Circuit Switched VM Networks for Zero-Copy IO

Johannes Krude, Mirko Stoffers, Klaus Wehrle

https://comsys.rwth-aachen.de/
VM Networks

- VMs are used for Isolation
  - Multiple Tenants on the same Host
  - Compartmentalization
  - Fault Isolation

Isolation complicates Communication

Until now: Performance and Isolation are mutually exclusive

Circuit Switched VM Networks enable Zero-Copy IO with Isolation
VM Networks

- VMs are used for Isolation
  - Multiple Tenants on the same Host
  - Compartmentalization
  - Fault Isolation

![Diagram]

- Circuit Switched VM Networks enable Zero-Copy IO with Isolation
VM Networks

- **VMs are used for Isolation**
  - Multiple Tenants on the same Host
  - Compartmentalization
  - Fault Isolation

```plaintext
VM1
- HTTP Proxy
- Application Server

VM2
- Database
```

Isolation complicates Communication

- Until now: Performance and Isolation are mutually exclusive

**Circuit Switched VM Networks** enable Zero-Copy IO with Isolation
VM Networks

- VMs are used for Isolation
  - Multiple Tenants on the same Host
  - Compartmentalization
  - Fault Isolation

- Isolation complicates Communication
VM Networks

• VMs are used for Isolation
  ◦ Multiple Tenants on the same Host
  ◦ Compartmentalization
  ◦ Fault Isolation

• Isolation complicates Communication

• Until now: Performance and Isolation are mutually exclusive

![Diagram of VM Networks]

- NIC
- VM1
  - HTTP Proxy
  - Application Server
- VM2
  - Database

Krude et al.
VM Networks

- VMs are used for Isolation
  - Multiple Tenants on the same Host
  - Compartmentalization
  - Fault Isolation

- Isolation complicates Communication
- Until now: Performance and Isolation are mutually exclusive

Circuit Switched VM Networks enable Zero-Copy IO with Isolation
VM Packet Processing

- Problem: Packet Switching
VM Packet Processing

- Problem: Packet Switching
- Unnecessary Overhead

Goals
- Remove Overhead
- Keep Application Compatibility
- Keep Network Isolation

VM Packet Processing

---

Krude et al.
VM Packet Processing

- Problem: Packet Switching
- Unnecessary Overhead
  - Multiplexing
  - Packetization

Goals
- Remove Overhead
- Keep Application Compatibility
- Keep Network Isolation
VM Packet Processing

- Problem: Packet Switching
- Unnecessary Overhead
  - Multiplexing
  - Packetization
  - Congestion Control
  - Retransmissions
  - Reordering

Goals
- Remove Overhead
- Keep Application Compatibility
- Keep Network Isolation
VM Packet Processing

- Problem: Packet Switching
- Unnecessary Overhead
  - Multiplexing
  - Packetization
  - Congestion Control
  - Retransmissions
  - Reordering
  - (Copying)

Goals
- Remove Overhead
- Keep Application Compatibility
- Keep Network Isolation
VM Packet Processing

- Problem: Packet Switching
- Unnecessary Overhead
  - Multiplexing
  - Packetization
  - Congestion Control
  - Retransmissions
  - Reordering
  - (Copying)

Goals
- Remove Overhead
- Keep Application Compatibility
- Keep Network Isolation
Removing Overhead

- No Packet Processing in VM Kernels
  - Move to Host if Still Needed
  - Remove if Possible
Removing Overhead

• No Packet Processing in VM Kernels
  ▶ Move to Host if Still Needed
  ▶ Remove if Possible

VM₁
- HTTP Proxy
- Application Server
- Socket

VM₂
- Database
- Socket

TCP/UDP Proxy Stack
NIC

Krude et al.
Removing Overhead

• No Packet Processing in VM Kernels
  ► Move to Host if Still Needed
  ► Remove if Possible

• Keep Socket API
  ► Provides Access to Streams & Datagrams
  ► Required to Support Legacy Applications
  ► Provides Isolation between Applications
Removing Overhead

- **No Packet Processing in VM Kernels**
  - Move to Host if Still Needed
  - Remove if Possible

- **Keep Socket API**
  - Provides Access to Streams & Datagrams
  - Required to Support Legacy Applications
  - Provides Isolation between Applications

- **Provide Zero-Copy API**
  - As Optional Extension to Socket API
Circuit Switched VM Networks

- **Separate Shared-Memory based Circuit for each Connection**
  - from VM to Proxy Stack
  - or Direct from VM to VM
Circuit Switched VM Networks

- **Separate Shared-Memory based Circuit for each Connection**
  - from VM to Proxy Stack
  - or Direct from VM to VM
- **Switch Operator**
  - Mediates Connection Establishment
  - Enforces Connection Policies
Circuits

- Control Area: Read & Write Pointers, Flags, …

• Protocol Features
  - TCP Flow Control: Ring Buffers
  - UDP Datagrams: Prepend some kind of Header
- **Protocol Features**
  - TCP Flow Control: Ring Buffers
  - UDP Datagrams: Prepend some kind of Header
- **Zero-Copy Circuit**
  - Map Circuit Memory into Application
  - Optional ⇒ Compatible with Legacy Applications
Network Isolation

- No Access to Communication of other Applications
  - Keeps Socket Isolation
  - Even when doing Zero-Copy IO
Network Isolation

• No Access to Communication of other Applications
  ▶ Keeps Socket Isolation
  ▶ Even when doing Zero-Copy IO

• Connection Policies enforced on Connection Setup
  ▶ No Inspection of Individual Packets needed
  ▶ No Redundant State for Stateful Firewalls
Network Isolation

• **No Access to Communication of other Applications**
  ▶ Keeps Socket Isolation
  ▶ Even when doing Zero-Copy IO

• **Connection Policies enforced on Connection Setup**
  ▶ No Inspection of Individual Packets needed
  ▶ No Redundant State for Stateful Firewalls

• **Denying Raw Packet Access**
  ▶ Same Level of Access as Containers
  ▶ No Crafting of Malicious Packet Headers
  ▶ No Unfair Congestion Control Algorithms
Implementation & Evaluation

- **Xen Hypervisor**
  - Allows for Shared-Memory between any consenting VM
- **Linux VM Kernel & Linux Host OS**
  - No VM User-Space Modifications Required
  - Use Regular Linux Sockets for Proxy Stack

- Works for Real-World Applications
  - NGINX, BIND, Tor, Firefox, Transmission, Quake 3, Mutt, openssh, git, aptitude, wget, …

- Reduced VM Size
  - Minimum Linux VM: 17% Memory Reduction, 48 MiB to 40 MiB
  - Especially Relevant for Unikernels in high density Deployments

- Measured Goodput & Response Times
  - Hardware: Xeon E5-4610 v4 (10 Cores), Intel X710-T4 (10 Gbit)
Implementation & Evaluation

• Xen Hypervisor
  ▶ Allows for Shared-Memory between any consenting VM

• Linux VM Kernel & Linux Host OS
  ▶ No VM User-Space Modifications Required
  ▶ Use Regular Linux Sockets for Proxy Stack

• Works for Real-World Applications
  ▶ NGINX, BIND, Tor, Firefox, Transmission, Quake 3, Mutt, openssh, git, aptitude, wget, …

Reduced VM Size

▶ Minimum Linux VM: 17% Memory Reduction, 48 MiB to 40 MiB

Especially Relevant for Unikernels in high density Deployments

Measured Goodput & Response Times

▶ Hardware: Xeon E5-4610 v4 (10 Cores), Intel X710-T4 (10 Gbit)
Implementation & Evaluation

• **Xen Hypervisor**
  ▶ Allows for Shared-Memory between any consenting VM

• **Linux VM Kernel & Linux Host OS**
  ▶ No VM User-Space Modifications Required
  ▶ Use Regular Linux Sockets for Proxy Stack

• **Works for Real-World Applications**
  ▶ NGINX, BIND, Tor, Firefox, Transmission, Quake 3, Mutt, openssh, git, aptitude, wget, …

• **Reduced VM Size**
  ▶ Minimum Linux VM: 17 % Memory Reduction, 48 MiB to 40 MiB
  ▶ Especially Relevant for Unikernels in high density Deployments
Implementation & Evaluation

- **Xen Hypervisor**
  - Allows for Shared-Memory between any consenting VM

- **Linux VM Kernel & Linux Host OS**
  - No VM User-Space Modifications Required
  - Use Regular Linux Sockets for Proxy Stack

- **Works for Real-World Applications**
  - NGINX, BIND, Tor, Firefox, Transmission, Quake 3, Mutt, openssh, git, aptitude, wget, ...

- **Reduced VM Size**
  - Minimum Linux VM: 17% Memory Reduction, 48 MiB to 40 MiB
  - Especially Relevant for Unikernels in high density Deployments

- **Measured Goodput & Response Times**
  - Hardware: Xeon E5-4610 v4 (10 Cores), Intel X710-T4 (10 Gbit)
Stream Goodput

<table>
<thead>
<tr>
<th>#VMs</th>
<th>32</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodput (Gbit/s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>packet switched</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circuit + legacy app</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circuit + zero-copy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

95% Confidence

+ Circuit
+ Proxy Stack
+ NIC

• suitable beyond 10 GBit NICs
• up to 137.2 Gbit/s with an Improvement of up to 15.4×
Stream Goodput

- VMs to External Host
- VMs to Host OS

- Goodput (Gbit/s) for different numbers of VMs (32, 128)
- Comparison of packet switched, circuit + legacy app, and circuit + zero-copy

- 95% Confidence

- Suitable beyond 10 GBit NICs

Krude et al.
Stream Goodput

- suitable beyond 10 GBit NICs
- up to 137.2 Gbit/s with an Improvement of up to $15.4 \times$
Response Times

Stream Response

-3779 µs
-31 µs
-79 µs
+50 µs

Size 1 256 64Ki 16Mi connect

-10-5 -10-4 -10-3 -10-2

10-5 10-4 10-3 10-2

Time (s)

packet switched
circuit + legacy app
circuit + zero-copy
95% Confidence

10 Krude et al.
Response Times

- faster for Streams after 1-2 Rounds
Response Times

- faster for Streams after 1-2 Rounds
- faster for Datagrams after 1 Round

Packet switched
Circuit + legacy app
Circuit + zero-copy
95% Confidence

Krude et al.
Conclusion

- Remove Packet Processing from VM Kernels
- Circuit Switched VM Networks with Zero-Copy IO
- Network Isolation & Performance
- up to 137.2 Gbit/s with up to 15.4 × Improvement
Conclusion

• Remove Packet Processing from VM Kernels
• Circuit Switched VM Networks with Zero-Copy IO
• Network Isolation & Performance
• up to 137.2 Gbit/s with up to 15.4 × Improvement

Thank you for Listening!
Socket API

socket (PF_INET, SOCK_STREAM)
- connect (AF_UNSPEC)
- listen()
- shutdown(),
  connect (AF_UNSPEC)
- accept ()
- Success
- Failure

socket (PF_INET, SOCK_DGRAM)
- bind (),
  connect ()
- recv (),
  send ()
- connect (AF_UNSPEC)
- send (),
  connect ()
- accept ()
- Success
- Failure