TCP Flow Control & Congestion Control

Arvind Krishnamurthy
Spring 2003

Transmission Control Protocol

- TCP relies on IP's message delivery protocol

TCP properties:
- Program-to-program connection
- Sequenced and flow controlled
- End-to-end reliable

TCP transmission strategy:
- Have a number of packets outstanding
- If a packet is not acknowledged in a short period of time, retransmit

Window Based Protocol

- Send up to N messages at a time without waiting for acks
- "Window" also reflects storage at receiver - sender shouldn’t overrun receiver's buffer space
- Send a new message only when a new ack is received
- Each message is identified by its sequence number

Flow Control Details

Sender messages
- sent, acked | sent, not acked | not sent | not sent
- "in window" | not "in window"

Receiver messages
- received | received, acked | maybe received | not yet received
- given to app | buffered | not asked

- Receiver asks: "got all messages up to #"
- Duplicate acks implies holes in received message sequence
- Sender can perform "fast retransmit"

TCP Flow Control

- Assume:
  - Receiver window size = 4K
  - TCP minimum threshold for sending = 1K
  - Initial sequence number = 3000

TCP Flow Control 2

- Acknowledgement message: “expecting byte # x”
TCP Flow Control 3

User
Send 4K bytes
Buffer: 1K → Seq: 4000, Size: 3K → Buffer: 1K
User

Buffer: 1K → Ack: 7000 → Buffer: 4K

- Send only up to the receiver window

TCP Flow Control 4

User
Window: 2K
Buffer: 1K → Ack: 7000 → Buffer: 2K
User

Buffer: 5K → Seq: 7000, Size: 1K → Buffer: 3K

- Receiver sends a window update when user picks up data

TCP Flow Control 5

User
Window: 3K
Buffer: 0K → Ack: 9000 → Buffer: 1K
User

3 x Send: 1K bytes
Buffer: 0K → Seq: 8000, Size: 1K → Buffer: 0K
Buffer: 0K → Seq: 10000, Size: 1K → Buffer: 3K

TCP Flow Control 6

User
Ack: 9000, Wdw: 1K
Buffer: 0K → Ack: 11000 → Buffer: 2K
User

Buffer: 0K → Seq: 9000, Size: 1K → Buffer: 2K

Connections

- Requires three-way handshakes
- Setup:
  - Open request packet (SYN, initial sequence number)
  - Acknowledgement (SYN, own sequence number, ack number)
  - Acknowledgement of the acknowledgement
- SYN occupies 1 byte of sequence space

Failure Scenarios

- Cannot reuse sequence number if there are some old live data
- Keep track of previous recent connections
- What if machines go up and down?
  - Wait for a while when machine reboots
  - Let old packets die
- What if connection packets get lost?
  - Timeout and retransmit
  - But initially very conservative estimate of RTT
Connection Tear Down

- Keep connection state around for some more time
  - FIN occupies 1 byte in sequence space
  - Connection state is last byte received in sequence
- Typically kept for 2*RTT duration
- No clean solution as to when state can be forgotten
- A distributed consensus problem

General’s Paradox

- Two generals on separate mountains
- Can communicate only via messengers
  - Messengers can be captured
- Need to coordinate an attack
  - If they attack at the same time, they win
  - Else they will all die
- Devise a protocol to coordinate the two generals

Announcements

- Assignment 4 has been posted online
- Change in readings:
  - Authentication in distributed systems: next Monday
  - Friday: Remote Procedure Calls and Lightweight Remote Procedure Calls

Congestion Control

- Window size controls flow and congestion
  - Receiver advertised window is maximum amount of data that can be outstanding
  - Have a smaller window if there is congestion in the system
- Canonical congestion problem:
  - Flow between A-B uses up link capacity
  - Flow between C-D starts, resulting in congestion on the link

Issues

- Flows need to find a fair use of link resources
  - When a flow starts, it needs to find what is available reasonably fast and under different network capacities
- Flows need to distinguish packet losses from packet delays
  - Spurious detection of packet loss results in more traffic, more congestion, more delays, and so on
- Flows need to adapt to changing network conditions
  - Sometimes increase its utilization, sometimes lower its utilization

Finding Equilibrium from Startup

- Two features:
  - Self-clocking mechanism
    - “Slow start” mechanism - actually ramps up rather fast!
  - Self-clocking:
    - Send a new packet only when a previous packet is acknowledged
    - Soon packets are sent at the rate they are received
Slow Start Mechanism
- Initially set cwnd to be 1
- Maintain the invariant that cwnd < window given by receiver
- Increment cwnd by 1 for every acknowledgement

Accurate Round Trip Time Estimates
- How long should timeout be?
  - Too long? Wastes time
  - Too short? Retransmits even though message is not lost
- Maintain running estimate of R*
  \[ R = (1-\alpha)R + \alpha M \]
  where M is new measurement, \( \alpha \) is decay constant
  - High \( \alpha \) makes it unstable
  - Low \( \alpha \) makes the system have too much history
- Also measure the error or variance in measurements
- Set timeout to be R + 4*variance

Congestion Avoidance Algorithm
- React to changing network conditions by modifying cwnd
- At loss: (multiplicative decrease)
  \[ cwnd = \frac{cwnd}{2} \]
  Better to have a drastic decrease when losses occur
- After loss: (additive increase)
  \[ cwnd += \frac{1}{cwnd} \]
  Results in slow increase; probes for available bandwidth
  Better to have a conservative increase policy

Question:
- How do you get good TCP Performance?

Congestion Control at Routers
- Router queues can fill up
- When they fill up?
  - What to drop?
  - When to drop?
- Random Early Detection (RED) algorithm: use randomization
- Router can be in one of three states:
  - Few packets in the queues: do not drop any packets (normal operating phase)
  - Lots of packets in the queues: drop for sure (congestion phase)
  - Intermediate number of packets: calculate probability for dropping based on queue length and number of packets since last drop (congestion avoidance phase)

RED Drop Probability
- Voodoo constants: \( \text{minQThresh}, \text{maxQThresh}, \text{maxp} \)
- Step 1:
  \[ p = \text{maxp} \times \frac{\text{avgQlen} - \text{minQThresh}}{\text{maxQThresh} - \text{minQThresh}} \]
  \( p \leq \text{maxp} \)
  \( \text{avgQlen} \) is calculated as a weighted average over time
- Step 2:
  Drop probability = \( p / (1 - \text{count} \times p) \)
  \( \text{count} \) is number of packets since last drop
  Try to avoid cascading drops