TCP Congestion Control

Study-Ready Notes

Compiled by Andrew Photinakis

October 17, 2025

Contents

1 TCP AIMD (Additive Increase Multiplicative Decrease)

1.1 Basic Approach

- **Approach**: Senders increase sending rate until packet loss (congestion) occurs, then decrease on loss event
- **Additive Increase**: Increase sending rate by 1 MSS every RTT until loss detected
- **Multiplicative Decrease**: Cut sending rate in half at each loss event
- Creates characteristic **sawtooth pattern** - probing for available bandwidth

1.2 Multiplicative Decrease Details

- **TCP Reno**: Cut rate in half on loss detected by triple duplicate ACK
- **TCP Tahoe**: Cut rate to 1 MSS when loss detected by timeout

1.3 Why AIMD?

- Distributed, asynchronous algorithm proven to:
  - Optimize congested flow rates network-wide
  - Have desirable stability properties
  - Converge to fair allocation among competing flows

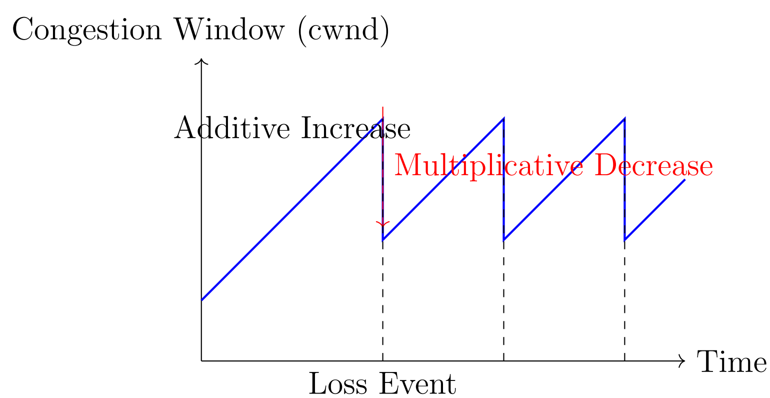

Figure 1: AIMD sawtooth behavior: probing for available bandwidth

[Summary: AIMD is TCP's core congestion control mechanism where senders gradually increase rates until congestion occurs, then dramatically reduce rates, creating a stable sawtooth pattern that fairly shares bandwidth.]

2 TCP Congestion Control Details

2.1 Congestion Window (cwnd)

- Dynamic limit on amount of unacknowledged data
- TCP sending rate approximately: Rate $\approx \frac{\text{cwnd}}{\text{RTT}}$ bytes/sec
- Sender constraint: $\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$

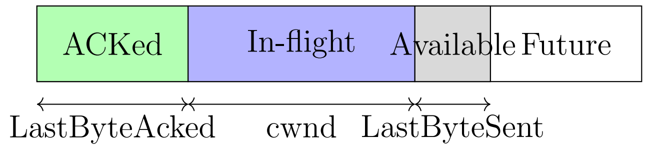

Figure 2: TCP sender sequence number space with congestion window

3 TCP Slow Start

3.1 Initial Phase

- When connection begins, increase rate exponentially until first loss
- Initial cwnd = 1 MSS
- Double cwnd every RTT
- Achieved by incrementing cwnd for every ACK received

3.2 Exponential Growth

$$\text{cwnd}_{\text{after RTT}} = \text{cwnd} \times 2$$
$$\text{cwnd}_{\text{after n RTTs}} = 2^n \times \text{MSS}$$

[Mnemonic: "Slow Start Speeds Up" - Despite the name, slow start actually increases the window exponentially fast, not slowly.]

4 TCP Congestion Avoidance

4.1 Transition from Slow Start

- Switch from exponential to linear increase when cwnd reaches half its pre-loss value
- **ssthresh** (slow start threshold) variable tracks this point
- On loss event: ssthresh $= \frac{\text{cwnd}}{2}$

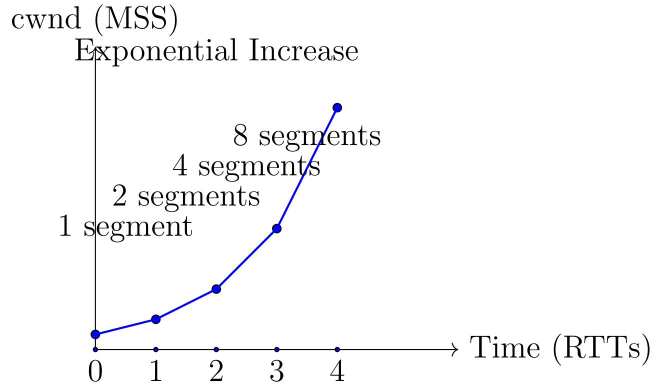

Figure 3: TCP slow start: exponential growth of congestion window

4.2 Congestion Control States

- **Slow Start**: Exponential increase until ssthresh
- **Congestion Avoidance**: Additive increase (AIMD)
- **Fast Recovery**: Handling duplicate ACKs

5 TCP Congestion Control Finite State Machine

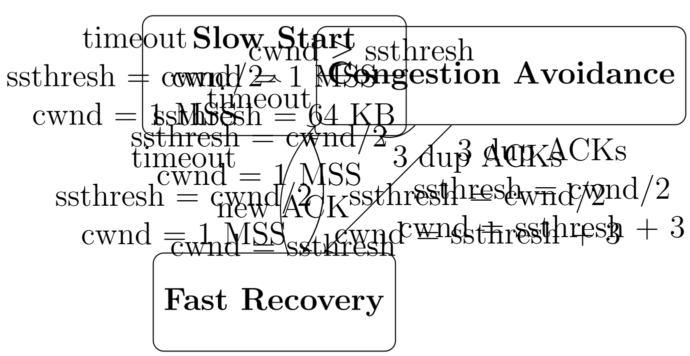

Figure 4: TCP congestion control finite state machine

6 TCP CUBIC

6.1 Motivation for Improvement

- Is there a better way than AIMD to "probe" for usable bandwidth?
- Key insight: After cutting rate on loss, initially ramp to $W_{\max}$ faster, then approach more slowly

- $W_{\max}$: Sending rate at which congestion loss was detected

6.2 CUBIC Algorithm

- K: Point in time when window will reach $W_{\max}$
- Increase window as cube of distance from current time to K
- Larger increases when further from K, smaller increases when nearer

$$W(t) = C(t - K)^3 + W_{\max}$$

Where C is a scaling constant.

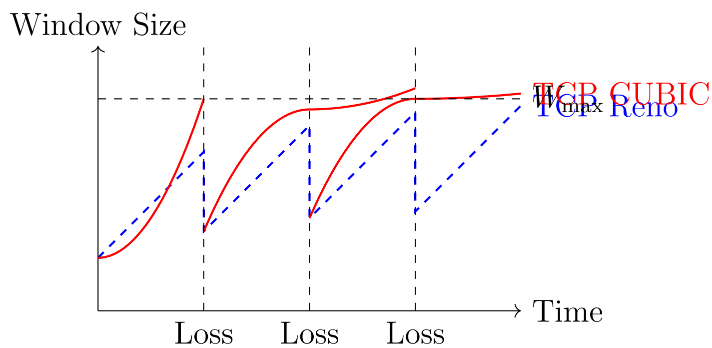

Figure 5: TCP CUBIC vs classic TCP: CUBIC achieves higher throughput

6.3 Deployment Status

- Default TCP in Linux
- Most popular TCP for popular web servers
- Provides better performance on high-bandwidth, high-latency networks

[Summary: TCP CUBIC improves upon AIMD by using a cubic growth function that initially recovers quickly after loss then approaches the previous maximum more cautiously, providing better performance on modern networks.]

7 Bottleneck Link Concept

7.1 Network Bottlenecks

- TCP increases sending rate until packet loss occurs at some router's output
- This congested router is the **bottleneck link**
- Understanding congestion requires focusing on bottleneck links

7.2 Key Insights

- Increasing TCP sending rate beyond bottleneck capacity won't increase end-to-end throughput
- Increasing TCP sending rate will increase measured RTT due to queueing
- Goal: "Keep the end-end pipe just full, but not fuller"

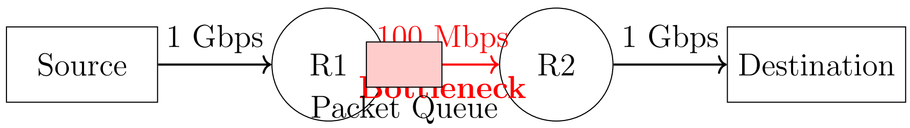

Figure 6: Bottleneck link concept: the slowest link determines maximum throughput

8 Delay-Based TCP Congestion Control

8.1 Philosophy

- Keep sender-to-receiver pipe "just full enough, but no fuller"
- Keep bottleneck link busy transmitting, but avoid high delays/buffering
- Congestion control without inducing/forcing loss

8.2 Algorithm Approach

- Measure RTT_{min}: Minimum observed RTT (uncongested path)
- Calculate uncongested throughput: $\frac{\text{cwnd}}{\text{RTT}_{\text{min}}}$
- Compare measured throughput with uncongested throughput:
  - If "very close": Increase cwnd linearly (path not congested)
  - If "far below": Decrease cwnd linearly (path congested)

8.3 Deployment Examples

- BBR (Bottleneck Bandwidth and Round-trip propagation time)
- Deployed on Google's internal backbone network
- Better performance for latency-sensitive applications

9 Explicit Congestion Notification (ECN)

9.1 Network-Assisted Approach

- Routers provide direct congestion feedback
- Two bits in IP header (ToS field) marked by network router
- Congestion indication carried to destination
- Destination sets ECE bit on ACK to notify sender

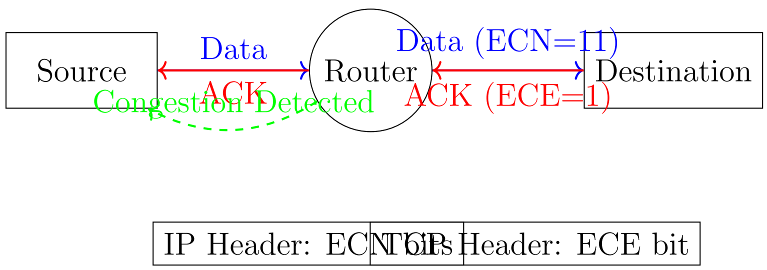

Figure 7: Explicit Congestion Notification (ECN) mechanism

10 TCP Fairness

10.1 Fairness Goal

- If K TCP sessions share bottleneck link of bandwidth R, each should have average rate of R/K
- TCP converges to fair allocation under idealized conditions

10.2 Fairness Analysis

- Additive increase gives slope of 1 as throughput increases
- Multiplicative decrease reduces throughput proportionally
- Result: Competing flows converge to equal shares

10.3 Limitations and Violations

- **UDP applications**: Multimedia apps don't use TCP congestion control
- **Parallel TCP connections**: Applications can open multiple connections
- Example: Link with rate R and 9 existing connections:

  - New app with 1 TCP gets rate R/10
  - New app with 11 TCPs gets rate R/2
- No "Internet police" enforcing congestion control use

Congestion Control Type	**Mechanism**	**Characteristics**
Classic TCP (Reno)	Loss-based AIMD	Sawtooth pattern, proven fairness
TCP CUBIC	Cubic growth function	Better high-speed performance
Delay-based (BBR)	RTT measurements	Lower latency, no forced loss
ECN-enabled	Network feedback	Early congestion notification

Table 1: Comparison of TCP congestion control variants

11 TCP Throughput Analysis

11.1 Average Throughput Formula

Ignoring slow start and assuming always data to send:

$$\text{Average TCP throughput} = \frac{3}{4} \cdot \frac{W}{\text{RTT}} \quad \text{bytes/sec}$$

Where:

- W: Window size where loss occurs
- Average window size is $\frac{3}{4}W$
- Average throughput is $\frac{3}{4}W$ per RTT

11.2 Throughput Example

For a connection with:

- Loss window $W = 10$ MSS
- RTT $= 100$ ms
- MSS $= 1460$ bytes

$$\text{Average throughput} = \frac{3}{4} \cdot \frac{10 \times 1460}{0.1} = 109,500 \text{ bytes/sec} \approx 876 \text{ kbps}$$

[Concept Map: TCP Congestion Control → AIMD (core algorithm) + Slow Start (initial phase) + Congestion Avoidance (steady state) + Enhancements (CUBIC, BBR, ECN) → Goals: Efficiency + Fairness + Stability → Prevents congestion collapse while maximizing network utilization.]

12 Study Aids and Exam Preparation

12.1 Key Concepts to Master

- Understand AIMD mechanism and why it creates sawtooth pattern
- Differentiate between slow start and congestion avoidance
- Explain TCP CUBIC improvements over classic TCP
- Compare loss-based vs delay-based congestion control
- Describe ECN mechanism and benefits
- Analyze TCP fairness and its limitations

12.2 Practice Questions

1. **Explain the AIMD mechanism** and draw the characteristic sawtooth pattern. Why does this pattern emerge, and what are its stability properties?
2. Compare **TCP slow start** and **congestion avoidance**. When does the transition occur, and what triggers it?
3. Calculate the **average TCP throughput** for a connection that experiences loss when cwnd = 16 MSS, with RTT = 50 ms and MSS = 1500 bytes.
4. Describe how **TCP CUBIC** improves upon classic TCP congestion control. What is the key insight behind its cubic growth function?
5. Explain the **fairness limitations** of TCP congestion control. How can applications "cheat" the system, and what are the implications?

[Mnemonic: "AIMD: Add Incrementally, Multiply Down" - Add 1 MSS per RTT when increasing, multiply by 0.5 when decreasing.]

13 Summary

- TCP congestion control prevents network collapse while maximizing utilization
- AIMD (Additive Increase Multiplicative Decrease) is the core algorithm
- Slow start provides exponential initial growth
- Congestion avoidance provides linear increase after threshold
- TCP CUBIC improves performance on high-speed networks
- Delay-based approaches (BBR) avoid forced packet loss
- ECN enables network-assisted congestion notification
- TCP achieves fairness under idealized conditions
- Various enhancements address limitations of classic approaches
- Throughput can be modeled mathematically based on loss window and RTT