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?

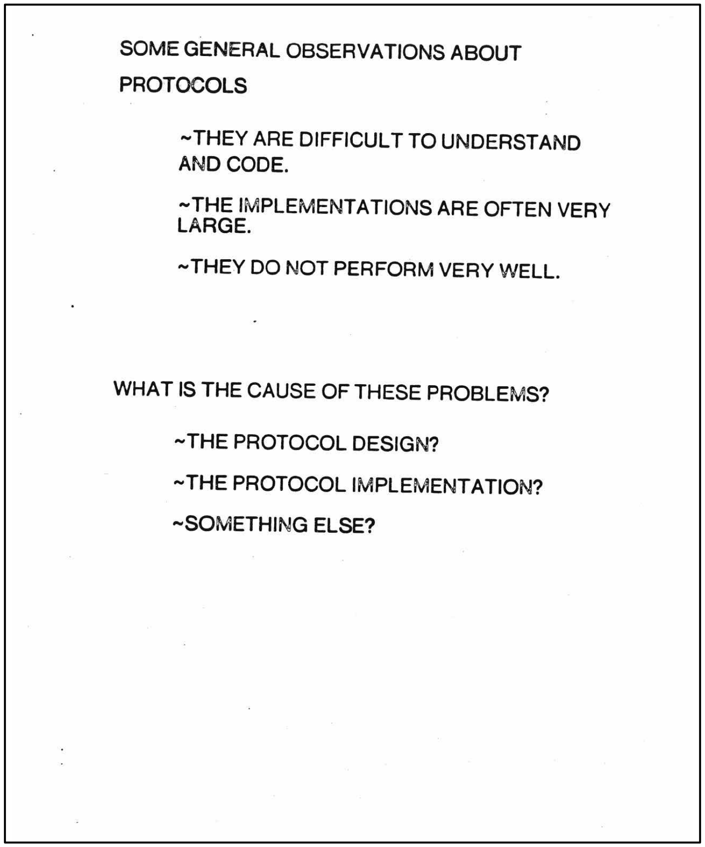
- Variable size units (packets and application data).
- Malformed content and size.
- Internet connected heterogeneous machines over heterogeneous networks.
    - First (and in some sense only) goal was interoperation.
    - Byte order, 9 bit bytes, etc.
- Unpredictable arrival/transmission.
- Must be 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.



SOME GENERAL OBSERVATIONS ABOUT PROTOCOLS

~THEY ARE DIFFICULT TO UNDERSTAND AND CODE.

~THE IMPLEMENTATIONS ARE OFTEN VERY LARGE.

~THEY DO NOT PERFORM VERY WELL.

WHAT IS THE CAUSE OF THESE PROBLEMS?

~THE PROTOCOL DESIGN?

~THE PROTOCOL IMPLEMENTATION?

~SOMETHING ELSE?

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?

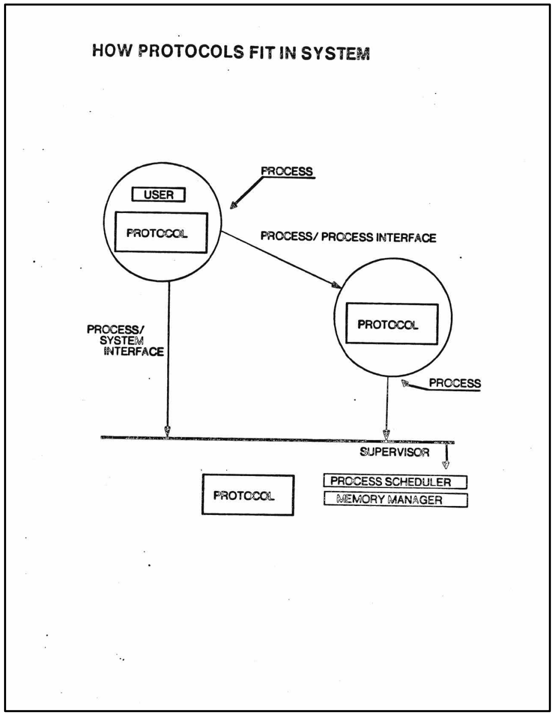
- Low overhead.
- Nasty programming environment.
- Run all the code at interrupt time?

Protocol in the application process?

- No asynchrony.
- Easy invocation.

Protocol in a separate process?

- High cost to invoke.
- Asynchronous execution.



HOW PROTOCOLS FIT IN SYSTEM

Waiting for events

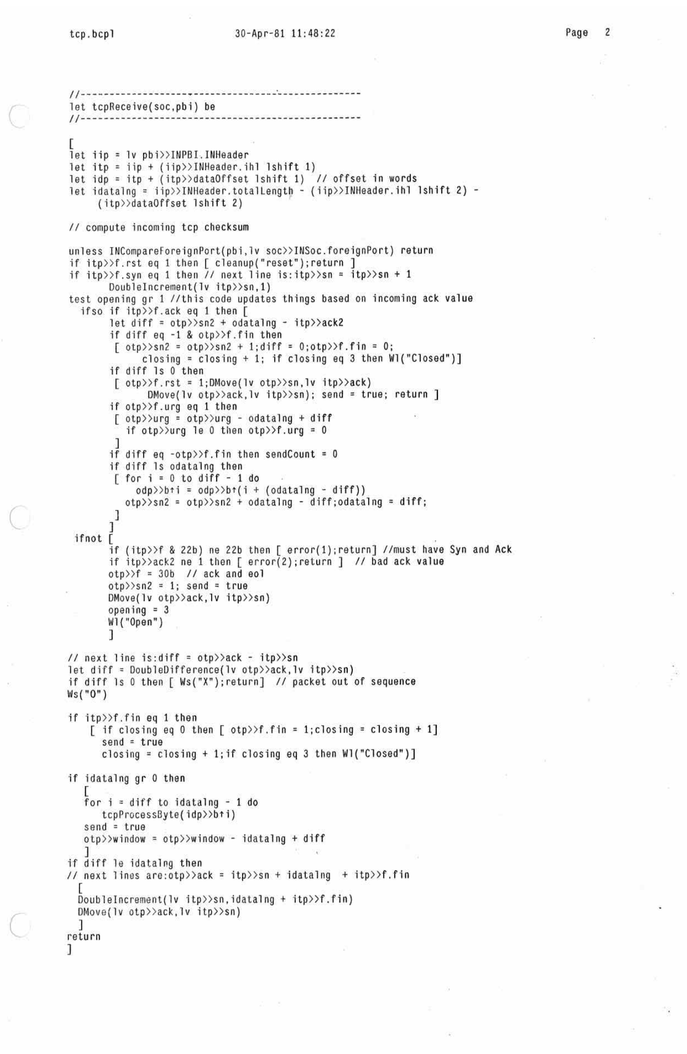
- Protocols have an odd (by the thinking of the day) structure.
  - They wait for multiple events.
  - A user event, a network event, a timer event.
- Many interprocess scheduling mechanisms required the waiting process to wait on one event.

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…



```
tcp.bcpl                 30-Apr-81 11:48:22                                   Page  2


//--------------------------------------------------
let tcpReceive(soc,pbi) be
//--------------------------------------------------

[
let iip = lv pbi>>INPBI.INHeader
let itp = iip + (iip>>INHeader.ihl lshift 1)
let idp = itp + (itp>>dataOffset lshift 1)  // offset in words
let idatalng = iip>>INHeader.totalLength - (iip>>INHeader.ihl lshift 2) -
     (itp>>dataOffset lshift 2)

// compute incoming tcp checksum

unless INCompareForeignPort(pbi,lv soc>>INSoc.foreignPort) return
if itp>>f.rst eq 1 then [ cleanup("reset");return ]
if itp>>f.syn eq 1 then // next line is:itp>>sn = itp>>sn + 1
       DoubleIncrement(lv itp>>sn,1)
test opening gr 1 //this code updates things based on incoming ack value
  ifso if itp>>f.ack eq 1 then [
       let diff = otp>>sn2 + odatalng - itp>>ack2
       if diff eq -1 & otp>>f.fin then
        [ otp>>sn2 = otp>>sn2 + 1;diff = 0;otp>>f.fin = 0;
             closing = closing + 1; if closing eq 3 then Wl("Closed")]
       if diff ls 0 then
        [ otp>>f.rst = 1;DMove(lv otp>>sn,lv itp>>ack)
              DMove(lv otp>>ack,lv itp>>sn); send = true; return ]
       if otp>>f.urg eq 1 then
        [ otp>>urg = otp>>urg - odatalng + diff
          if otp>>urg le 0 then otp>>f.urg = 0
        ]
       if diff eq -otp>>f.fin then sendCount = 0
       if diff ls odatalng then
        [ for i = 0 to diff - 1 do
            odp>>b↑i = odp>>b↑(i + (odatalng - diff))
          otp>>sn2 = otp>>sn2 + odatalng - diff;odatalng = diff;
        ]
       ]
 ifnot [
       if (itp>>f & 22b) ne 22b then [ error(1);return] //must have Syn and Ack
       if itp>>ack2 ne 1 then [ error(2);return ]  // bad ack value
       otp>>f = 30b  // ack and eol
       otp>>sn2 = 1; send = true
       DMove(lv otp>>ack,lv itp>>sn)
       opening = 3
       Wl("Open")
       ]

// next line is:diff = otp>>ack - itp>>sn
let diff = DoubleDifference(lv otp>>ack,lv itp>>sn)
if diff ls 0 then [ Ws("X");return]  // packet out of sequence
Ws("O")

if itp>>f.fin eq 1 then
    [ if closing eq 0 then [ otp>>f.fin = 1;closing = closing + 1]
      send = true
      closing = closing + 1;if closing eq 3 then Wl("Closed")]

if idatalng gr 0 then
   [
   for i = diff to idatalng - 1 do
      tcpProcessByte(idp>>b↑i)
   send = true
   otp>>window = otp>>window - idatalng + diff
   ]
if diff le idatalng then
// next lines are:otp>>ack = itp>>sn + idatalng  + itp>>f.fin
  [
  DoubleIncrement(lv itp>>sn,idatalng + itp>>f.fin)
  DMove(lv otp>>ack,lv itp>>sn)
  ]
return
]
```

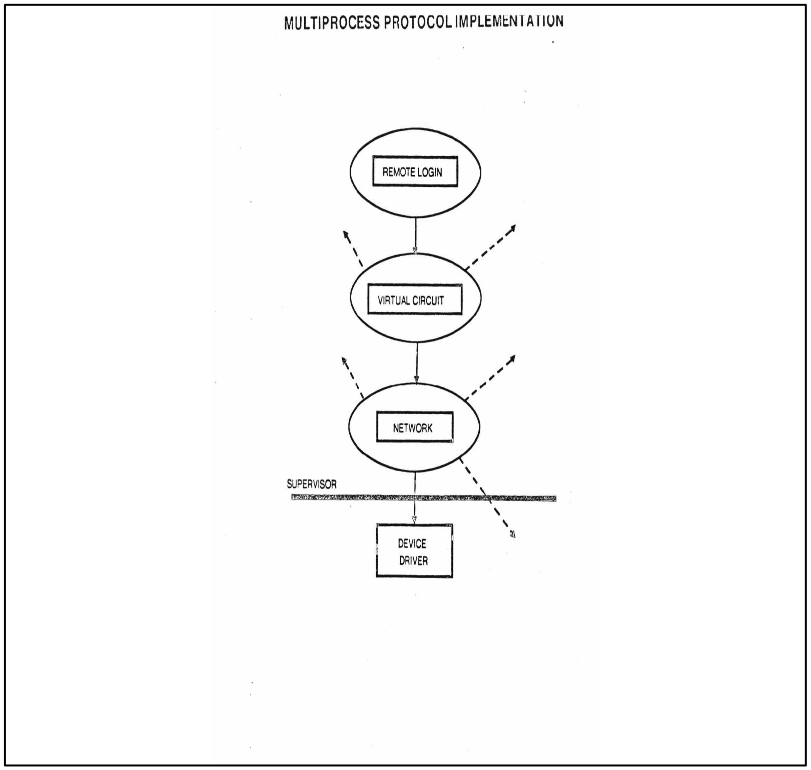
Layers of protocol

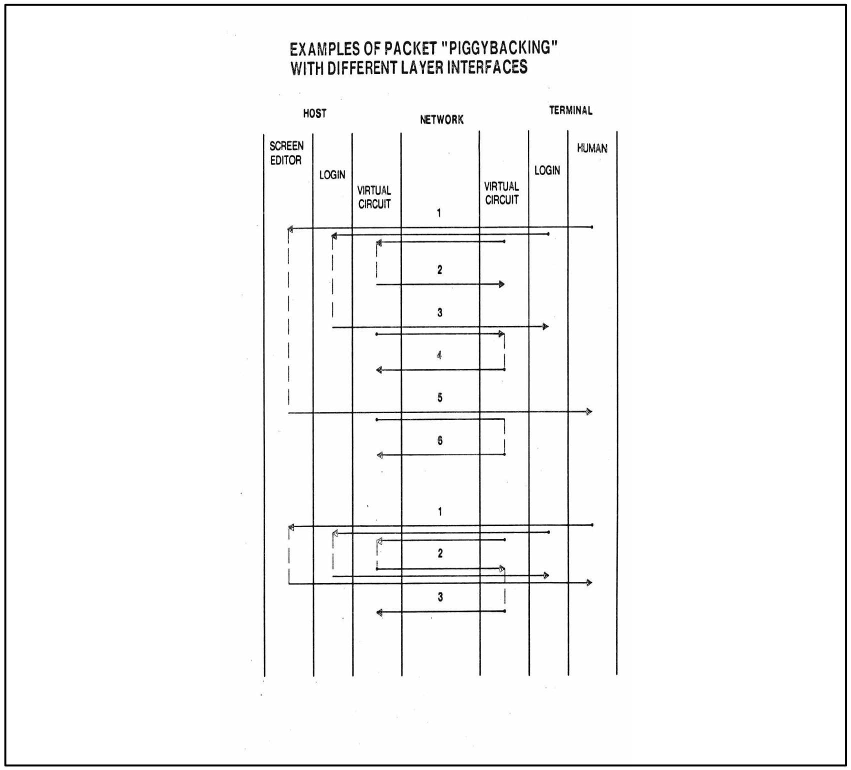
- Link, IP, TCP, app.
- How should the code be structured?
  - Obvious (but bad) idea: structure a layer as a process.
  - Why? It takes (much) longer to schedule a process than process a packet.
- Layering is a device for specification, not code structure.

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

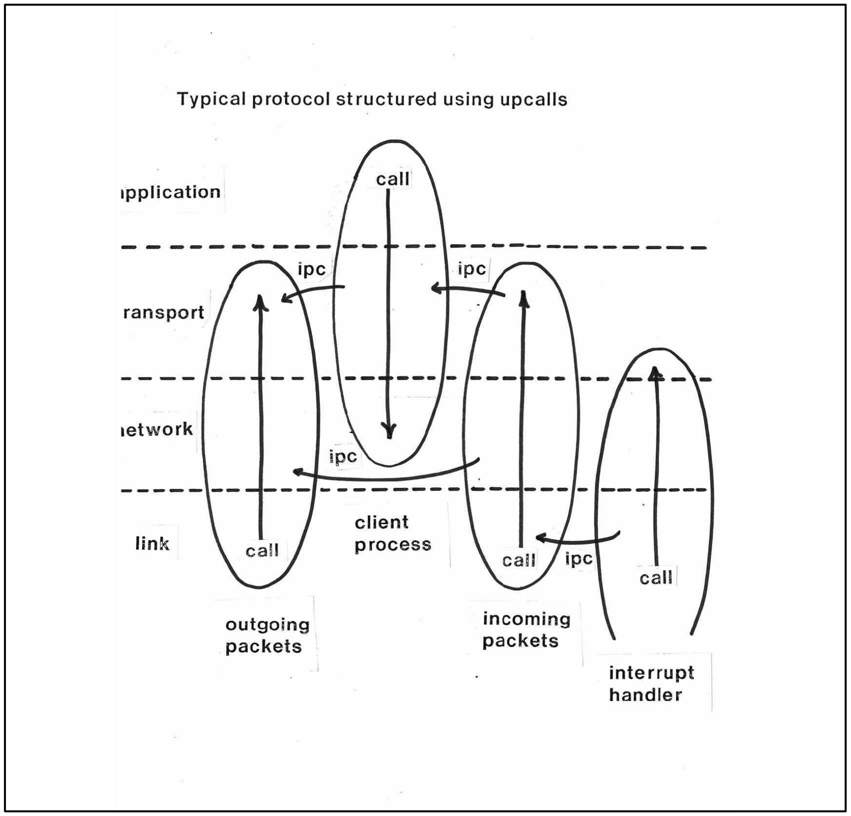

MULTIPROCESS PROTOCOL IMPLEMENTATION

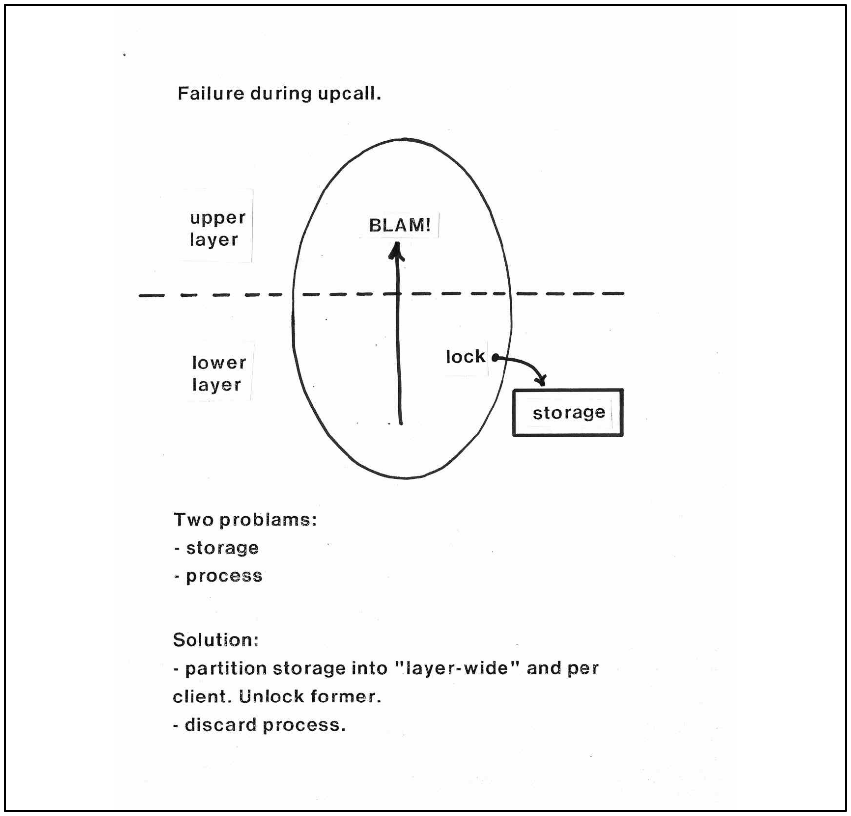

EXAMPLES OF PACKET "PIGGYBACKING" WITH DIFFERENT LAYER INTERFACES

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 problams:
- 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

- Clark, D.D.; Romkey, J.; Salwen, H., "An analysis of TCP processing overhead," in *Local Computer Networks, 1988., Proceedings of the 13th Conference on* , vol., no., pp.284-291, 10-12 Oct 1988

- 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.