Introduction to Computer Networks

Congestion Overview
(§6.3, §6.5.10)

Computer Science & Engineering

UNIVERSITY of WASHINGTON

Topic

• Understanding congestion, a “traffic jam” in the network
  — Later we will learn how to control it

What’s the hold up?
Nature of Congestion

• Simplified view of per port output queues
  – Typically FIFO (First In First Out), discard when full

Nature of Congestion (2)

• Queues help by absorbing bursts when input > output rate
• But if input > output rate persistently, queue will overflow
  – This is congestion
• Congestion is a function of the traffic patterns – can occur even if every link have the same capacity
Effects of Congestion

• What happens to performance as we increase the load?

![Graph showing effects of congestion on goodput and delay](image)

Effects of Congestion (2)

• What happens to performance as we increase the load?

![Graph showing effects of congestion on goodput and delay](image)
Effects of Congestion (3)

- As offered load rises, congestion occurs as queues begin to fill:
  - Delay and loss rise sharply with more load
  - Throughput falls below load (due to loss)
  - Goodput may fall below throughput (due to spurious retransmissions)

- None of the above is good!
  - Want to operate network just before the onset of congestion

Bandwidth Allocation

- Important task for network is to allocate its capacity to senders
  - Good allocation is efficient and fair

- Efficient means most capacity is used but there is no congestion
- Fair means every sender gets a reasonable share the network
Bandwidth Allocation (2)

• Why is it hard? (Just split equally!)
  – Number of senders and their offered load is constantly changing
  – Senders may lack capacity in different parts of the network
  – Network is distributed; no single party has an overall picture of its state

Bandwidth Allocation (3)

• Key observation:
  – In an effective solution, Transport and Network layers must work together

• Network layer witnesses congestion
  – Only it can provide direct feedback

• Transport layer causes congestion
  – Only it can reduce offered load
Bandwidth Allocation (4)

- Solution context:
  - Senders adapt concurrently based on their own view of the network
  - Design this adaptation so the network usage as a whole is efficient and fair
  - Adaptation is continuous since offered loads continue to change over time

Introduction to Computer Networks

Fairness of Bandwidth Allocation (§6.3.1)

Computer Science & Engineering

UNIVERSITY of WASHINGTON
Topic

• What’s a “fair” bandwidth allocation?
  – The max-min fair allocation

Recall

• We want a good bandwidth allocation to be fair and efficient
  – Now we learn what fair means

• Caveat: in practice, efficiency is more important than fairness
Efficiency vs. Fairness

• Cannot always have both!
  – Example network with traffic A→B, B→C and A→C
  – How much traffic can we carry?

\[ \text{Efficiency vs. Fairness (2)} \]

• If we care about fairness:
  – Give equal bandwidth to each flow
  – A→B: \( \frac{1}{2} \) unit, B→C: \( \frac{1}{2} \), and A→C, \( \frac{1}{2} \)
  – Total traffic carried is \( 1 \frac{1}{2} \) units
Efficiency vs. Fairness (3)

• If we care about efficiency:
  – Maximize total traffic in network
  – $A \rightarrow B$: 1 unit, $B \rightarrow C$: 1, and $A \rightarrow C$, 0
  – Total traffic rises to 2 units!

The Slippery Notion of Fairness

• Why is “equal per flow” fair anyway?
  – $A \rightarrow C$ uses more network resources (two links) than $A \rightarrow B$ or $B \rightarrow C$
  – Host $A$ sends two flows, $B$ sends one

• Not productive to seek exact fairness
  – More important to avoid starvation
  – “Equal per flow” is good enough
Generalizing “Equal per Flow”

- **Bottleneck** for a flow of traffic is the link that limits its bandwidth
  - Where congestion occurs for the flow
  - For A→C, link A–B is the bottleneck

![Diagram](image1)

Generalizing “Equal per Flow” (2)

- Flows may have different bottlenecks
  - For A→C, link A–B is the bottleneck
  - For B→C, link B–C is the bottleneck
  - Can no longer divide links equally ...

![Diagram](image2)
Max-Min Fairness

- Intuitively, flows bottlenecked on a link get an equal share of that link

- Max-min fair allocation is one that:
  - Increasing the rate of one flow will decrease the rate of a smaller flow
  - This “maximizes the minimum” flow

Max-Min Fairness (2)

- To find it given a network, imagine “pouring water into the network”
  1. Start with all flows at rate 0
  2. Increase the flows until there is a new bottleneck in the network
  3. Hold fixed the rate of the flows that are bottlenecked
  4. Go to step 2 for any remaining flows
Max-Min Example

• Example: network with 4 flows, links equal bandwidth
  – What is the max-min fair allocation?

Max-Min Example (2)

• When rate=1/3, flows B, C, and D bottleneck R4—R5
  – Fix B, C, and D, continue to increase A

Bottleneck
Max-Min Example (3)


Max-Min Example (4)

- End with A=2/3, B, C, D=1/3, and R2—R3, R4—R5 full
  - Other links have extra capacity that can’t be used
Adapting over Time

• Allocation changes as flows start and stop

Adapting over Time (2)

Flow 1 slows when Flow 2 starts
Flow 1 speeds up when Flow 2 stops

Flow 3 limit is elsewhere
Introduction to Computer Networks

Additive Increase
Multiplicative Decrease (AIMD)
(§6.3.2)

Recall

• Want to allocate capacity to senders
  – Network layer provides feedback
  – Transport layer adjusts offered load
  – A good allocation is efficient and fair

• How should we perform the allocation?
  – Several different possibilities ...
### Bandwidth Allocation Models

- **Open loop versus closed loop**
  - Open: reserve bandwidth before use
  - Closed: use feedback to adjust rates
- **Host versus Network support**
  - Who sets/enforces allocations?
- **Window versus Rate based**
  - How is allocation expressed?

> TCP is a closed loop, host-driven, and window-based

### Additive Increase Multiplicative Decrease

- **AIMD** is a control law hosts can use to reach a good allocation
  - Hosts additively increase rate while network is not congested
  - Hosts multiplicatively decrease rate when congestion occurs
  - Used by TCP

- Let’s explore the AIMD game ...
AIMD Game

- Hosts 1 and 2 share a bottleneck
  - But do not talk to each other directly
- Router provides binary feedback
  - Tells hosts if network is congested

AIMD Game (2)

- Each point is a possible allocation
AIMD Game (3)

- AI and MD move the allocation

AIMD Game (4)

- Play the game!
AIMD Game (5)

- Always converge to good allocation!

AIMD Sawtooth

- Produces a “sawtooth” pattern over time for rate of each host
  - This is the TCP sawtooth (later)
AIMD Properties

• Converges to an allocation that is efficient and fair when hosts run it
  – Holds for more general topologies
• Other increase/decrease control laws do not! (Try MIAD, MIMD, AIAD)
• Requires only binary feedback from the network

Feedback Signals

• Several possible signals, with different pros/cons
  – We’ll look at classic TCP that uses packet loss as a signal

<table>
<thead>
<tr>
<th>Signal</th>
<th>Example Protocol</th>
<th>Pros / Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss</td>
<td>TCP NewReno</td>
<td>Hard to get wrong</td>
</tr>
<tr>
<td></td>
<td>Cubic TCP (Linux)</td>
<td>Hear about congestion late</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet delay</td>
<td>Compound TCP (Windows)</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Need to infer congestion</td>
</tr>
<tr>
<td>Router indication</td>
<td>TCPs with Explicit Congestion</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td>Notification</td>
<td>Require router support</td>
</tr>
</tbody>
</table>


TCP Tahoe/Reno

- Avoid congestion collapse without changing routers (or even receivers)
- Idea is to fix timeouts and introduce a congestion window (cwnd) over the sliding window to limit queues/loss
- TCP Tahoe/Reno implements AIMD by adapting cwnd using packet loss as the network feedback signal

TCP Tahoe/Reno (2)

- TCP behaviors we will study:
  - ACK clocking
  - Adaptive timeout (mean and variance)
  - Slow-start
  - Fast Retransmission
  - Fast Recovery
- Together, they implement AIMD
Introduction to Computer Networks

TCP Ack Clocking (§6.5.10)

Sliding Window ACK Clock

- Each in-order ACK advances the sliding window and lets a new segment enter the network
  - ACKs “clock” data segments
Benefit of ACK Clocking

- Consider what happens when sender injects a burst of segments into the network

Benefit of ACK Clocking (2)

- Segments are buffered and spread out on slow link
Benefit of ACK Clocking (3)

- ACKs maintain the spread back to the original sender

Benefit of ACK Clocking (4)

- Sender clocks new segments with the spread
  - Now sending at the bottleneck link without queuing!
Benefit of ACK Clocking (4)

• Helps the network run with low levels of loss and delay!

• The network has smoothed out the burst of data segments

• ACK clock transfers this smooth timing back to the sender

• Subsequent data segments are not sent in bursts so do not queue up in the network

Introduction to Computer Networks

TCP Slow Start (§6.5.10)
TCP Startup Problem

• We want to quickly near the right rate, $cwnd_{\text{ideal}}$, but it varies greatly
  – Fixed sliding window doesn’t adapt and is rough on the network (loss!)
  – AI with small bursts adapts $cwnd$ gently to the network, but might take a long time to become efficient

Slow-Start Solution

• Start by doubling $cwnd$ every RTT
  – Exponential growth (1, 2, 4, 8, 16, ...)
  – Start slow, quickly reach large values
Slow-Start Solution (2)

- Eventually packet loss will occur when the network is congested
  - Loss timeout tells us cwnd is too large
  - Next time, switch to AI beforehand
  - Slowly adapt cwnd near right value

- In terms of cwnd:
  - Expect loss for $cwnd_C \approx 2BD+queue$
  - Use ssthresh = $cwnd_C/2$ to switch to AI

Slow-Start Solution (3)

- Combined behavior, after first time
  - Most time spend near right value

---

Diagram: 

- Window $cwnd_C$ 
- $cwnd_{IDEAL}$ 
- ssthresh 
- Fixed 
- Al phase 
- Slow-start 
- Al 
- Time
Slow-Start (Doubling) Timeline

Increment cwnd by 1 packet for each ACK

Additive Increase Timeline

Increment cwnd by 1 packet every cwnd ACKs (or 1 RTT)
TCP Tahoe (Implementation)

- Initial slow-start (doubling) phase
  - Start with cwnd = 1 (or small value)
  - cwnd += 1 packet per ACK

- Later Additive Increase phase
  - cwnd += 1/cwnd packets per ACK
  - Roughly adds 1 packet per RTT

- Switching threshold (initially infinity)
  - Switch to AI when cwnd > ssthresh
  - Set ssthresh = cwnd/2 after loss
  - Begin with slow-start after timeout

Timeout Misfortunes

- Why do a slow-start after timeout?
  - Instead of MD cwnd (for AIMD)
Timeout Misfortunes

• Why do a slow-start after timeout?
  – Instead of MD cwnd (for AIMD)

• Timeouts are sufficiently long that the ACK clock will have run down
  – Slow-start ramps up the ACK clock

• We need to detect loss before a timeout to get to full AIMD
  – Done in TCP Reno

Introduction to Computer Networks

TCP Fast Retransmit / Fast Recovery (§6.5.10)

Computer Science & Engineering
UNIVERSITY of WASHINGTON
Inferring Loss from ACKs

• TCP uses a cumulative ACK
  – Carries highest in-order seq. number
  – Normally a steady advance
• Duplicate ACKs give us hints about what data hasn’t arrived
  – Tell us some new data did arrive, but it was not next segment
  – Thus the next segment may be lost

Fast Retransmit

• Treat three duplicate ACKs as a loss
  – Retransmit next expected segment
  – Some repetition allows for reordering, but still detects loss quickly

\[
\begin{array}{cccccccc}
\text{Ack} & 1 & 2 & 3 & 4 & 5 & 5 & 5 & 5
\end{array}
\]
Fast Retransmit (2)

- Third duplicate ACK, so send 14
- ACK jumps after loss is repaired

Data 14 was lost earlier, but got 15 to 20
Retransmission fills in the hole at 14

Fast Retransmit (3)

- It can repair single segment loss quickly, typically before a timeout
- However, we have quiet time at the sender/receiver while waiting for the ACK to jump
- And we still need to MD cwnd ...
Inferring Non-Loss from ACKs

- Duplicate ACKs also give us hints about what data has arrived
  - Each new duplicate ACK means that some new segment has arrived
  - It will be the segments after the loss
  - Thus advancing the sliding window will not increase the number of segments stored in the network

Fast Recovery

- First fast retransmit, and MD cwnd
- Then pretend further duplicate ACKs are the expected ACKs
  - Lets new segments be sent for ACKs
  - Reconcile views when the ACK jumps
Fast Recovery (2)

- Third duplicate ACK, so send 14
  - Set ssthresh, cwnd = cwnd/2
  - More ACKs advance window; may send segments before jump
  - Data 14 was lost earlier, but got 15 to 20
  - Retransmission fills in the hole at 14
  - Exit Fast Recovery

Fast Recovery (3)

- With fast retransmit, it repairs a single segment loss quickly and keeps the ACK clock running
  - This allows us to realize AIMD
    - No timeouts or slow-start after loss, just continue with a smaller cwnd
- TCP Reno combines slow-start, fast retransmit and fast recovery
  - Multiplicative Decrease is ½
TCP Reno, NewReno, and SACK

- Reno can repair one loss per RTT
  - Multiple losses cause a timeout
- NewReno further refines ACK heuristics
  - Repairs multiple losses without timeout
- SACK is a better idea
  - Receiver sends ACK ranges so sender can retransmit without guesswork
Introduction to Computer Networks

Explicit Congestion Notification
(§5.3.4, §6.5.10)

Congestion Avoidance vs. Control

• Classic TCP drives the network into congestion and then recovers
  – Needs to see loss to slow down
• Would be better to use the network but avoid congestion altogether!
  – Reduces loss and delay
• But how can we do this?
Feedback Signals

• Delay and router signals can let us avoid congestion

<table>
<thead>
<tr>
<th>Signal</th>
<th>Example Protocol</th>
<th>Pros / Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss</td>
<td>Classic TCP</td>
<td>Hard to get wrong</td>
</tr>
<tr>
<td></td>
<td>Cubic TCP (Linux)</td>
<td>Hear about congestion late</td>
</tr>
<tr>
<td>Packet delay</td>
<td>Compound TCP (Windows)</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Need to infer congestion</td>
</tr>
<tr>
<td>Router indication</td>
<td>TCPs with Explicit Congestion Notification</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Require router support</td>
</tr>
</tbody>
</table>

ECN (Explicit Congestion Notification)

• Router detects the onset of congestion via its queue
  — When congested, it marks affected packets (IP header)
ECN (2)

- Marked packets arrive at receiver; treated as loss
  - TCP receiver reliably informs TCP sender of the congestion

![Diagram of ECN](image)

ECN (3)

- Advantages:
  - Routers deliver clear signal to hosts
  - Congestion is detected early, no loss
  - No extra packets need to be sent

- Disadvantages:
  - Routers and hosts must be upgraded