High Performance Networking

U-Net and FaRM

U-Net (1995)

- Thorsten von Eicken
  - Ph.D Berkeley, Prof Cornell, now CTO at RightScale
- Anindya Basu
  - Ph.D Cornell
- Vineet Buch
  - MS Cornell, now at Google
- Werner Vogels
  - Cornell, Amazon

Motivation

The Traditional Network

A message explicitly managed by the kernel

Traverse network “stack”

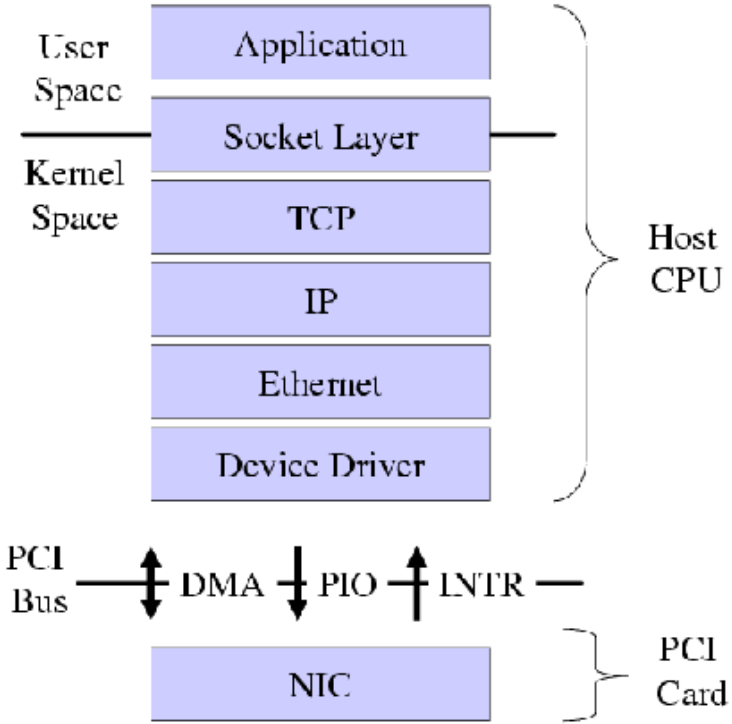
The Traditional Network

Send

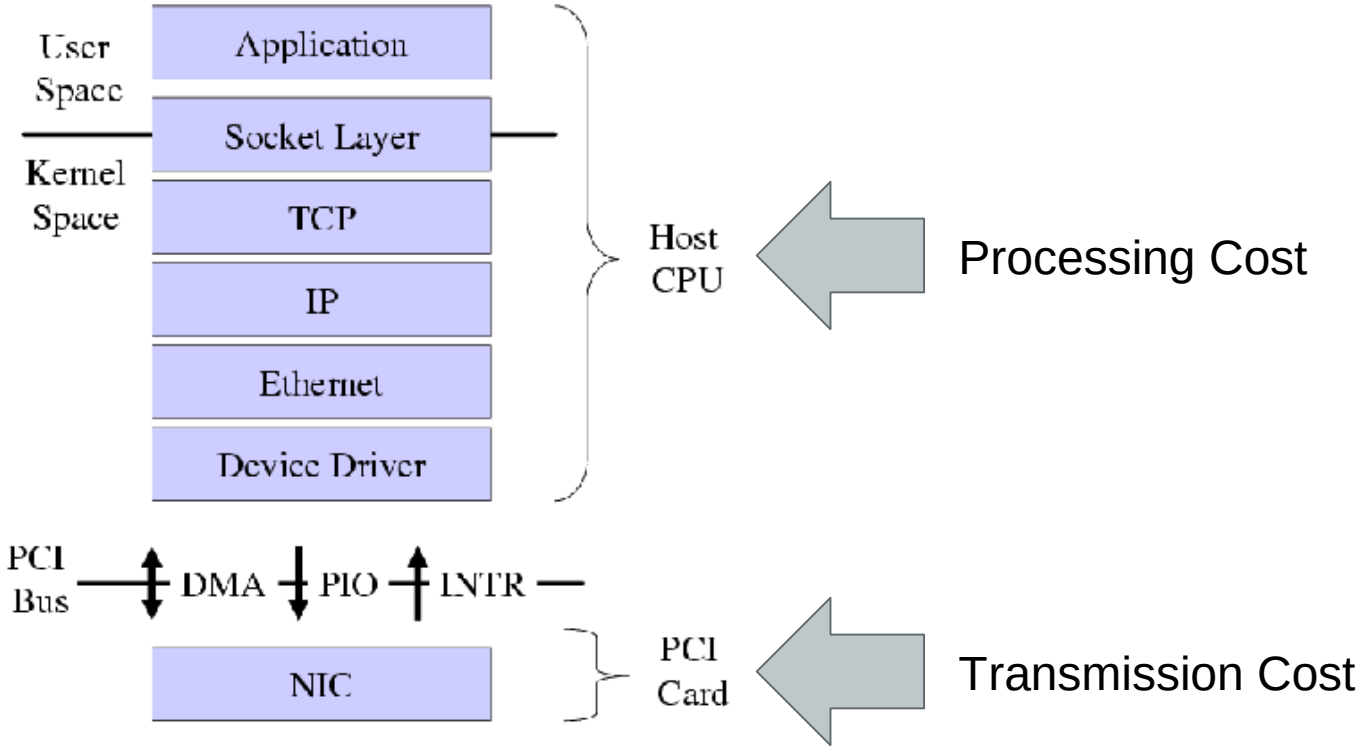
- Application buffer -> Socket Buffer
- Attach headers
- Move to Network Interface Card (NIC)

Receive

- Same story
- NIC buffer -> Socket Buffer
- Parse Headers



Costs


User Space
Application
Socket Layer
Kernel Space
TCP
IP
Ethernet
Device Driver
Host CPU
Processing Cost
PCI Bus
DMA
PIO
INTR
NIC
PCI Card
Transmission Cost

Transmission cost vs Processing cost

Network layers in a kernel are not a problem if transmission cost dominate!

- Large messages
- For example, video streaming
- Transmission cost >> Processing cost

More often than not, messages are small

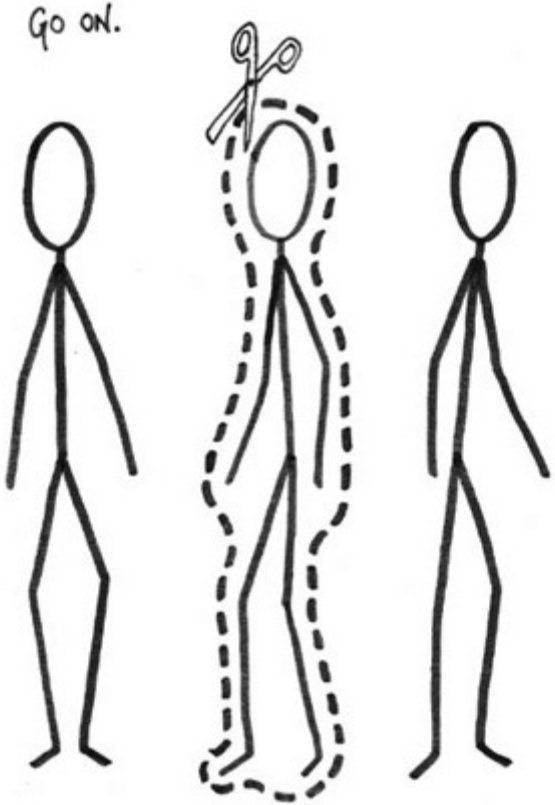
- Processing cost >> Transmission cost
- Layers increasing latency, decrease bandwidth

What are the problems with traditional networking?

What are the problems with this model?

- Messages go through Kernel
  - More processing
  - Applications have to interface with Kernel
- Multiple copies of same message
  - Unnecessary replication
- Low Flexibility
  - Protocol processing inside Kernel means no new interfaces

Goals, Takeaways, and Secret Sauce


Go on.
CUT OUT THE MIDDLE MAN.

U-Net Goals

- Put network processing at user level
  - Sort of like an Exokernel
  - This bypasses the Kernel (the middleman)
- Decrease number of copies
  - Holy Grail: Zero Copy Networking
  - No copying in network activity
- High protocol flexibility
  - U-Net should be able to implement existing protocols for legacy reasons

U-Net Takeaways

Putting networking at user level increases performance

- Both Latency and Bandwidth

Networking analogy of exokernel

U-Net Secret Sauce

Create sockets at the user level

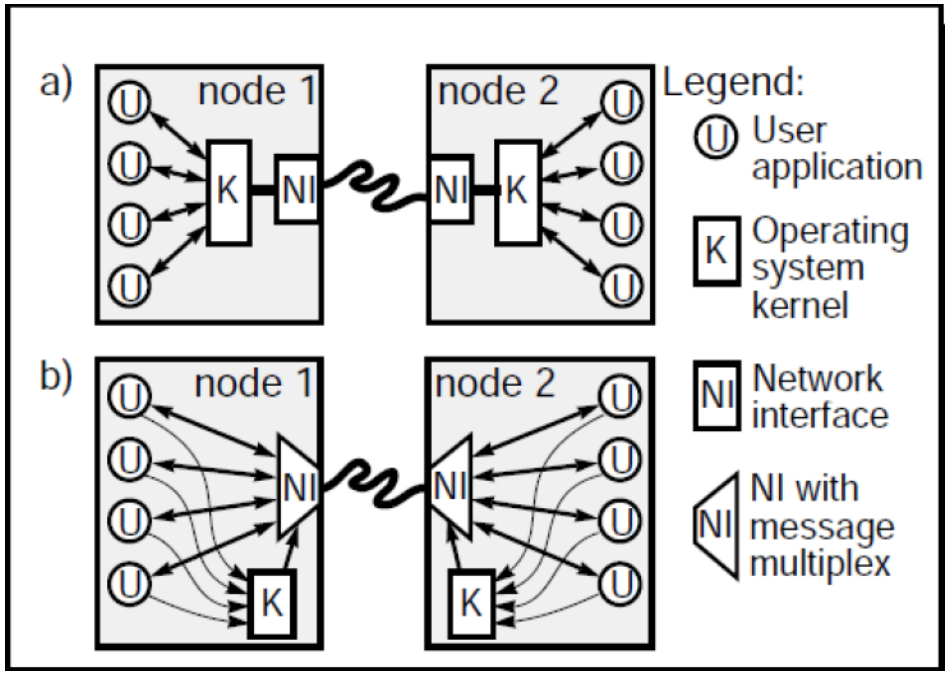
- Called endpoints in U-Net

Let Network Interface Card (NIC) handle networking instead of CPU

Related Work

- Mach3 (Exokernel)
  - User level implementation of TCP/IP
  - Not done for performance -no choice
- Parallel computing/HPC community
  - Required specialized hardware and software (e.g. no TCP)
  - Custom Machines = Expensive
  - Still holds today

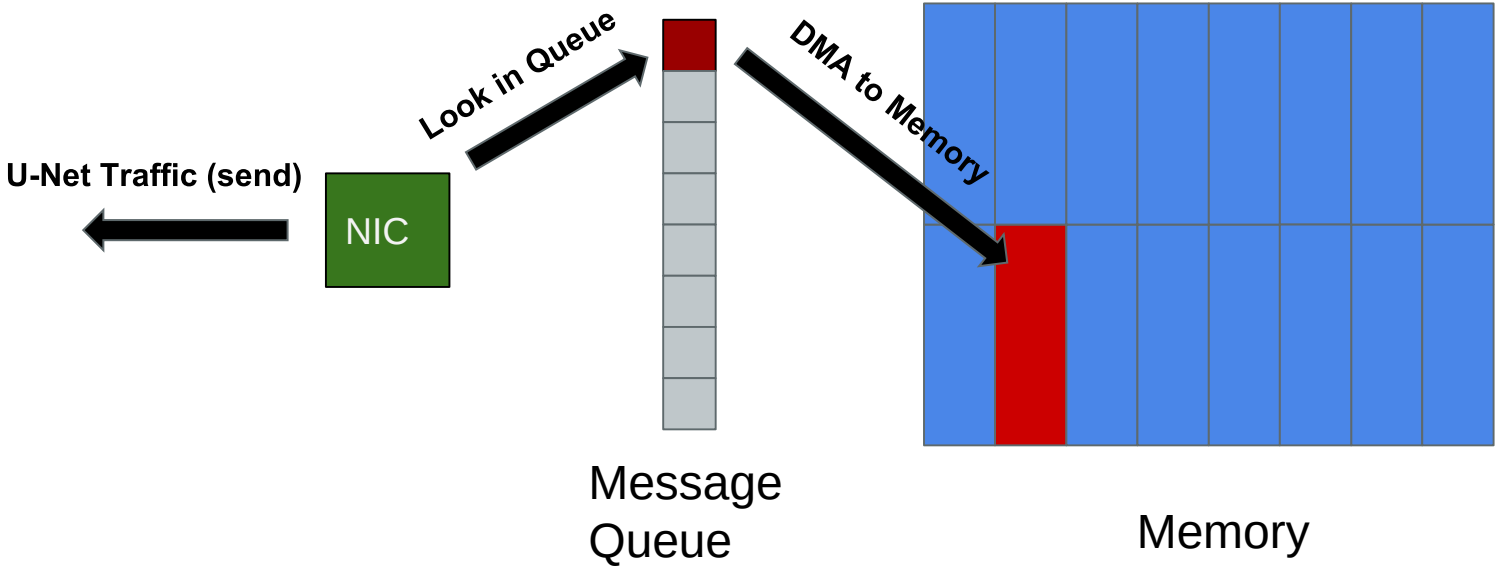
Inner Workings



(a) The traditional network. Kernel as a middleman

(b) U-Net. Direct access to network interface

A Simplified View of U-Net



U-Net Design

Endpoints are a handle into the network

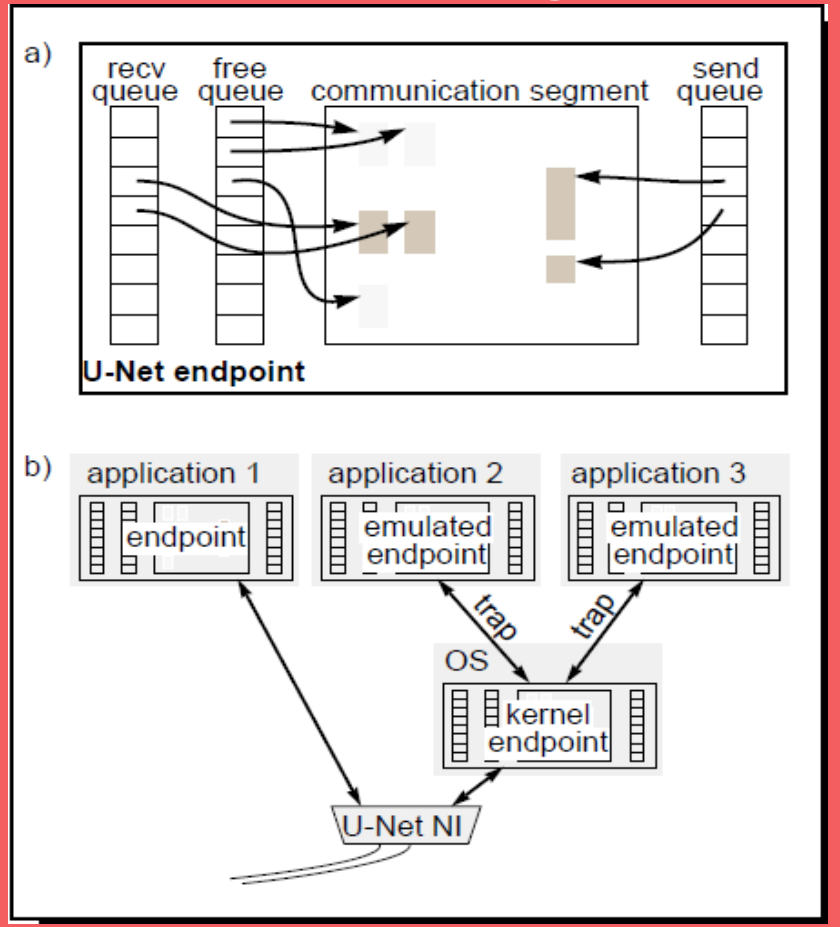
Sort of like socket

Communication Segments,

Sections of memory holding message contents

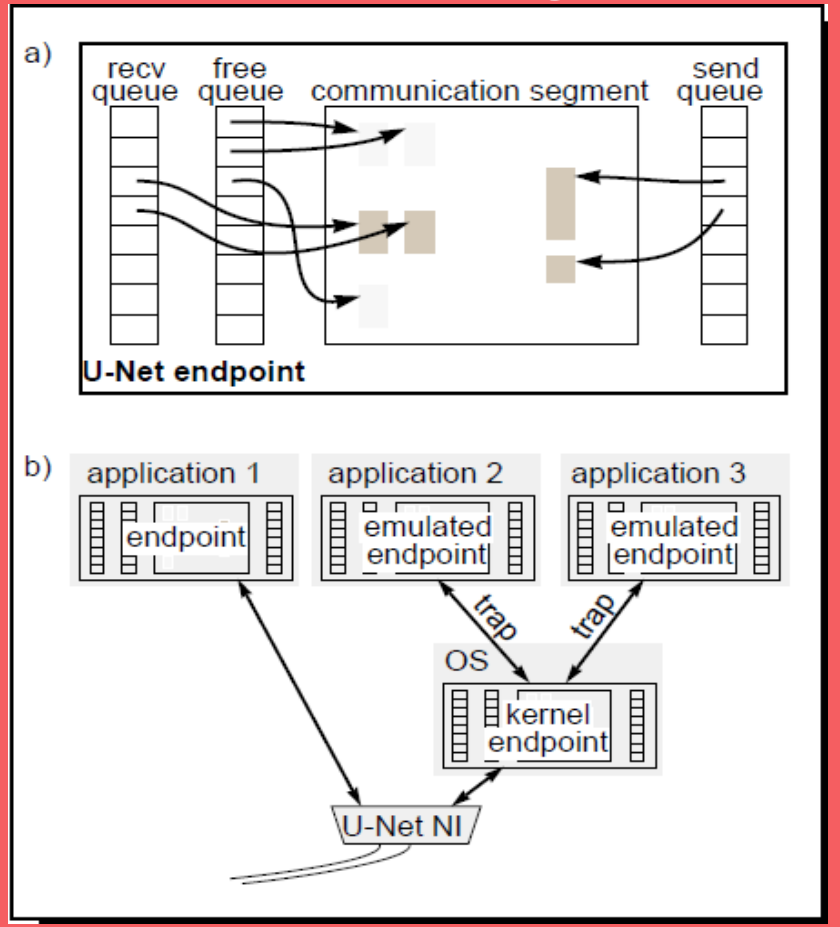
Message Queues hold descriptors of messages

Hold pointers, not data



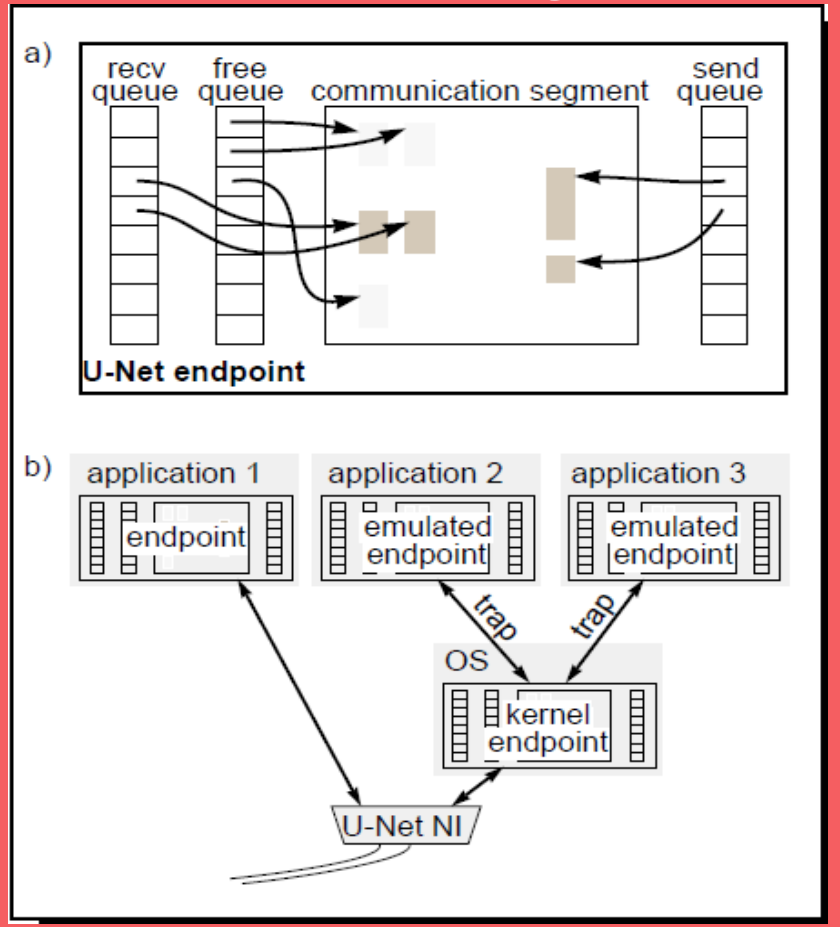
U-Net Design

User accessing U-Net create
endpoint, queues, alloc memory



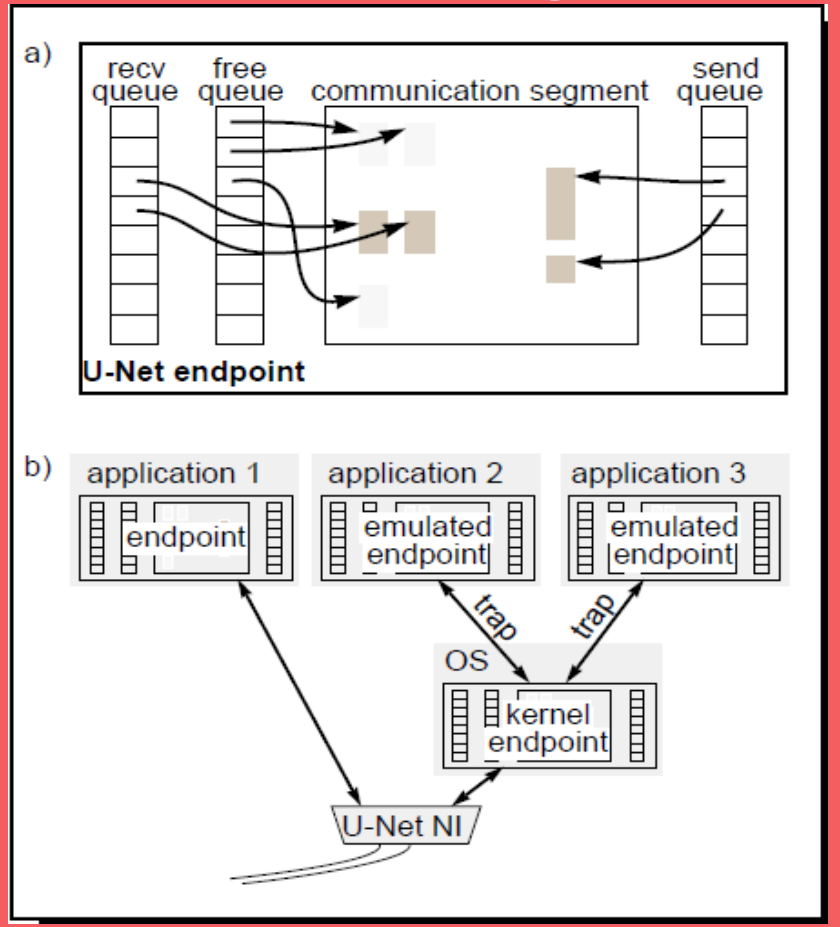
U-Net Design

1. To send, user puts message in communication segment
2. A descriptor (pointer) gets put into send queue
3. Looking in send queue, NIC grabs descriptor
4. Using a DMA, NIC retrieves message from communication segment and sends it out to receive endpoint



U-Net Design

1. Descriptor put onto receive queue
2. To receive, user looks in free queue for empty communication segment
3. Message then written into empty communication segment by NIC



NIC handles everything else!

Zero Copy and “True” Zero Copy

In the literature, a zero copy is one that does not use any excess memory

- I.e. Is not copied to a buffer first

Zero Copies in traditional networking isn’t “true”

- Buffering MUST occur between kernel and application
- Communication buffer (kernel) to application buffer

U-Net attempts to support true zero copying

- Authors note need for specialized hardware

Performance Numbers

Performance

Two measures of “Performance”: Latency and Bandwidth

- Latency: Delay in messages
- Bandwidth: bits/sec
- Highway analogy

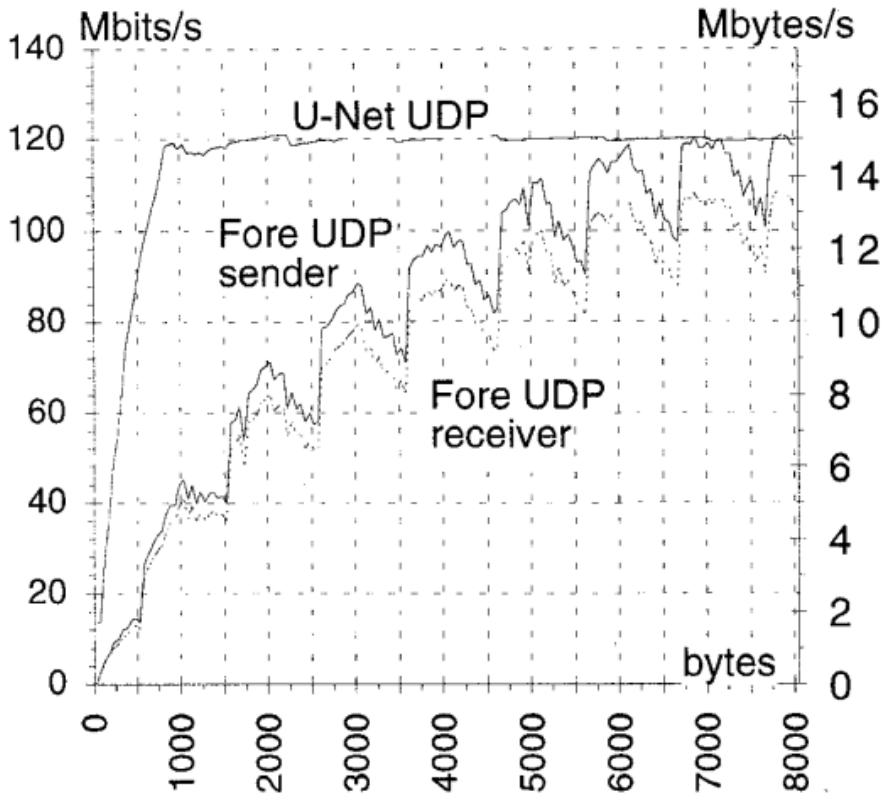
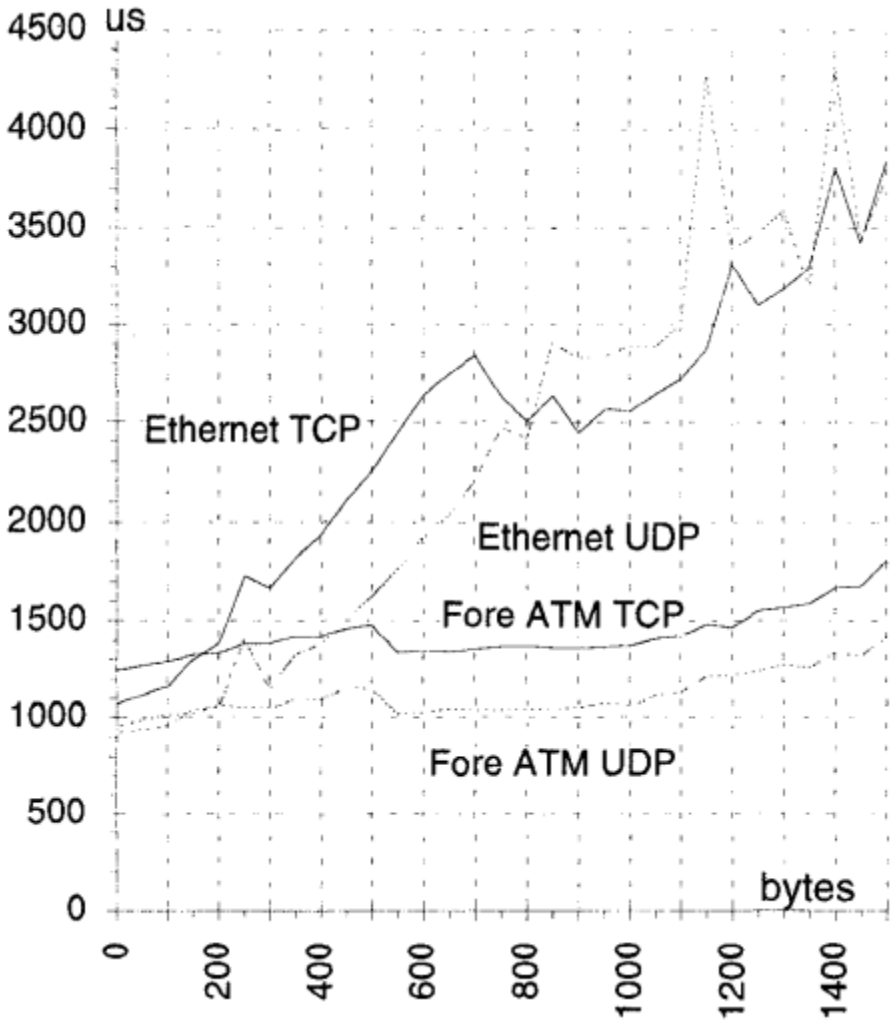


Figure 7: UDP bandwidth as a function of message size.


4500
us
4000
3500
3000
2500
2000
1500
1000
500
0
Ethernet TCP
Ethernet UDP
Fore ATM TCP
Fore ATM UDP
bytes
0
200
400
600
800
1000
1200
1400

Food for Thought: Trade-offs

What are some trade-offs associated with switching from OS level to application level networking?

Why is OS level networking far more popular?

Food for Thought: Trade-offs

- Development time vs performance
  - Application development requires re-implementation of key features
- Why is OS level networking far more popular?
  - Same reason exokernels/microkernels aren't successful!
  - Standard interface makes life easy for developer
  - Security
    - Without kernel, more things to worry about
  - Multiplexing

FaRM (2014)

Aleksandar Dragojevic (Ph.D EPFL)

Dushyanth Narayanan (Ph.D Carnegie Mellon)

Orion Hodson (Ph.D University College London)

Miguel Castro (Ph.D MIT)

FaRM

Relatively modern Distributed Computing Platform

Uses Remote Direct Memory Accesses (RDMA) for performance

RDMA


2016 RDMA
WINNERS
SEE ALL

RDMA: A History

- U-Net (1995)
- Virtual Interface Architecture (VIA) (1997)
  - U-Net interface + Remote DMA service
- RDMA (2001-2002)
  - Succeeds where prior work didn't
    - Widely adopted kernel-bypass networking
    - Standard "interface" (known as verbs)
      - Not a real interface, verbs define legal operations

RDMA over Infiniband

Infiniband networks = HPC Networks

RDMA traditionally used in Infiniband Networks

- Infiniband has a number of vendors, including Intel, Qlogic, and Mellanox
- Used extensively in HPC machines (Supercomputers)
- Expensive, requires specialized hardware (physical network and NIC)
- 100Gb/s standard

RDMA over Ethernet/TCP

- RoCE: RDMA done over Ethernet instead of Infiniband
  - (RDMA over Converged Ethernet)
  - Still requires specialized hardware
  - Cheaper because need only specialized NICs
  - 40Gb/s (and maybe 60Gb/s)
  - RoCE seems to scale worse
- iWARP
  - RDMA over TCP

RDMA Today

Widely used in Data Centers, HPC, Storage Systems, Medical Imaging, etc

RoCE seems to be the most popular.

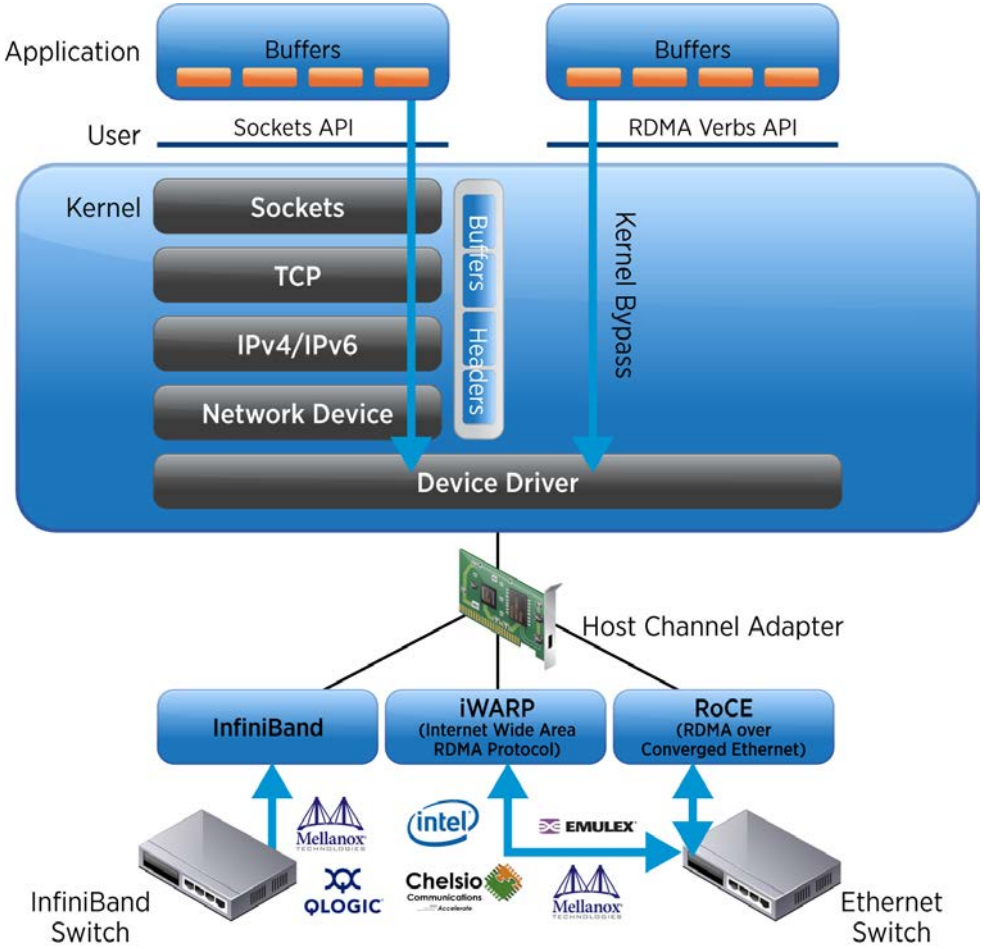
- Azure offers RDMA over Infiniband as well

Supported natively in (newer) Linux, some Windows, and OSX

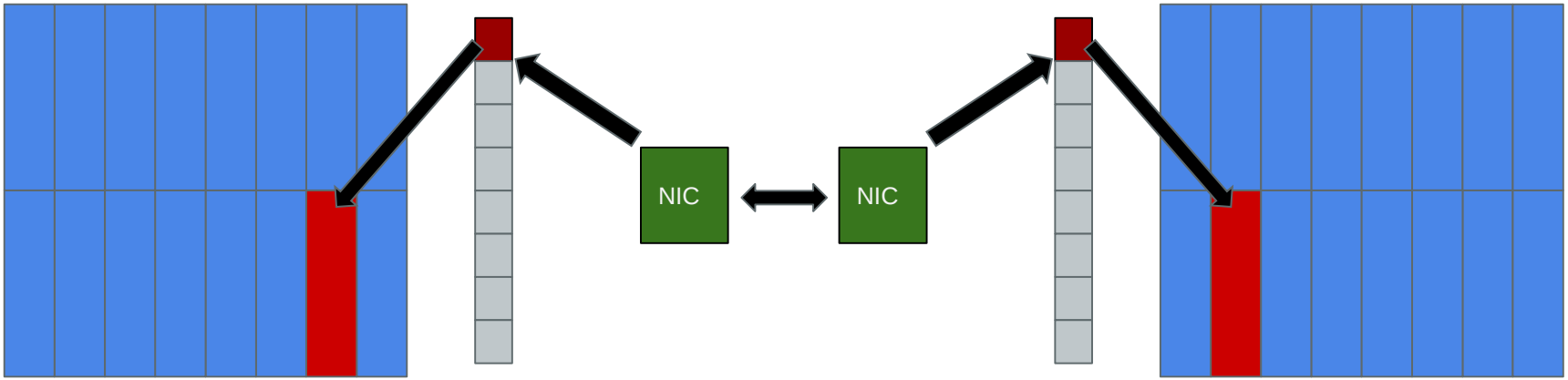
Bandwidth growing…1Tb/s in the future!

How RDMA Works

- RDMA traffic sent directly to NIC without interrupting CPU
- A remote memory region registers with the NIC first
  - NIC records virtual to physical page mappings.
  - When NIC receives RDMA request, it performs a Direct Memory Access into memory and returns the data to client.
- Kernel bypass on both sides of traffic


Application
Buffers
Buffers
User
Sockets API
RDMA Verbs API
Kernel
Sockets
TCP
IPv4/IPv6
Network Device
Buffers
Headers
Kernel Bypass
Device Driver
Host Channel Adapter
InfiniBand
iWARP
(Internet Wide Area
RDMA Protocol)
RoCE
(RDMA over
Converged Ethernet)
Mellanox
QLOGIC
intel
Chelsio
Communications
EMULEX
Mellanox
InfiniBand
Switch
Ethernet
Switch

A Simplified View of RDMA



Machine 1

Machine 2

Goals and Secret Sauce

FaRM Goals

If one has control over all machines, why worry about networking?

- Just write to memory directly between machines
  - Not an original idea (Sprite OS, Global Shared Memory, etc)

Streamline memory management

- The user should not worry about machine-level memory management

FaRM Secret Sauce

Use RDMA. Leads to massive performance gains in terms of latency

Use a Shared Memory Address Space

- Treat cluster memory as a shared resource
- Memory management is far easier
- Powerful Abstraction!

FaRM Design

FaRM is a distributed supports two main communication primitives

- One-sided RDMA reads
- RDMA message passing between nodes

FaRM Design

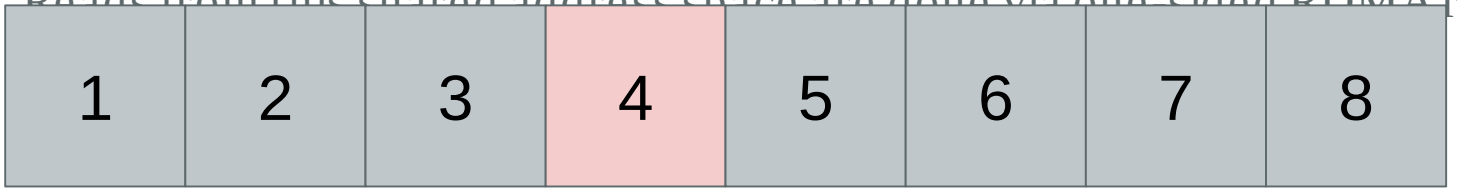
Shared address space

Memory of all machines in the cluster exposed as shared memory

This is powerful!

Requires some management

Reads from this shared address space are done via one-sided RDMA reads



FaRM Implementation

This shared memory abstraction must be maintained internally

- A machine associated with a piece of shared memory goes down
- Despite this, shared memory must still be consistent

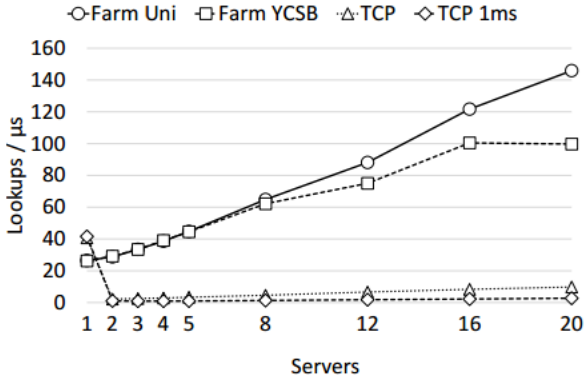
Map shared memory to machines via ring

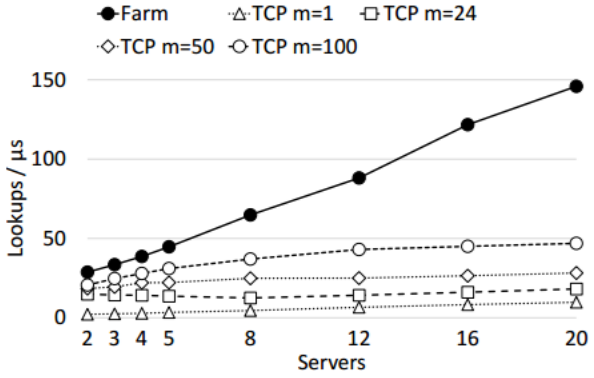
- Replication to guarantee fault-tolerance
- Membership determined using Zookeeper

Analogy

- DHT but where keys are memory addresses

Performance Numbers





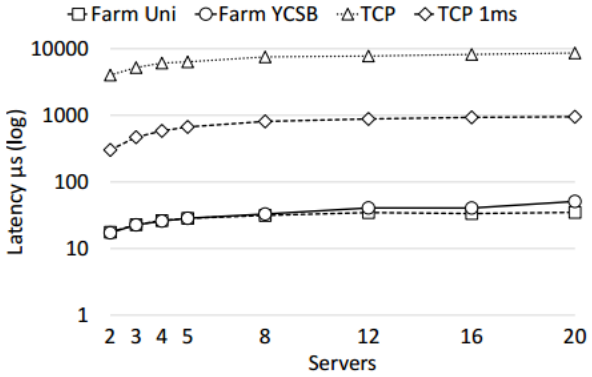


Figure 12: Key-value store: lookup scalability

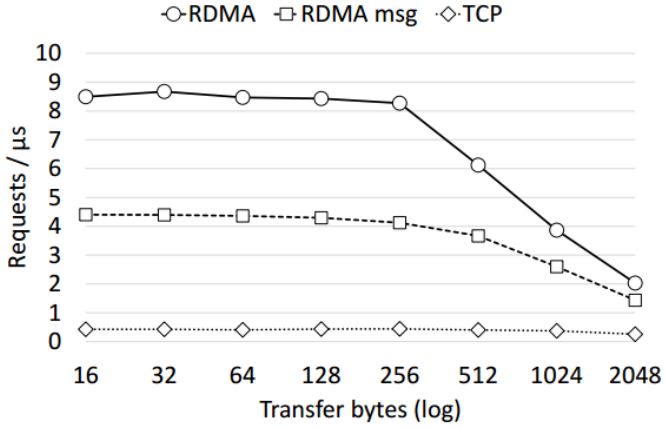
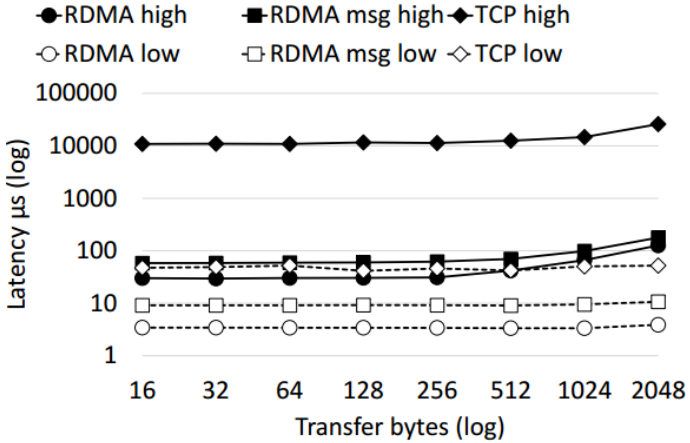


Figure 2: Random reads: request rate per machine



Food for Thought

RMDA improved upon a good idea by providing a standard interface

Is there any analogous thing in the exokernel case?

FaRM's shared memory abstraction is convenient

What are the trade-offs with this approach?