HO CHI MINH CITY UNIVERSITY OF SCIENCE
FACULTY OF ELECTRONICS AND TELECOMMUNICATIONS
DEPARTMENT OF TELECOMMUNICATIONS AND NETWORKS

COURSE

COMPUTER NETWORKS

Chapter 08

TCP Reno and Congestion Management

Reference: Peter L Dordal, "An Introduction to Computer Networks," Feb 05, 2022

Lecturer: Nguyen Viet Ha, Ph.D. Email: nvha@fetel.hcmus.edu.vn

1

Basics of TCP Congestion Management

1. Basics of TCP Congestion Management

❖TCP's congestion management is **window-based**; that is, TCP adjusts its window size to adapt to congestion.

- ➢**Window size *(winsize)*** can be thought of as the number of packets out there in the network.
- ➢Or representing how many packet buffers it could allocate. (via ***Advertised*** *Window Size*)

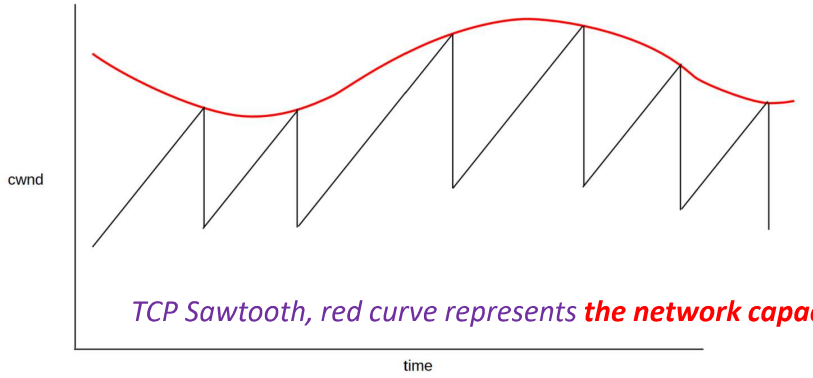
❖When *winsize* is adjusted downwards for this reason, it is generally referred to as the Congestion Window, or ***cwnd***.

winsize = min(cwnd, AdvertisedWindow)

1. Basics of TCP Congestion Management

❖***Additive Increase, Multiplicative Decrease (AIMD)***

- ➢If there were no losses, **cwnd = cwnd +1**.
- ➢If packets were lost, **cwnd = cwnd/2**.



TCP Sawtooth, red curve represents ***the network capacity***

1. Basics of TCP Congestion Management

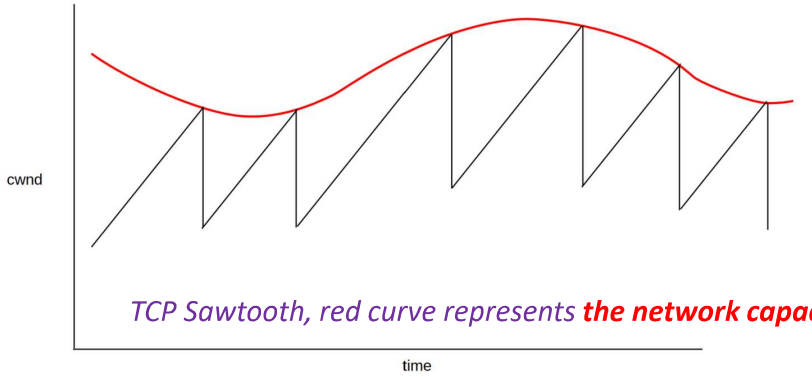
❖ ***Additive Increa***

➢ If there were n

➢ If packets were

❖ How to guess the **network capacity**?

❖ What value of ***cwnd*** should begin with?



TCP Sawtooth, red curve represents ***the network capacity***

2 Slow start

2. Slow start

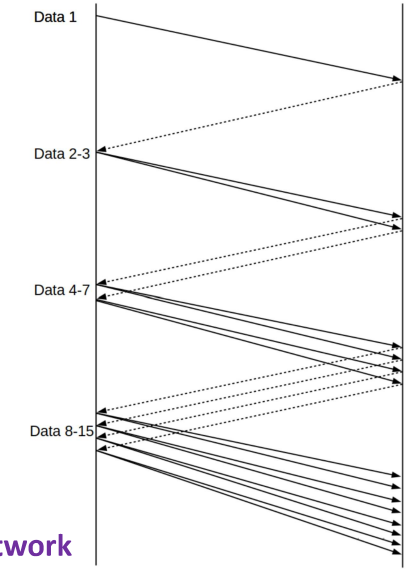
❖ A good strategy is to initially set ***cwnd=1***.

❖ Keep doubling until exceeding the network capacity.

➢ ***cwnd=cwnd x 2*** after each ***windowful***

○ *Or* ***cwnd+=1*** after each *packet*.

❖ Try increasing *cwnd* slowly from the previous *cwnd* (***$cwnd_{old}/2$***)



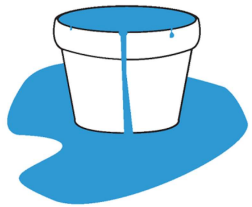
Slow Start with discrete packet flights

When is the network capacity exceeded?

2. Slow start

❖ Eventually the bottleneck queue gets full

➔ **drops a packet**.



❖ **Dropping event** will be determined at source based on:

➢ **Packet loss detected** or

➢ **TCP timeout**

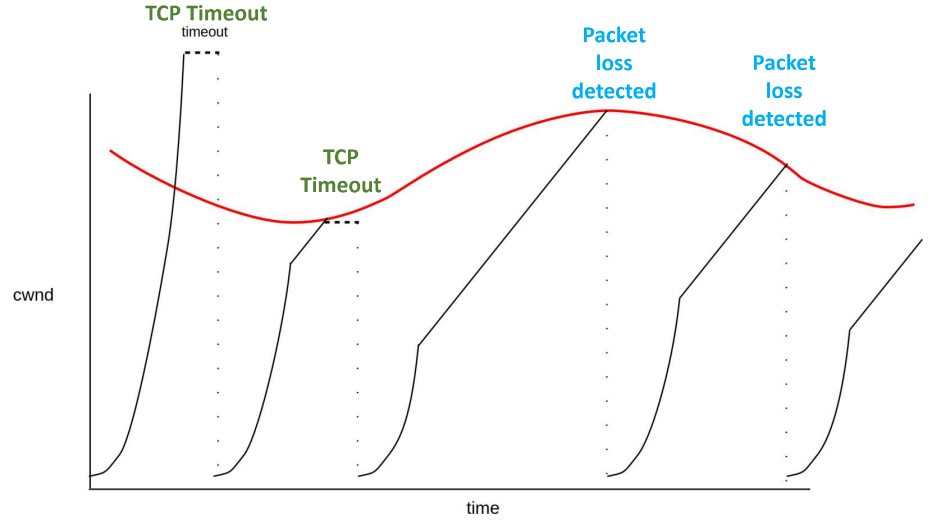
2. Slow start

❖In **TCP Tahoe** (**packet loss detected** or **TCP timeout**)

- ***cwnd = 1***
- **Threshold slow start** (***ssthresh***) = $cwnd_{old}/2$
- Enter to **slow start phase** ***(cwnd+=1)***

- If ***cwnd > ssthresh***
  - Enter the **congestion-avoidance phase**.
  - ***cwnd+=1*** after each **windowful**
  - *Or* ***cwnd = cwnd + 1/cwnd$_0$*** after each *packet*.
    - where ***cwnd$_0$*** is the value of ***cwnd*** at the start of that particular windowful.
  - Use ***floor(cwnd)*** when actually sending packets.

2. Slow start



TCP Tahoe Sawtooth, red curve represents the ***network capacity***

Slow Start is used after each packet loss until ***ssthresh*** is reached

2. Slow start

❖In **TCP Reno** (Case 1: If **packet loss detected**)

- $cwnd_{new} = cwnd_{old}/2$
- **Threshold slow start** (***ssthresh***) = $cwnd_{old}/2$

- If ***cwnd > ssthresh***
  - Enter the **congestion-avoidance phase**.
  - ***cwnd+=1*** after each **windowful**
  - *Or* ***cwnd = cwnd + 1/cwnd$_0$*** after each *packet*.
    - where ***cwnd$_0$*** is the value of ***cwnd*** at the start of that particular windowful.
  - Use ***floor(cwnd)*** when actually sending packets.

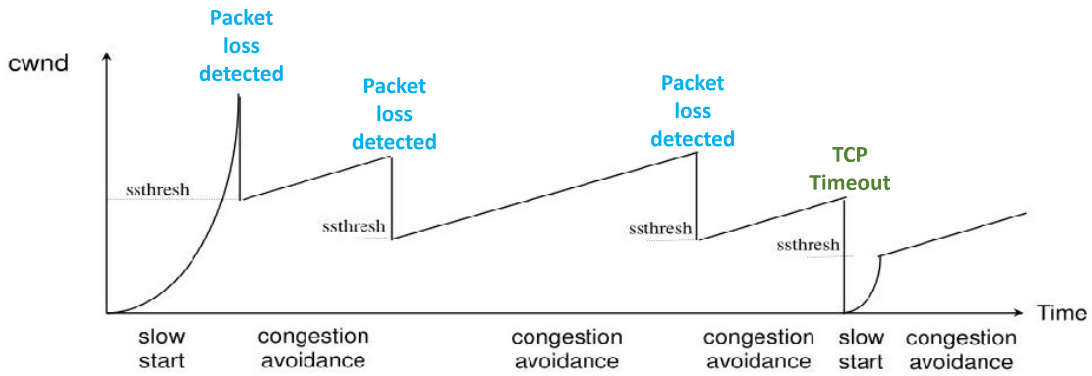
2. Slow start

❖In **TCP Reno** (Case 2: If **TCP timeout**)

- ***cwnd = 1***
- **Threshold slow start** (***ssthresh***) = $cwnd_{old}/2$
- Enter to **slow start phase** ***(cwnd+=1)***

- If ***cwnd > ssthresh***
  - Enter the **congestion-avoidance phase**.
  - ***cwnd+=1*** after each **windowful**
  - *Or* ***cwnd = cwnd + 1/cwnd$_0$*** after each *packet*.
    - where ***cwnd$_0$*** is the value of ***cwnd*** at the start of that particular windowful.
  - Use ***floor(cwnd)*** when actually sending packets.

2. Slow start

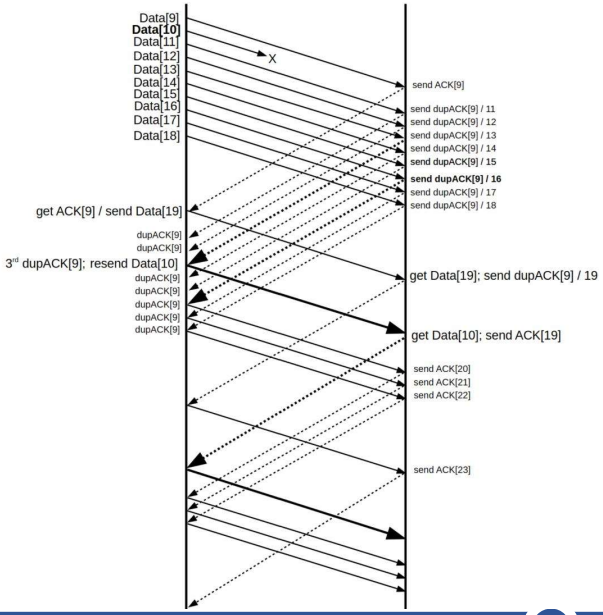


TCP Reno Sawtooth

3 Fast Retransmit

3. Fast Retransmit

❖Resend Data[N] when we have received **three dupACKs** for Data[N-1]; that is, four ACK[N-1]'s in all. (called **packet loss detected** in previous slide)



3. Fast Retransmit

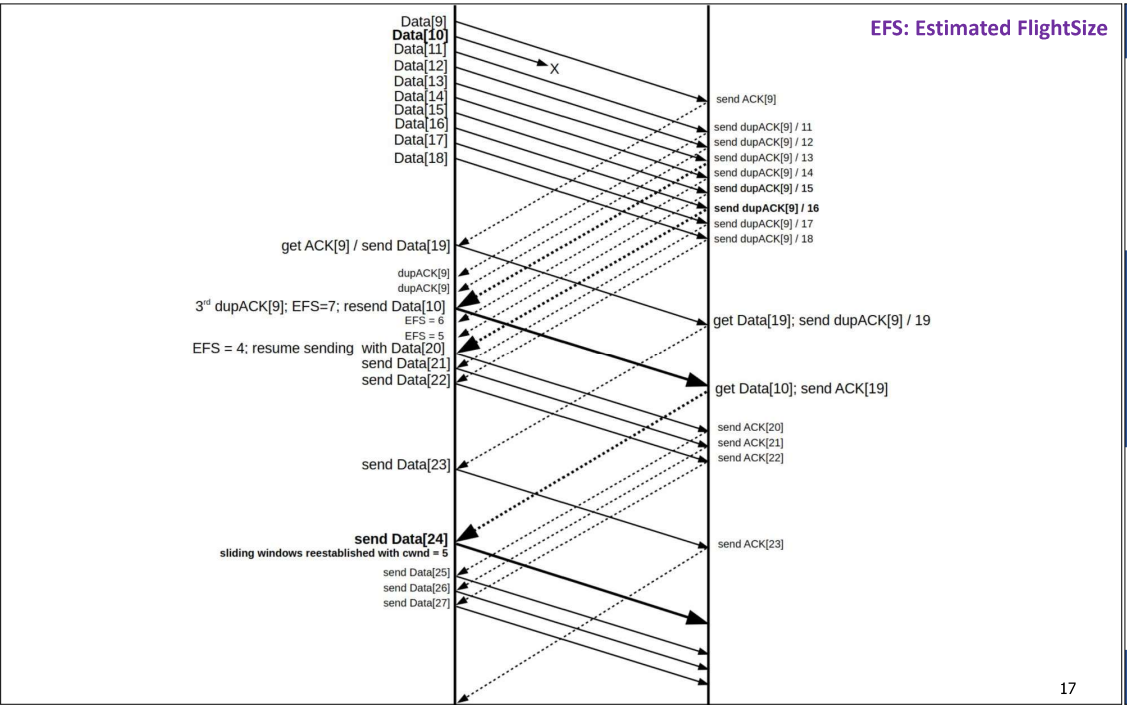
❖In **TCP Tahoe**, packet loss → ***cwnd=1***

➢*Increase **cwnd** only if receiving the ACK for the retransmission.*

❖**BUT,** in **TCP Reno**, packet loss → $cwnd_{new} = cwnd_{old}/2$

➢How **many dupACKs** we have to wait for before we can **resume transmissions of new data?**

➢**Estimated FlightSize**, or **EFS**, which is the sender's best guess at the number of outstanding packets (Packets on the Flight).



4

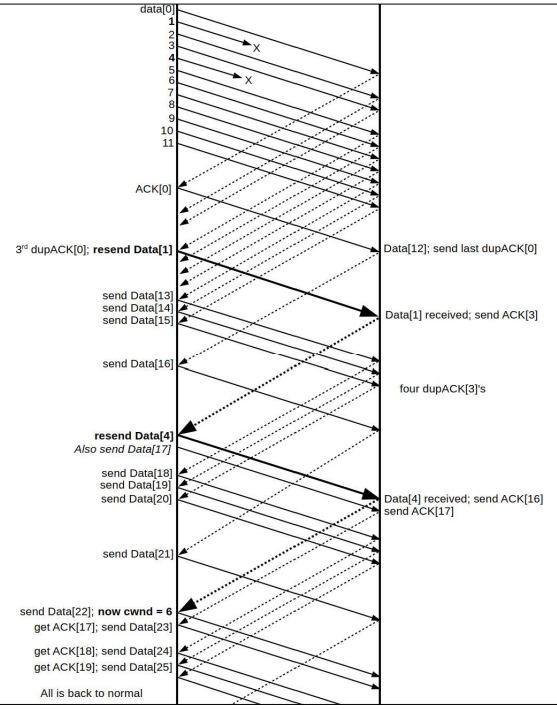
TCP NewReno

4. TCP NewReno

- Improves handling of the case when **two or more packets are lost** in a windowful.

- **Partial ACKs.**
  - If two (or more) data packets are lost and the first is retransmitted, the receiver will acknowledge data up to just before the second packet, and then continue sending **dupACKs** of this until the second lost packet is also retransmitted.

  - Because retransmission of the first lost packet did not result in an ACK of all the outstanding data.
    - These partial ACKs as evidence to retransmit later lost packets, and also to keep pacing the **Fast Recovery process**.



5

Selective Acknowledgments (SACK)

5. Selective Acknowledgments (SACK)

❖A traditional TCP ACK is a cumulative acknowledgment of all data received up to that point.

➢Only use **Triple duplicated ACKs** to detect ONE the packet loss at a time.

➢Ex:

- Data[1002] is received. Data[1001] is lost
- The receiver sends the duplicate ACK[1000].
  - This does indicate that *something* following Data[1001] made it through, **but nothing more.**

5. Selective Acknowledgments (SACK)

❖**Selective ACK** (SACK) option (implemented at the receiver).

➢If this is available, the sender does not have to **guess** from *dupACKs* what has gotten through.

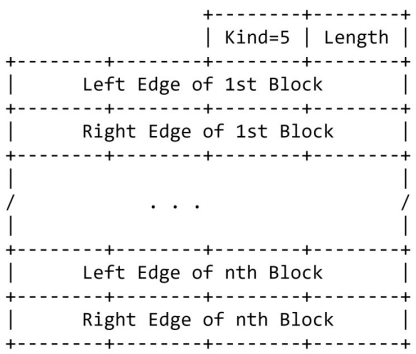
➢The receiver can send an ACK that says (example):

- All packets up through 1000 have been received (the ***cumulative ACK***)
- All packets up through 1050 have been received *except for* 1001, 1022, and 1035. (the ***Selective ACK***)

❖Almost all TCP implementations now support SACK.

➢Use the TCP Option field.

5. Selective Acknowledgments (SACK)



❖**Left Edge of Block:** the first sequence number of this block.

❖**Right Edge of Block:** the sequence number immediately following the last sequence number of this block.

❖ A SACK option that specifies n blocks will have a length of ***8*n+2*** bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks.

➢3 losses event (including burstiness loss)

5. Selective Acknowledgments (SACK)

❖In practice,

➢**Selective ACKs** provide at best a modest performance improvement in many situations.

➢**TCP NewReno** does rather well, in moderate-loss environments.

QA

Lecturer: Nguyen Viet Ha, Ph.D.
The University of Science, Vietnam National University, Ho Chi Minh City
Faculty of Electronics and Communications
Department of Telecommunication and Networks
Email: nvha@fetel.hcmus.edu.vn