HIGH-PERFORMANCE NETWORKING
:: USER-LEVEL NETWORKING
:: REMOTE DIRECT MEMORY ACCESS

CS6410

Moontae Lee (Nov 20, 2014)

Part 1

Overview

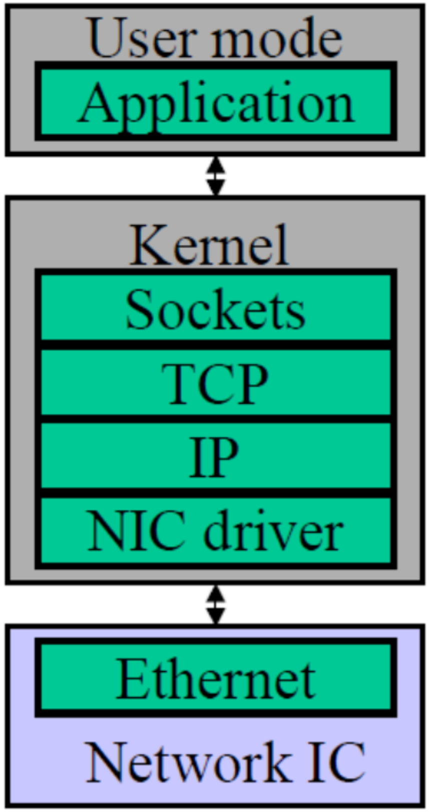
- □ Background
- □ User-level Networking (U-Net)
- □ Remote Direct Memory Access (RDMA)
- □ Performance

Index

00

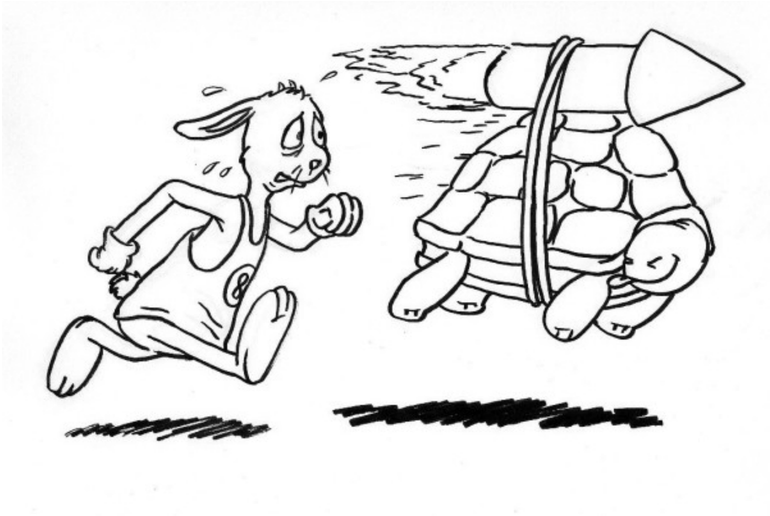
Background

Network Communication



- Send
  - Application buffer → Socket buffer
  - Attach headers
  - Data is pushed to NIC buffer
- Receive
  - NIC buffer → Socket buffer
  - Parsing headers
  - Data is copied into Application buffer
  - Application is scheduled (context switching)

Today's Theme



Faster and lightweight communication!

Terms and Problems

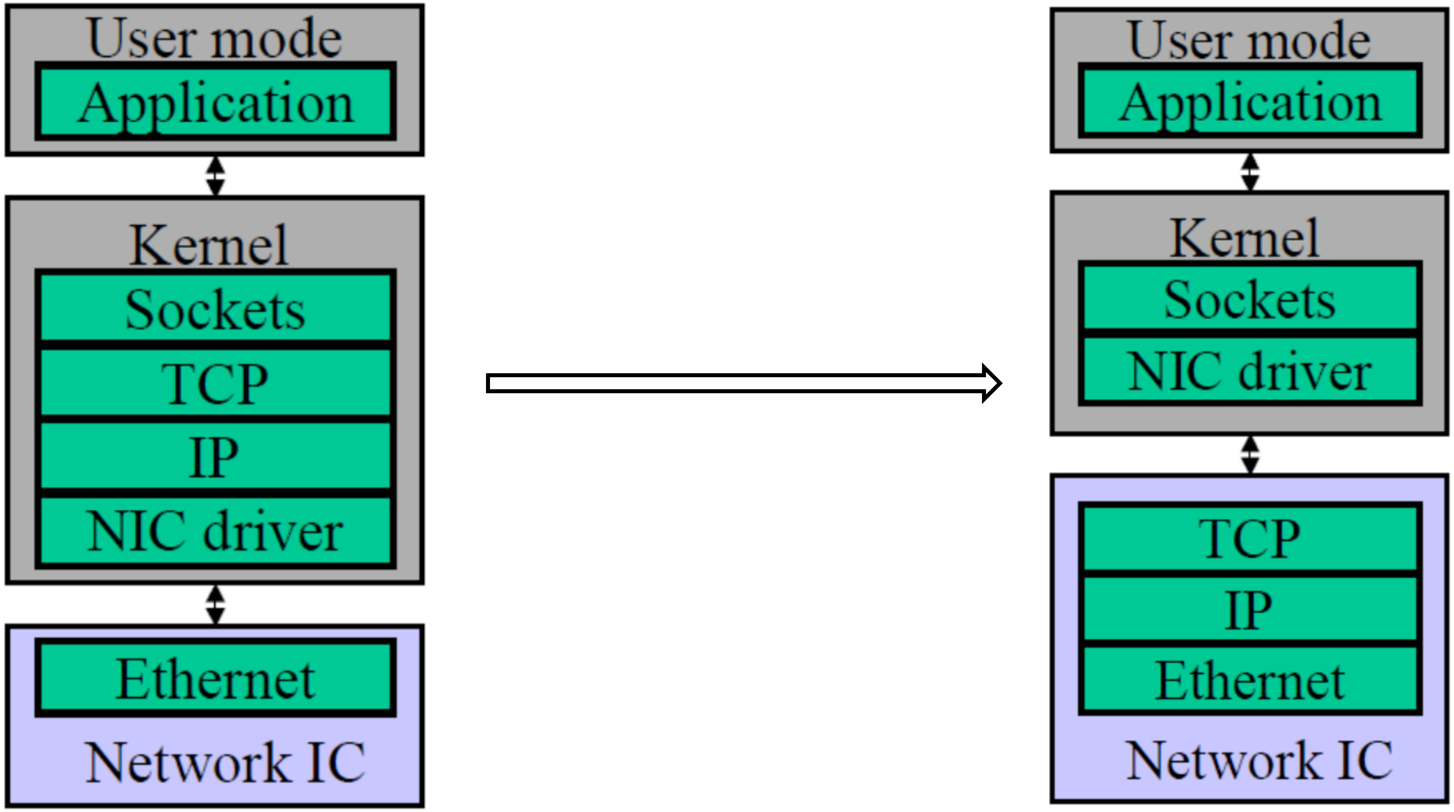
- Communication latency
  - Processing overhead: message-handling time at sending/receiving ends
  - Network latency: message transmission time between two ends (i.e., end-to-end latency)

Terms and Problems

- Communication latency
  - Processing overhead: message-handling time at sending/receiving ends
  - Network latency: message transmission time between two ends (i.e., end-to-end latency)

- If network environment satisfies
  - High bandwidth / Low network latency
  - Long connection durations / Relatively few connections

TCP Offloading Engine (TOE)



THIS IS **NOT** OUR STORY!

Our Story

- Large vs Small messages
  - Large: transmission dominant → new networks improves (e.g., video/audio stream)
  - Small: processing dominant → new paradigm improves (e.g., just a few hundred bytes)

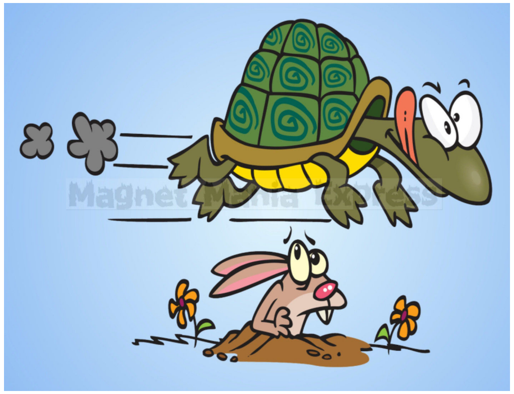
Our Story

- Large vs Small messages
  - Large: transmission dominant → new networks improves
    (e.g., video/audio stream)
  - Small: processing dominant → new paradigm improves
    (e.g., just a few hundred bytes)

- Our underlying picture
  - Sending many small messages in LAN
  - Processing overhead is overwhelming
    (e.g., buffer management, message copies, interrupt)

Traditional Architecture

- Problem: Messages pass through the kernel
  - Low performance
    - Duplicate several copies
    - Multiple abstractions between device driver and user apps
  - Low flexibility
    - All protocol processing inside the kernel
    - Hard to support new protocols and new message send/receive interfaces



History of High-Performance

- User-level Networking (U-Net)
  - One of the first kernel-bypassing systems

- Virtual Interface Architecture (VIA)
  - First attempt to standardize user-level communication
  - Combine U-Net interface with remote DMA service

- Remote Direct Memory Access (RDMA)
  - Modern high-performance networking
  - Many other names, but sharing common themes

Index

00

U-Net

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

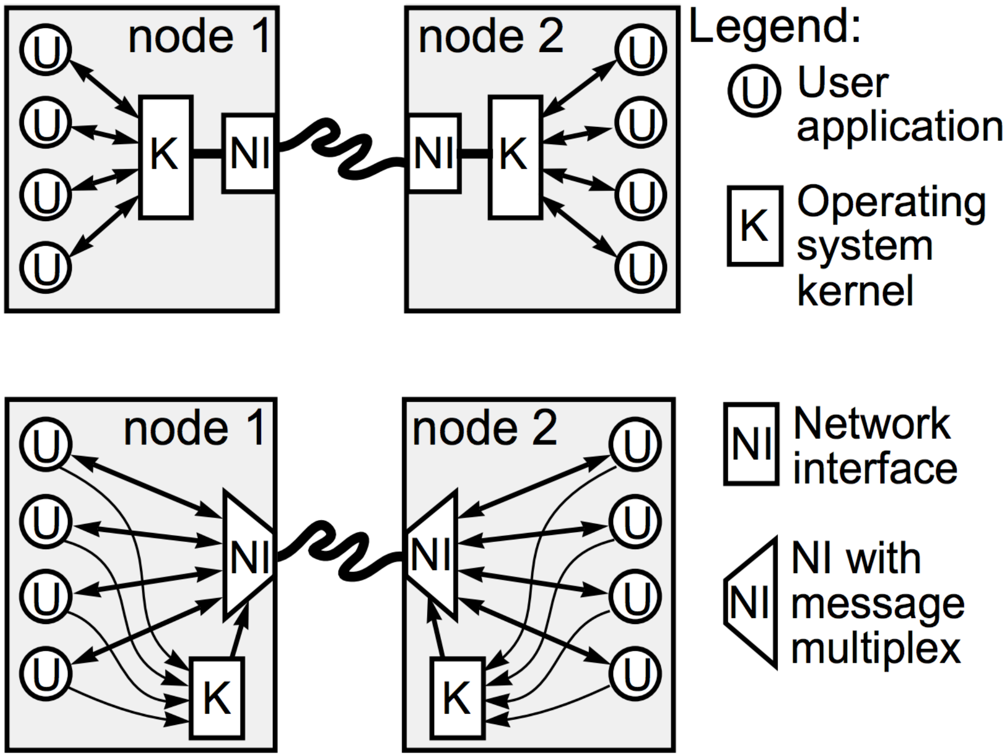
- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

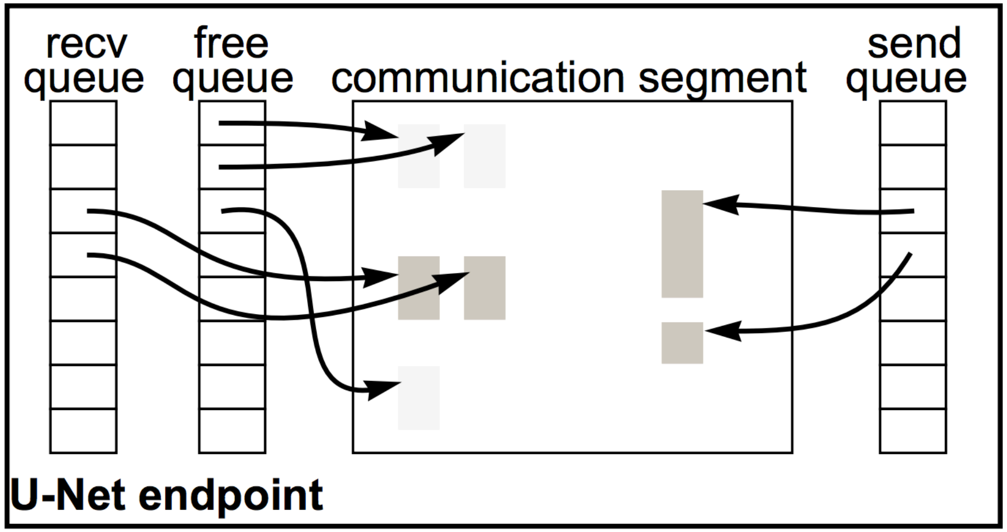
U-Net Architecture



- Traditionally
  - Kernel controls network
  - All communications via the kernel
- U-Net
  - Applications can access network directly via MUX
  - Kernel involves only in connection setup

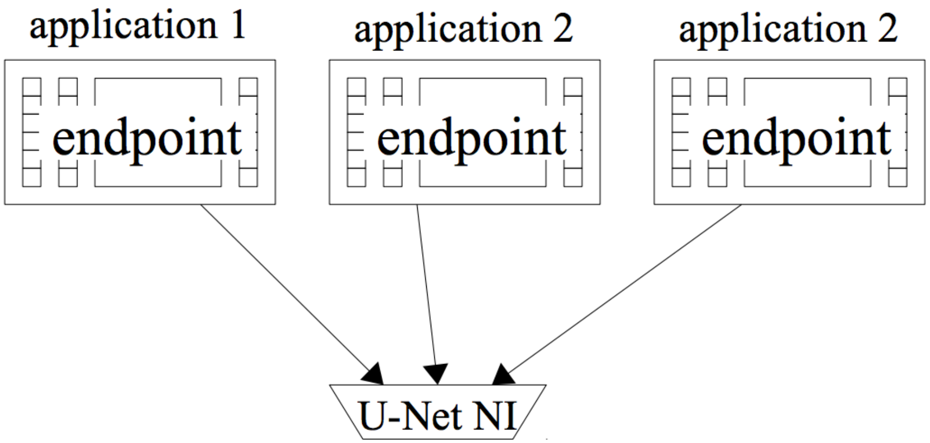
* Virtualize NI → provides each process the illusion of owning interface to network

U-Net Building Blocks



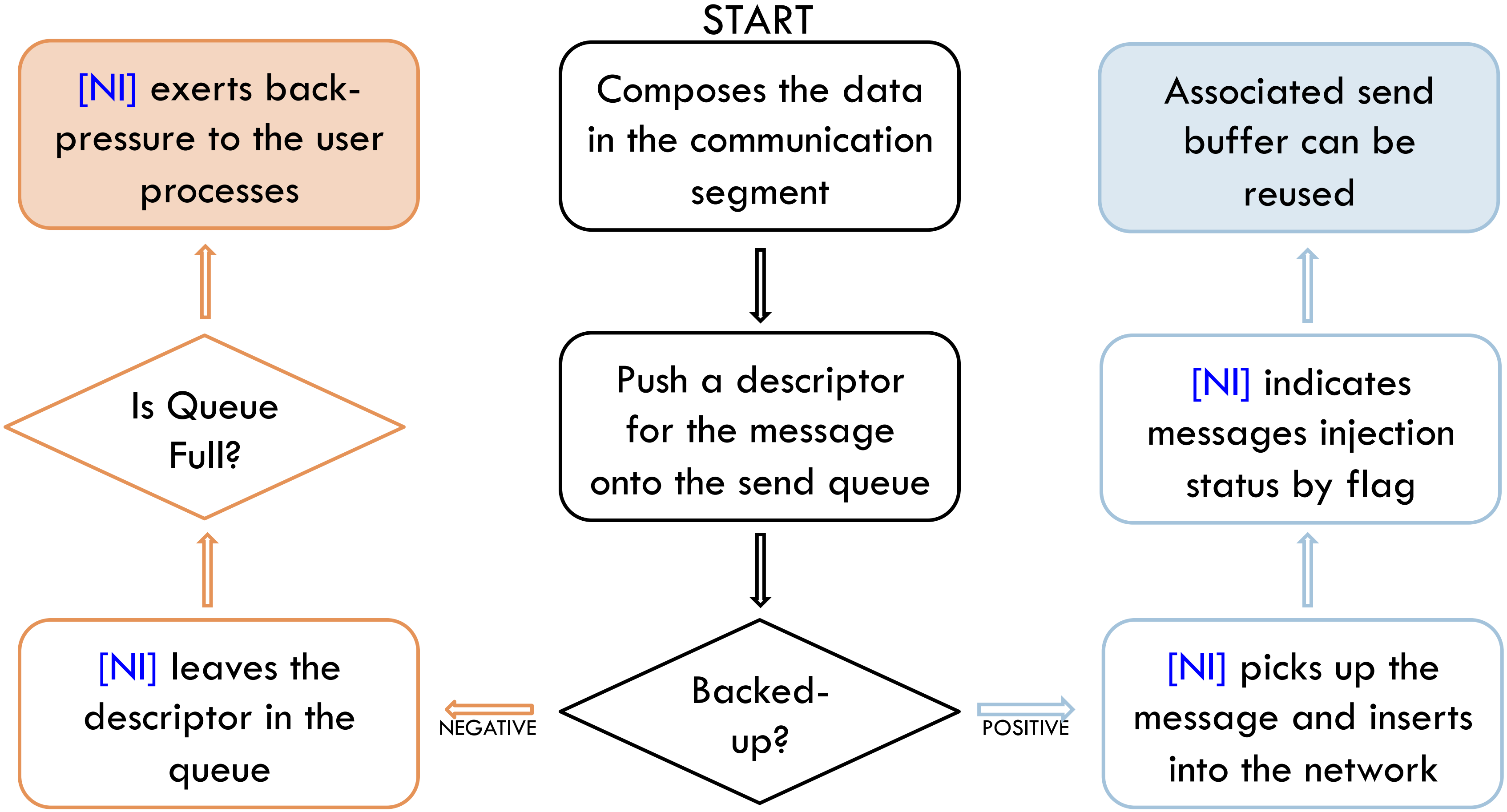
- □ **End points:** application's / kernel's handle into network
- □ **Communication segments:** memory buffers for sending/receiving messages data
- □ **Message queues:** hold descriptors for messages that are to be sent or have been received

U-Net Communication: Initialize



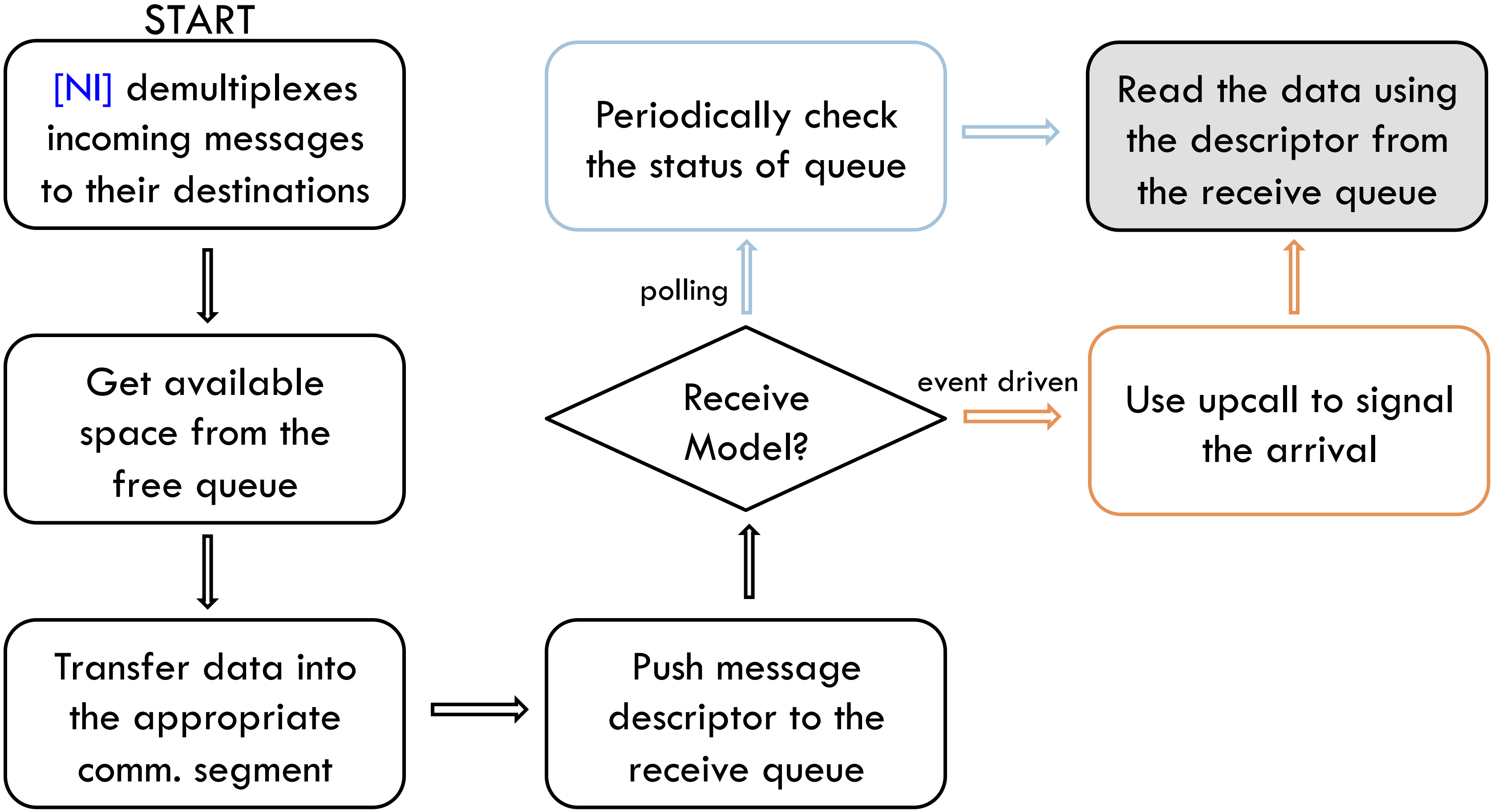
- Initialization:
  - Create single/multiple endpoints for each application
  - Associate a communication segment and send/receive/free message queues with each endpoint

U-Net Communication: Send


START
Composes the data in the communication segment
Push a descriptor for the message onto the send queue
Backed-up?
NEGATIVE
[NI] leaves the descriptor in the queue
Is Queue Full?
[NI] exerts back-pressure to the user processes
POSITIVE
[NI] picks up the message and inserts into the network
[NI] indicates messages injection status by flag
Associated send buffer can be reused

Send as simple as changing one or two pointers!

U-Net Communication: Receive



START

[NI] demultiplexes incoming messages to their destinations

Get available space from the free queue

Transfer data into the appropriate comm. segment

Push message descriptor to the receive queue

Receive Model?

polling

Periodically check the status of queue

event driven

Use upcall to signal the arrival

Read the data using the descriptor from the receive queue

Receive as simple as NIC changing one or two pointers!

U-Net Protection

- Owning process protection
  - Endpoints
  - Communication segments
  - Send/Receive/Free queues

Only owning process can access!

- Tag protection
  - Outgoing messages are tagged with the originating endpoint address
  - Incoming messages are only delivered to the correct destination endpoint

U-Net Zero Copy

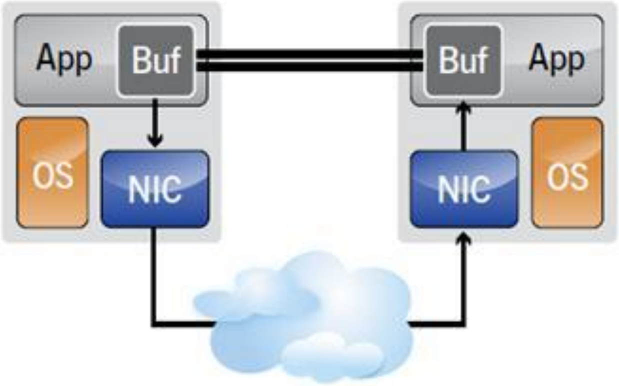
- Base-level U-Net (might not be ‘zero’ copy)
  - Send/receive needs a buffer
  - Requires a copy between application data structures and the buffer in the communication segment
  - Can also keep the application data structures in the buffer without requiring a copy
- Direct Access U-Net (true ‘zero’ copy)
  - Span the entire process address space
  - But requires special hardware support to check address

Index

RDMA

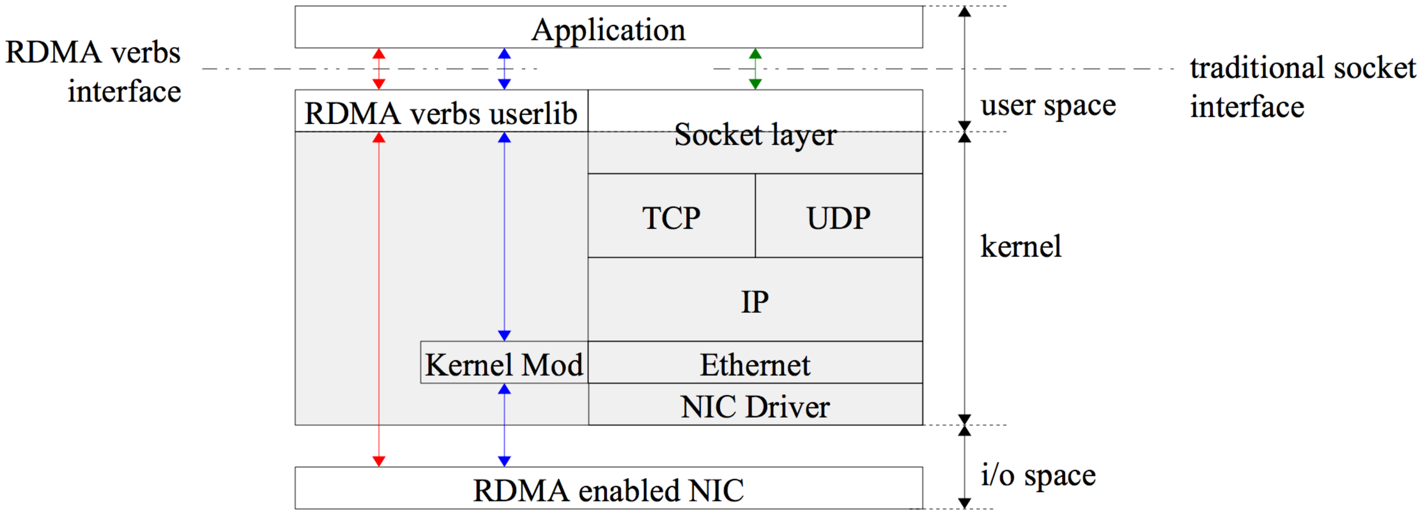
RDMA Ideas and Goals

- Move buffers between two applications via network



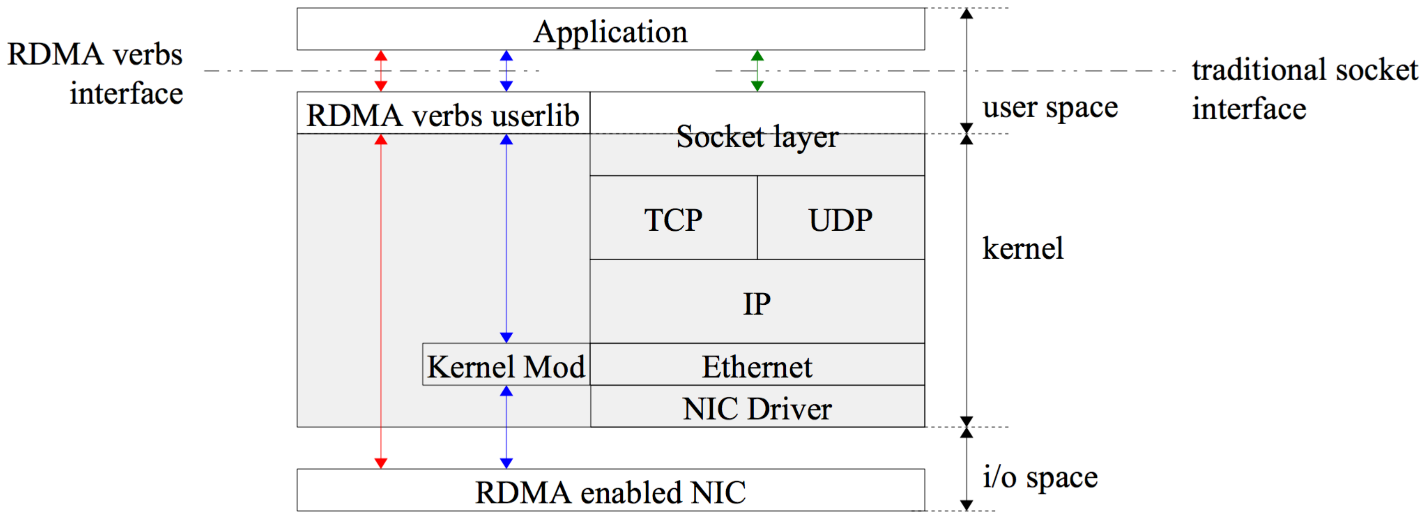
- Once programs implement RDMA:
  - Tries to achieve lowest latency and highest throughput
  - Smallest CPU footprint

RDMA Architecture (1/2)



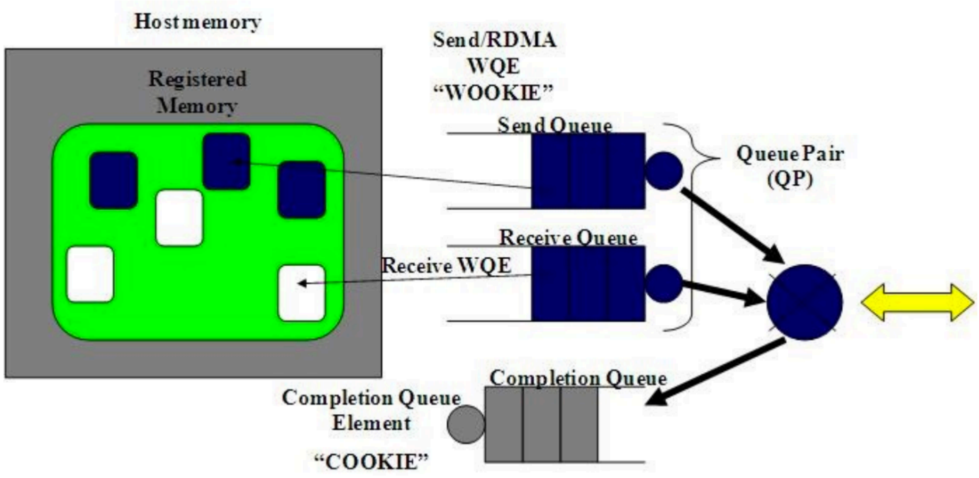
- Traditionally, socket interface involves the kernel
- Has a dedicated **verbs interface** instead of the socket interface
- Involves the kernel only on control path
- Can access rNIC directly from user space on data path bypassing kernel

RDMA Architecture (2/2)



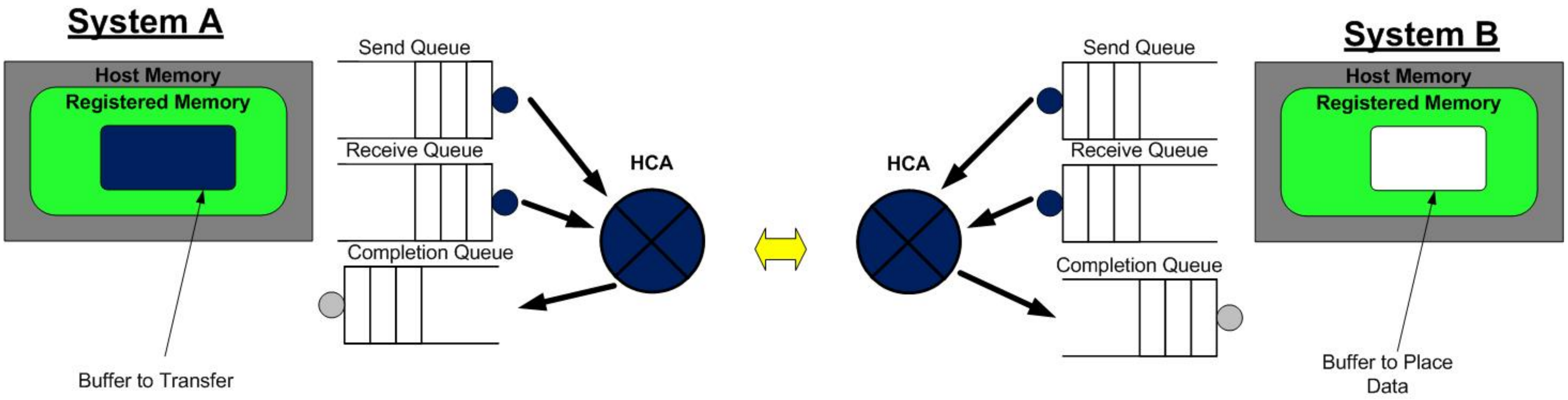
- To initiate RDMA, establish data path from RNIC to application memory
- **Verbs interface** provide API to establish these data path
- Once data path is established, directly read from/write to buffers
- Verbs interface is different from the traditional socket interface.

RDMA Building Blocks



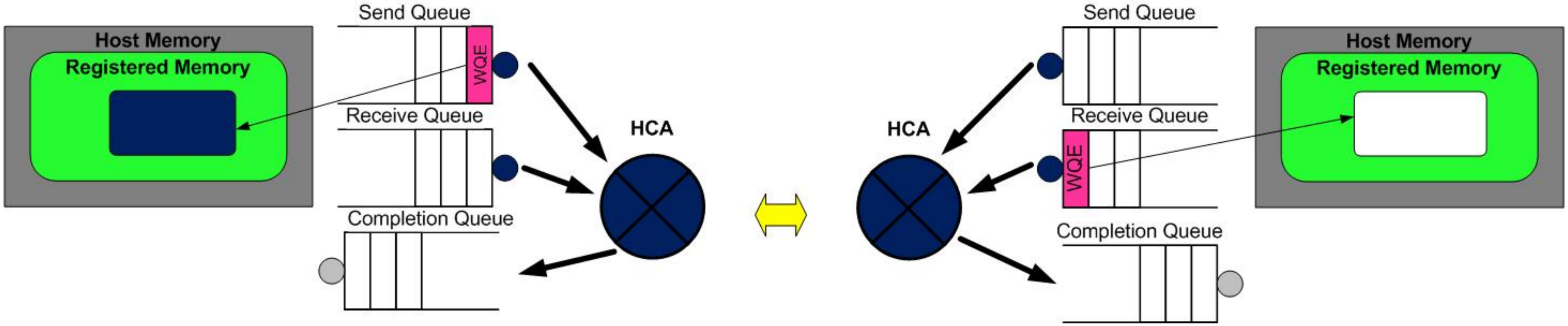
- Applications use **verb interfaces** in order to
  - Register memory: kernel ensures memory is pinned and accessible by DMA
  - Create a queue pair (QP): a pair of send/receive queues
  - Create a completion queue (CQ): RNIC puts a new completion-queue element into the CQ after an operation has completed.
  - Send/receive data

RDMA Communication (1/4)



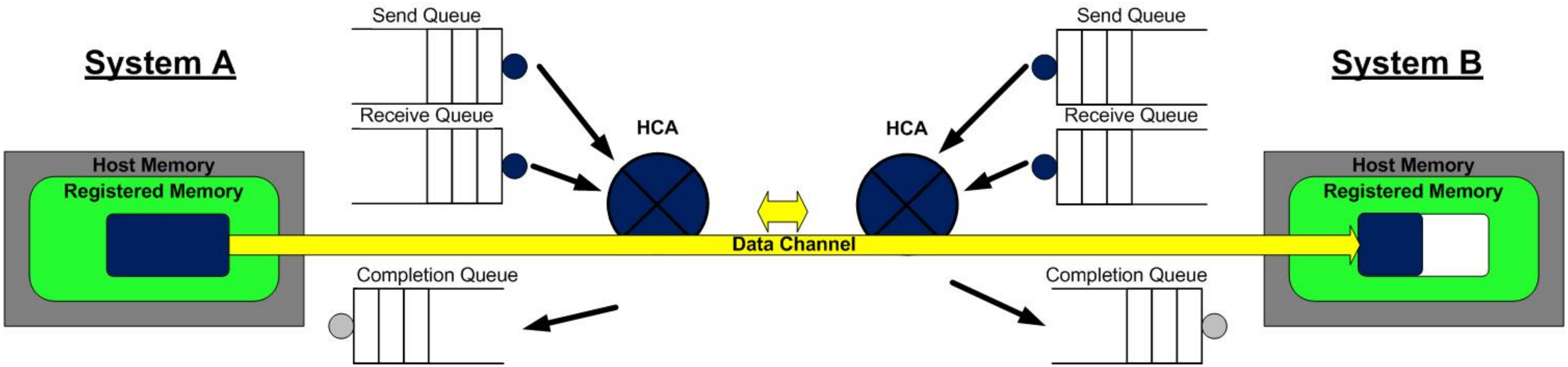
- Step 1

RDMA Communication (2/4)



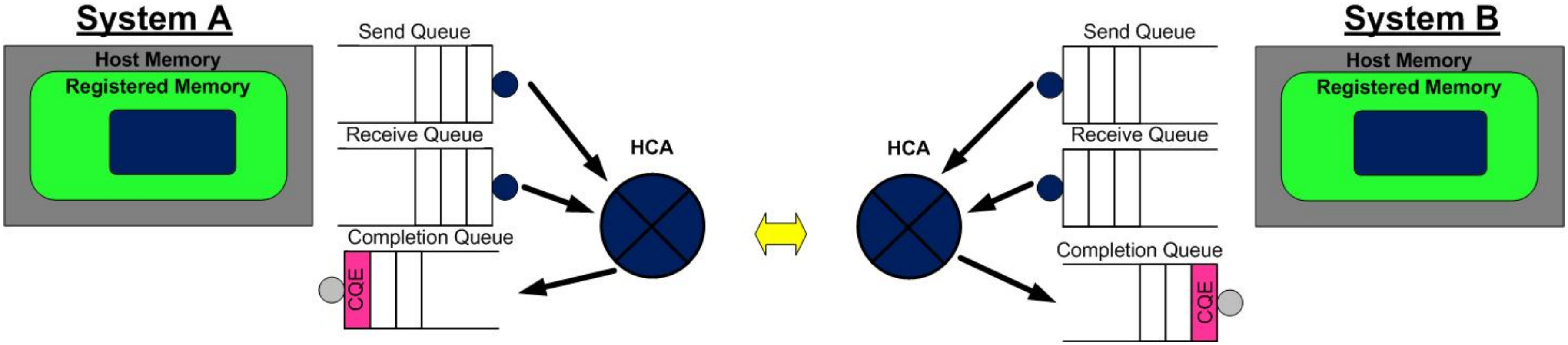
- Step 2

RDMA Communication (3/4)



- Step 3

RDMA Communication (4/4)



- Step 4

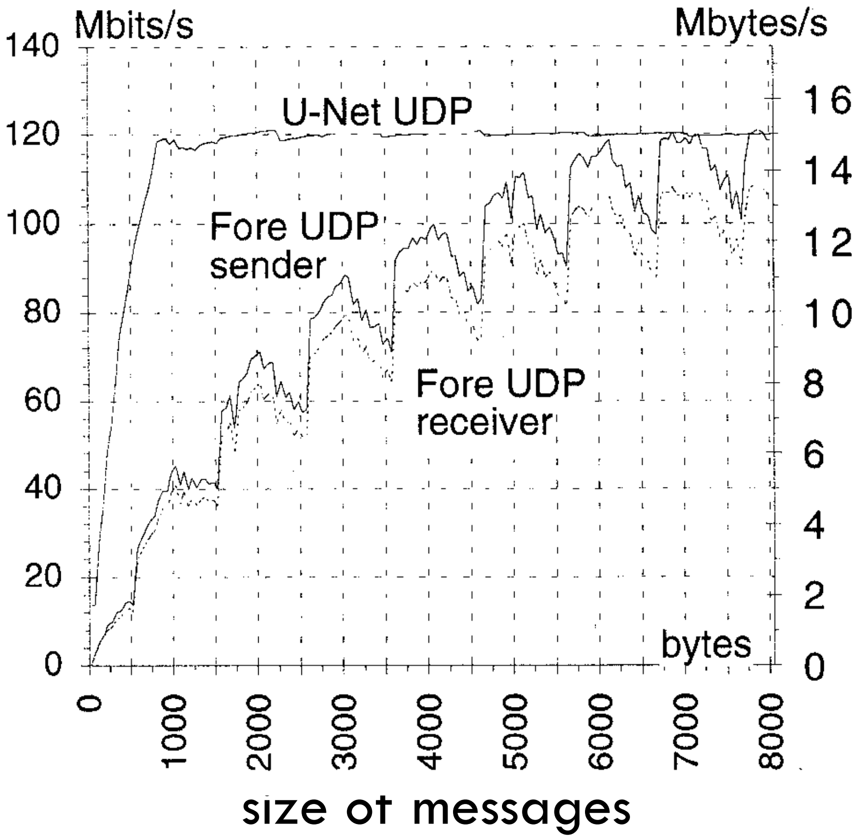
Index

00

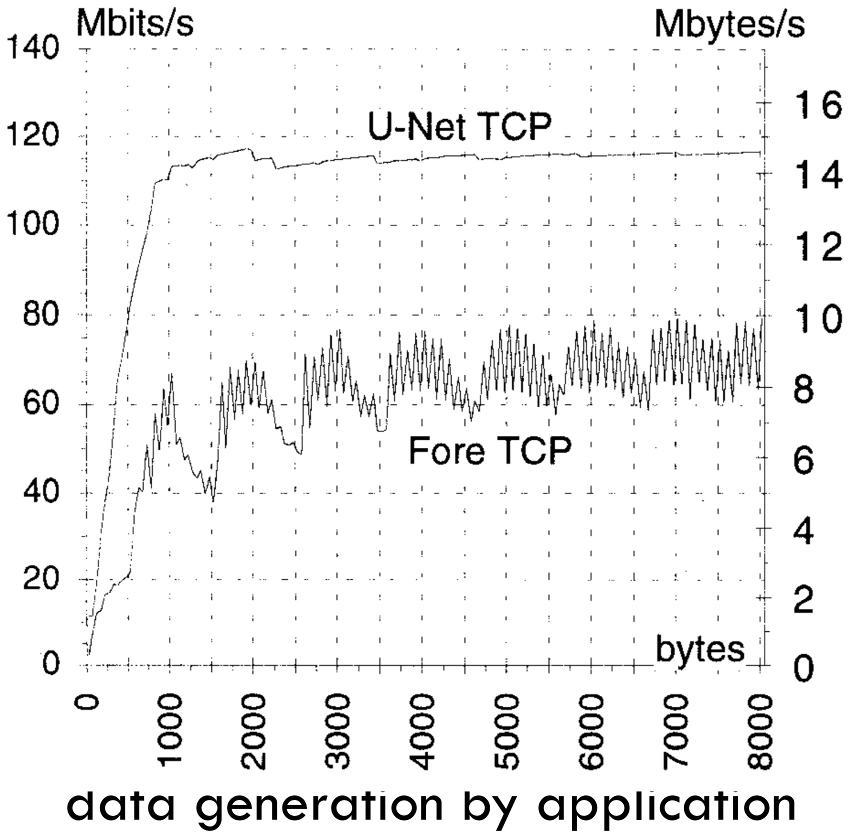
Performance

U-Net Performance: Bandwidth

* UDP bandwidth

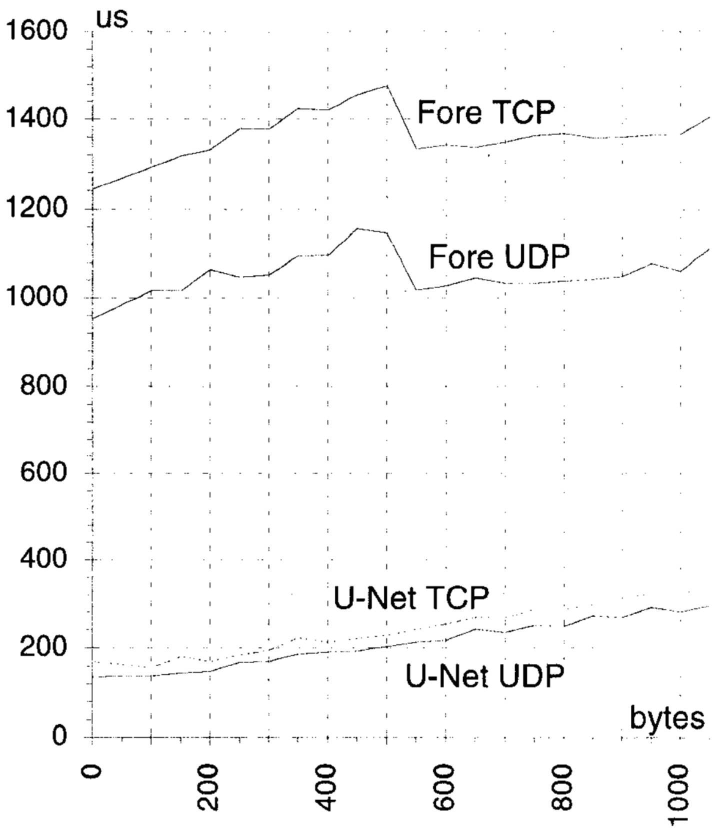


* TCP bandwidth



U-Net Performance: Latency

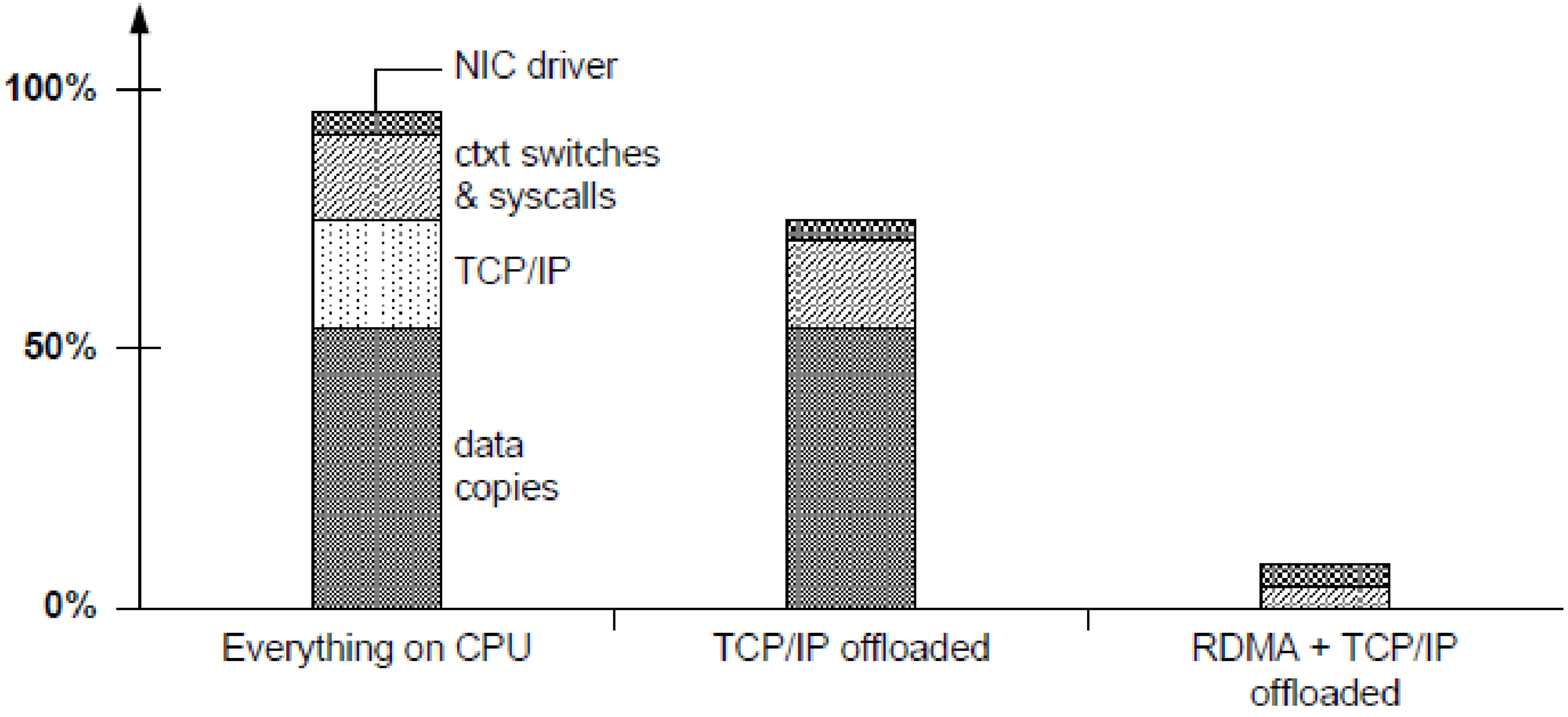
- End-to-end round trip latency



size of messages

RDMA Performance: CPU load

- CPU Load


NIC driver
ctxt switches & syscalls
TCP/IP
data copies
Everything on CPU
TCP/IP offloaded
RDMA + TCP/IP offloaded