Internet Congestion Control (1988-2024): A Primer

COS 597S: *Recent Advances in Wireless Networks*

Fall 2024

Kyle Jamieson

Transport Layer: Context & Motivation

Layer	Service	
Application Layer (L7)	Applications	
Transport Layer (L4)	Reliable streams	Messages
Network Layer (L3)	Best-effort *global* packet delivery	
Link Layer (L2)	Best-effort *local* packet delivery	

- Most applications want to exchange messages between different remote processes
- Further, many applications want a **reliable stream of bytes between different remote processes**

Transport Protocols

- Provide **logical communication** between **remote application processes**
  - Sender application divides a message into ***segments***
  - Receiver application **reassembles** segments into message
- Transport layer services
  - (De)multiplexing packets
  - Detecting corrupted data
  - **Optional:** reliable byte stream delivery, flow control, congestion avoidance...

Transmission Control Protocol (TCP)

- Reliable byte stream service
  - **all data** reach receiver: **in order** they were sent, with **no data corrupted**
- Reliable, in-order delivery
  - Corruption: checksums
  - Detect loss/reordering: sequence numbers
  - Reliable delivery: acknowledgments and retransmissions
- Connection oriented
  - Explicit set-up and tear-down of TCP connection
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for greater good

Fundamental Problem: Estimating RTT

- **Round-Trip Time (RTT):** end-to-end delay for data to reach receiver + ACK to reach sender, including:
  - propagation delay on links
  - serialization delay at each hop
  - queuing delay at routers

- **Design alternative:** use fixed timer (*e.g.*, 250 ms)
  - What if **the route changes?**
  - What if **congestion at one or more routers?**

TCP: Retransmit Timeouts

- Sender sets timer for each sent packet
  - when ACK returns, timer canceled
  - if timer expires before ACK returns, packet resent

- Expected time for ACK to return:
  - Round Trip Time (RTT)

- TCP estimates round-trip time using EWMA
  - ***measurements*** m_i from timed packet :: ACK pairs
  - $RTT_i = ((1-\alpha) \times RTT_{i-1} + \alpha \times m_i)$
  - Original TCP retransmit timeout:
    - $RTO_i = \beta \times RTT_i$ (original TCP: $\beta = 2$)

Mean and Variance: Jacobson's RTT Estimator

- Above link load of 30% at router, $\beta \times RTT_i$ will retransmit too early!
- Response to increasing load: waste bandwidth on duplicate packets
- Result: **congestion collapse!**

- [Jacobson 88]: estimate v_i, mean deviation (EWMA of $|m_i - RTT_i|$), stand-in for variance

  $$v_i = v_{i-1} \times (1-\gamma) + \gamma \times |m_i - RTT_i|$$

- **All modern TCPs: use $RTO_i = RTT_i + 4v_i$**

Connection Startup Behavior

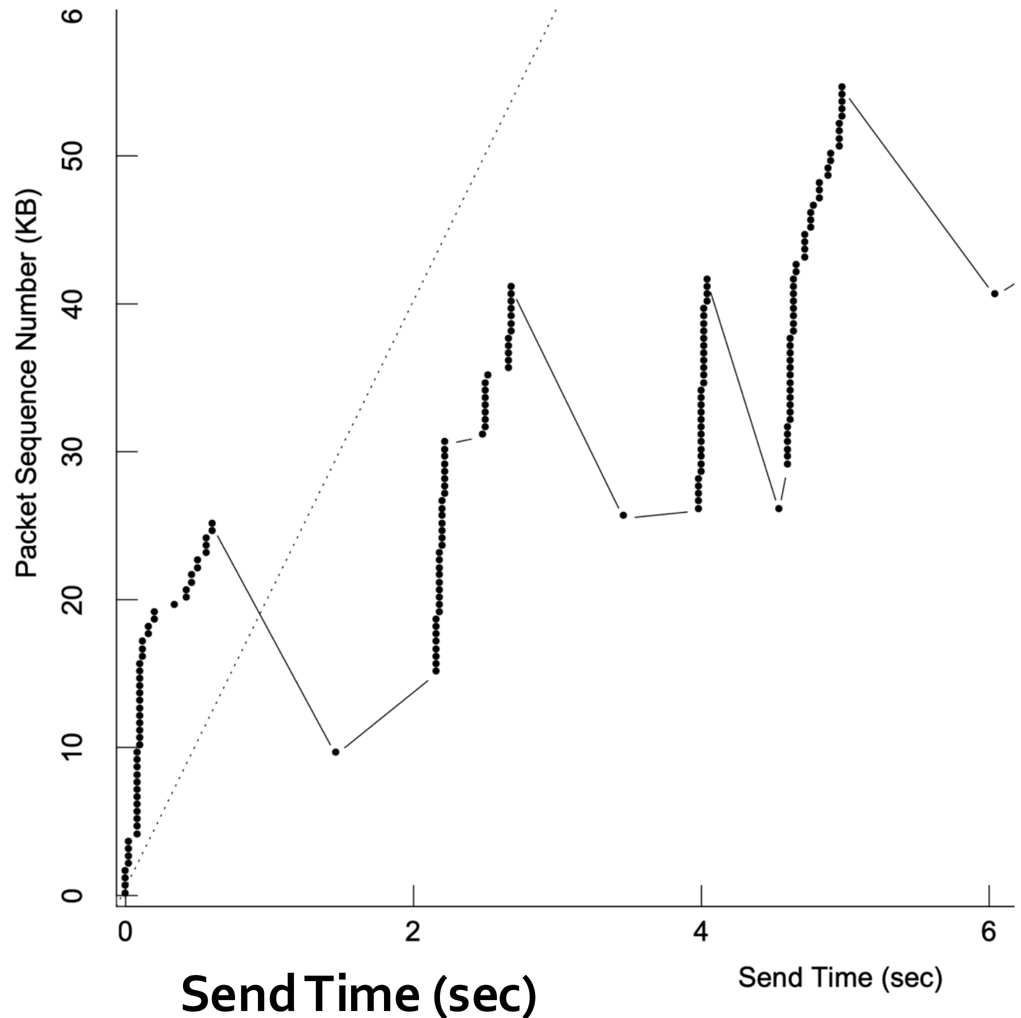
- TCP control of window size: *Slow Start*

- Original TCP, before [Jacobson 88]:
  - At connection start, send full window of packets
  - retransmit each packet just after timer expires

- **Result:** window-sized **packet bursts** sent into network

Pre-Jacobson TCP (Obsolete!)

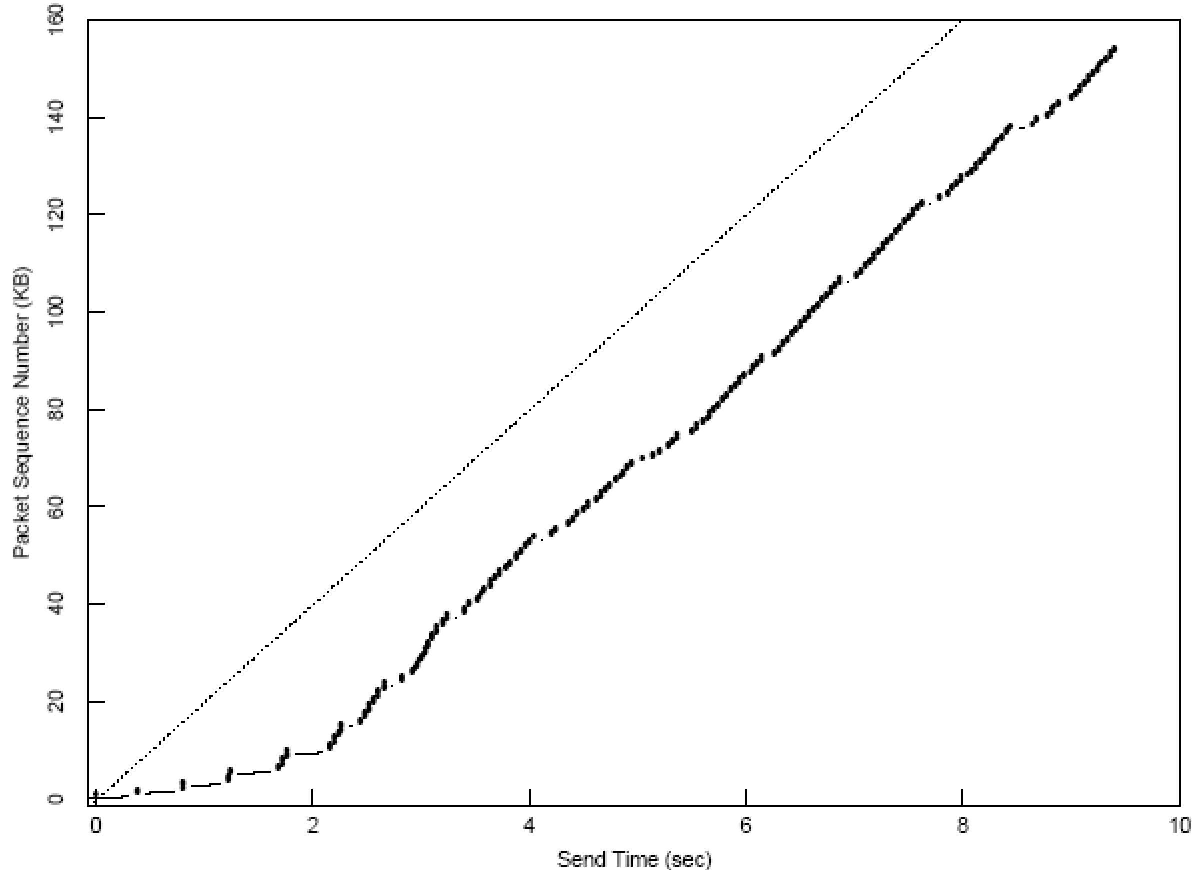
- Time-sequence plot taken at sender
- Bursts of packets: vertical lines
- Spurious retransmits: repeats at same y-value (enough buffer on path)
- Dashed line: available 20 Kbps capacity



Reaching Equilibrium: Slow Start

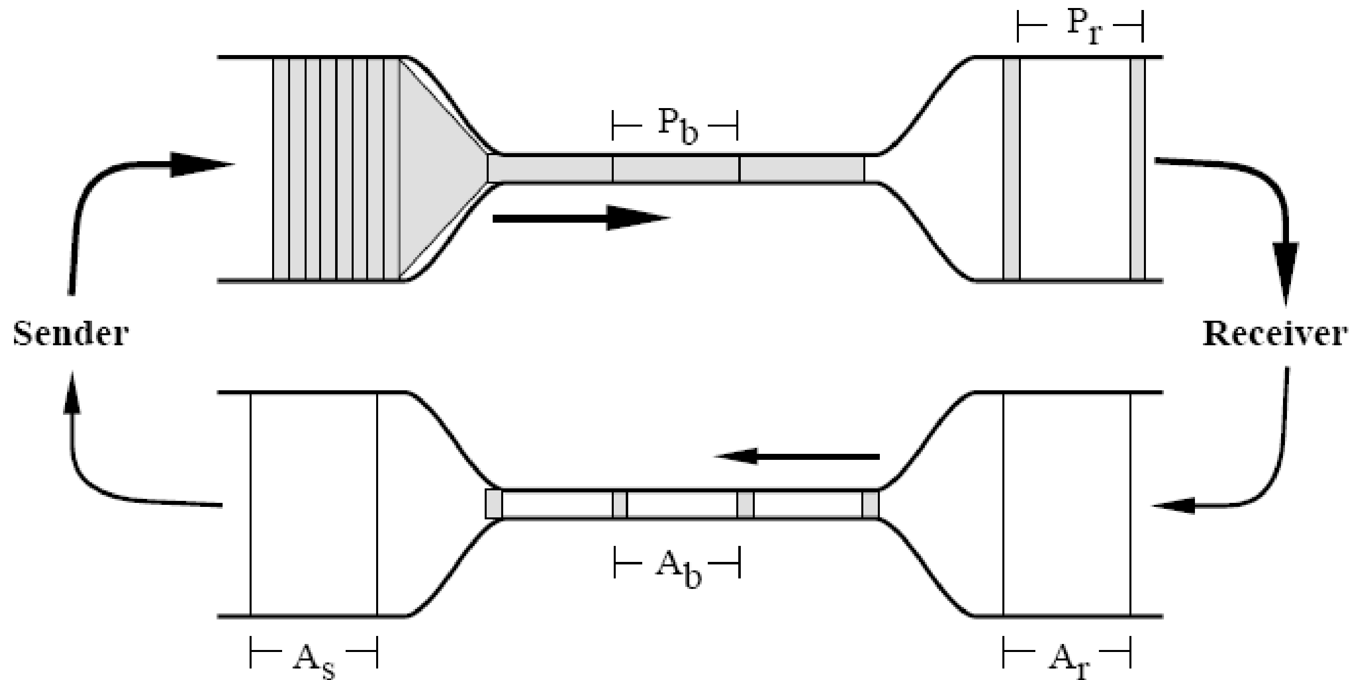
- At connection start: sender sets congestion window size, cwnd, to pktSize, not whole window
- Sender sends up to minimum of receiver's advertised window size W and cwnd
- Upon return of each ACK until receiver's advertised window size reached, increase cwnd by pktSize bytes
- "Slow" means exponential window increase!
  - Takes $\log_2(W/pktSize)$ RTTs to reach receiver's advertised window size W

Post-Jacobson TCP: Slow Start and Mean+Variance RTT Estimator



- Time-sequence plot at sender; dashed line = available capacity
- "Slower" start
- No spurious retransmits

Self-Clocking: Conservation of Packets



- **Goal: self-clocking transmission**
  - each ACK returns, one data packet sent
  - spacing of returning ACKs: matches spacing of packets in time at slowest link on path P_b

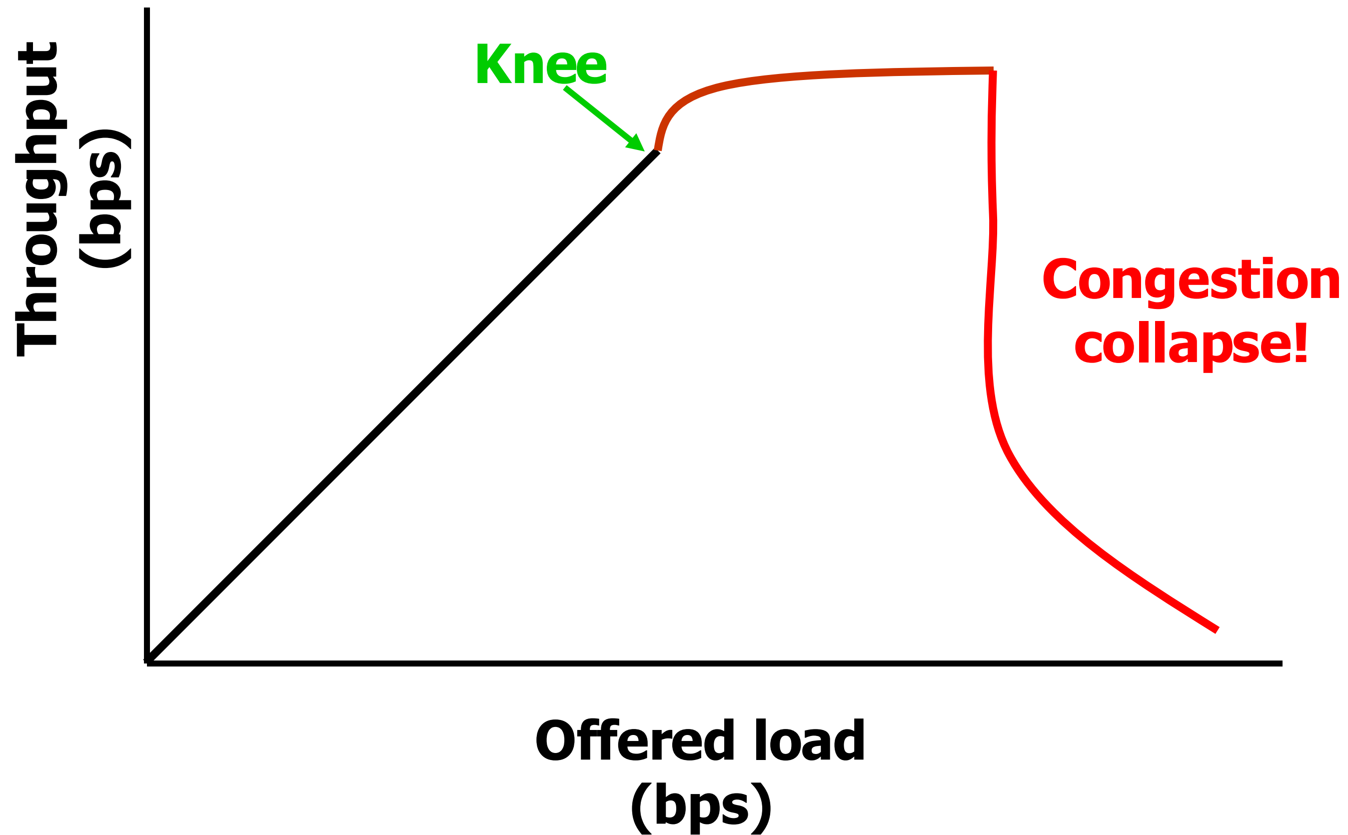
Today

- Pacing Transmissions
- Slow Start and Self-clocking
- **Congestion control**
- Learning to Share: Chiu-Jain phase plots
- Modeling Throughput

Goals in Congestion Control

- Achieve high link utilization; don't waste capacity!
- Divide bottleneck link capacity fairly among users
- Be stable: converge to steady allocation among users
- Avoid congestion collapse

Congestion Collapse



- Cliff behavior observed in [Jacobson 88]

Congestion Requires Slowing Senders

- Recall: big buffers can't prevent congestion collapse
  - Senders must **slow down** to alleviate congestion. How?
  - **Absence of ACKs** implicitly indicates congestion

- TCP sender's window size determines sending rate

- *How can sender learn the right cwnd?*
  - **Search** for it, by adapting window size
  - **Feedback** from network: ACKs return (window OK) or do not return (window too big)

Avoiding Congestion: Multiplicative Decrease

- Upon **timeout** for sent packet, sender presumes packet lost to congestion, and:
  - sets ssthresh = cwnd / 2
  - sets cwnd = pktSize
  - uses slow start to grow cwnd up to ssthresh
- End result: cwnd = cwnd / 2, via slow start
- Sender sends one window per RTT
  - Halving cwnd halves transmit rate

Avoiding Congestion: Additive Increase

- No feedback to indicate TCP using **less** than its fair share of bottleneck

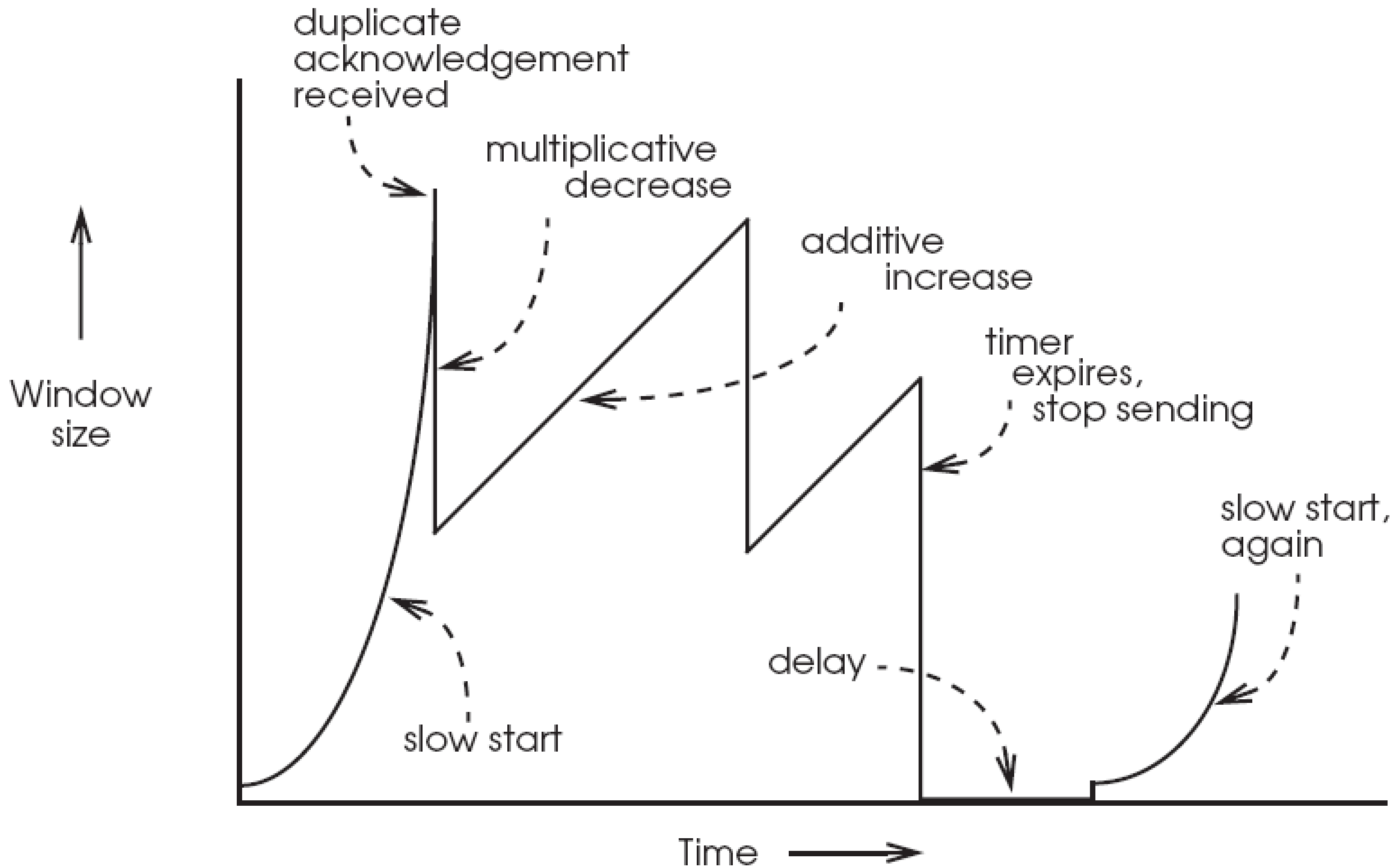
- Solution: speculatively increase window size as ACKs return
  - Additive increase: for each returning ACK,

    cwnd = cwnd + (pktSize × pktSize)/cwnd

    - Increases cwnd by ~pktSize bytes per RTT

Combined algorithm: Additive Increase, Multiplicative Decrease (AIMD)

AIMD in Action



- Sender **searches** for correct window size

Why AIMD?

- Other control rules possible
  - E.g., MIMD, AIAD, …
- Recall goals:
  - Links fully utilized (efficient)
  - Users share resources fairly
- TCP adapts all flows' window sizes independently
- Must choose a control that will always converge to an efficient and fair allocation of windows

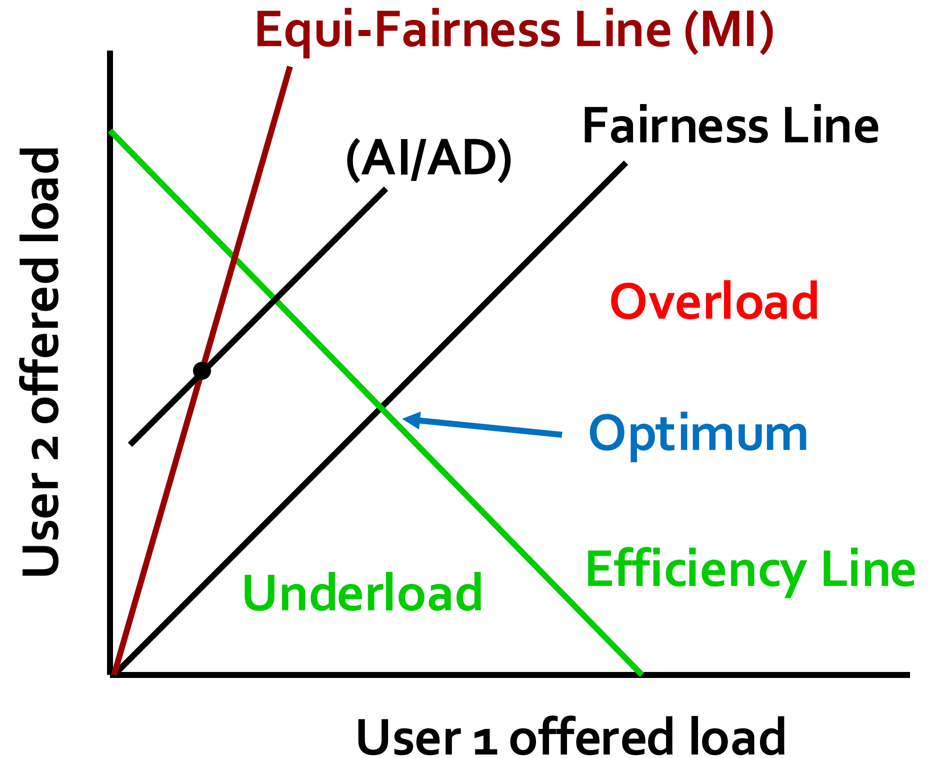
Chiu-Jain Phase Plots

- Consider two users sharing a bottleneck link
  - Plot bandwidths allocated to each

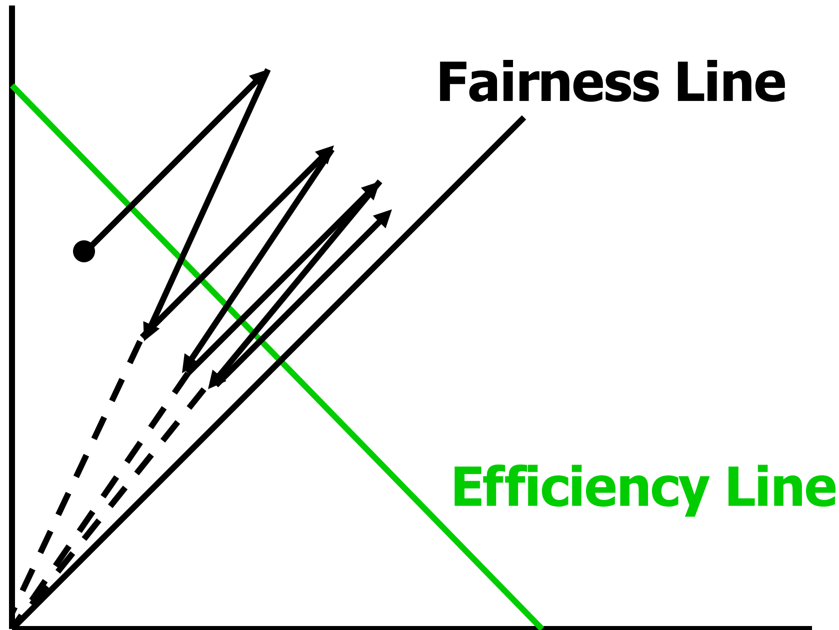
- **Efficiency Line:** sum of two users' rates = bottleneck capacity

- **Fairness Line:** two users' rates equal

- **Equi-Fairness Line:** ratio of two users' rates fixed

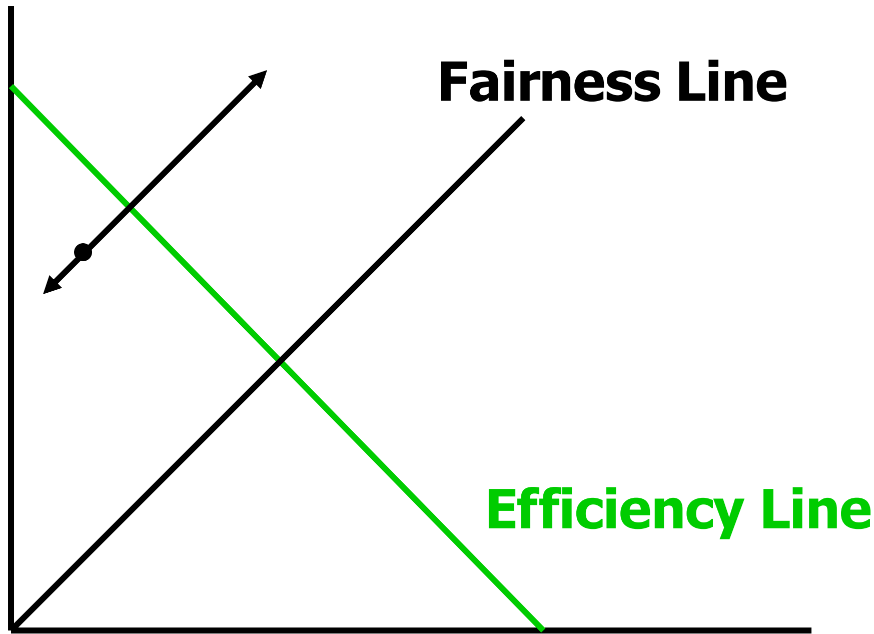


Chiu Jain: AIMD



- AIMD converges to optimum efficiency and fairness

Chiu Jain: AIAD



- AIAD doesn’t converge to optimum point!
- Similar oscillations for MIMD

Summary: TCP and Congestion Control

- Connection establishment and teardown
  - Robustness against delayed packets crucial
- Round-trip time estimation
  - EWMAs estimate both RTT mean and deviation
- Congestion detection at sender
  - Timeout: half window, slow start from one packet
  - Fast retx: three dup ACKs, half window, no slow start
- Search for optimal sending window size
  - Additive increase, multiplicative decrease (AIMD)
  - AIMD converges to high utilization, fair sharing

High Bandwidth-Delay Product

- Key Problem: TCP performs poorly when
  - The capacity of the network (bandwidth) is large
  - The delay (RTT) of the network is large
  - Or, when bandwidth * delay is large
    - b * d = maximum amount of in-flight data in the network
    - a.k.a. the bandwidth-delay product
- Why does TCP perform poorly?
  - Slow start and additive increase are slow to converge
  - TCP is ACK clocked
    - i.e. TCP can only react as quickly as ACKs are received
    - Large RTT → ACKs are delayed → TCP is slow to react

TCP CUBIC Implementation

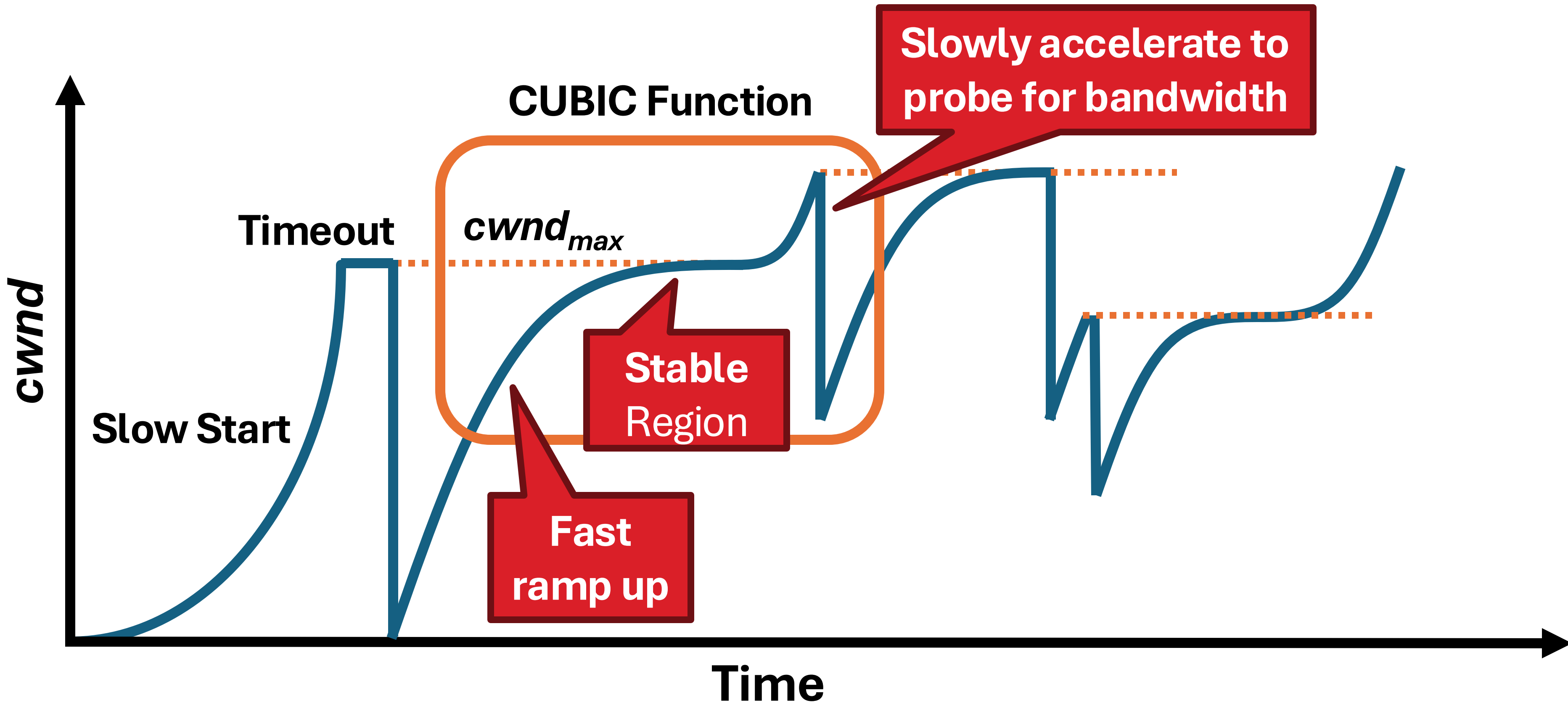
- Default TCP implementation in Linux
- Replace AIMD with cubic function

$$W_{cubic} = C(T - K)^3 + W_{max} \qquad (1)$$

C is a scaling constant, and $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$

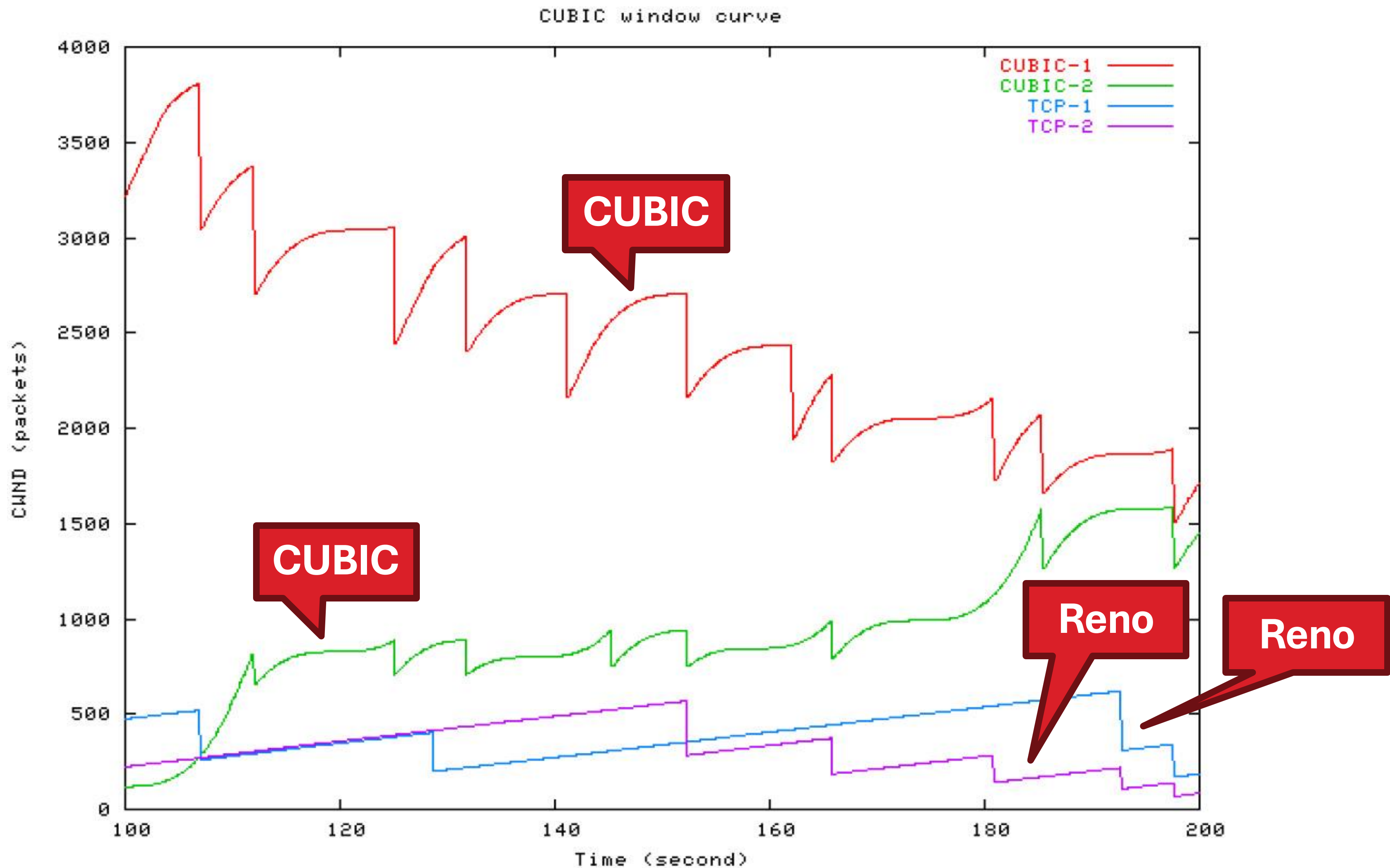
  – B → a constant fraction for multiplicative increase
  – T → time since last packet drop
  – W_max ➔ cwnd when last packet dropped

TCP CUBIC Example



- Less wasted bandwidth due to fast ramp up
- Stable region and slow acceleration help maintain fairness
  - Fast ramp up is more aggressive than additive increase
  - To be fair to Tahoe/Reno, CUBIC needs to be less aggressive

Simulations of CUBIC Flows



Recent BBR Performance Studies

- Available in Zotero ("*Weeks 3-5 Wireless Cognizant CC"* Folder)
- D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, **"Promises and Potential of BBRv3,"** in *Passive and Active Measurement*, vol. 14538, P. Richter, V. Bajpai, and E. Carisimo, Eds., in Lecture Notes in Computer Science, vol. 14538. , Cham: Springer Nature Switzerland, 2024, pp. 249–272. doi: 10.1007/978-3-031-56252-5_12.
- Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, **"When to use and when not to use BBR: An empirical analysis and evaluation study,"** in *Proceedings of the Internet Measurement Conference*, Amsterdam Netherlands: ACM, Oct. 2019, pp. 130–136. doi: 10.1145/3355369.3355579.
- R. Drucker, G. Baraskar, A. Balasubramanian, and A. Gandhi, **"BBR vs. BBRv2: A Performance Evaluation,"** in *2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, Bengaluru, India: IEEE, Jan. 2024, pp. 379–387. doi: 10.1109/COMSNETS59351.2024.10427175.
- S. Vargas, G. Gunapati, A. Gandhi, and A. Balasubramanian, **"Are mobiles ready for BBR?,"** in *Proceedings of the 22nd ACM Internet Measurement Conference*, in IMC '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 551–559. doi: 10.1145/3517745.3561438.