THE U-NET USER-LEVEL NETWORK ARCHITECTURE

A Case for User-Level Network Protocols

SOSP — December 4th, 1995

Joint work with Werner Vogels, Anindya Basu, and Vineet Buch
U-Net Goal

Improve Communication in Networks of Workstations
• key for running parallel and tightly coupled distributed programs

Proposed solution: user-level network access
• overcome bottlenecks in current centralized networking layers

Motivation 1: Performance
• fully utilize high speed network fabrics
• support parallel processing
• tighter application-network coupling

Motivation 2: Flexibility
• variants of standard protocols
• new communication semantics
• different types of communication paradigms
Inspiration: Massively Parallel Machines

Key idea: allow user-level access to network interface device

• Examples: TMC CM-5, IBM SP-2, Meiko CS-2

• Advantages
  – bypasses the kernel for send/receive
  – no copy: DMA in/out of user memory
  – application-specific protocols

• Shortcomings
  – custom network interface, custom reliable network
  – assumes homogeneous nodes
  – restricts multi-user capabilities
Related Work

User-level protocol implementations
• Mach 3.0
• HP Jetstream
• AN1 TCP

Kernel bypass
• UW remote read/write instructions
• Active Messages over ATM
• Arizona ADCs
• Shrimp

Packet filter in the network interface
• Arizona Pathfinder
• HP Jetstream
The U-Net Approach

Targets
- Performance: 10x bandwidth and 0.1x latency
- Flexibility: TCP&Co. as well as parallel computations

Techniques
- use parallel machine technology
  - protected user-level access to the network
- build user-level protocols
  - thin networking software layers

Invariants
- off-the-shelf hardware and software
  - workstations, operating system, and network
- no compromise on protection
  - process A cannot inspect or corrupt process B’s messages
  - process A cannot impersonate process B
U-Net: Basic Idea

Traditional:
• kernel controls the network
• all communication goes via kernel

U-Net:
• applications send and receive directly via simple mux in NI
• kernel only involved in route set-up

Legend:
- User application
- Operating system kernel
- Message mux/demux
The U-Net Interface

- virtual device-like interface
  - buffers, queues, interrupts
- U-Net Endpoint: virtual device data structure

communication segment

recv queue

send queue

free bufs
Sending & Receiving Messages

Sending a message
- allocate buffer in communication segment
- assemble message in buffer
- push message descriptor onto send queue

Waiting for a message to arrive
- a) poll receive queue
- b) block until message arrival — select()
- c) asynchronous notification — signal()

Receiving a message
- pop message descriptor from receive queue
- consume message
- push (free) buffer descriptors onto free queue
U-Net implementation

Invariant 1: off-the-shelf hardware/OS
- Sun SPARCstations 10 and 20, SunOS 4.1.3
- Fore Systems SBA-200 ATM network interface (140Mbit/s)

U-Net software
- SBA-200 firmware
  - on-board i960 processor + 256Kb RAM + DMA
  - implements U-Net message mux/demux
- SunOS device driver
  - manages communication segments and queues (currently pinned)
  - controls application access to routes (ATM -> VCIs)

Invariant 2: protection
- VM system used to protect communication segments and queues
- table in NI maps between <endpt;channel> and VCIs
Raw U-Net Performance

Round-trip latency
(application to application and back)

- Single-cell messages: 65μs round-trip
  - 20μs: ATM switch & fiber
  - 40μs: due to i960&DMA

Bandwidth
(comm segment to comm segment)

- Fiber saturated with <1Kbyte msgs
TCP and UDP Bandwidth

Round-trip latency

- **U-Net TCP**
  - proof-of-concept user-level implem. (IP hdr, chksum, ICMP, IPoverATM)
  - 2Kbyte segments, 8Kbyte TCP window

- **Kernel TCP**
  - SunOS network stack, Fore ATM driver
  - 8Kbyte segments, 64Kbyte TCP window
U-Net Active Messages: Micro-benchmarks

• communication layer for parallel language implementations
• message = data + ptr to handler
• Active Message handler
  – small, custom code
  – moves data into data structures
    may send a reply back
• Small message latency
  – up to 32 bytes of data
  – 71\,\mu s round-trip
  – \sim 10\,\mu s CPU overhead
• Block transfer bandwidth
  – arbitrary source & dest virtual address
  invoke handler upon reception

![Graph showing U-Net and UAM performance](image-url)
Split-C: Application benchmarks

Machines

<table>
<thead>
<tr>
<th>Machine</th>
<th>CPU</th>
<th>Network bandwidth</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM-5</td>
<td>33Mhz</td>
<td>80Mbits/s</td>
<td>12µs</td>
</tr>
<tr>
<td></td>
<td>sparc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS-2</td>
<td>40Mhz</td>
<td>320Mbits/s</td>
<td>25µs</td>
</tr>
<tr>
<td></td>
<td>supersparc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U-Net</td>
<td>50/60Mhz</td>
<td>120Mbits/s</td>
<td>70µs</td>
</tr>
<tr>
<td></td>
<td>supersparc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results

- on 8 processors
- normalized to the CM-5
- ATM cluster is comparable in performance to CM-5 and CS-2
- But: ATM cluster ≠ parallel machine
Work in progress

Complete U-Net implementation
• kernel endpoint, kernel networking stack over U-Net
• ATM signalling (connection set-up)

U-Net ports to new platforms
• 100MBit ethernet, x86/PCI, Linux
  – kernel traps/interrupts for message mux/demux
• ATM, SPARCstation, Solaris

Future plans
• couple message reception and process scheduling
• framework for developing user-level protocols
• parallel/distributed program manager
Conclusions

Target 1: performance
- 140Mbit/s fiber saturated using <1Kbyte messages
- 65μs round-trip latency for small messages – key to parallel computing – key to high bandwidth

Target 2: flexibility
- TCP: saturates fiber, offers low round-trip latency
- Active Messages & Split-C: comparable to parallel machines

Invariants:
- off-the-shelf components
- U-Net firmware & driver are simpler than Fore’s
- multi-user network access with protection

=> Proof that Network of Workstations can perform
U-Net TCP/IP modifications

• Simple connection mux/demux based on VCIs
• Custom buffering (not *mbufs* or *streams bufs*):
  – no buffer copy, no strange fragmentation, simple allocation, pre-aligned
• Straight-forward acks:
  – traditional TCP has “delayed ack” algorithm: delay every other ack by up to 200ms to try and piggy-back onto “reply” message
  – with U-Net/AAL5 an ack uses one cell — “no overhead, no bandwidth”
• Reasonable timers
  – kernel uses 200ms and 500ms timers (e.g. for retransmission)
• Small window and message sizes
  – large window = expensive retransmission
  – large messages = high probability of loss and of slow-start
  – TCP/IP over U-Net buffers: 8Kb window & 2Kb segments