# Faster!

Presenter: Saikat Guha

Cornell University

CS 614, Fall 2005

# Blast from the Past: circa 1995

- NOW : Networks of Workstations
  - Aggregate DRAM
  - Muliple CPUs
  - Network as I/O backplane
- Cluster Computing : Commodity supercomputing
- Gigabit network interconnects
  - Ethernet, ARP, IP ... solved problem. Right?

# Down with IP!

- Cluster computing
  - Few (thousands of) hosts
  - Simple, small topology
  - Network packet = function call
- IP solves a different problem
  - Global inter network
  - Planetary scale, multi-hop
  - IP data generally interactive, or bulk

# Down with IP!

- Baked into the kernel
  - Death by contention (Ethernet)
  - Death by congestion (ARP)
  - Death by latency (IP)
  - Death by processing overhead (Kernel)
- ATM to the rescue
  - Circuit switched
  - Low maximum overhead (high minimum overhead)
    - ATM: 10%
    - Ethernet: 30%
  - Supported by kernels ... as IP over ATM. D'oh!

# Look Ma, no kernel!

- By the power of: $\mu$-Kernel
  - sans user-space FS
  - sans user-space VM
  - sans all but user-space networking

*U-Net: A User-Level Network Interface for Parallel and Distributed Computing*

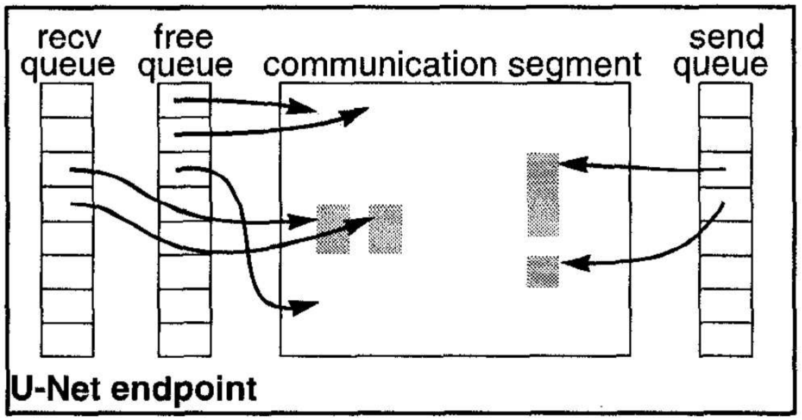
Thorsten von Eicken, Anindya Basu, Vineet Buch and Werner Vogels, Cornell University

# Back to the Future

- Zero-copy, *true* Zero-copy
  - Shared buffer (IO-Lite '99)
- Multiplex Network Interface (Exokernel '95)
- Input and Output queues (SEDA '01)
- Save on context switches ($L^4$ '97)

# U-Net is born

- User app makes syscall, creates endpoint




- Setup (ATM-like) channels to demultiplex
- Get a user-kernel (or user-hardware) shared buffer
- Compose data in buffer, send scatter-gather descriptor to Tx queue
- Trap to kernel
- For receive, poll or register upcall

# U-Net, fantastic! Fore, not so much.

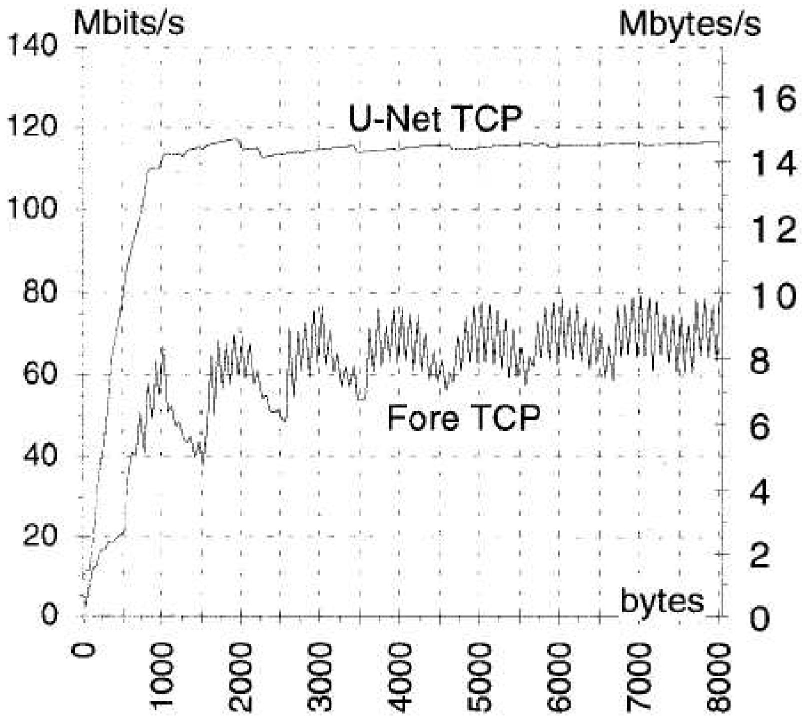



Figure 8: TCP bandwidth as a function of data generation by the application.

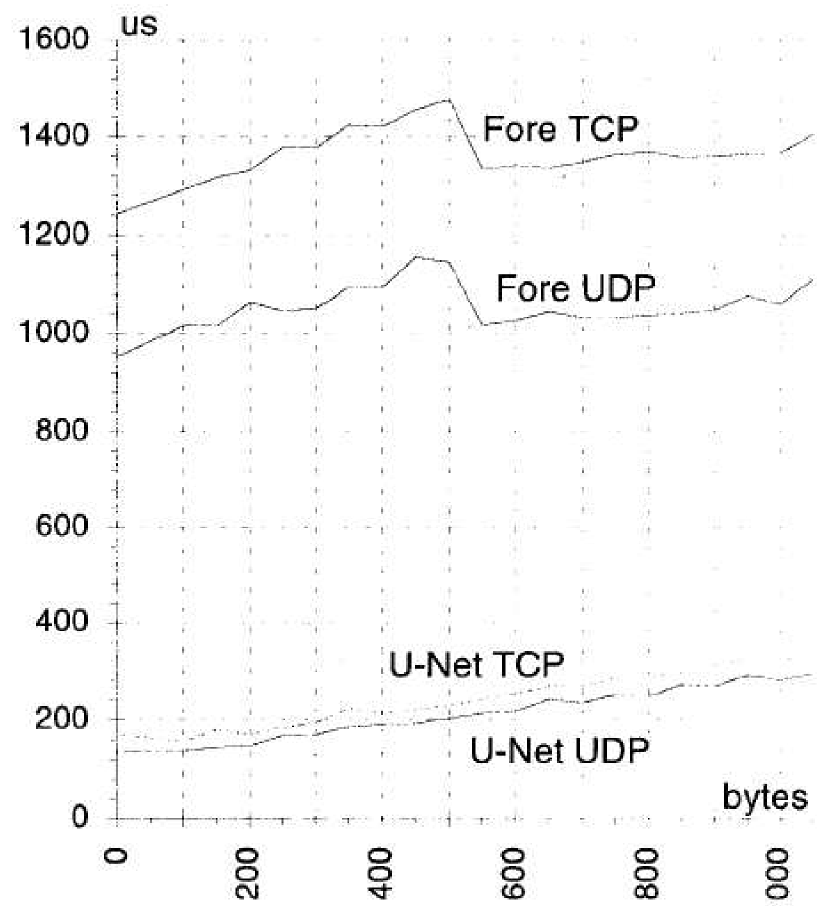



Figure 9: UDP and TCP round-trip latencies as a function of message size.

# Long live U-Net

- Restricts user application
  - U-Net with buffer management ’97. Welsh et al.
- Scalable?
  - Connections
  - Nodes
  - Interfaces
- Reinvent the wheel
  - Naming, Routing, Discovery
  - Reliability, QoS

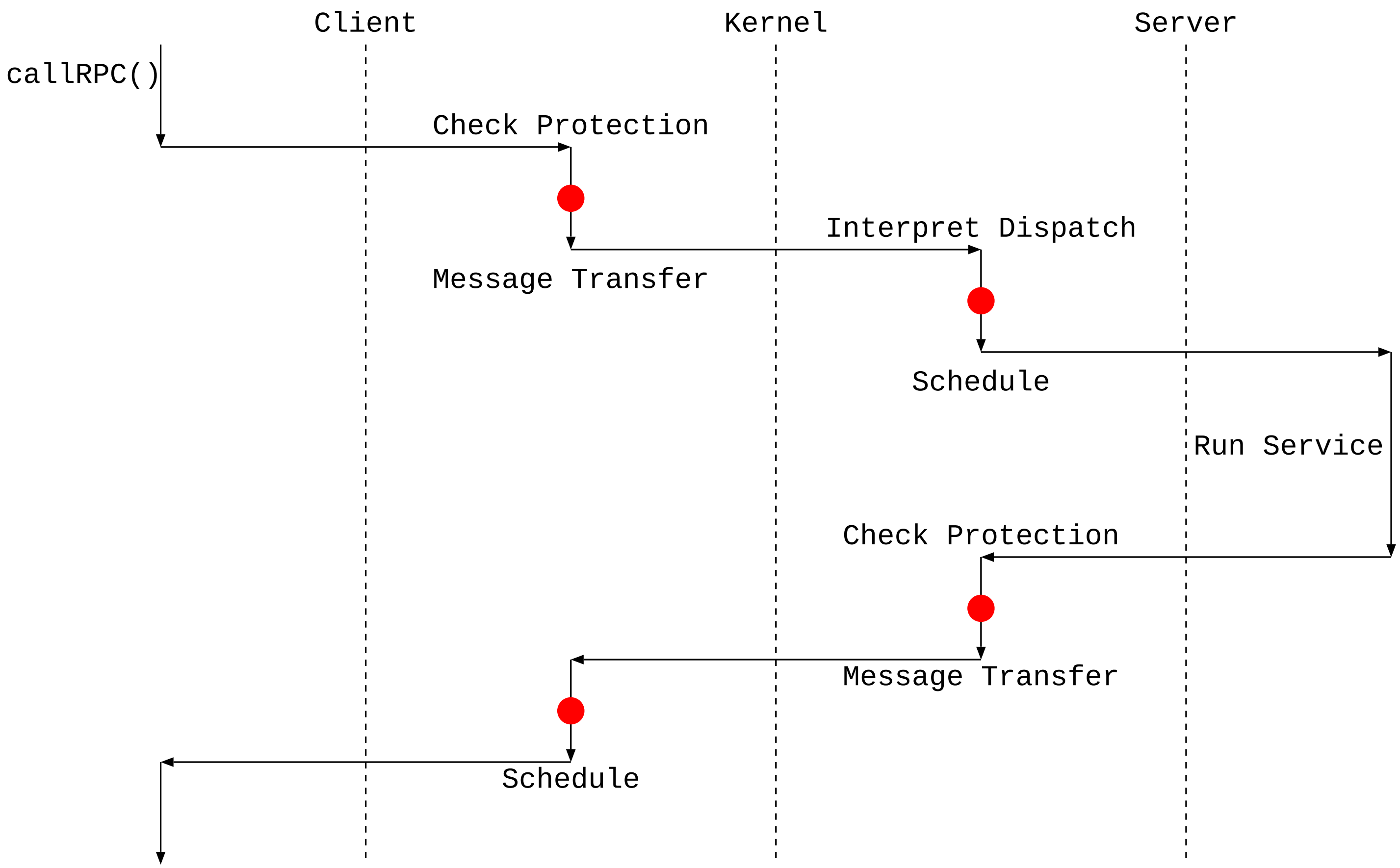
# U-Net meets Amdahl, Moore

- Does it really matter?
  - Cross-machine RPC: 0.6% – 5.3%
  - Are nodes still slower than networks?
- LRPC saves the world
  - Exploit machine-local RPC ($>$ 94%)
  - Reduce message copies
  - Reduce scheduling lag

*Lightweight Remote Procedure Call*

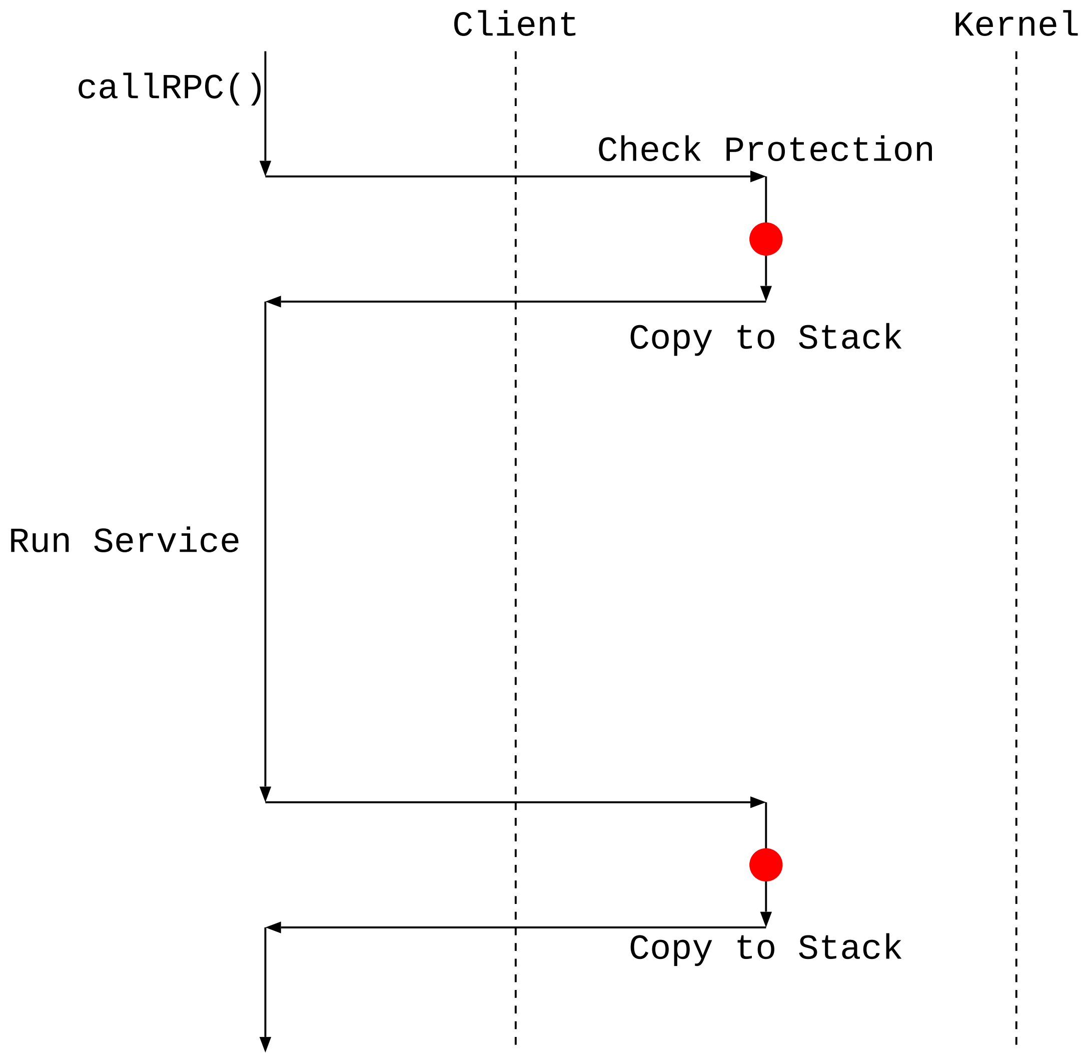
Brian Bershad, Thomas Anderson, Edward Lazowska, Henry Levy, UWash

# Copy-happy RPC




- Stub generation
- Buffer Overhead
- Context Switch $\times$ 2
- Scheduling $\times$ 2

# LRPC. Or perhaps just, *PC*




No scheduling required, just switch.

# Context Switch be Gone

- Optimization for multiprocessors
  - Cache contexts on idle processor
  - Instead of context switch, run cached proc.
  - Saves on TLB misses, cache misses etc
- No pessimization for remote calls
  - Fallback to *real* RPC
  - for complex local calls too

# Proof by Numbers

Table IV. LRPC Performance of Four Tests (in microseconds)

| Test | Description | LRPC/MP | LRPC | Taos |
|---|---|---|---|---|
| Null | The Null cross-domain call | 125 | 157 | 464 |
| Add | A procedure taking two 4-byte arguments and returning one 4-byte argument | 130 | 164 | 480 |
| BigIn | A procedure taking one 200-byte argument | 173 | 192 | 539 |
| BigInOut | A procedure taking and returning one 200-byte argument | 219 | 227 | 636 |

Table V. Breakdown of Time (in microseconds) for Single-Processor Null LRPC

| Operation | Minimum | LRPC overhead |
|---|---|---|
| Modula2+ procedure call | 7 | — |
| Two kernel traps | 36 | — |
| Two context switches | 66 | — |
| Stubs | — | 21 |
| Kernel transfer | — | 27 |
| Total | 109 | 48 |

# Under the rug

- Memory management costs
  - Allocate A-stack at bind time
- Resource migration
- Server control of degree of concurrency