Shared, Multi-Hop Networks and End-to-End Arguments

Brad Karp
UCL Computer Science

CS 3035/GZ01
17th October 2011
Networks Are Shared

• Any host may want to reach any other
• Dedicated links: N hosts $\rightarrow$ $O(N^2)$ links
  – “Fewer links” implies “shared links”
• CPUs, memory, disks, network cards ever-cheaper, quickly upgradeable
• Adding network links different:
  – Non-decreasing costs of tearing up street, laying cable, launching satellite, radio spectrum...
  – Approval often involves regulatory bodies $\rightarrow$ glacial pace
  – Users economically motivated to share network resources, despite declining costs of network electronics
Sharing of Multi-Hop Networks

- Link multiplexing among users
- Packet forwarding and delay
- Best-effort delivery: packet buffering, buffer overflow, packet dropping
- Packet duplication
- Packet corruption
- Link breakage
- Packet reordering
Link Multiplexing

- Link between cities carries many conversations simultaneously; **multiplexed**
- Earthquake in SF; heavy call load from Boston to SF
- Link capacity limited; some Boston callers will be told “network busy”
- $N^{th}$ caller’s call **completes**, $n+1^{st}$’s fails; why?
Telephony: Isochronous Link Multiplexing

- Assume shared link capacity 45 Mbps
- Voice call: 8-bit samples (frames) spaced perfectly evenly to achieve 64 Kbps
  - one frame per 5624 bit times, or 125 us; 8000 frames per second
- Between one call’s successive frames, room for 702 other frames; 703 calls total capacity
- Hard-edged: 1st 703 win, 704th loses

figure: [Saltzer and Kaashoek]
Telephony: Isochronous Link Multiplexing

- Assume shared link capacity 45 Mbps
- Voice call: 8-bit samples (frames) spaced perfectly evenly to achieve 64 Kbps – one frame per 5624 bit times, or 125 us; 8000 frames per second
- Between one call’s successive frames, room for 702 other frames; 703 calls total capacity
- Hard-edged: 1st 703 win, 704th loses

**Time-Division Multiplexing (TDM):** equal-sized frames at equal intervals, perfectly predictable rate and delay per second.

- Between one call’s successive frames, room for 702 other frames; 703 calls total capacity
- Hard-edged: 1st 703 win, 704th loses
Connection-Oriented Forwarding

• **Connection setup**: Boston switch asks SF switch to forward B3’s frames to S2
• **Switches store state** concerning how to forward frames for the call; slot determines forwarding
• **Connection tear down**: at end of call, two switches delete state concerning call
Data Networks: Asynchronous Link Multiplexing

- Computers (and users) send data in bursts; not in a steady stream, like telephony
  - Sender may have nothing to send when its TDM slot comes around
  - Sender may have more than one frame to send at once, but can only use its own TDM slots
- More fitting to send data as they become available, rather than in scheduled fashion

figure: [Saltzer and Kaashoek]
Data Networks: Asynchronous Link Multiplexing

- Computers (and users) send data in bursts; not in a steady stream, like telephony – Sender may have nothing to send when its TDM slot comes around – Sender may have more than one frame to send at once, but can only use its own TDM slots

Asynchronous Multiplexing: give up predictable data rate and latency in favor of delivering entire message more quickly

- Sender may have more than one frame to send at once, but can only use its own TDM slots

- More fitting to send data as they become available, rather than in scheduled fashion

figure: [Saltzer and Kaashoek]
Async Link Multiplexing (cont’d)

- Frames of any length (up to link limit)
- Frames may be sent anytime link not already in use
- Timing of frame doesn’t imply forwarding; must explicitly include guidance information
- Variable-length frames require framing to demarcate inter-frame boundaries
- No need for per-connection state at switches; connectionless

figure: [Saltzer and Kaashoek]
Async Link Multiplexing (cont’d)

- Frames of any length (up to link limit)
- Frames may be sent anytime link not already in use
- Timing of frame doesn’t imply forwarding; must explicitly include guidance information
- Variable-length frames require framing to demarcate inter-frame boundaries
- No need for per-connection state at switches; connectionless

Asynchronous Multiplexing may be usable for telephony; depends on variability of delay and rate offered by network

figure: [Saltzer and Kaashoek]
Sharing of Multi-Hop Networks

- Link multiplexing among users
- Packet forwarding and delay
- Best-effort delivery: packet buffering, buffer overflow, packet dropping
- Packet duplication
- Packet corruption
- Link breakage
- Packet reordering
Packet Forwarding

- Packet switches, or routers, connected by async links
- Multiple paths, or routes, possible
- **Forwarding:** examine destination of arrived packet, look it up in table, forward on appropriate link

figure: [Saltzer and Kaashoek]
Transit Time

- **Propagation delay**: speed of light over medium across link length; *fixed* for given distance and medium
- **Transmission delay**: serialization of packet’s bits at link rate on each transmission; *varies* depending on packet length and link speed

[figure: Saltzer and Kaashoek]
Transit Time (cont’d)

- Processing delay:
  - forwarding decision: fixed component
  - other packet processing (e.g., checksum, copying): varying, length-proportional component
- Queuing delay:
  - output link may be busy when packet arrives
  - store packet in memory, or queue
  - varies with total traffic volume and pattern
How can we estimate queuing delay?

Assume
- Packets arrive according to random, memoryless process
- Packets have randomly distributed service times (i.e., transmission delays)
- Utilization of outgoing link is $\rho$

Average queuing delay in packet service times (including this packet’s) is $\frac{1}{1-\rho}$

figure: [Saltzer and Kaashoek]
Utilization and Delay

• Trade-off: bounding max delay also bounds max utilization
• Network operators like high utilization; links cost money
• Users like low latency; e.g., interactive voice
• Isochronous: abrupt exclusion of \( N+1 \) user
• Asynchronous: delay grows as load grows

figure: [Saltzer and Kaashoek]
Utilization and Delay

- Trade-off: bounding max delay also bounds max utilization
- Network operators like high utilization; links cost money
- Users like low latency; e.g., interactive voice
- Isochronous: abrupt exclusion of N+1st user
- Asynchronous: delay grows as load grows

Delay is **average**; to bound worst-case delay, must target utilization below $\rho_{\text{max}}$

\[ \frac{1}{1 - \rho} \]

figure: [Saltzer and Kaashoek]
Queuing Theory: Final Words

• Utilization/delay trade-off true throughout computer systems (CPU, disk scheduling)

• Warning: queuing theory assumptions don’t hold on real data networks!
  – Packet arrivals not Poisson process; burstier
  – Router queues of finite length; bounds queuing delay
Sharing of Multi-Hop Networks

- Link multiplexing among users
- Packet forwarding and delay
- **Best-effort delivery:** packet buffering, buffer overflow, packet dropping
- Packet duplication
- Packet corruption
- Link breakage
- Packet reordering
Queue Sizes (and Packet Drops)

- A router’s packet queue is just memory
- How much memory does a router need?
- Strategies:
  - Plan for worst case: enough memory to buffer longest possible queue length
  - Plan for average case, slow senders: enough memory for common case; when queue full, tell senders to slow down
  - Plan for average case, drop extra packets: enough memory for common case; when queue full, drop extras
Worst-Case Memory Size

- Memory is relatively cheap
- How can we predict worst-case queue size?
- Bursts of load caused by users (and bugs in code!)
  - highly unpredictable
  - orders of magnitude worse than average case
- Very long queues mean very long delays
  - Would you wait 2 minutes to learn if you had new email?
Average-Case Memory Size, with Quench

- Central worry is congestion: sometimes, memory not big enough, queue will fill
- When queue fills, send a quench message on incoming link, asking sender (router or host) to slow down
- Problems:
  - Respond to congestion by generating traffic?
  - Whom should be quenched?
  - Quenched source may no longer be sending
  - Quench itself delayed by queuing; worse congestion is, longer the delay
- Essentially not used in practice
Average-Case Memory Size, with Dropping

- When queue full, drop packets!
- Some entity must resend dropped packet
- Lack of end-to-end acknowledgement now indicates congestion!
  - Implicit signal to sender—without adding to traffic load
  - Introduces possibility of sender slowing automatically in response to congestion
- What the Internet does
Isochronous vs. Asynchronous

• Isochronous:
  – when capacity remains, get fixed transmission rate, independent of other users’ activity
  – when capacity fully subscribed, get nothing

• Asynchronous:
  – transmission rate depends on instantaneous load from those with whom you share links

• Preferable regime depends on application
  – After earthquake, better to be able to send “I’m alive” slowly than not to be able to send at all
  – Remote surgery requires guaranteed bit rate
Best Effort

• Networks that never discard packets termed **guaranteed-delivery**
  – more mechanism required to guarantee (and track) delivery than to drop packets

• Networks willing to discard packets under congestion termed **best-effort**

• Fuzzy meaning of “best effort”: **far greater chance of undetected loss than with guaranteed delivery**

• Internet delivers packets **best-effort**

• Internet email **guarantees delivery** (atop best-effort packet delivery service)
Sharing of Multi-Hop Networks

- Link multiplexing among users
- Packet forwarding and delay
- Best-effort delivery: packet buffering, buffer overflow, packet dropping
  - Packet duplication
  - Packet corruption
  - Link breakage
  - Packet reordering
Packet Duplication

- Best-effort delivery drops packets when queues overflow
- End-to-end principle suggests original sender should retry
  - destination could also be down

figure: [Saltzer and Kaashoek]
Packet Duplication (cont’d)

- **Responses** can be dropped
- Consequence: client’s resent request appears as **duplicate** to server
- Problem? Depends on application and request type:
  - Bank withdrawal
  - File write
Packet Duplication (cont’d)

- Delay can also cause duplication

figure: [Saltzer and Kaashoek]
Duplicate Suppression

• A marks requests with monotonically increasing sequence numbers (any non-repeating sequence suffices)
• B remembers request sequence numbers for requests previously processed
• B ignores requests with sequence numbers already completed; repeats response to A
Packet Corruption and Link Breakage

• Noise on links, errors in router memory, software errors, &c., may corrupt packets en route
  – Given best-effort service, error detection is sufficient; drop packets that contain errors

• Links may break (excavation cuts fiber; power failure)
  – Networks typically offer more than one path; routing must find a currently working path
Packet Reordering

• Consider two paths, R1 and R2, with delays D1 and D2, where D1 < D2
  – Sender may send packet P0 along R2
  – Sender may send packet P1 along R1
  – Packets may arrive at receiver in order P1, P0

• For messages divided into packets, reassembly at receiver must be done with care
**Summary: Multi-Hop Networks**

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Application characteristics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>isochronous (e.g., telephone network)</td>
<td>Continuous stream (e.g., interactive voice)</td>
<td>good match</td>
</tr>
<tr>
<td>asynchronous (e.g., Internet)</td>
<td>Bursts of data (most computer-to-computer data)</td>
<td>wastes capacity</td>
</tr>
<tr>
<td></td>
<td>Response to load variations</td>
<td>(hard-edged) either accepts or blocks call</td>
</tr>
<tr>
<td></td>
<td>variable latency upset application</td>
<td>good match</td>
</tr>
</tbody>
</table>
|                                  |                                                  | (gradual)  
|                                  |                                                  | 1. variable delay  
|                                  |                                                  | 2. discards data  
|                                  |                                                  | 3. rate adaptation |

figure: [Saltzer and Kaashoek]
Outline

• Shared, Multi-hop Networks
• End-to-End Arguments
Motivation: End-to-End Argument

• 7 layers in OSI model
• 7 places to solve many of same problems:
  – In-order delivery
  – Duplicate-free delivery
  – Reliable delivery (retransmission) after corruption or loss
  – Encryption
  – Authentication
• In which layer(s) should a particular function be implemented?
Example: Careful File Transfer

- **Goal:** accurately copy file on A’s disk to B’s disk
- **Straw man:**
  - Read file from A’s disk
  - A sends stream of packets containing file data to B
  - Link-layer retransmission of lost/corrupted packets at each hop
  - B writes file data to disk
- **Does this system meet design goal?**
  - Bit errors on links not a problem
Where Can Errors Happen?

• On A’s or B’s disk
• In A’s or B’s RAM or CPU
• In A’s or B’s software
• In the RAM, CPU, or software of any router that forwards packet \((\text{MIT example!})\)

• Why might errors be likely?
  – Drive for CPU speed and storage density: pushes hardware to EE limits, engineered to tight tolerances
  – e.g., today’s disks return data that are the output of an MLE!
Solution: End-to-End Verification

- A stores checksum with data on disk
  - Why not compute freshly on read?
- B computes checksum over received data, sends to A (or vice-versa)
- Compare two checksums; A resends if not identical
- Can we eliminate hop-by-hop error detection?
  - Suppose there’s a router with bad RAM; how will you find it?
- Is a whole-file checksum enough?
  - Poor performance: must resend whole file each time one packet (bit) corrupted!
End-to-End Principle

- Only application at communication endpoints can completely and correctly implement a function
- Processing in middle alone cannot provide function
- Processing in middle may, however, be important performance optimization
- Engineering middle hops to provide guaranteed functionality often wasteful of effort and inefficient
Perils of Low-Layer Implementation

- Entangles application behavior with network internals
- Suppose each IP router reliably transmits to next hop
  - lossless delivery, variable delay
  - ftp: OK, move huge file reliably (just end-to-end TCP works fine, too, though)
  - Skype: terrible, jitter packets when a few corruptions or drops not a problem anyway
- Complicates deployment of innovative applications
  - phone network vs. Internet
Advantages of Low-Layer Implementation

• Each application author needn’t recode a shared function
• Overlapping error checks (e.g., checksums) at all layers invaluable in debugging and fault diagnosis
• If end systems not cooperative (increasingly the case), only way to enforce resource allocation!
Challenge: End-to-End Authentication and Encryption

- Use a public PC to check your email using IMAP & SSL
  - Authenticaes server to you and you to server robustly
  - Encrypts between you robustly
- Key security consideration: threat model
  - which attacks are you explicitly defending against?
  - which are you ignoring?
  - what does it cost your adversary to mount an attack?
- What are you trusting?
  - mail reading application (could be trojaned)
  - OS (could also be trojaned)
  - hardware (e.g., "fake ATM" cases)
- End-to-end notion of security must consider integrity of software and hardware at endpoints, possibly even of users!
End-to-End Violation: Firewalls

- Box in middle of network that blocks “malicious” traffic
  - End-host software often vulnerable to worms
  - Users naive, may not keep desktop patched up-to-date
- Clearly violates e2e principle
  - Endpoints capable of deciding what traffic to ignore
  - Firewall entangled with design of network and higher protocol layers and apps, and vice-versa
  - Example: new ECN bit to improve TCP congestion control; many firewalls filtered all such packets!
- Probably need firewalls
- But beware entangling network edge and network interior
Summary: End-to-End Principle

• Many functions **must** be implemented at application endpoints to provide desired behavior, even if implemented in “middle” of network

• End-to-end approach **decouples design** of components in network interior from design of applications at edges

• Some functions still **benefit from implementation in network interior**; at **cost of entangling interior and core**
  – Performance (e.g., link-layer retransmission)
  – Security (e.g., firewalls)
  – Scalability (e.g., routing)

• End-to-end principle is **not sacred**; it’s a way to think critically about design choices in communication systems