Layer Optimization: Congestion Control
CS 118
Computer Network Fundamentals
Peter Reiher

We can lose packets for many reasons

- Corruption
- Not delivered to receiver
- Poor flow control
- But also because of overall network conditions
- If there's too much traffic in the net, not all packets can be delivered
  - Can happen locally at one link or one part of network

Congestion control

- Receiver might be ready, but is the net?
  - Don't want to overwhelm the network
- We have some windows
  - Send = how much info *can* be outstanding
  - Recv = how much info *can* be reordered
- *Can* isn't the same as *should*

How much SHOULD be outstanding?

A network problem

- Congestion control is not directly about the sender and receiver
- It's about the network path they use
  - And share with others
- The shared paths can only handle so much traffic
- A given sender might send less
- But all the senders using the path in combination might overwhelm it
  - Perhaps just part of it

How to address the congestion control problem?

- A global problem, so perhaps a global solution?
- But who is in charge of the problem?
- And how does that party enforce its dictates?
- Instead, if everyone cooperates, maybe we can solve it without global control
- Everyone does his part to solve the problem, leading to a better global solution

But what can I do?

- You can only change your own behavior
- But if everyone does, that will reduce the congestion
- And life becomes better for everyone
- OK, so how do I change my behavior to help?
- And how much should I change it?

Recall the two windows

- Receiver window
  – Reorder out-of-order arrivals
  – Buffer messages until receiver catches up
- Send window
  – Hold for possible retry until ACKed
  – Emulate how the channel delays/stores messages in a pipeline until ACKed

Send window maximum

- Round-trip to the receiver
  - "BW * delay" product
  - Really "fill the pipe until you get an ACK", presuming there isn't any loss
- Once you fill the pipe, send at the rate you get ACKs
  - ACK clocking
  - Forces sender to pace to the receiver

TCP and congestion control

- TCP is one protocol that addresses congestion control
- Probably the most important congestion control factor in the Internet
- Essentially a cooperative approach
- When congestion occurs, all TCP senders slow down

TCP's CWND

- Another window used by TCP
- Not the same as the send window
- Not intended to handle flow control
- Rather, to handle congestion control

TCP MSS and RTT

- Two important parameters for TCP use
- MSS – Maximum Segment Size
  - Biggest TCP payload you can fit into one IP packet
  - By default, 536 “octets” (essentially bytes)
  - Find it by trial and error
- RTT – Round Trip Time
  - Time to send a TCP packet and receive an ACK

Adjusting the congestion window

- TCP CWND management
  - CWND is the send window max
    - Starts at 1, 4, 10K, or 10 packets
- Additive Increase
  - Until you see loss, increase CWND by a constant amount for every ACK
- Multiplicative decrease
  - When you see loss, halve CWND

AIMD feedback

- A conservative approach
- Grow slowly by probing
- Backoff faster than you grow if there’s signs of trouble

The slow start phase

- New TCP connection starts in a slow start phase
  - Until CWND reaches SSTRESH
    - A parameter of TCP
- CWND grows by 1 for each ACK
  - I.e., CWND doubles* each RTT

Why's that exponential?

- Sender sends out some number of packets N
  – Without waiting for an ACK
- If all goes well, N ACKs come back quickly
- You add one to CWND for each ACK
- So the next time, you send out 2*N packets
- And expect back 2*N ACKS
- In which case, you add 2*N to CWND
  – Getting 4*N
- That's exponential

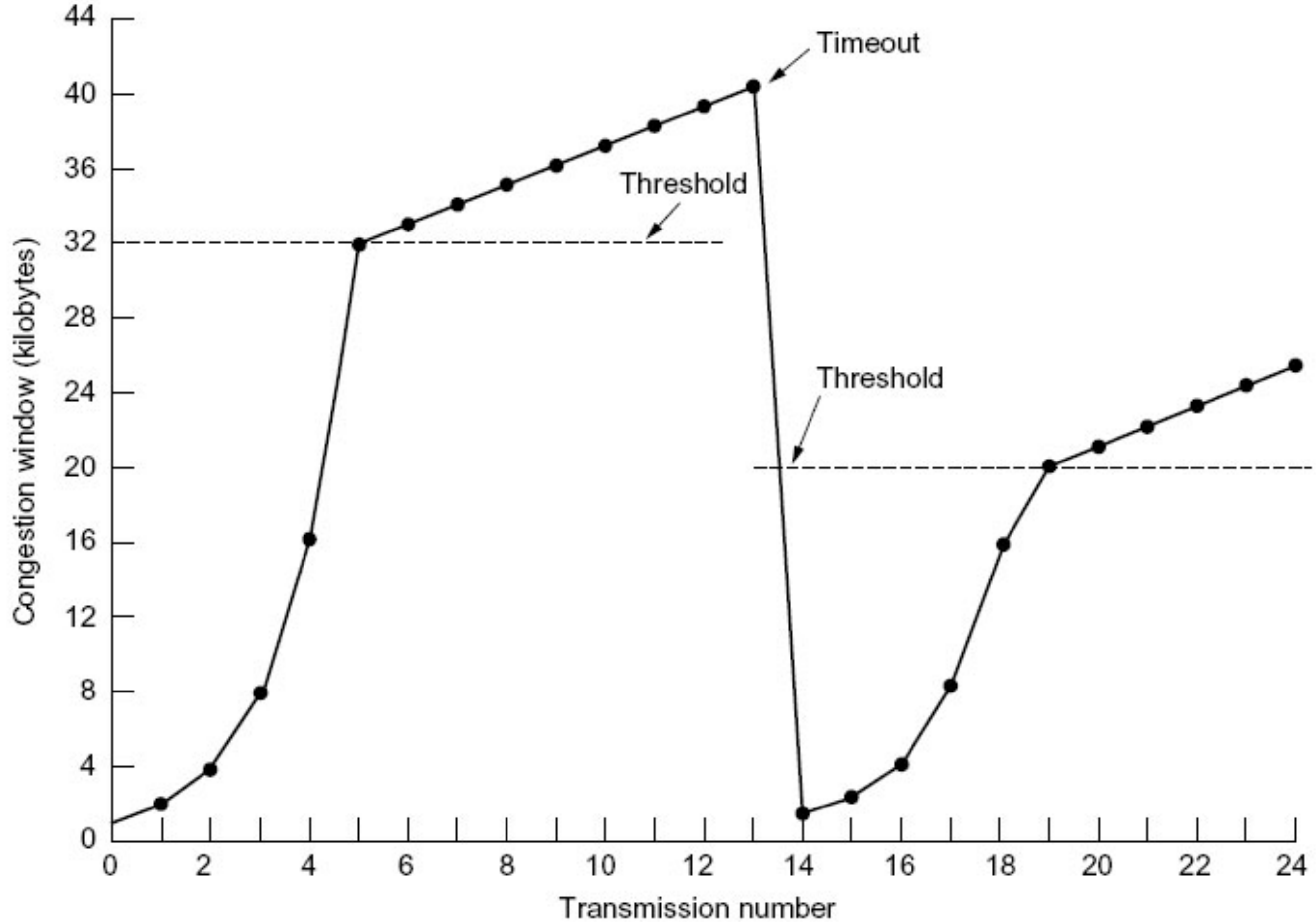
Why does it stop?

- Either you hit the limit to change TCP congestion control behavior
  - Your CWND reaches SSTHRESH
- Or you time out waiting for an ACK
  - Assuming that the packet is lost
  - Due to congestion
  - Will that assumption always be true . . . ?
- In latter case, also halve SSTHRESH
  - Depending on TCP variant

Congestion avoidance phase

- Happens once SSTHRESH is reached
- Assumption is that there is no congestion so far
- Inch up a bit further to see if more can be sent
  - Until you reach MAX
- CWND grows by 1 for each RTT
  - NOT each ACK received

Visualization



Details

- CWND doesn't double per RTT in slow start
  - Because receiver doesn't ACK every segment
  - It ACKs every other ("ACK compression")
  - CWND increases by 50% each RTT in slow start
- This is one TCP variant
  - There are dozens, and they keep changing!

TCP's biggest assumption

- TCP only knows:
  - What arrived
  - A timeout happened
- TCP measures:
  - RTT directly (timestamps)
    - Based on sent packets and ACKs
  - Max receive window (window)
  - ***Network congestion (via timeout!)***

What does a loss mean?

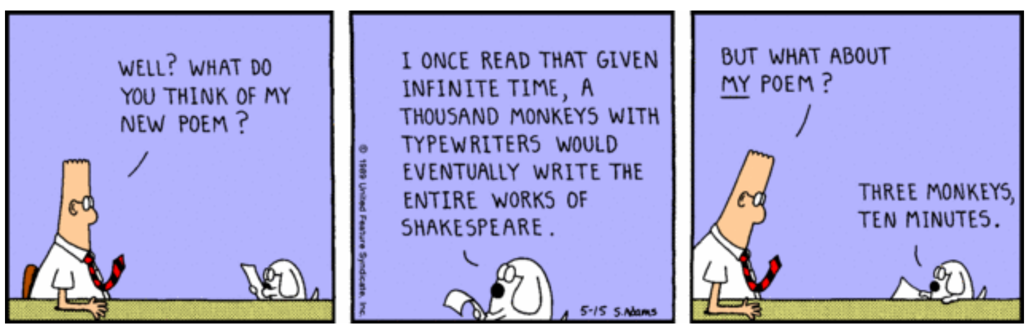
- Corruption
  - Should send more, i.e., send another copy
- Congestion
  - Should send less
- TCP assumes loss implies congestion
  - I.e., the more conservative interpretation

Impact of loss=congestion

- TCP works poorly when corruption is high
  - I.e., wireless networks
  - When corruption is not due to load
- TCP is aggressive
  - It keeps sending more until something is lost
  - Two TCP flows always fight each other
- But TCP loses to cheaters
  - TCP backs off
  - Others might not

Congestion control algorithms

- Many of them
  - Lots of variations
  - Lots of incremental tweaks
  - Many based on fluid flow, feedback theory
  - Many based on whomever types it in…



Latency management

- Networks have buffers
  - Buffers adjust for bursts
- Most networks “tail drop”
  - I.e., keep as many messages as the buffer can hold, and drop ones that arrive once full
- Tail drop favors keeping buffers full
  - Full buffers mean high delays

Solutions to latency management

- Explicit network congestion signals
  - Routers tell endpoints when buffers are filling
- Progressive loss
  - Drop probability increases as buffer grows
  - Don’t just wait for “full” and drop all
  - “Random Early Drop” and variants

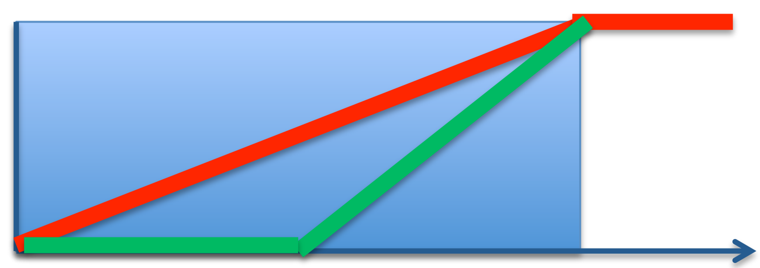
Explicit congestion notification (ECN)

- ECN routers (relays) indicate congestion
  - Mark instead of drop
  - Implies space to hold marked packets
  - So really more like “mark before drop”
  - E.g., mark packets arriving when queue is more than half full
- Endpoints react to ECN flags as if congestion was noticed
  - For TCP, ECN makes the CWND smaller
  - TCP can react to congestion without losing packets

What if ECN isn't available?

- Tail-drop queue
  - Do not drop if there's room
  - Drop if queue is full

- Random Early Detection
  - Drop probability increases as queue grows
  - Various curves



Better buffering

- Relays can cause problems
  - Connections compete one packet at a time
  - Maybe separate buffering by connections is better
    - "Fair queuing"
  - Need better use of buffers
    - Memory is cheap, but has a cost

Space

- Compression
- Caching

Compression

- Translate a set of long messages into a set of short ones
  - Take a set of messages
  - Represent frequent ones with fewer bits, longer ones with more bits
- Translate a long message into a short one
  - Take a set of groups of symbols in a message
  - Represent frequent groups with fewer bits, longer ones with more bits

Compression examples

- Web traffic
- E-mail
- TCP/IP headers

Web traffic

- HTTP 1.1
  - Compress content of responses
  - E.g., zip images, large text areas
  - Inside Google Chrome browser
- HTTP 2.0
  - Compress headers

E-mail

- By the program
  - Postscript, Word
- By the user in advance
  - Zip folders
- By the email system
  - Compress attachments

TCP/IP headers

- Compress the TCP and IP headers
  - 40 bytes down to 16
  - Most of the header is predictable within a single connection
- Typical for PPP and SLIP (dial-up lines)
  - I.e., over path that doesn't examine the header

TCP/IP compression

- When is it useful?
  - What benefit?
    - For 40B ACK packets, saves 60%
    - For 512B payload data, saves 4%
    - For 1500B segments (Ethernet), saves 1.6%
  - Where useful?
    - ACK-only, BW-limited returns
    - For 2400bps modems (1990), saves 87ms

Required compression information

- Patterns and frequencies of those patterns
  - Usually from a set of previous messages
    - E.g., Morse code
  - Or from previous use on this channel
    - E.g., LZW, used in GIFs
  - Or just obvious patterns
    - Run-length encoding, used for faxes and JPEG

Compression trade-offs

- Trade (consume)
  - Effort
    - CPU works harder
  - Energy
    - CPU burns power
  - Time
    - Encode/decode needs to delay the stream
    - Encode/decode operation takes time
- Gain (produce)
  - Space
    - Smaller message takes up less memory
  - Capacity
    - Smaller message uses less bandwidth
  - Time
    - Smaller message takes less time to transfer

Compression caveats

- Works once
  - Compression removes patterns
  - Works at only ONE layer or over ONE hop
- Obscures information
  - Can’t modify or easily read until undone
  - Uncompress/recompress is expensive
- Small returns if used on only part of large messages
  - HTTP/2 header compression is controversial

Caching

- Save via reuse
  - Over time within one stream
    - If you have the answer from before, use it again
  - Across a set of streams
    - Don’t ask if your friends know the answer

Caching examples

- Inside a protocol
  - TCP control block sharing, TCP/IP compression
- Content
  - ARP, DNS, Web

TCP control block sharing

- New connections start from "zero"
  - Why?

- New connections can reuse
  - From past (reuse CWND, RTT, MSS)
  - From peers (reuse RTT, MSS, split CWND)

Why reuse?

- Change is unlikely
  - Path (routing) tends to be stable
  - Endpoints tend to be stable
  - Aggregate traffic patterns tend to be stable
  - So RTT, MSS tend to be stable
- Why infer when you can share?
  - Endpoints within the same machine can share
  - No need to have CWNDs fight and balance; can just "split at start"

Net effect of TCP sharing

- Less blind probing
  - No need to send large segments to find MSS
  - No need to use RTT over-estimates
- No need to compete via loss
  - Shared info can "rebalance" CWND
- Safe
  - Tries to anticipate transients – only at connection start/end
  - Tries to jump closer to convergence, then lets existing feedback take over

More complex sharing

- Endpoints within a LAN
  - Can share their experience
  - Can explicitly coordinate rather than compete
- Inherently harder
  - No longer just sharing information on a single computer
  - Which means it must be communicated

Information delineation

- Boundaries
- Flows

Boundaries

- Message vs. packet alternatives
  - *Span*: messages longer than a packet
  - *Preserve*: message matches packet
  - *Pack*: packet carries multiple messages
  - *None*: no boundary support (e.g., TCP)

Adding markers is easy…

- Length indicator
  - E-mail attachments, IP packets, HTTP chunks
  - Efficient (rapid jump), but fixed max
- Special symbols (“escape” sequences)
  - Not used for data
  - Arbitrary chunk size, but need to scan

Deciding marker use is hard

- Costs
  - Gathering small chunks can cause delays
  - Picking the wrong size increases overheads
  - Cost to split/merge or merge/split
- Risks
  - Lack of fate-sharing
    - Different chunks via different paths

Marker examples

- Length
  - *Pack*: HTTP, e-mail, SCTP
  - *Preserve*: UDP, DCCP
  - *Span*: ATM AAL5, IP frag., multipart MIME
- Special symbols ("escape" sequences)
  - ATM, Ethernet preamble

Flows

- Like a channel…
  - Information shared between parties
- …with multiple viewpoints simultaneously
  - One channel
  - Several separate channels

Examples of multiple flows

- Multiplexing
- Striping (inverse multiplexing)
- Partitioning

Multiplexing

- Using one flow to emulate many
  - HTTP chunking and muxing
  - Allows one TCP connection to support concurrent web transfers
- Hazards
  - "Fair sharing"
  - Head-of-line blocking

Fair-sharing

- Merging multiple flows onto one
  - Who goes next?
- Various strategies
  - Shortest-first, largest-first, round-robin, proportional
- How is “fair” defined?
  - Each according to their needs?
  - Each gets the same?
- How is “each” defined?
  - Per human? Per endpoint? Per application?

Head-of-line blocking

- Consider lines at a market
  - Large basket arrives before 2-item
  - BLOCKS system
- Avoiding HOL blocking?
  - Limit chunksize
    - E.g., everyone pays 10 items at a time
    - Leaves when done paying for entire basket
  - Use separate connections
    - E.g., multiple TCP connections for web clients

Striping

- Making multiple channels appear as one
  - Increased bandwidth
  - Increased reliability
- Examples
  - Multipath TCP
  - SCTP
  - Various datacenter optimizations

Partitioning

- Split one info stream into separate ones
  - To avoid HOL blocking
  - To manage differently (loss vs. recovery)
- Examples
  - Teleconference audio vs. video
  - FTP control vs. content

Translation

- Formats
- Conversion
- Marshalling

Recall encodings

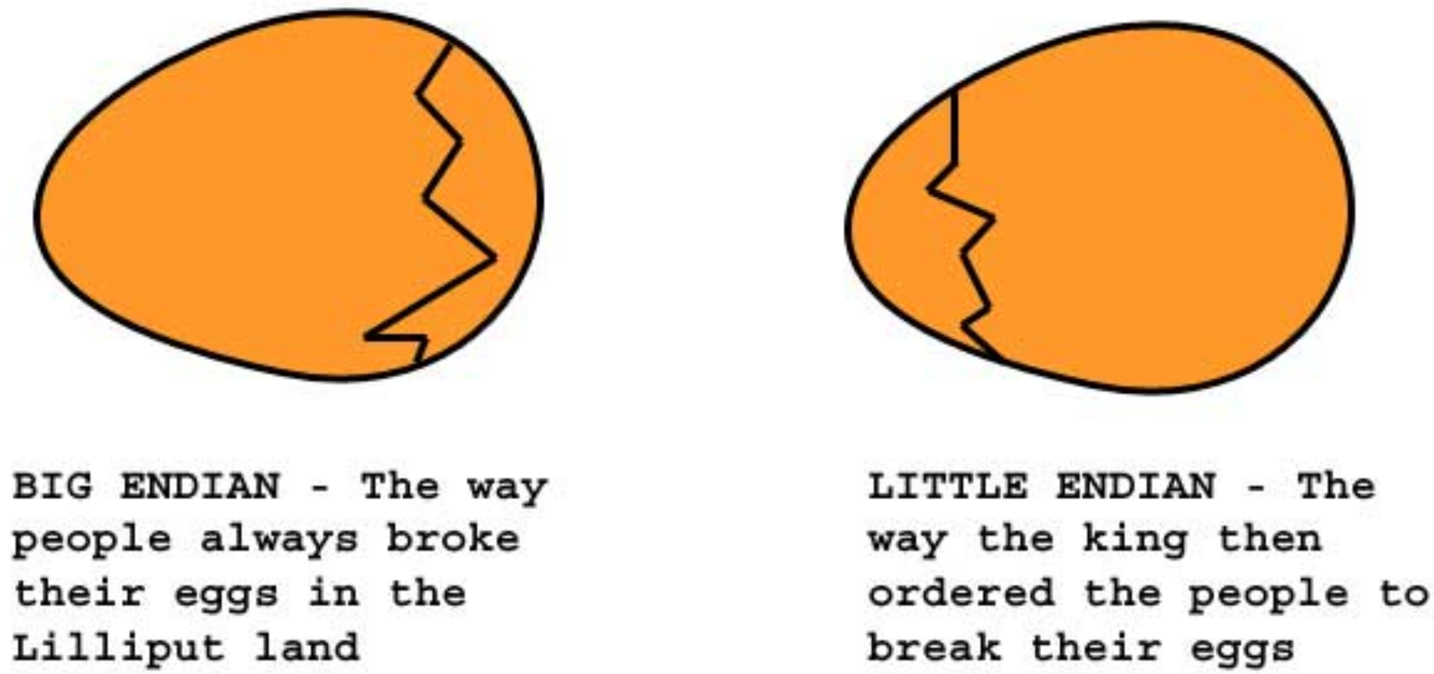
- Represent information with symbols
  - Various strategies
  - Earlier lectures focused on physical, error
- More encoding issues
  - More encoding variants
  - Coordinating the endpoints

Bit order and formats

- Many channels exchange bit sequences
  - Upper layers exchange bytes, words, etc.
  - What order?
- LSB vs. MSB
  - LSB-first: enables serial arithmetic
    - Ethernet, Token bus
  - MSB-first:
    - Token ring

On holy wars and plea for peace

- Gulliver's Travels



Endianess

- Big-endian: ABCD stored as A, B, C, D
  - The Internet
  - Motorola 68000, RISC (PowerPC, SPARC)
  - Telephone numbers
- Little-endian: ABCD stored as D, C, B, A
  - Intel and AMD processors
- Both (configurable)
  - ARM

Conversion

- Host to net, net to host
  - Long, short, etc.
  - Converts from Internet (big-endian) to local

Marshalling

- Packing and unpacking
  - Format conversion
  - Sequencing
  - Labeling
- All for what?
  - Same as for a function call
  - A way to know the meaning of shared bits

Why is marshalling hard?

- Expensive
  - Conversion takes time
- Tedious
  - Many steps to mess up
- Exacting
  - All the steps have to match to work

Summary

- Lots more optimizations and features
  - The details depend on the implementation
- Details matter and they don't
  - Parties must agree on details to communicate
  - Detail differences affect performance
  - But particulars of details not always otherwise critical
  - Things can be done many ways