and Conclusions
Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
Insights and Conclusions

• Kernel does not provide **per-packet level parallelization**.
• Kernel does not efficiently handle **various packet processing tasks**.
• Bottlenecks become more severe for **small packets**.
Insights and Conclusions

• Kernel does not provide per-packet level parallelization.
• Kernel does not efficiently handle various packet processing tasks.
• Bottlenecks become more severe for small packets.

Thinking about future works:
Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Thinking about future works:

- Is it feasible to provide packet-level parallelization for a **single network flow**?
Insights and Conclusions

• Kernel does not provide *per-packet level parallelization*.
• Kernel does not efficiently handle *various packet processing tasks*.
• Bottlenecks become more severe for *small packets*.

Thinking about future works:

• Is it feasible to provide packet-level parallelization for a *single network flow*?
• How can the kernel perform a *better isolation among multiple flows* especially for efficiently utilizing shared hardware resources?
Insights and Conclusions

• Kernel does not provide **per-packet level parallelization**.
• Kernel does not efficiently handle **various packet processing tasks**.
• Bottlenecks become more severe for **small packets**.

Thinking about future works:

• Is it feasible to provide packet-level parallelization for a **single network flow**?
• How can the kernel perform a **better isolation among multiple flows** especially for efficiently utilizing shared hardware resources?
• Can the packets be **further coalesced** with optimized network path for reduced interrupts and context switches?
Thank you!