Reliable Transport: The Prequel

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson and other colleagues at Princeton and UC Berkeley

Question

- How many people have not yet participated?

Don't be intimidated….

- Wide spectrum of backgrounds
- But that's just a head start in context, not content
- When we get to the real algorithms, everyone will be on the same page

Don't parse my words too carefully

- "Networking" is not a set of precise rules
  - ***It is a state of mind….***

- The principles of networking help you build scalable and robust systems
  - ***But they don't provide a detailed instruction manual***

Outline for Today

- Fate Sharing
- Course So Far
- Reliable Delivery

Decisions and Their Principles

- How to break system into modules
  - **Dictated by Layering**
- Where modules are implemented
  - **Dictated by End-to-End Principle**
- Where state is stored
  - **Dictated by Fate-Sharing**

Fate-Sharing

Fate-Sharing

- Note that E2E principle relied on “fate-sharing”
  - Invariants break only when endpoints themselves break
  - Minimize dependence on other network elements

- This should also dictate placement of storage

General Principle: *Fate-Sharing*

- When storing state in a distributed system, co-locate it with entities that rely on that state
- Only way failure can cause loss of the critical state is if the entity that cares about it also fails ...
  - … in which case it doesn't matter
- Often argues for keeping *network state* at end hosts rather than inside routers
  - In keeping with End-to-End principle
  - E.g., packet-switching rather than circuit-switching
  - E.g., NFS file handles, HTTP "cookies"

A Cynical View of Distributed Systems

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"

---Leslie Lamport

- ***This is precisely what fate-sharing is trying to avoid…..***

The Course So Far

We Are In the “Conceptual” Phase

- Three phases to course:
  - Basic concepts
  - Making these concepts real
  - Various topics

- The conceptual phase has three steps

First Step: Basic Decisions

- Packet Switching winner over circuit switching

- Best-effort service model

Second Step: Architectural Principles

- Layering
- End-to-End Principle
- Fate-Sharing

Third Step: Design Challenges

- Let’s go layer by layer
  - Physical
  - Datalink
  - Network
  - Transport
  - Application

- What function does each layer need to implement?

Two Layers We Don’t Worry About

- Physical:
  - Technology dependent
  - Lots of possible solutions
  - Not specific to the Internet

- Application:
  - Application-dependent
  - Lots of possible solutions

Datalink and Network Layers

- Both support best-effort delivery
  - Datalink over local scope
  - Network over global scope

- Key challenge: scalable, robust **routing**
  - How to direct packets to destination

Transport Layer

- Provide reliable delivery over unreliable network

We Only Have Two Design Challenges

- **Routing**: to be covered next week (+project 2)
- **Reliable delivery**: to be covered today (+project 1)
- You will then know everything you need to know
  - Conceptually…..
- Lecture on “missing pieces” will complete picture

Purpose of Today

- Understand reliable transport conceptually
  - *What are the fundamental aspects of reliable transport?*
- The goal is not to understand TCP
  - TCP involves lots of detailed mechanisms, covered later
- Ground rules for discussion:
  - No mention of TCP
  - No mention of detailed practical issues
  - Focus only on “ideal” world of packets and links

Two Pedagogical Approaches

1. Understand why given algorithm works (textbook)
2. Understand the space of possible algorithms

- The first: you understand why the Internet works
  - And get a job at Cisco…
- The second: you could design the next Internet
  - Or start the next Cisco...
- **The second is what we do at Berkeley!**

You Must Think For Yourself

- Today's lecture requires you to engage
  - How would I design a reliable service?
- I will ask questions, want you to think about them
  - If you think you already know this, you are wrong
  - If you think you don't know enough, you are wrong
  - If you think you can learn this asleep, you are wrong

Reliable Delivery

Best Effort Service

- Packets can be lost
- Packets can be corrupted
- Packets can be reordered
- Packets can be delayed
- Packets can be duplicated
- ….

How can you possibly make anything work with such a service model?

Making Best Effort Work

- Engineer **network** so that average case is decent
  - No guarantees, but you must try….
- Engineer **apps** so they can tolerate the worst case
  - They don't have to thrive, they just can't die
- A classical case of architecting for flexibility
  - Engineering for performance
- Internet enabled app innovation and competition
  - Only the hardy survived, and doomsayers were ignored

Reliable Transport Is Necessary

- Some app semantics involve reliable transport
  - E.g., file transfer

- How can we build a reliable transport service on top of an arbitrarily unreliable packet delivery?

- A central challenge in bridging the gap between
  - **the abstractions application designers want**
  - **the abstractions networks can easily support**

Important Distinctions

- Functionality implemented in network
  - Keep minimal (easy to build, broadly applicable)

- Functionality implemented in the application
  - Keep minimal (easy to write)
  - Restricted to application-specific functionality

- Functionality implemented in the "network stack"
  - The shared networking code on the host
  - This relieves burden from both app and network
  - **This is where reliability belongs**

Two Different Statements

- **Applications need reliable service**
  - This means that the application writers should be able to assume this, to make their job easier
- **The network must provide reliable service**
  - This contends that end hosts cannot implement this functionality, so the network must provide it
- Today we are making the first statement, and refuting the second…

Challenge for Today

- Building a stack that supports reliable transfer
  - So that individual applications don't need to deal with packet losses, etc.

Fundamental Systems Question

- How to build reliable services over unreliable components
  - File systems, databases, etc.

- Reliable transport is the *simplest* example of this

Four Goals For Reliable Transfer

- Correctness
- Timeliness
  - Minimize time until data is transferred
- Efficiency
  - Would like to minimize use of bandwidth
  - i.e., don't send too many packets
- "Fairness"
  - How well does it play with others?

Start with transfer of a single packet

- We can later worry about larger files, but in the beginning it is cleaner to focus on this simple case

Correctness Condition?

- Packet is delivered to receiver.

WRONG!

- What if network is partitioned?

Correctness Condition?

- Packet is delivered to receiver if and only if it was possible to deliver packet.

WRONG!

- If the network is only available at one instant of time, only an Oracle would know when to send.

Correctness Condition?

- Resend packet if and only if the previous transmission was lost or corrupted

WRONG!

- Impossible
  - "Coordinated Attack" over an unreliable network
- Consider two cases:
  - Packet delivered; all packets from receiver are dropped
  - Packet dropped; all packets from receiver are dropped
- They are indistinguishable to sender
  - **Does it resend, or not?**

Correctness Condition?

- Packet *is always* resent if the previous transmission was lost or corrupted.
- Packet *may* be resent at other times.
- Note:
  - This invariant gives us a simple criterion for deciding if an implementation is *correct*
  - Efficiency and timeliness are separate criteria….

We have correctness condition

- How do we achieve it?

Two Choices for Corruption

- Have applications do integrity check
  - Ignore it in transport protocol
- Do per-packet checksum
  - Won't be perfectly reliable, still have app-level check
  - So why do it? **What does the E2E principle say**?
- This is all implemented in the ends!
  - But E2E reasoning about correctness still applies
- Today, we will ignore corruption, treat it as loss

Solution v1

- Send every packet as often and fast as you can….

- Definitely correct
- Optimal timeliness
- Infinitely bad efficiency

What's Missing?

- Feedback from receiver!

- If receiver does not respond, no way for sender to tell when to stop resending.
  - Cannot achieve efficiency + correctness w/out feedback.

Forms of Feedback

- ACK: Yes, I got the packet
- NACK: No, I did not get the packet
- When is NACK a natural idea?
  - Corruption
- Ignore NACKs for rest of lecture….

Solution v2

- Resend packet until you get an ACK
  - And receiver resends ACKs until data flow stops

- Optimal timeliness

- Efficiency: how much bandwidth is wasted?

  ~ B x RTT

  - ok for short latencies
  - bad for long latencies

Solution v3

- Send packet
  - Set a timer
- If receive ACK: done
- If no ACK by time timer expires, resend.
- Timeliness would argue for small timeout
- Efficiency would argue for larger timeout
  - May want to increase timer each time you try
  - May want to cap the number of retries

Have "solved" the single packet case

- Send packet
- Set timer
- If no ACK when timer goes off, resend packet
  - And reset timer

5 Minute Break

Multiple Packets

- Service Model: reliable stream of packets
  - Hand up contiguous block of packets to application
- Why not use single-packet solution?
  - Only one packet in flight at any time
  - Very poor timeliness (but very good efficiency)
- Use window-based approach
  - Allow for W packets in-flight at any time **(unack'ed)**
  - Sliding Window implies W packets are contiguous
    - Makes sense if window is related to receiver buffer (later)

Window-based Algorithms

- See textbook or the web for animations….
  - Will implement in project

- Very simple concept:
  - Send W packets
  - When one gets ACK’ed, send the next packet in line

- Will consider several variations….
  - But first….

How big should the window be?

- Windows serve three purposes
  - Taking advantage of the bandwidth on the link
  - Limiting the bandwidth used (congestion control)
  - Limiting the amount of buffering needed at the receiver

- If we ignore all but the first goal, then we want to keep the sender always sending (in the ideal case)
  - RTT: sending first packet until receiving first ACK

Condition: RTT x B ~ W x Packet Size

Design Considerations

- Nature of feedback
- Detection of loss
- Response to loss

Possible Feedback From Receiver

- Ideas?

ACK Individual Packets

- Strengths:
  - Know fate of each packet
  - Impervious to reordering
  - Simple window algorithm
    - W independent single-packet algorithms
    - When one finishes, grab next packet
- Weaknesses?
  - Loss of ACK packet requires a retransmission

Cumulative ACK

- ACK the highest sequence number for which all previous packets have been received
  - Implementations often send back "next expected packet", but that's just a detail
- Strengths:
  - Recovers from lost ACKs
- Weaknesses?
  - Confused by reordering
  - Incomplete information about which packets have arrived

Full Information

- List all packets that have been received
  - Give highest cumulative ACK plus any additional packets
  - Feasible if only small holes

- Strengths:
  - As much information as you could hope for
  - Resilient form of individual ACKs

- Weaknesses?
  - Could require sizable overhead in bad cases

Detecting Loss

- If packet times out, assume it is lost….
- How else can you detect loss?

Loss with individual ACKs

- Assume packet 5 is lost, but no others

- Stream of ACKs will be:
  - 1
  - 2
  - 3
  - 4
  - 6
  - 7
  - 8
  - ….

Loss with individual ACKs

- Could resend packet when k “subsequent packets” are received
- Response to loss:
  - Resend missing packet
  - Continue window based protocol

Loss with full information

- Same story, except that the "hole" is explicit

- Stream of ACKs will be:
  - Up to 1
  - Up to 2
  - Up to 3
  - Up to 4
  - Up to 4, plus 6
  - Up to 4, plus 6,7
  - Up to 4, plus 6,7,8
  - ….

Loss with full information

- Could resend packet when k “subsequent packets” are received

- Response to loss:
  - Resend missing packet
  - Continue window based protocol

Loss with cumulative ACKs

- Assume packet 5 is lost, but no others

- Stream of ACKs will be:
  - 1
  - 2
  - 3
  - 4
  - 4 (when 6 arrives)
  - 4 (when 7 arrives)
  - 4 (when 8 arrives)
  - ….

Loss with cumulative ACKs

- "Duplicate ACKs" are a sign of an isolated loss
  - The lack of ACK progress means 5 hasn't been delivered
  - The stream of ACKs means that some packets are being delivered
- Therefore, could trigger resend upon receiving k duplicate ACKs
- But response to loss is trickier….

Loss with cumulative ACKs

- Two choices:
  - Send missing packet and optimistically assume that subsequent packets have arrived
    - i.e., increase W by the number of Dup ACKs
  - Send missing packet, and wait for ACK

- Timeout-detected losses also problematic
  - If packet 5 times out, packet 6 is about to time out also
  - Do you resend both?
  - Do you resend 5 and wait?
  - ….

Go-Back-N

- Simple algorithm (not advisable, but simple)
- Sliding window (only W contiguous packets)
- When a loss is detected by timeout, resend all W packets starting with loss
- Receiver discards out-of-order packets

All the bad things best effort can do…

- Packets can be lost
- Packets can be corrupted
- **Packets can be reordered**
- **Packets can be delayed**
- **Packets can be duplicated**

Effect of Reordering?

- Individual ACKs: not a problem
- Full information: not a problem
- Cumulative ACKs: create Dup ACKs

Effect of Long Delays?

- Possible timeouts

Effect of Duplication?

- Produce Duplicate ACKs
  - Could be confused for loss with cumulative ACKs
  - But duplication is rare….

Possible Design

- Full information ACKs
- Window-based, with retransmissions after:
  - Timeout
  - K subsequent ACKs
- This is correct, timely, efficient

Fairness?

- Adjust W based on losses….

- In a way that flows receive same shares

- Short version:
  - Loss: cut W by 2
  - Successful receipt of window: W increased by 1

Summary

- Window-based flow control separates concerns
  - Size of W:
  - Nature of feedback:
  - Response to loss:
- Can design each aspect relatively independently
- Can be correct, efficient, timely, and fair

Are We Done?

- There are other approaches….

Alternate Strategy: Rateless Codes

- Use special encoding
  - Receipt of any set of M packets allows you to recover file
  - Where M is close to the size of the original file
- Receiver only sends ACK when M are received
  - Sender keeps sending until receives ACK
- Timely, Correct
  - How efficient is it?

The Paradox of Internet Traffic

- The majority of flows are short
  - A few packets
- The majority of bytes are in long flows
  - MB or more
- And this trend is accelerating…

Inefficiency

- The wasted bandwidth ~ BxRTT
- For long flows, this is small compared to total file
- For short flows, this is large compared to file
  - But most of the bandwidth is in long flows!
- This is not a terrible idea
- What is missing?

Next Lecture

- Routing