# Problem 7: Transmission Control Protocol

### (i) Describe the congestion control algorithms for (a) Cubic TCP (b) Compound TCP.

## What is Congestion?

Load on the network is higher than the capacity

- Capacity is not uniform across networks
    - Modem vs. Cellular vs. Cable vs. Fiber Optics
- There are multiple flows competing for bandwidth
    - Residential cable modem vs. corporate datacenter
- The load is not uniform over time
    - 10 pm, Saturday night = Movie on Netflix

## Goals of Congestion Control:

- Adjusting to the bottleneck bandwidth
- Adjusting to variations in bandwidth
- Sharing bandwidth between flows
- Maximizing throughput

## Idea:

- TCP introduces the concept of “windows” to establish traffic flow control and manage connections between two devices: a sender and a receiver.
- **Idea: vary the window size to control the send rate**
- Sending rate is ~ window/RTT  (where RTT is Round Trip Time)
- Congestion control is a sender-side problem, so introduce a congestion window at the sender

## Implementing Congestion Control:

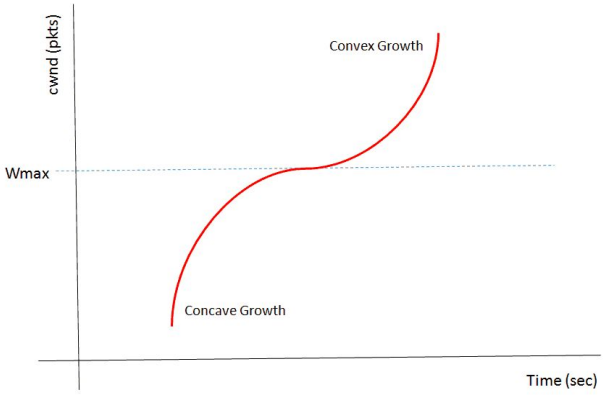
- Maintains three variables:
    - **cwnd**:  congestion window (limits how much data is in transit)
    - **adv_wnd**: receiver advertised window
    - **ssthresh**:  threshold size (used to update cwnd)
- For sending, use: **wnd = min(cwnd, adv_wnd)**
- Two phases of congestion control
    - Slow start (cwnd < ssthresh)
        - Probe for bottleneck bandwidth
    - Congestion avoidance (cwnd >= ssthresh)
        - AIMD

## Popular TCP Congestion Control Algorithms:

- TCP performs poorly on high bandwidth-delay product networks (like the modern Internet)
- **CUBIC TCP (Linux)**
  - Enhancement of BIC (Binary Increase Congestion Control)
  - Window size controlled by a cubic function
  - Parameterized by the time T since the last dropped packet
- **Compound TCP (Windows)**
  - Based on Reno
  - Uses two congestion windows: delay-based and loss based
  - Thus, it uses a compound congestion controller

# Cubic TCP

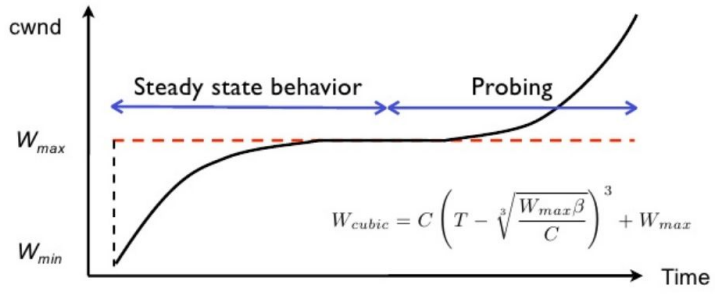
- CUBIC is a congestion control protocol for TCP (transmission control protocol) and the current default TCP algorithm in **Linux**.
- TCP CUBIC arises with the idea of taking advantage of the fact that today's communications links tend to have increasingly **higher bandwidth levels**. In a network composed of wide bandwidth links, a congestion control algorithm that **slowly** increases the transmission rate may end up wasting the capacity of the links.
- The protocol modifies the linear window growth function of existing TCP standards to be a **cubic function** in order to improve the scalability of TCP over fast and long-distance networks.
- It can achieve high bandwidth connections over networks **more quickly** and reliably in the face of high latency than earlier algorithms.




- During steady-state, **CUBIC increases the window size aggressively when the window is far from the saturation point, and slowly when it is close to the saturation point.** This feature allows CUBIC to be very scalable when the bandwidth and delay product of the network is large, and at the same time, be highly stable and also fair to standard TCP flows.

## CUBIC TCP Implementation:

- At the time of experiencing congestion events, the window size for that instant will be recorded as **Wmax** or the maximum window size.
- The Wmax value will be set as the **inflection point** of the cubic function that will govern the growth of the congestion window.




The following formula determines the congestion window size (cwnd):
cwnd = $C(T - K)^3 + Wmax$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

where

- C - scaling constant
- T - the time since the last loss event
- β - multiplicative decrease factor after a loss event
- Wmax - congestion window (cwnd) before a loss event

- The transmission will then be restarted with a smaller window value and, if no congestion is experienced, this value will increase according to the **concave portion** of the cubic function.
- As the window approaches Wmax the increments will **slow down**.
- Once the tipping point has been reached, i.e. Wmax, the value of the window will continue to increase discreetly.
- Finally, if the network is still not experiencing any congestion, the window size will continue to increase according to the **convex portion** of the function.

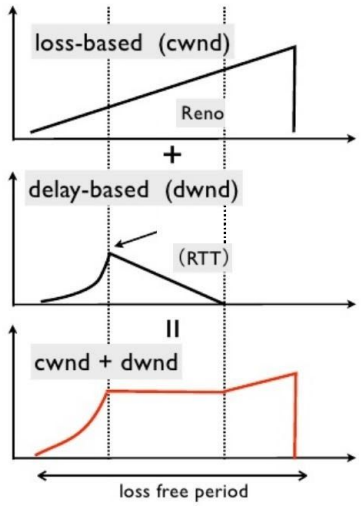
As we can see, CUBIC implements schemes of large increments at first, which decrease around the window size that caused the last congestion, and then continue to increase with large increments.

# Compound TCP

- Compound TCP(CTCP) is a Microsoft implementation of TCP which maintains **two different congestion windows simultaneously**. It was introduced as part of the Windows Vista and Windows Server 2008 TCP stack.
- It is designed to aggressively adjust the sender's congestion window to optimize TCP for connections with large bandwidth-delay products while trying **not to harm fairness**.
- CTCP incorporates a scalable **delay-based component** into the standard TCP congestion avoidance algorithm.
- This scalable delay-based component has a **rapid window increase rule** when the network is sensed to be under-utilized and gracefully reduces the sending rate once the bottleneck queue is built.
- CTCP has good TCP-fairness. It gracefully reduces the sending rate when the link is fully utilized.

## Compound TCP Implementation:

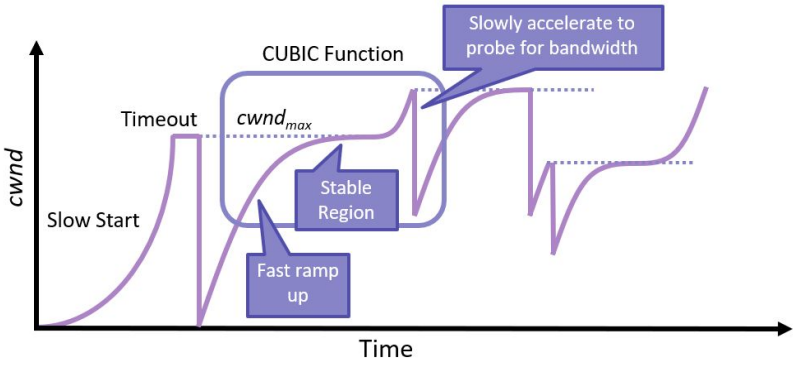
- Maintains two windows: cwnd and dwnd.
- win = min ( cwnd + dwnd, awnd)




- **cwnd:** conventional congestion window, controls the loss-based component.
- **dwnd:** new variable, controls the delay-based component.
- **awnd:** advertised window from the receiver.
- RTT: Round Trip Time
- Rules for adjusting **dwnd**:
    1. If RTT is increasing, decrease dwnd (dwnd >= 0)
    2. If RTT is decreasing, increase dwnd
    3. Increase/decrease are proportional to the rate of change

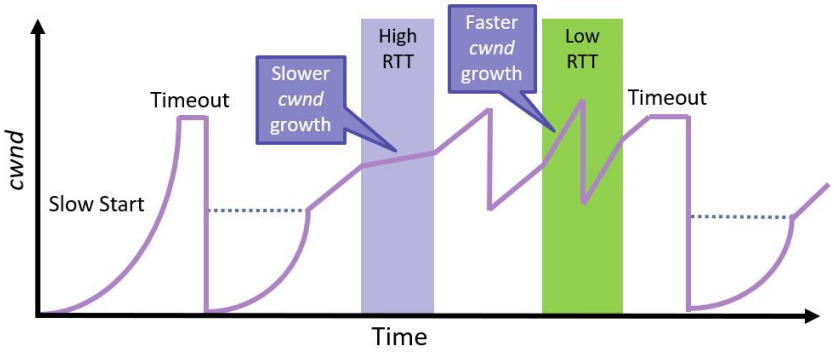
# CUBIC vs Compound TCP

## 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

## Compound TCP Example




- Aggressiveness corresponds to changes in RTT
- Advantages: fast ramp up, more fair to flows with different RTTs
- Disadvantage: must estimate RTT, which is very challenging

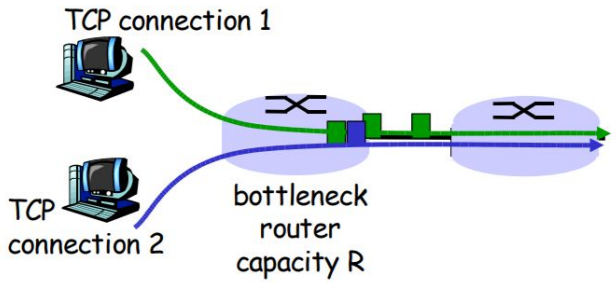
**(ii) What does it mean for a TCP to be fair? And how might one evaluate fairness when TCP operates over a large scale network, like the Internet?**

## Fairness:

- Fairness metrics are used in network engineering to determine whether users or applications are receiving a fair share of system resources.

- TCP fairness requires that a new protocol receive a no larger share of the network than a comparable TCP flow. This is important as TCP is the dominant transport protocol on the Internet, and if new protocols acquire unfair capacity they tend to cause problems such as congestion collapse.

- If K TCP sessions share the same bottleneck link of bandwidth R, then each should have an average rate of R/K




## Problem:

- TCP throughput depends on RTT
- ACK clocking makes TCP inherently unfair

## Possible solution:

- Maintain a separate delay window
    - Implemented by Microsoft's Compound TCP

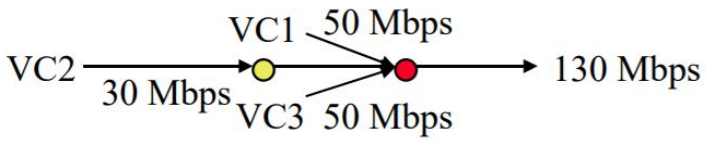
# Quantitative Fairness Measure:

## Jain's Index:

Four desired properties of the Jain's index (or simply fairness index) are:

- **Independent of population size:**
  The index should be scalable with the number of users
- **Independent of scale and metric:**
  The index should not change with measures or metrics used. This property implies that variance can also be a fairness index
- **Boundedness:**
  The index should be finite and it can be a ratio between 0 and 1
- **Continuity:**
  The index function should be continuous on allocations, and it should have the ability to measure different allocations.

**Example:**
A scheme gives 50, 30, 50 Mbps when the optimal is 50, 10, 10 Mbps. How fair is it?




Measured Throughput: (T1 , T2 , ..., Tn )
Use any criterion (e.g., max-min optimality) to find the Fair Throughput (O1 , O2 , ..., On)
Normalized Throughput: xi = Ti /Oi

$$\text{Fairness Index} = \frac{(\Sigma x_i)^2}{n\Sigma x_i^2}$$

**Example**: 50/50, 30/10, 50/10 $\Rightarrow$ 1, 3, 5

$$\text{Fairness Index} = \frac{(1+3+5)^2}{3(1^2+3^2+5^2)} = \frac{9^2}{3(1+9+25)} = 0.81$$

Index=0.81 means 81% of the users are treated fairly and 20% are starved.

# Qualitative Fairness Measure:

Qualitative fairness measures are not able to provide a measurement of fairness with a real number representation however, they can judge whether the allocations achieve fairness. One of the representative measures is max-min fairness.

## Max-min:

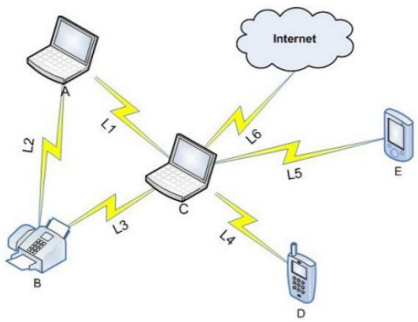
- Max-min fairness is said to be achieved by an allocation if and only if the allocation is feasible and an attempt to increase the allocation of any flow necessarily results in the decrease in the allocation of some other flow with an equal or smaller allocation.

    - A feasible allocation of resource x to n users is max-min fair if for each user i, $x_i$ cannot be increased (while maintaining the feasibility) without decreasing $x_j$ where $x_j \leq x_i$ , $(i \neq j)$.
    - In other words, a system reaches max-min fairness, if it cannot increase an individual's resource without decreasing another individual's resource allocation which is already **less than** the previous ones.

- Max-min fairness (or bottleneck optimality) has been studied widely and implemented in many applications, such as flow control, bandwidth sharing, radio channel accessing, etc

- A max-min fair allocation is achieved only when **bandwidth is allocated equally** i.e. in a system with perfect max-min fairness, based on the above definition, every individual gets exactly the same amount of resource. Explanation:

    - For example in the figure




      if nodes do not get the equal network capacity such as 10%, 20%, 30%, 30% and 10% for A, B, C, D and E, respectively, it is not max-min fair, because we can increase A's capacity (10%) by decreasing B's (20%) which is not **less than** A's.

    - On the other hand, if all nodes obtain an equal amount of network capacity (20%), then none of the nodes can increase its capacity without decreasing the capacity of other nodes which is no more than its own capacity.