TCP Congestion Control

- When we discussed flow control in TCP, we mentioned that the ***size of the send window is controlled by the receiver using the value of rwnd***, which is advertised in each segment traveling in the opposite direction. The use of this strategy guarantees that the receive window is never overflowed with the received bytes (no congestion at end systems).
- This, however, does not mean that the intermediate buffers, buffers in the routers, do not become congested.
- TCP needs to worry about congestion in the middle because many segments lost may seriously affect the error control. More segment loss means resending the same segments again, resulting in worsening the congestion, and finally the collapse of the communication.
- TCP is an end-to-end protocol that uses the services of IP Protocol of Network layer. The congestion in the router is in the IP territory and should be taken care by IP. But we will discuss in the next unit that IP does not provide any congestion control mechanism.

CONGESTION WINDOW

- Since IP does not handle congestion at router, TCP itself has to take the responsibility and send the data segments at appropriate rate so that network does not become congested while utilizing the network bandwidth efficiently.
- *TCP needs to define policies that accelerate the data transmission when there is no congestion and decelerate the transmission when congestion is detected.*
- To control the number of segments to transmit, TCP uses another variable called a **congestion window, *cwnd*,** whose size is controlled by the congestion situation in the network.
- The ***cwnd*** variable and the ***rwnd*** variable together define the size of the send window in TCP. The first is related to the congestion in the middle (network); the second is related to the congestion at the end.

Actual send window size = minimum (rwnd, cwnd)

Congestion Detection

- How does TCP sender detect the possible existence of congestion in the network?
- The TCP sender uses the occurrence of two events as signs of congestion in the network: *time-out* and *receiving three duplicate ACKs*.
- **Time-Out Event**
  - If the sender doesn't get an acknowledgment (ACK) for a segment within a certain time (timeout), it assumes that segment got lost due to congestion.
  - This is a sign of strong congestion. It means the network is struggling and TCP needs to be careful.

Congestion Detection

- **Three Duplicate ACKs Event**
  - If the sender receives three duplicate ACKs (four ACKs with the same acknowledgment number), it means that one segment is missing, but the receiver got the other three segments which were sent later by sender.
  - This could indicate a less severe congestion.
  - This could indicate that network is either slightly congested or has recovered from the severe congestion.
- Earlier versions of TCP, called *Taho TCP*, treated both events similarly. But the newer version of TCP, called *Reno TCP*, treats these two signs differently based on their severity.
- A very interesting point in TCP congestion is that the TCP sender uses only one feedback from the other end to detect congestion: ACKs. The lack of regular, timely receipt of ACKs, which results in a time-out, is the sign of a strong congestion; the receiving of three duplicate ACKs is the sign of a weak congestion in the network.

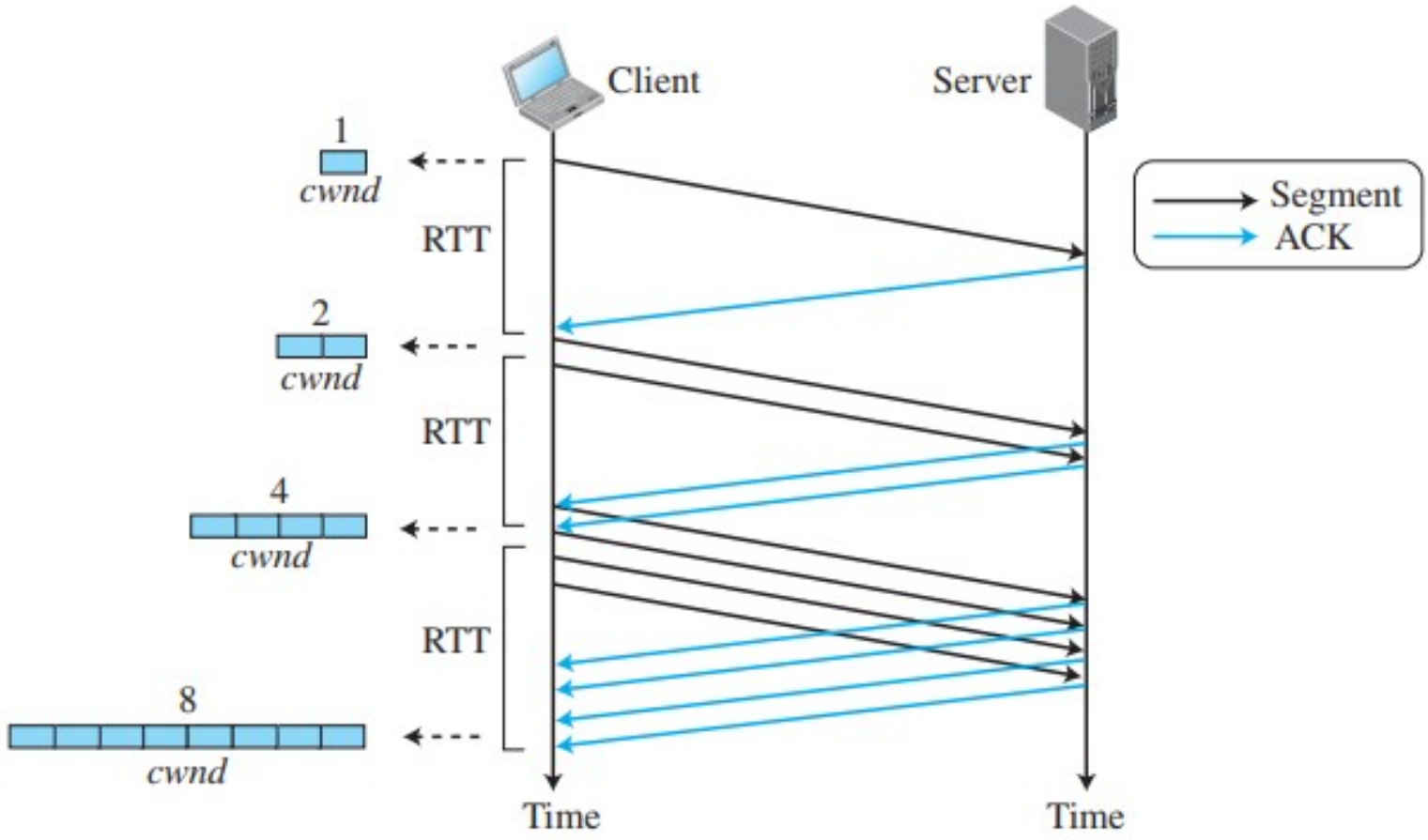
Congestion Policies

- TCP's general policy for handling congestion is based on three algorithms -
  - slow start
  - congestion avoidance
  - fast recovery.

SLOW START: EXPONENTIAL INCREASE

- The slow-start algorithm is based on the idea that the size of the congestion window (cwnd) starts with one *maximum segment size (MSS)*, but it increases one MSS each time one acknowledgment arrives.
- The Maximum Segment Size (MSS) is a parameter of the *Options* field of the TCP header that specifies the largest amount of data (in bytes) that can be sent in a single TCP Segment. It only counts data not headers.
- The MSS is decided during the connection establishment phase in TCP and can't be changed later till the end of connection.
- **Starting Slowly:**
  - At begining, you can only send one segment (a chunk of data).
  - Each time you get an acknowledgment (ACK) that your sent data was received, you can send double the amount next time.
  - So, it starts slow but grows pretty fast.

SLOW START: EXPONENTIAL INCREASE



Slow Start: Exponential Increase

- **Window Size Calculation:**
  - The congestion window (cwnd) is like the number of segments you can send.
  - If an ACK arrives, cwnd increases by 1.
  - So, if you get more ACKs, you can send more data.
- **Exponential Growth:**
  - The cwnd grows exponentially with each round-trip time (RTT).
  - It's a bit like a very eager approach—growing really quickly.
- **Stopping Slow Start:**
  - It can’t go on foreever, though. There is a limit called slow-start threshold (ssthresh).
  - When the window size reaches this threshold, slow start stops amd a new phase begins.

Slow Start: Exponential Increase

- **Consideration for Delayed Acknowledgements:**
  - If acknowledgments are delayed, the growth is a bit slower.
  - For every ACK, cwnd increases by only 1, even if multiple segments are acknowledged together.
  - So, the growth is still fast but not as fast as in ideal conditions.

If an ACK arrives, cwnd = cwnd + 1.

SLOW START: EXPONENTIAL INCREASE

- **In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.**

Start	$\rightarrow$	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	$\rightarrow$	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

Congestion Avoidance: Additive Increase

After the initial slow start, TCP has another trick called "congestion avoidance" to prevent things from getting out of control.

1. Congestion Avoidance:
- Instead of growing really fast (exponentially), TCP wants to be more careful.
- It uses an algorithm called congestion avoidance to increase the congestion window (cwnd) more steadily.

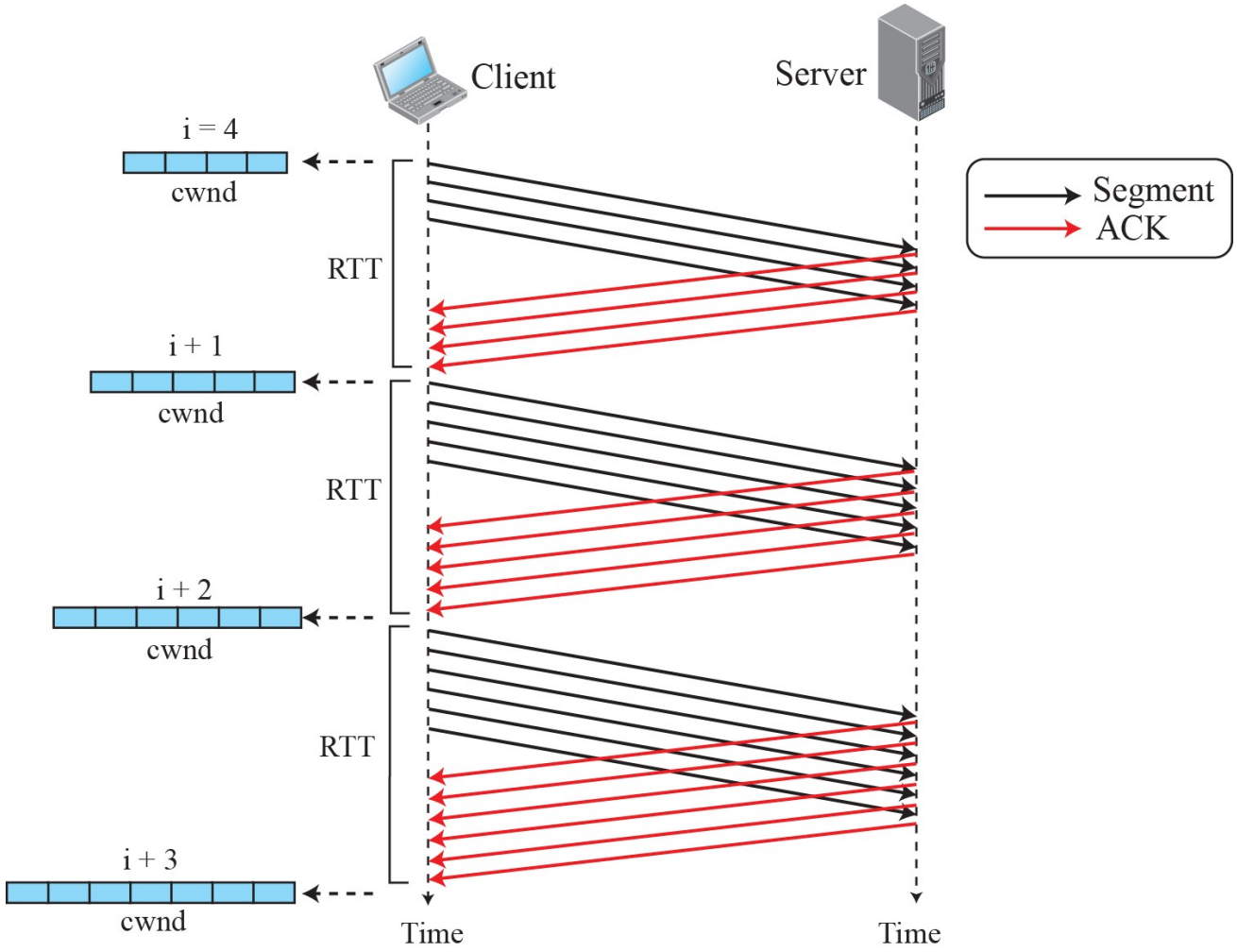
2. How It Works:
- After the slow start, when cwnd reaches a certain point (slow-start threshold), it switches to the congestion avoidance phase.
- **In this phase, each time a whole "window" of segments is acknowledged, cwnd increases by one.**
- A window is the number of segments sent during one round-trip time (RTT).

Congestion Avoidance: Additive Increase

3. Example:
- Let's say the sender starts with cwnd = 4, meaning it can send four segments initially.
- After getting four acknowledgments, it can send one more segment, and the window becomes 5.
- The process continues, increasing the window each time the whole window is acknowledged.

4. Math Behind It:
- The size of the window increases by a fraction of the maximum segment size (MSS), specifically 1/cwnd.
- **This means that all segments in the previous window must be acknowledged to increase the window by one MSS.**

5. Growth Rate:
- If you look at the size of cwnd in terms of round-trip times (RTTs), **the growth is more steady and linear compared to the earlier slow start.**


Client
Server
Segment
ACK
i = 4
cwnd
RTT
i + 1
cwnd
RTT
i + 2
cwnd
RTT
i + 3
cwnd
Time
Time

Congestion Avoidance: Additive Increase

Start	→	$cwnd = i$
After 1 RTT	→	$cwnd = i + 1$
After 2 RTT	→	$cwnd = i + 2$
After 3 RTT	→	$cwnd = i + 3$

In the congestion-avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

If an ACK arrives, cwnd = cwnd + (1/cwnd).

FAST RECOVERY

- The fast-recovery algorithm is optional in TCP. The old version of TCP did not use it, but the new versions try to use it.
- It starts when three duplicate ACKs arrives that is interpreted as light congestion in the network.
- Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm).

If a duplicate ACK arrives, $cwnd = cwnd + (1 / cwnd)$.