NetKernel: Making Network Stack Part of the Virtualized Infrastructure

Zhixiong Niu, Hong Xu, Peng Cheng, Qiang Su, Yongqiang Xiong, Tao Wang, Dongsu Han, Keith Winstein
Current architecture in the cloud

VM

APP

Guest OS

Network Stack

Hypervisor infrastructure

DCN Infrastructure

VM

APP

Guest OS

Network Stack
What’re the fundamental limitations?
**Motivation: Tenants**

Have to deal with the network stack all by myself

<table>
<thead>
<tr>
<th>TCP parameters</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>initcwnd</td>
<td>net.ipv4.tcp_rmem</td>
</tr>
<tr>
<td>initialRTO (ms)</td>
<td>net.ipv4.tcp_wmem</td>
</tr>
<tr>
<td>minRTO (ms)</td>
<td>net.core.rmem_max</td>
</tr>
<tr>
<td>DelayedAckTimeout (ms)</td>
<td>net.core.wmem_max</td>
</tr>
</tbody>
</table>

- BBR
- MPTCP
- CTCP
- CUBIC
- PCC
- DCTCP
- StackMap
- MegaPipe
- mTCP
- FastSocket
- FlexSC
- Kernel
Tenants are primarily concerned with performance and functionality, not implementation details.
Motivation: Operator

I know everything here.
I can really help my tenants (and make some money!)
Motivation: Operator

- Can’t deploy new stacks (DCTCP)
- Difficult to perform management tasks
- Difficult to even define performance SLA
- Difficult to troubleshoot

Zero visibility or control of the network stack
Is there a better way?
Making Network Stack Part of the Virtualized Infrastructure

Current architecture

Interface unchanged (BSD sockets, etc.)

Network stack module

Packets handled in the NSM

Current architecture
Benefits

• Better efficiency in management for the operator
  • Orchestrate the resource provisioning strategies more flexibly
  • Implement management functions as a part of user’s network stack

• Deployment and performance gains for users without efforts
  • Enforce various kernel stack optimizations
  • Enforce high-performance userspace stacks
  • Use advanced hardware
Design Challenges

• How to transparently redirect socket API calls without changing applications?
• How to transmit the socket semantics between the VM and NSM?
• How to ensure high performance with semantics transmission (e.g., 100 Gbps)?
Transparent socket API redirection

- A new sock type, SOCK_NETKERNEL
- GuestLib: A complete implementation of BSD socket APIs
A lightweight semantics channel

- NQE: NetKernel queue elements for semantics

<table>
<thead>
<tr>
<th>Opcode</th>
<th>VM ID</th>
<th>Queue set ID</th>
<th>VM socket ID</th>
<th>op_data</th>
<th>data pointer</th>
<th>size</th>
<th>rsved</th>
</tr>
</thead>
</table>

- NQE queues for semantics transmission and hugepages for data transmission in NetKernel device

---

Tenant VM

BSD Socket API
socket(), send(), ...

GuestLib
nk_bind(), nk_sendmsg(), ...

(4) return to app
(1) NetKernel socket

(3) response NQE
(2) translate to NQE

NetKernel device

Huge pages

NQE
Scalable lockless queues

- Per-core queue set, lockless queues
- NQE switching via CoreEngine
VM based NSM.

• Supports existing kernel and userspace stacks from various Oses
• Provide good isolation to guarantee the performance
• Run stacks independent of the hypervisor
NetKernel

- Tenant VM
  - APP1
  - APP2
  - BSD Socket
  - GuestLib (NetKernel Socket)

- NetKernel device
  - NetKernel CoreEngine
  - Queues
  - Huge pages

- NSM
  - Network Stack
  - ServiceLib
  - vNIC
  - Queues
  - Huge pages
  - Virtual Switch or Embedded Switch (SR-IOV)
  - pNICs

- Stripped area indicates a shared memory region
Implementation

• QEMU KVM 2.5.0, Linux Kernel 4.9
• Intel(R) Xeon(R) 16-core CPU @ 2.30GHz x 2
• 256GB DDR4 2133MHz
• Mellanox ConnectX-4 100G single port NIC
Use Cases #1: Multiplexing

Application Gateway (AG): L7 proxy and load balancing services

AG1
4 core

AG2
4 core

AG3
4 core

Normalized RPS Performance of a trace from a large cloud
Use Cases #1: Multiplexing

NetKernel: 9 Cores
Baseline: 12 Cores

Benefit: NetKernel can help operator perform network management more efficiently
Use Case #2: Deploying mTCP without API Change

- mTCP doesn't support Nginx yet
- mTCP ported as an NSM, fixed a bug in DPDK mlx5_core driver
- Unmodified Nginx on mTCP without any tenant effort

![Graph showing performance comparison between Kernel Stack NSM and mTCP NSM. mTCP NSM brings ~1.8x performance gain.](image-url)
Use Case #3: Shared Memory Networking

• The operator can easily detect the on-host traffic with NetKernel.
• For on-host traffic, it can use shared memory NSM to avoid TCP and bridge overhead.

Deployment and performance gains for users

**Shared memory NSM can achieve >2x performance gain for on-host traffic**

Benefit: NetKernel can help users achieve deployment and performance gains.
Microbenchmarks: Throughput

- Baseline (a VM) and NetKernel (a VM with a Linux Kernel) using the same setting
- 8 TCP connections, 8KB messages

*Can achieve 100Gbps with 3 cores (send), 8 cores (receive)*
Microbenchmarks: RPS

- Simple epoll server, short TCP conn.
- 64B request/response

mTCP NSM brings 2x performance gain
Discussion and future directions

• How can I do Netfilter?
  • Hard to support for multiple-tenant NSM

• What about troubleshooting performance issues?
  • Operator can easily monitor their NSMs by deploy additional mechanisms in the NSMs

• Does NetKernel increase the attack surface?
  • Own address spaces for NK device
  • Isolated channel between NSM and VM

• Future directions
  • Performance isolation
  • Charging policies
  • FPGA/SoC
Recap

• Designed and implemented NetKernel
  • Decouples the network stack from the guest
  • Making it part of the virtualized infrastructure in the cloud

• Enabled several new usecases
  • Multiplexing, mTCP NSM, Shared mem. NSM

• Conducted comprehensive testbed evaluation with commodity 100G NICs

• Website
  • https://netkernel.net