CSE 222
Graduate Networking

Winter 2001

Lecture 3: Internet Architecture

Geoffrey M. Voelker

Overview

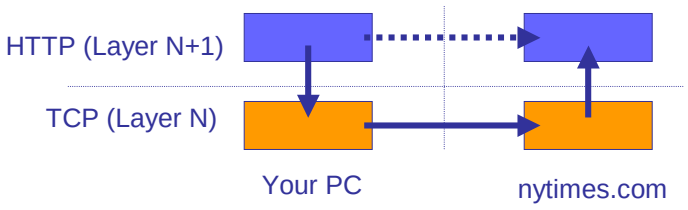
- Network architecture and layering
- End-to-end arguments in system design
  - Network architecture
- Architectural considerations
  - Application level framing (ALF)
  - Integrated layer processing (ILP)

Network Architecture

- Protocols and their specifications
  - Semantics
  - APIs
  - Packet formats
- Network architectures are often layered
  - To deal with the complexity
- Two important questions when designing a network architecture
  - What functionality goes in each layer? (Saltzer84)
  - How are the layers implemented efficiently and with good performance? (Clark90)

Layering

- Layering provides abstractions (c.f. Dijkstra's THE)
  - Higher layers can build upon the API and semantics defined and implemented in lower layers
  - Peer layers communicate with each other
  - Effective approach to managing complexity in systems



Layering in Networking

- Network layering manifests itself in multiple ways
  - Reference models
  - Encapsulation
  - Protocol graphs
  - Implementations

OSI Reference Model

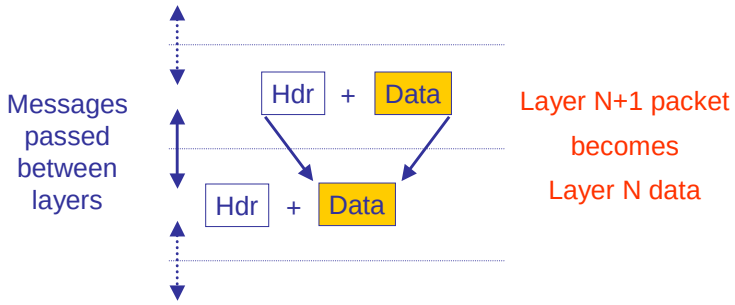
- International Standards Organization (ISO) Open Systems Interconnection (OSI) 7-layer reference model

Application
Presentation
Session
Transport
Network
Link
Physical

- Defined by programmer
- Encode/decode messages
- Manage connections
- Reliability, congestion control
- Routing
- Framing, multiple access
- Symbol coding, modulation

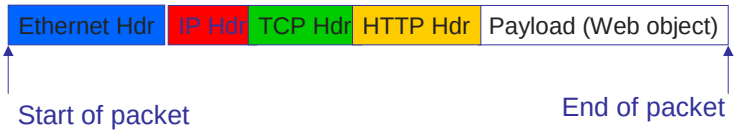
Encapsulation

- Protocols encapsulated within others



Recursive Encapsulation

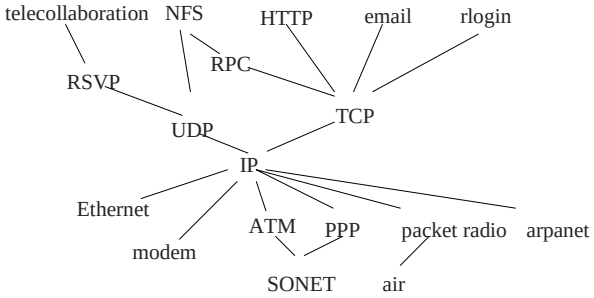
- Encapsulation is done at every layer



- Not completely accurate
  - Ignores segmentation and reassembly, trailers, etc.
- But notice that layers add overhead
  - Space (headers), effective bandwidth
  - Time (processing headers, peeling the onion), latency

Protocol Graph

- Any given protocol provides an abstraction on which multiple higher-level protocols are built



- Note that all “protocol paths” funnel through IP

Layered Implementations

- Layering is useful as a model
  - Clear specification of semantics and APIs
  - But it also adds overhead (space, time)
- Tension between layering as a model vs. implementation
  - Layer boundaries ignored in implementations
    - » Ease of implementation, better performance
    - » Ex: TCP and IP headers often accessed by same layer code
  - PD ~ Layering is a good slave, but a poor master
- We’ll talk about this more in the Clark90 paper

Saltzer84: End-to-End Principle

- Where should functionality be placed in the network architecture?
  - Functionality should be implemented at a lower layer iff it can be correctly and completely implemented there
  - Generality can get in the way if a higher layer does not need it
- Early example
  - ARPANet provided reliable link transfers between switches
  - Packets could still get corrupted on host-switch link, or inside of the switches
  - Hence, still need reliability at higher layers

Reliable File Transfer

- From server disk over network to client disk
- Many places where errors can be introduced
  - Disk can introduce bit errors
  - Host I/O bus can introduce bit errors
  - Packets can be corrupted, dropped, reordered at any node
- Conclusion
  - Still need integrity checks on entire file, at application level, not per packet or per hop
  - Impossible to design “perfect” layers because perfect requires support from higher layers

Other Examples

- Network
  - Suppressing duplicate messages
  - Reordered messages
  - Retransmission
- Distributed transactions
  - Exactly one vs. at most once vs. at least once
- Security
  - Only as strong as weakest assumption

Performance Optimizations

- Functionality at lower layers then viewed as performance optimizations
  - Not required for correct operation
  - Can be required for reasonably efficient operation
- Back of the envelope
  - N hops (average hops on Internet route = 15 hops)
  - Prob(corrupted packet per link) = p
  - Prob(packet lost end to end)
    - » p = 0.0001% -> Prob(e2e loss) = 0.0015%
    - » p = 1% -> Prob(e2e loss) = 14%

E2E Discussion

- Do you agree with the E2E argument?
  - "Network protocols should not strive to satisfy the needs of those driven only by efficiency, they should strive to satisfy the needs of the largest number of people. There are many more people who want functionality that there are who want efficiency."
- Do you think any of the following violate E2E?
  - Network address translators (NATs)?
  - Web proxy caches?
  - Active Networks?

Clark90: ALF and ILP

- Problem: Networks must be flexible and efficient
  - Operate at very high speeds (gigabit) – will existing protocols be a bottleneck?
  - New technologies – ISDN (goodbye), DSL, cable, fiber
  - Service integration – single hosts have to support various media (text, graphics, audio, video) and access patterns (interactive, bulk transfer, real-time rendering) – will existing protocols be able to meet application needs?
- Solution
  - Application layer framing – speed up presentation layer
  - Integrated layer processing – speed up implementations

Protocol Processing

- Two overheads to protocol processing
  - Data manipulation
    - » Buffering, crossing protection domains, presentation conversion
  - Control transfer
    - » Multiplexing, flow/congestion control, acks, framing
- Claim:
  - Control transfer requires little processing
  - Most of the time spent in data manipulation
    - » Copying, checksumming
    - » Presentation conversion
  - Focus on optimizing data manipulation

Performance of What?

- What is the bottleneck?
- Depends on where you measure
  - Layer-to-layer performance – TCP to TCP
  - End-to-end performance – Application to application
    - » Always ask what “end” means in “end-to-end”
  - Ex: HTTP
    - » HTTP latency often measured round trip from socket to socket
    - » Ignores time spent processing and displaying by browser
    - » But this is what users care about!
      - Complex tables in netscape take forever to display, so who cares how long it took to download from the server

Presentation Conversion

- Many techniques have been developed to optimize layer-to-layer performance
  - Zero-copy TCP, PCB templates, offloading onto hardware, combine checksum and copying (I worked on this, as did Pasquale, in early 90s), etc.
- Claim: The real bottleneck is presentation conversion
  - Ex: Copy at 130 Mb/s, conversion to ANS.1 at 28 Mb/s (4.6x)
  - Ex: HTTP: all HTTP communication is in strings
    - » A lot of code and time spent doing string management (take a peek at Apache code some time)
- Need new network protocol structuring to support optimizations in presentation conversion

Application Level Framing

- This is where application level framing (ALF) comes in
- The tough problem with presentation conversion is that lower layers need application-specific information to "do things right"
  - Esp. with lost or reordered packets
- Want applications to process data out of order
  - Need something to enable applications to tell lower layers what the minimum unit of data ordering it can handle

Application Data Units

- This is where Application Data Units (ADUs) come in
  - "Each ADU can be processed out of order with respect to other ADUs"
- Issues
  - Applications define ADU (more work for app, arguably good)
  - Need a mechanism to define ADU to lower layers (tough)
  - Receiver needs info from sender about where and/or when an ADU needs to be delivered
    - » Offset in file, frame in video, argument in RPC, etc.
    - » Cannot use sequence #s of lower layers, has to be application-defined

HTTP: Chunked Encoding

- HTTP supports a mechanism called chunked encoding that can be used as ALF

  http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.6.1
  - With chunked encoding, a given HTTP object can be requested in "chunks" (start and length range)
  - Each chunk is requested as a separate request/response
  - Application can deal with chunks in any order
    - » However, lower protocol layers do not take advantage of the reordering semantics except the benefit you get with separate connections
  - Can also be used to continue downloading where a previous connection was broken (e.g., over a modem)
- I have never seen it used seriously yet, though

ILP: Motivation

- Many ways to structure an implementation of a networking stack (c.f. Clark's upcalls OS paper)
  - Communicating processes/threads at each layer exchanging messages
  - Single process/thread per packet executes across layers
- Either way, processing is still serial across layers
  - Bad for performance
  - Not a good use of the memory hierarchy
- Serialization necessary when there are ordering constraints across layers
  - Data must be in order to checksum, decrypt, copy to app, etc.

ILP: Approach

- Perform all protocol manipulations across layers as one or two integrated processing loops
  - Ex: Combine copying and checksumming (and decryption…)
- Need to reduce ordering constraints
  - Use ADUs – can be handled out-of-order
- Structure using two manipulation stages
  1. Receive data from network and determine which ADU to put it in (demultiplex) and where in the ADU (reorder)
  2. Pass ADU to application so that it can do error checking and presentation conversion

ALF/ILP Discussion

- ALF
  - Why don't we use it today?
- ILP
  - Is ILP a general approach?
    - » "Sure, you can get performance by touching each piece of memory only once and performing all needed calculations at the same time. But how feasible is this in the common case? The authors hardly give any examples of how to combine operations for any real applications and only talk about applications in the most abstract way."
  - Where is retransmission done?
  - TCP checksumming?
  - When can both steps be combined into one?
  - Can you do it without ALF?

For Next Time...

- Read Chapters 2.5, 5.2, 6.1, and 6.3
- Read Jacobson88