Lecture 6.2

Transport Layer: Congestion Control

Dr. Vandana Kushwaha

Department of Computer Science

Institute of Science, BHU, Varanasi

Network Congestion

- **Network congestion** is one of the **serious issues** in **communication network**.
- There are several **side effects** of **network congestion** which **severely affects** the **network performance** and **quality of service.**
- **Congestion** is an **important issue** in a **packet-switched network.**
- **Congestion** in a **network** may occur if the **load on the network** (*the number of packets sent to the network*) **is greater than the *capacity* of the network** (*the number of packets a network can handle*).
- **Congestion control** refers to the **mechanisms** and **techniques** to **control the congestion** and **keep** the **load below** the **capacity.**
- **Congestion** in a **network** or internetwork occurs because **routers** and **switches have queues- buffers** that hold the packets before and after processing.

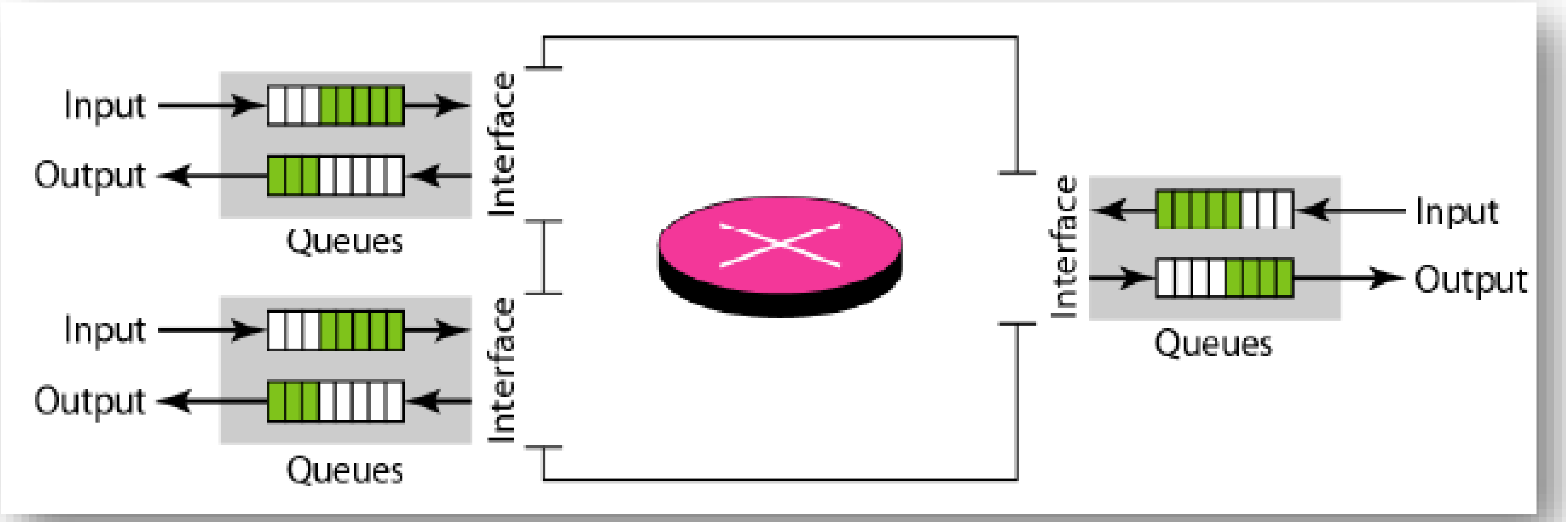
Network Congestion

- A **router,** for **example**, has an **input queue** and an **output queue** for **each interface.** When a packet arrives at the **incoming interface**, it undergoes **three steps** before departing.

  1. The **packet** is put at the end of the **input queue** while waiting to be checked.
  2. The **processing module** of the router removes the packet from the input queue once it reaches the front of the queue and uses its **routing table** and the **destination address** to find the **route.**
  3. The **packet** is put in the appropriate **output queue** and **waits its tum** to be **sent.**

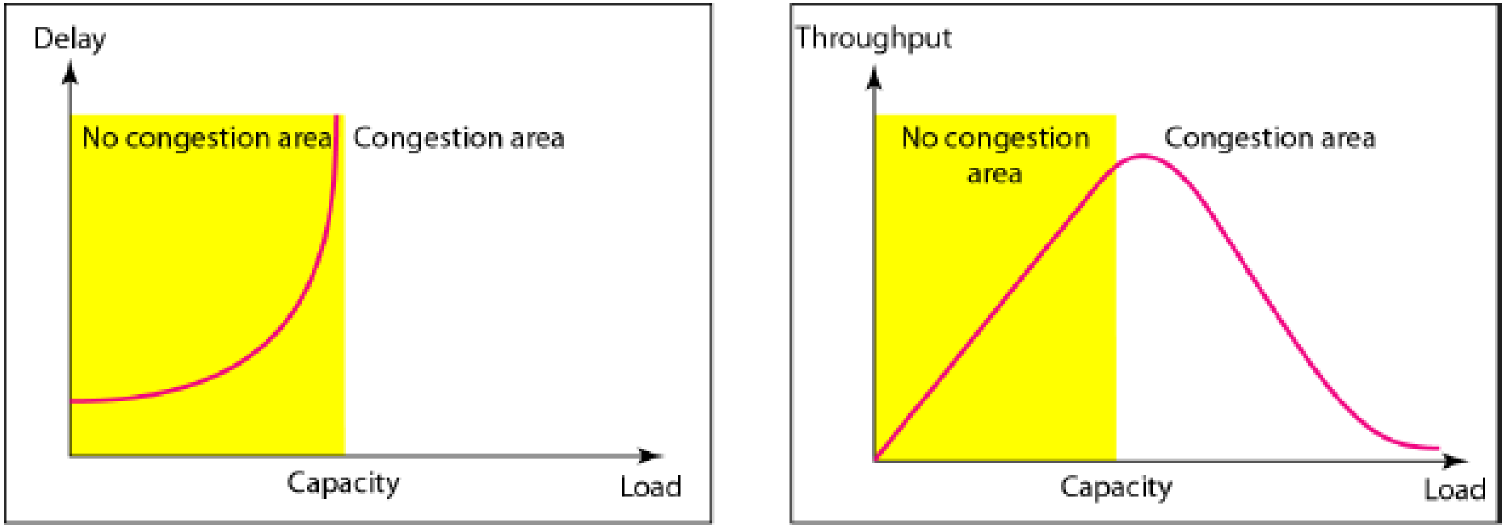
- If the **packet arrival rate** is **higher than the packet processing rate**, the **input queues** become **longer** and **longer.**
- If the **packet departure rate** is **less than the packet processing rate**, the **output queues** become **longer** and **longer.**

Queues in a Router



Network Performance

- **Congestion control** involves **two factors** that measure the **performance** of a **network**: **delay** and **throughput.**
- **Figure** below shows these two **performance measures** as **function** of **load.**



a. Delay as a function of load

b. Throughput as a function of load

Network Performance: Delay versus Load

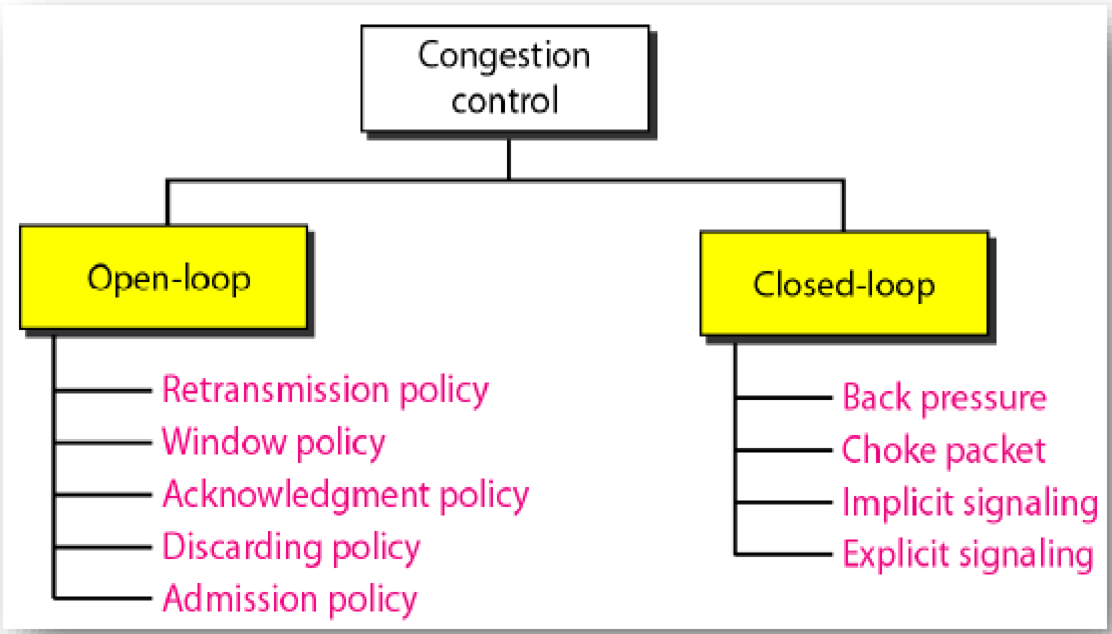
- When the **load** is much **less than** the **network capacity,** the **delay** is at a **minimum.**
- This **minimum delay** is composed of **propagation delay** and **processing delay**, both of which are **negligible.**
- However, when the **load reaches** the **network capacity**, the **delay increases sharply** because we now need to add the **waiting time in** the **queues** (for all routers in the path) to the **total delay.**
- The **delay becomes infinite** when the **load is greater than the capacity**.
- Due to **infinite delay**; the **queues become longer and longer.**
- **Delay** has a **negative effect on the load** and consequently the **congestion occurs**.
- When a **packet is delayed**, the **source**, **not receiving** the **acknowledgment, retransmits the packet**, which makes the delay, and the **congestion**, **worse.**

Network Performance: Throughput versus Load

- **Throughput** in a **network** is **defined** as **the number of packets passing through the network in a unit of time.**
- When the **load is below the network capacity**, the **throughput increases proportionally with the *load* .**
- We expect the **throughput** to remain constant after **the load reaches the capacity**, but instead the **throughput declines sharply**. The reason is the **discarding of packets by the routers.**
- When the **load exceeds the capacity**, the **queues become full** and the **routers** have to **discard some packets.**
- **Discarding packets** does **not reduce the number of packets** in the **network** because the **sources retransmit the packets**, using **time-out mechanisms**, when the packets do not reach the **destinations.**

Congestion Control

- **Congestion control** refers to **techniques and mechanisms** that can **either prevent congestion, before it happens, or remove congestion, after it has happened**.
- In general, we can divide **congestion control mechanisms** into **two broad categories**:
  - ***Open-loop congestion control*** (**prevention**)
  - ***Closed-loop congestion control*** (**removal**)



Open-Loop Congestion Control

- In **Open-loop congestion control**, policies are applied to ***prevent congestion before it happens***.
- In these mechanisms, **congestion control** is **handled** by either the **source** or the **destination.**

Open-Loop Congestion Control

1. Retransmission Policy

- If the **sender** feels that a **sent packet** is **lost** or **corrupted**, the packet needs to be **retransmitted.**
- **Retransmission** in general **may increase congestion** in the network as it **increases** the **network load.**
- The **retransmission policy** and the **retransmission timers** must be designed to **optimize efficiency (minimize load)** and at the same time **prevent congestion**.
- For **example**, the **retransmission policy** used by **TCP** is designed to prevent or alleviate congestion.

Open-Loop Congestion Control

2. Window Policy

- The **type of window** at the **sender** may also **affect congestion**.
- The **Selective Repeat window** is **better than the Go-Back-N window** for **congestion control.**
- In the ***Go-Back-N* window**, when the timer for a packet times out, **several packets may be resent**, although some may have arrived safe and sound at the receiver.
- This **duplication** may make the **congestion worse**.
- The **Selective Repeat window**, on the other hand, tries to **send** the **specific packets** that have been **lost or corrupted.**

Open-Loop Congestion Control

3. Acknowledgment Policy

- The **acknowledgment policy** imposed by the receiver may also **affect congestion**.
- If the **receiver** does **not acknowledge every packet** it receives, it may **slow down** the **sender** and help **prevent congestion.**
- A **receiver** may decide to **acknowledge** only ***N* packets at a time** which is called **cumulative acknowledgement policy**.
- We need to know that the **acknowledgments** are also part of the **load** in a **network.**
- **Sending fewer acknowledgments** means imposing **fewer loads on the network**.

Open-Loop Congestion Control

4. Discarding Policy

- A **good discarding policy** by the **routers** may **prevent congestion** and at the same time **may not harm the integrity of the transmission**.
- For **example,** in **audio transmission**, if the policy is to **discard less sensitive packets when congestion** is likely to happen, the **quality of sound** is **still preserved** and **congestion is prevented** or **alleviated.**

5. Admission Policy

- An **admission policy** can also **prevent congestion** in **virtual-circuit networks**.
- **Routers** first **check** the **resource requirement** of a **flow before admitting it to the network.**
- A **router** can **deny establishing a virtual circuit connection** if there is **congestion** in the network or if there is a **possibility of future congestion**.

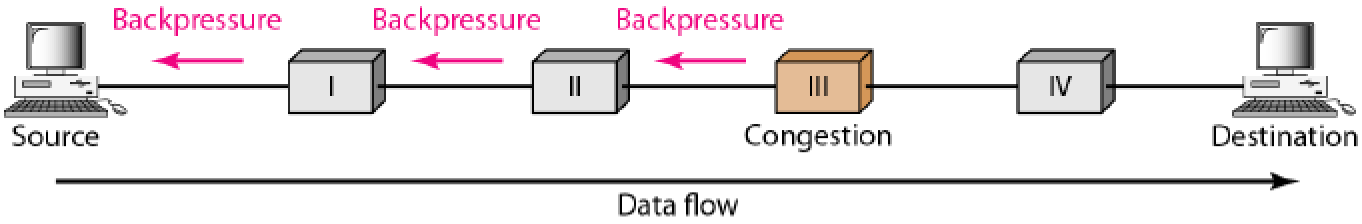
Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to **alleviate congestion after it happens**.

1. Backpressure

- The technique of ***backpressure*** refers to a **congestion control** mechanism in which a **congested node stops receiving data** from the **immediate upstream node** or nodes.
- This may cause the **upstream node or nodes to become congested**, and they, in turn, **reject data** from their **upstream nodes** or **nodes**. And so on.
- **Backpressure** is a **node-to-node congestion control** that starts with a node and **propagates, in the opposite direction of data flow, to the source**.
- The **backpressure technique** can be **applied** only to **virtual circuit networks**, in which **each node knows the upstream node** from which a flow of data is corning.

Closed-Loop Congestion Control



- **Node III** has **more input data** than it can **handle.**
- It **drops some packets** in its **input buffer** and **informs node II** to **slow down**.
- **Node II,** in turn, may be **congested** because it is **slowing down** the **output flow** of **data.**
- If **node II** is **congested**, it informs **node I** to **slow down**, which in turn may **create congestion.**
- If so, **node I** informs the **source of data to slow down.**
- This, in time, **alleviates** the **congestion.**
- Note that the **pressure on node III** is **moved backward** to the **source to remove** the **congestion.**

Closed-Loop Congestion Control

- **Backpressure technique** was **implemented** in the first **virtual-circuit network, X.25**.
- The technique **cannot be implemented in a datagram network** because in this **type** of **network**, a **node (router) does not have** the slightest **knowledge** of the **upstream router.**

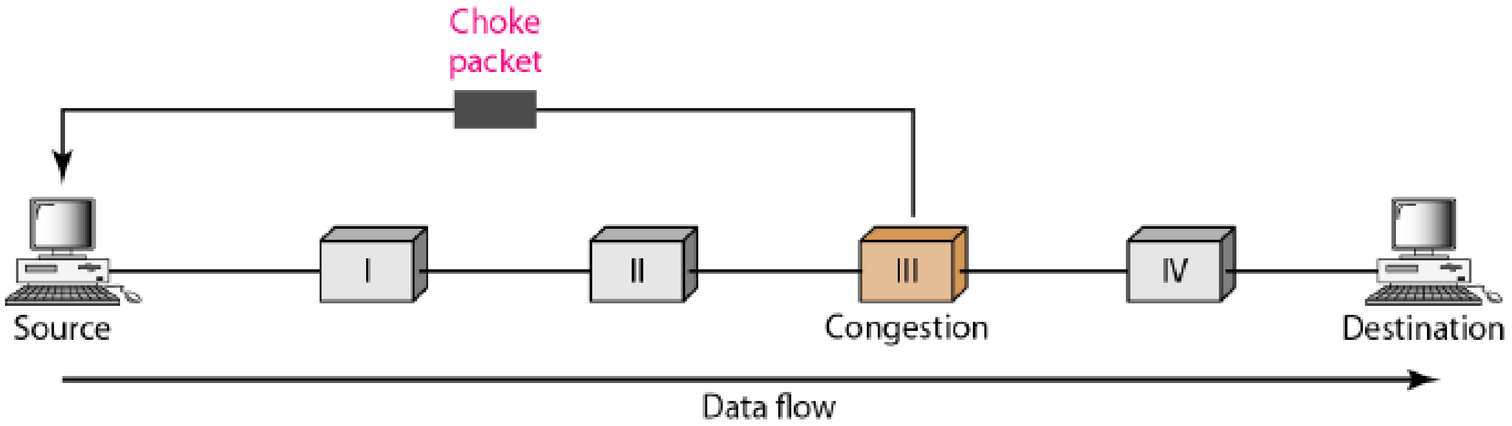
Closed-Loop Congestion Control

2. Choke Packet

- A **choke packet** is a **packet sent by a node** to the **source** to **inform** it of **congestion.**
- Note the **difference** between the **backpressure** and **choke packet** methods.
- *In **backpressure**, the **warning** is **from one node** to its **upstream node**, although the **warning** may **eventually reach** the **source station.***
- In the **choke packet** method, the ***warning** is **from** the **router**, which has **encountered congestion**, to the **source station** **directly.***
- The **intermediate nodes** through which the **packet** has **traveled** are **not warned.**
- We have seen an **example** of this **type** of control in **ICMP.**

Choke packet

- When a **router** in the **Internet** is **overwhelmed** with **IP datagrams**, it may **discard** some of them; but it **informs** the **source host**, using a **source quench** **ICMP message.**
- The **warning message** goes **directly to the source station**; the **intermediate routers** does not take **any action.**



Closed-Loop Congestion Control

3. Implicit Signaling

- In **implicit signaling**, there is **no communication between the congested node or nodes and the source**.
- The **source guesses** that **there** is a **congestion somewhere** in the **network from** other **symptoms**.
- For **example**, when a **source** sends **several packets** and there is **no acknowledgment for a while,** one **assumption** is that the **network** is **congested.**
- The **delay in receiving an acknowledgment** is **interpreted** as **congestion** in the **network**; the **source** should **slow down.**
- **TCP** uses such **signaling** for **congestion control.**

Closed-Loop Congestion Control

4. Explicit Signaling

- The **node** that **experiences** **congestion** can **explicitly** **send a** **signal** to the **source** **or** **destination.**
- The **explicit signaling** method, however, is **different** from the **choke packet** **method.**
- In the **choke packet** method, a **separate packet** **is used for this purpose**; in the **explicit signaling** **method**, the **signal** is **included** in the **packets** that **carry** **data**.
- **Explicit signaling**, as used in **Frame Relay congestion control**, can occur in either the **forward** or the **backward direction.**

Closed-Loop Congestion Control

4.1. Backward Signaling

- A **bit** can be **set** in a **packet moving** in the **direction** **opposite** to the **congestion.**
- This **bit** can **warn the** **source** that there is **congestion** and that it **needs** to **slow down** to avoid the discarding of packets.

4.2. Forward Signaling

- A **bit** can be **set** in a **packet moving** in the **direction** of the **congestion.**
- This **bit** can **warn** the **destination** that there is **congestion.**
- The **receiver** in this **case** can use **policies**, such as **slowing down** the **acknowledgments,** to **alleviate the congestion.**

Congestion Control in TCP

- **TCP** uses **congestion control** to **avoid congestion** or **alleviate congestion** in the **network.**

Congestion Window

- The **sender window size** is determined by the available buffer space in the receiver ***(rwnd).***
- In addition to the receiver, the **network** is a second entity that determines the size of the sender's window.
- Thus, *the **sender's window size** is determined not only by the **receiver** but also by **congestion** in the **network***.
- The **sender has two pieces of information**: the **receiver-advertised window size** and the **congestion window size.**
- The actual size of the window is the **minimum of these two**.
- **i.e. Sender’s window size= minimum (rwnd, cwnd)**

TCP Congestion Policy

- **TCP's general policy** for handling **congestion** is based on **three phases**:
  1. ***Slow start(SS)***
  2. ***Congestion avoidance(CA)***
  3. ***Congestion detection(CD)***
- In the **slow-start phase**, the **sender starts with a very slow rate** of **transmission**, but **increases the rate rapidly** to **reach** a **threshold**.
- When the **threshold is reached**, the **data rate is reduced to avoid congestion**.
- Finally **if congestion is detected**, the **sender goes back to the slow-start** or **congestion avoidance phase** based on **how the congestion is detected**.

1. Slow Start: Exponential Increase

- One of the **algorithms** used in **TCP congestion control** is called ***slow start*.**
- This **algorithm** is based on the idea that the **size** of the **congestion window *(cwnd)* starts with one maximum segment size (MSS).**
- The **MSS** is **determined** during **connection establishment** phase.
- The **size** of the **window increases one MSS** each time an **acknowledgment is received.**
- As the name implies, the **window starts slowly, but grows exponentially**.
- We have assumed that ***rwnd* is much higher than *cwnd*,** so that the **sender window size always equals *cwnd*.**
- We have assumed that each **segment** is **acknowledged** individually.

1.Slow Start: Exponential Increase

- The **Sender starts** with ***cwnd* =1 MSS.**
- This means that the **sender** can send only **one segment.**
- After **receipt** of the **acknowledgment** for **segment 1**, the **size** of the **congestion window** is **increased by 1**, which means that ***cwnd* is now 2.**
- Now two more **segments** can be **sent.**
- When each **acknowledgment is received**, the size of the **window** is increased by **1 MSS.**
- When all **seven segments** are **acknowledged**, ***cwnd* = 8.**

Slow Start: Exponential Increase

- If we look at the **size of *cwnd*** in terms of rounds (acknowledgment of the whole window of segments), we find that the rate is exponential as shown below:

  Start ..... *cwnd=1*

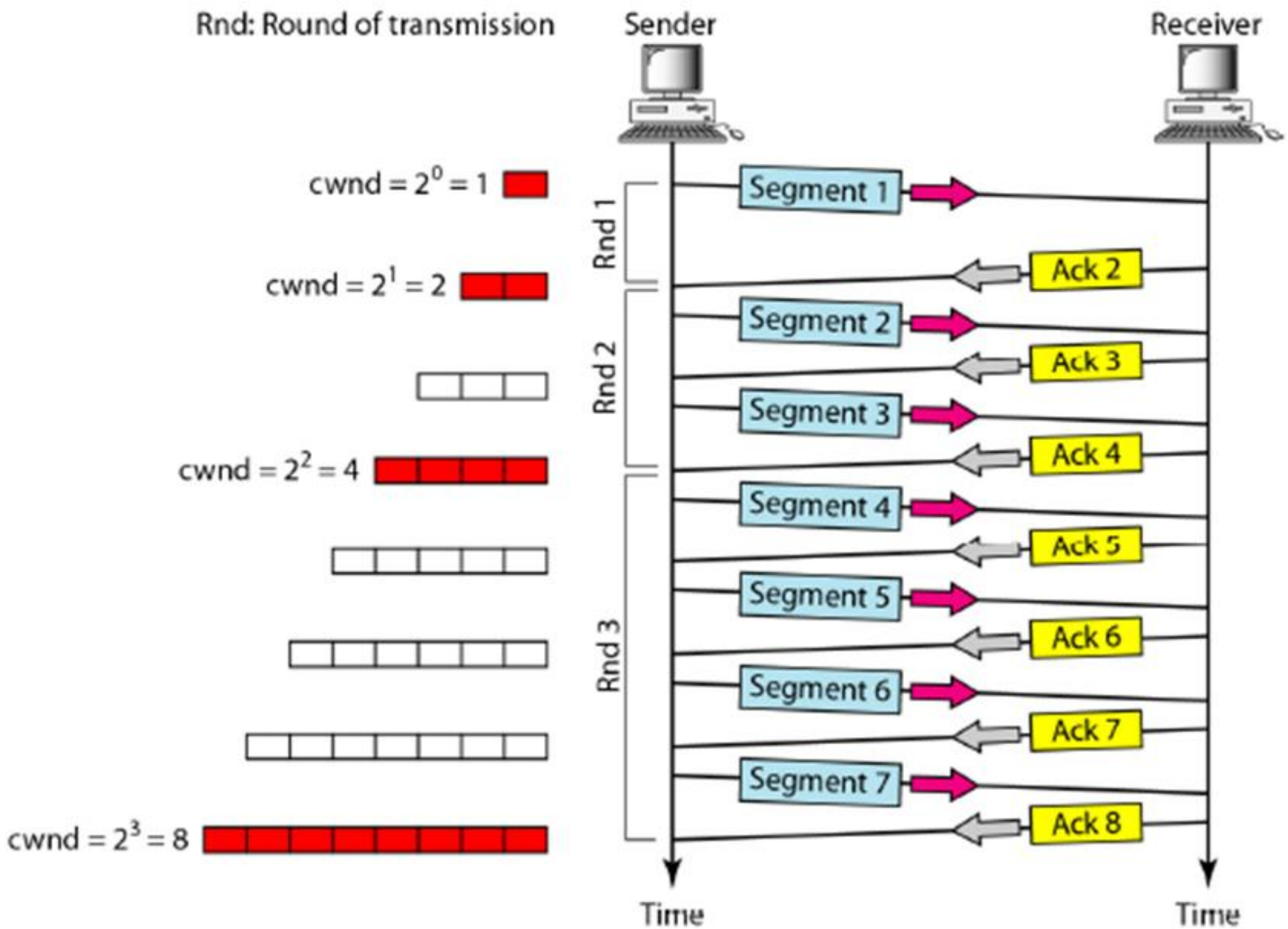
  After round 1 ..... *cwnd*=2^1 *=2*

  After round 2 ..... *cwnd*=2^2 *=4*

  After round 3 ..... *cwnd*=2^3 *=8*

- We need to mention that **if there is delayed ACKs,** the **increase in the size of the window is less than power of 2.**
- **Slow start** cannot continue **indefinitely**. There must be a **threshold to stop this phase**.
- The **sender** keeps track of a **variable named *ssthresh*** (slow-start threshold).
- When the **size of window** in **bytes reaches this threshold**, **slow start stops and the next phase starts.**
- In most implementations the value of ***ssthresh*** is **65,535 bytes**.

Slow Start: Exponential Increase



Congestion Avoidance: Additive Increase

- **Congestion avoidance** undergoes an **additive increase** instead of an exponential one.
- When the size of the **congestion window reaches the slow-start threshold**, the **slow-start phase stops and the additive phase begins.**
- In this **algorithm**, each time the whole window of segments is acknowledged (one round), the **size of the congestion window is increased by 1**.
- **Congestion avoidance** algorithm usually starts when the size of the window is much greater than 1.

Congestion Avoidance: Additive Increase

- In this case, after the sender has **received acknowledgments** for a **complete window size** of **segments,** the size of the window is **increased by one segment**. If we look at the size of *cwnd* in terms of rounds, we find that the rate is additive as shown below:

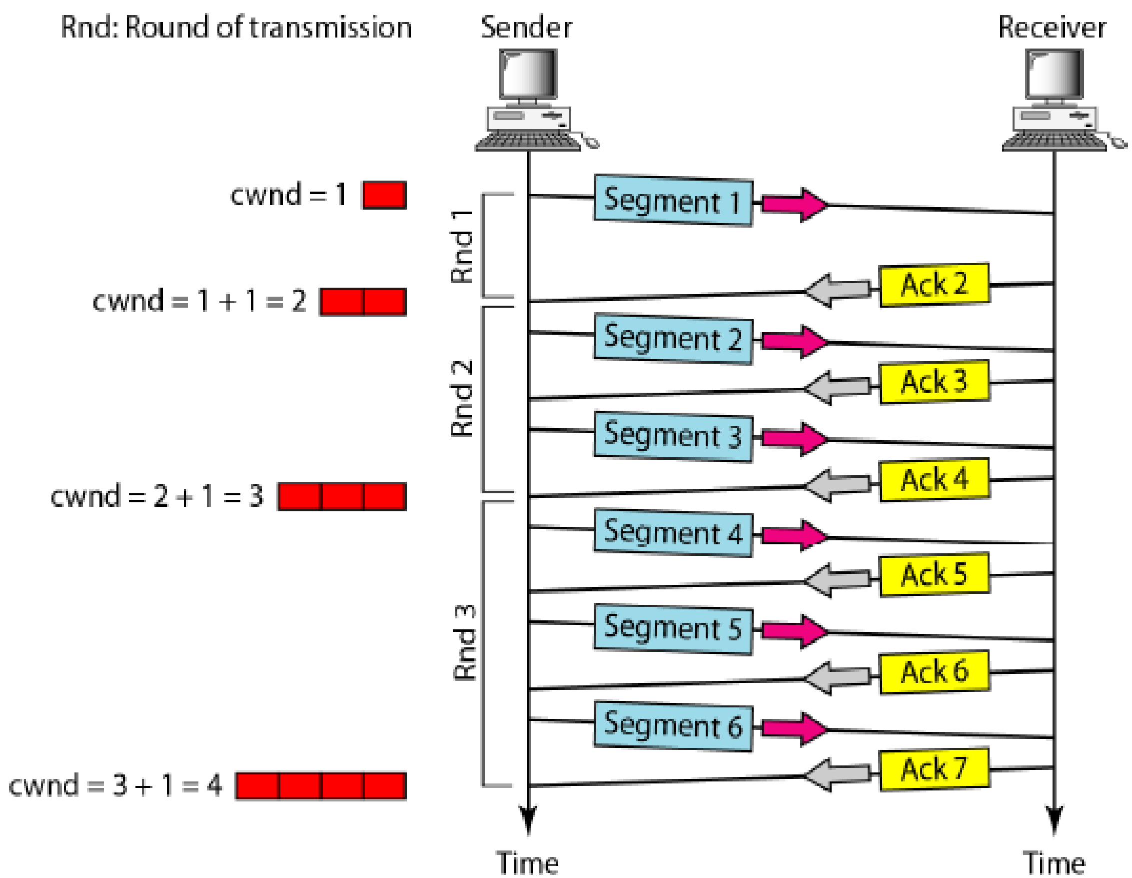
Start 1 *cwnd=1*

After round 1 *cwnd=* 1+ 1 =2

After round 2 *cwnd=2+* 1 =3

After round 3 *cwnd=3+* 1 =4

Congestion avoidance, additive increase



Congestion Detection: Multiplicative Decrease

- If **congestion occurs**, the **congestion window size must be decreased**.
- The only way the **sender can guess** that **congestion has occurred** is by the **need to retransmit a segment.**
- However, **retransmission can occur in one of two cases**: when a **timer times out** or when **three duplicate ACKs** are **received**.
- In both cases, the size of the **threshold is dropped to one-half**, a **multiplicative decrease.**
- Most **TCP implementations** have **two cases:**

Congestion Detection: Multiplicative Decrease

1. If a **time-out** occurs, there is a **stronger possibility of congestion**; a segment has probably been **dropped in the network**, and there is **no news about the sent segments**.

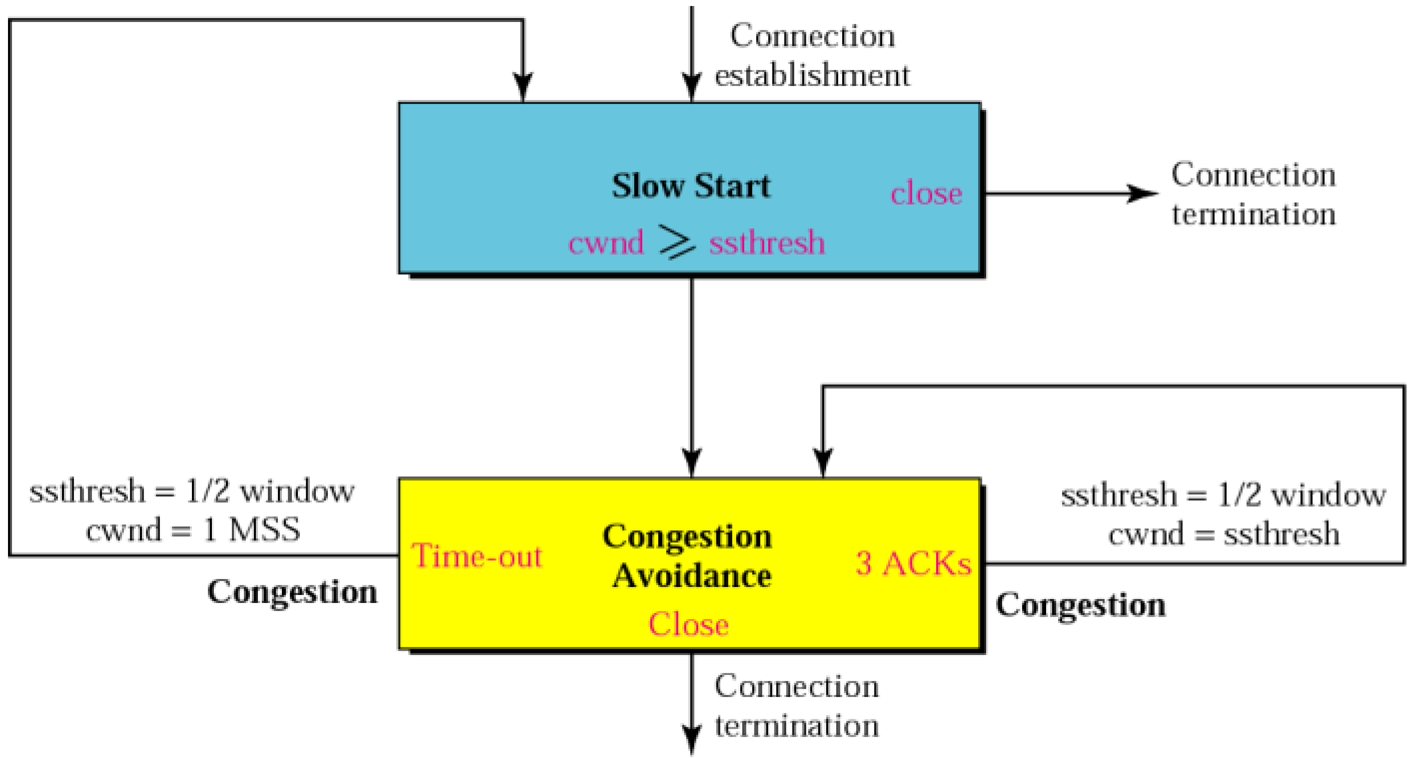
- In this case **TCP reacts strongly**:
- **a.** It sets the value of the **threshold** to **one-half** of the **current window size**.
- **b.** It sets ***cwnd*** to the **size** of **one segment**.
- **c.** It **starts** the **slow-start phase** again.

Congestion Detection: Multiplicative Decrease

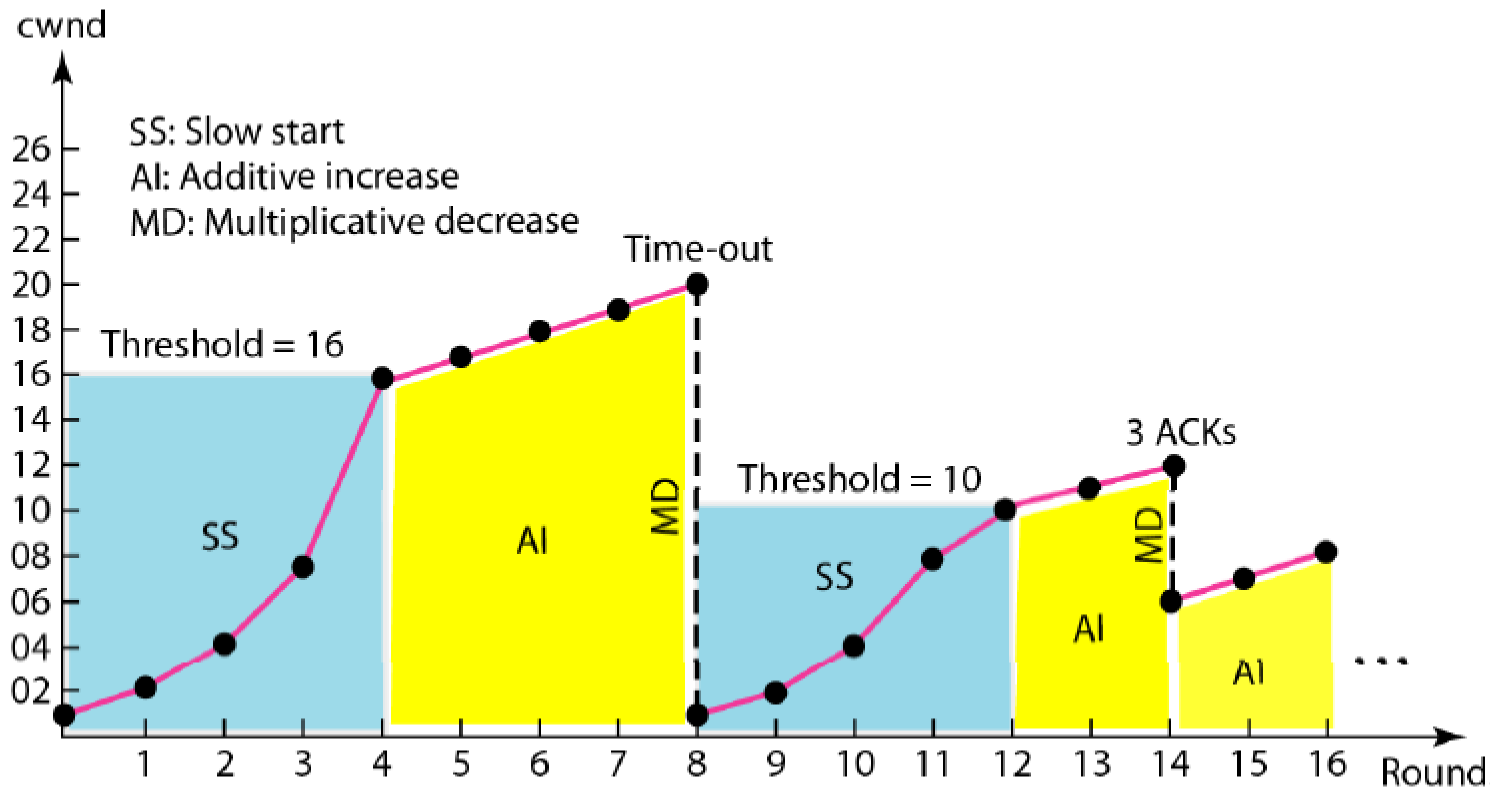
2. If **three DUP ACKs** are received, there is a **weaker possibility of congestion**; a segment **may have been dropped, but some segments after that may have arrived safely since three ACKs are received.** This is called **fast transmission** and **fast recovery**. In this case, TCP has a **weaker reaction:**

- **a.** It sets the value of the **threshold** to **one-half** of the **current window size**.
- **b.** It sets ***cwnd*** to the **value** of the **threshold**
- **c.** It **starts** the **congestion avoidance phase**.

TCP Congestion Policy Summary



TCP Congestion Control Example



- We assume that the **maximum window size is 32 segments**. The **threshold** is set to **16 segments** (one-half of the maximum window size).
- In the ***slow-start* phase** the window size **starts from 1** and grows **exponentially** until it reaches the threshold.

TCP Congestion Control Example

- After it reaches the **threshold**, the ***congestion avoidance*** ***(additive increase)*** procedure allows the window size to increase linearly until a timeout occurs or the maximum window size is reached. In Figure, the **time-out occurs** when the **window size** is **20.**
- At this moment, the ***multiplicative decrease*** procedure takes over and reduces the threshold to one-half of the previous window size. The previous window size was 20 when the time-out happened so the new threshold is now 10.
- TCP moves to **slow start again** and starts with a window size of 1, and TCP moves to additive increase when the new threshold is reached.
- When the window size is 12, a **three-DUPACKs** event happens. The **multiplicative decrease** procedure takes over again.
- The **threshold** is set to **6** and **TCP** goes to the **additive increase phase** this time.
- It remains in this phase until another time-out or another three ACKs happen.