The network and the OS

David Clark
MIT CSAIL
October, 2015
From the specific to the cosmic

• Early issues were pragmatic and “mechanical”.
  – How to structure and position the code that implemented the protocols.
  – Performance.

• Later issues were more fundamental:
  – What does it mean for a machine to be connected to the rest of the world?
  – Security, availability
Structure

• To understand the issues of structure, must understand what is distinctive about implementing network protocols.
  – Start there, then look at implications for the OS.
What is different about net I/O?

• Variable size units (packets and application data).
• Malformed content and size.
• Internet connected heterogeneous machines over heterogeneous networks.
  • First (and in some sense only) goal was interoperation.
  • Byte order, 9 bit bytes, etc.
• Unpredictable arrival/transmission.
• Must be processed to demultiplex.
  — Trustworthy processing.
A 1986 perspective

Our state of understanding in 1986:
• A slide of mine from the time.

There was deep confusion as to how to move from protocol specification to protocol implementation.
Implementing a protocol

• The stages in our understanding. What was the challenge?
  – Implementing the state machine.
  – Marshalling the packet fields.
  – Dealing with errors.
  – Processing 32 bit numbers.
  – Copying the data.
  – Dealing with congestion control.
  – Dispatching the packet to correct connection.
  – Dealing with layers
Where to put the software?

Protocol in the OS?
  • Low overhead.
  • Nasty programming environment.
  • Run all the code at interrupt time?
Protocol in the application process?
  • No asynchrony.
  • Easy invocation.
Protocol in a separate process?
  • High cost to invoke.
  • Asynchronous execution.
Waiting for events

• Protocols have an odd (by the thinking of the day) structure.
  – They wait for multiple events.
  – A user event, a network event, a timer event.

• Many interprocess scheduling mechanisms required the waiting process to wait on one event.
Performance

• We had to learn the relative cost of different actions.
  – Processing a header.
  – Scheduling a process/thread.
  – Setting a timer.
  – Taking an interrupt.
  – Copying the data.
  – Dispatching the packet.
Protocols can be simple

Implementation of TCP input routine for Xerox Alto.

It fit on one page.

– It does call subroutines...
Layers of protocol

• Link, IP, TCP, app.
• How should the code be structured?
  – Obvious (but bad) idea: structure a layer as a process.
  – Why? It takes (much) longer to schedule a process than process a packet.
• Layering is a device for specification, not code structure.
An example--TRIPOS

- TRIPOS (Cambridge University) was wonderful little OS that used processes for most system functions. (The micro-kernel philosophy.)
  - Interprocess communication by pointer, not copy.
    - highly efficient.
  - Network code structured as three processes.
    - Network, transport, remote login.
  - 54 process wakeups to exchange a character.
  - Recoding as one process: 10x smaller, 10x faster
The consequence of processes
Emerging ideas

• The Structuring of Systems Using Upcalls”
  • David Clark, SOSP, 1985

• “Layered Multiplexing Considered Harmful”
  • David Tennenhouse, First International Workshop on High Speed Networking, 1989
Some pictures of upcalls

Typical protocol structured using upcalls

Failure during upcall.

Two problems:
- storage
- process

Solution:
- partition storage into "layer-wide" and per client. Unlock former.
- discard process.
Fixing other performance problems

- G. Varghese and T. Lauck. Hashed and hierarchical timing wheels: data structures for the efficient implementation of a timer facility. In *Proceedings of the eleventh ACM Symposium on Operating systems principles* (SOSP '87). ACM, New York, NY, USA,
Packet processing


- TCP packet receipt:
  - Sender of data: 191-235 instructions
  - Receiver of data, 186 instructions.
  - Set a timer: 35 (used timing wheel algorithm)
  - Internet protocol: ~60
A range of topics

- Early issues were performance
- Network software design
- Homogeneity
- Co-processing
- Small machines
  - From Alto, PC, (to IoT).
- Parallel machines
- Alternative network semantics
- High-level implications of connectivity to the world
  - Security, availability, etc.
- Virtual networks and virtual computers
- Speed of light
The recurring structural issue

• Networks have a distinct set of issues to solve.
  – Resource allocation, security, managing delivery.

• But they do not know what they are being used for. (The end to end model).
  – What is core and what is overlay?

• TCP persists because we found no other general service model.
  – The alternative is to push to the app the implementation of the desire semantics. (UDP.)
  – But then app designer is implementing the protocol. See earlier part of talk.
  – Is the protocol (e.g., transport) a core service?

• The net cannot trust the host, the OS cannot trust the app, the app cannot trust any of them, and the resulting system should have some sort of reliability.