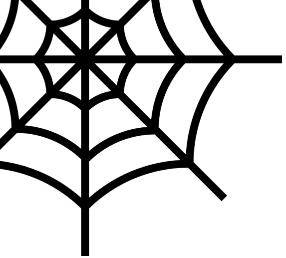
Congestion Control

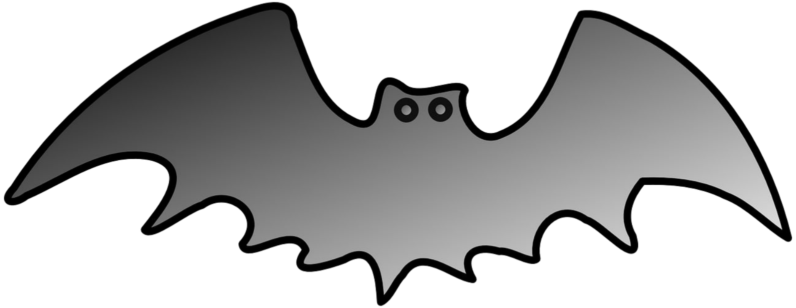
CS 168 – Fall 2022 – Discussion 9

Agenda

- Once Upon a Time...
- Congestion Control



October 1986



TCP Circa 1986

- What happens when router buffers fill up?
  - Packets get dropped
- Then what happens at the **sender**?
  - Increased RTT, timeouts, retransmits
    - $\rightarrow$**More packets in the network!!**
    - **… So more retransmits!**
- Eventually, useful throughput approaches **zero**
- This is the **congestion collapse** of 1986!

Congestion Control

Goal of Congestion Control

- Limit the # of packets in flight
  - Utilize our fair share of bandwidth…
  - But don't overload the network
- Adapt to the right bandwidth
- Be fair
  - Links are shared among many hosts

A naïve solution

- **Overwhelmed? Tell the sender to slow down!**
  - Both routers & receiver
- ICMP “Source Quench”
- Problems?
  - If the link is already overwhelmed, these extra messages may be dropped too! (or add more traffic)

TCP: loss-based feedback

Idea: drop implies congestion

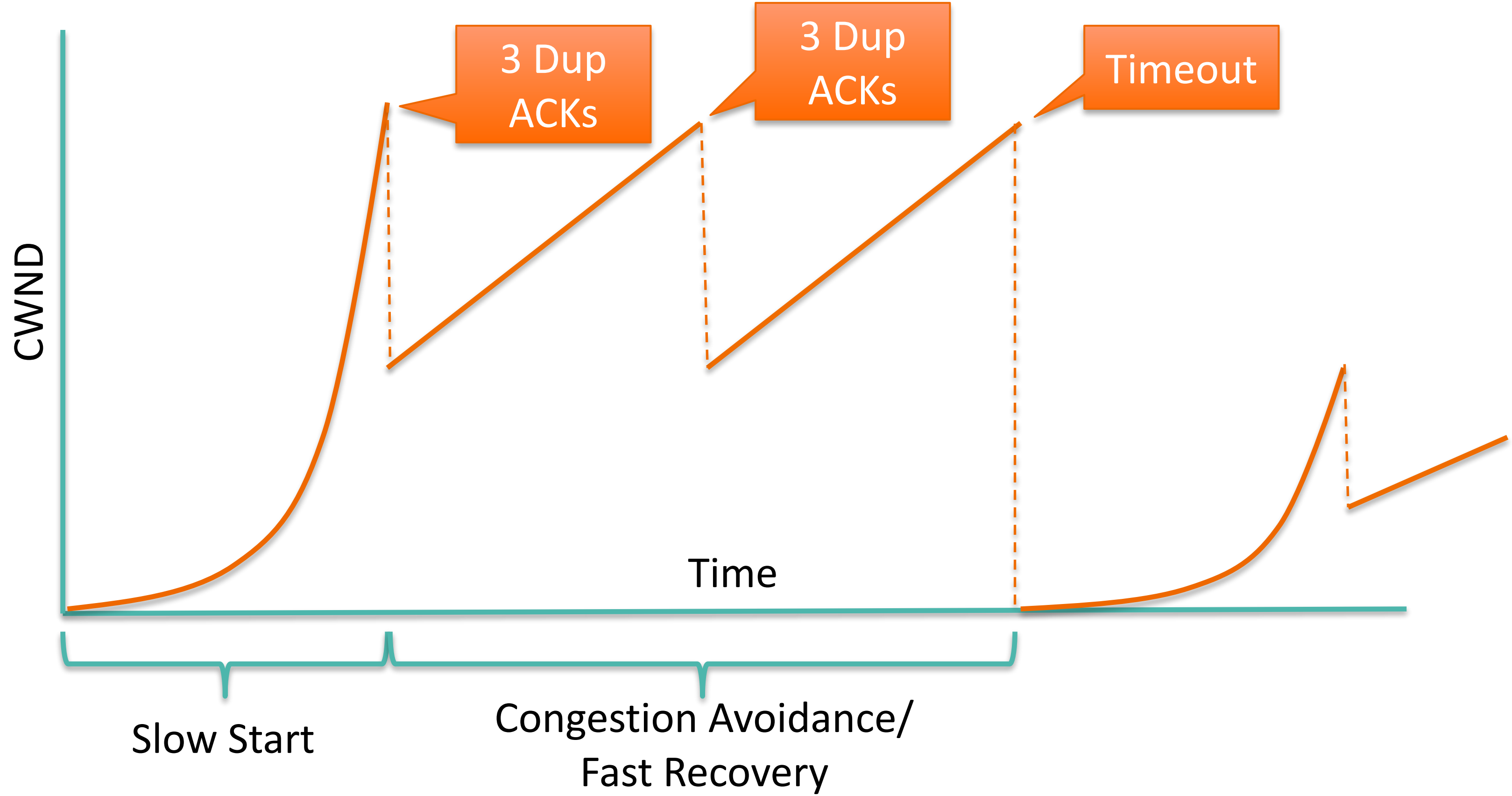
- 3 dupACK: minor congestion (ACKs get through)
- Timeout: major congestion (*nothing* gets through)

TCP's response depends on the *kind* of loss.

Congestion Control: Windows

- Receive Window (RWND)
  - What rate at which the **receiver** can process packets
- Congestion Window (CWND)
  - What rate at which the **network** can process packets
- Sending rate
  - Smaller of the two
- In this class, we assume CWND << RWND
  - **Network** will determine our sending rate

TCP Sawtooth

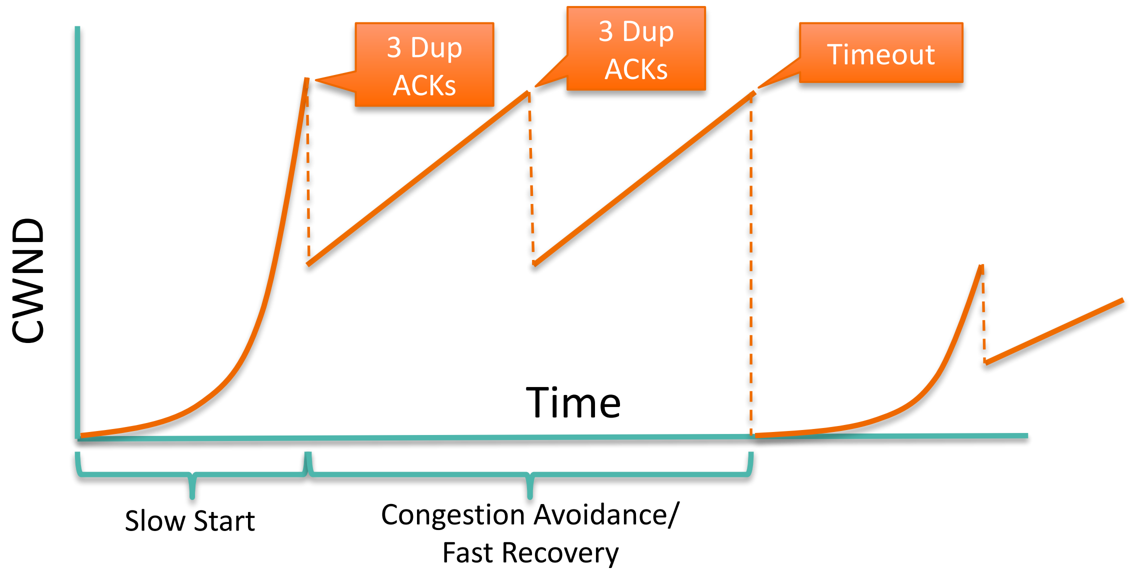


Utilization

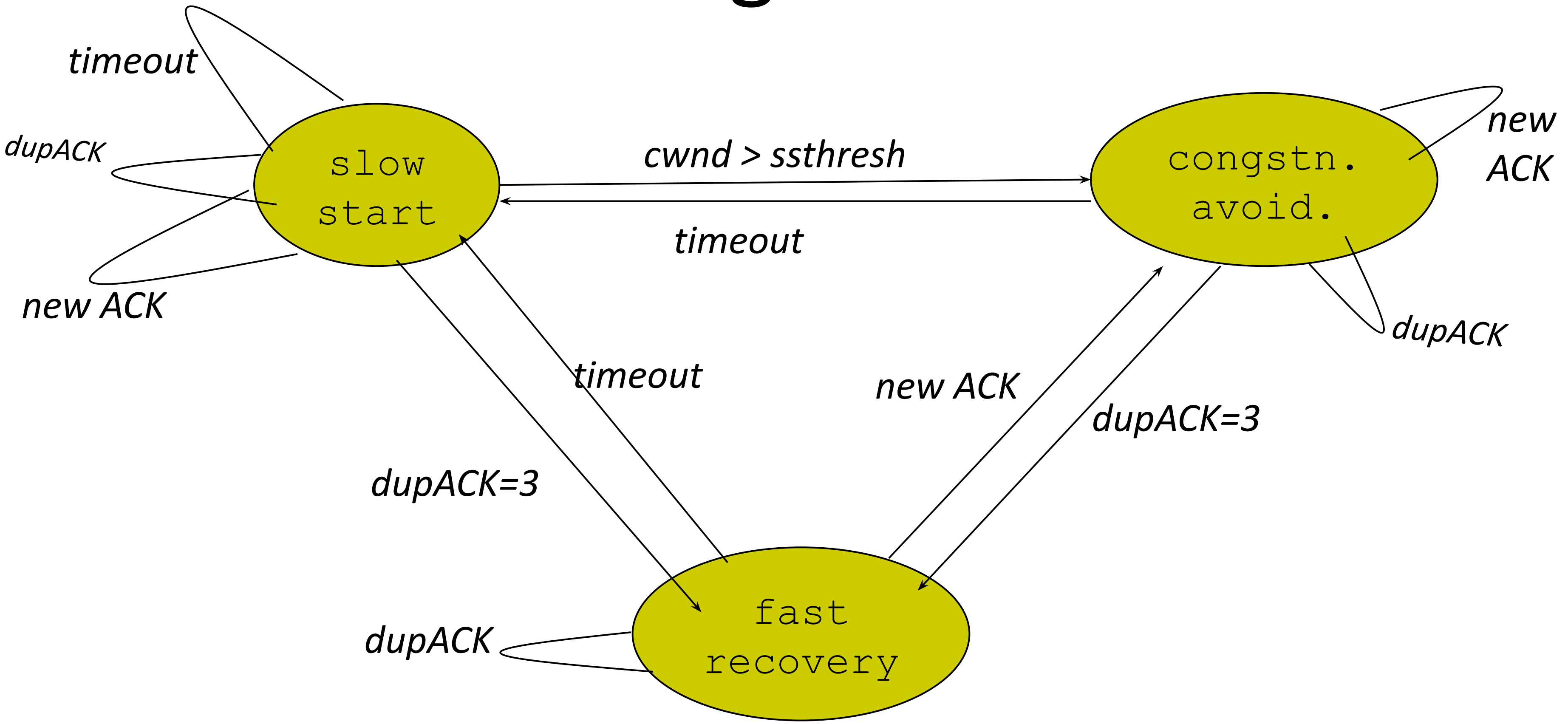
- The TCP sawtooth alternates between:
  - *Over-utilizing* bandwidth (causing drops)
  - *Under-utilizing* bandwidth
- Smart choices around buffering can result in higher utilization by absorbing the increase in window size

Three States

1. Slow Start
2. Congestion Avoidance
3. Fast Recovery



The Big Picture



Implementation

- State at sender
  - CWND
    - Max sending rate without congesting network (assuming CWND << RWND)
  - ssthresh
    - Threshold CWND for exiting slow start
  - dupACKcount
    - Count of contiguous duplicate ACKs received
  - timer

Congestion Control Mechanics

1. Slow Start
   – *Rapidly* increase our initial sending rate until we hit bottleneck
2. Congestion Avoidance
   – Adapting our sending rate to current network conditions
   – AIMD (**A**dditive **I**ncrease, **M**ultiplicative **D**ecrease)
3. Fast Recovery
   – Optimizing recovery from isolated loss
   – Detected through Duplicate ACKs

Implementation

- Events at sender
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout
- … receiver just receives packets and sends ACKs

Now the Details

- Thanks Alex Triana, our amazing F’15 TA!

Slow Start

- Value of CWND starts at (small constant) * MSS
- For each packet that is acknowledged, increase the CWND by 1
  - Effectively **doubles** CWND every **RTT**!

- Window goes from 1 → 2 → 4 → …

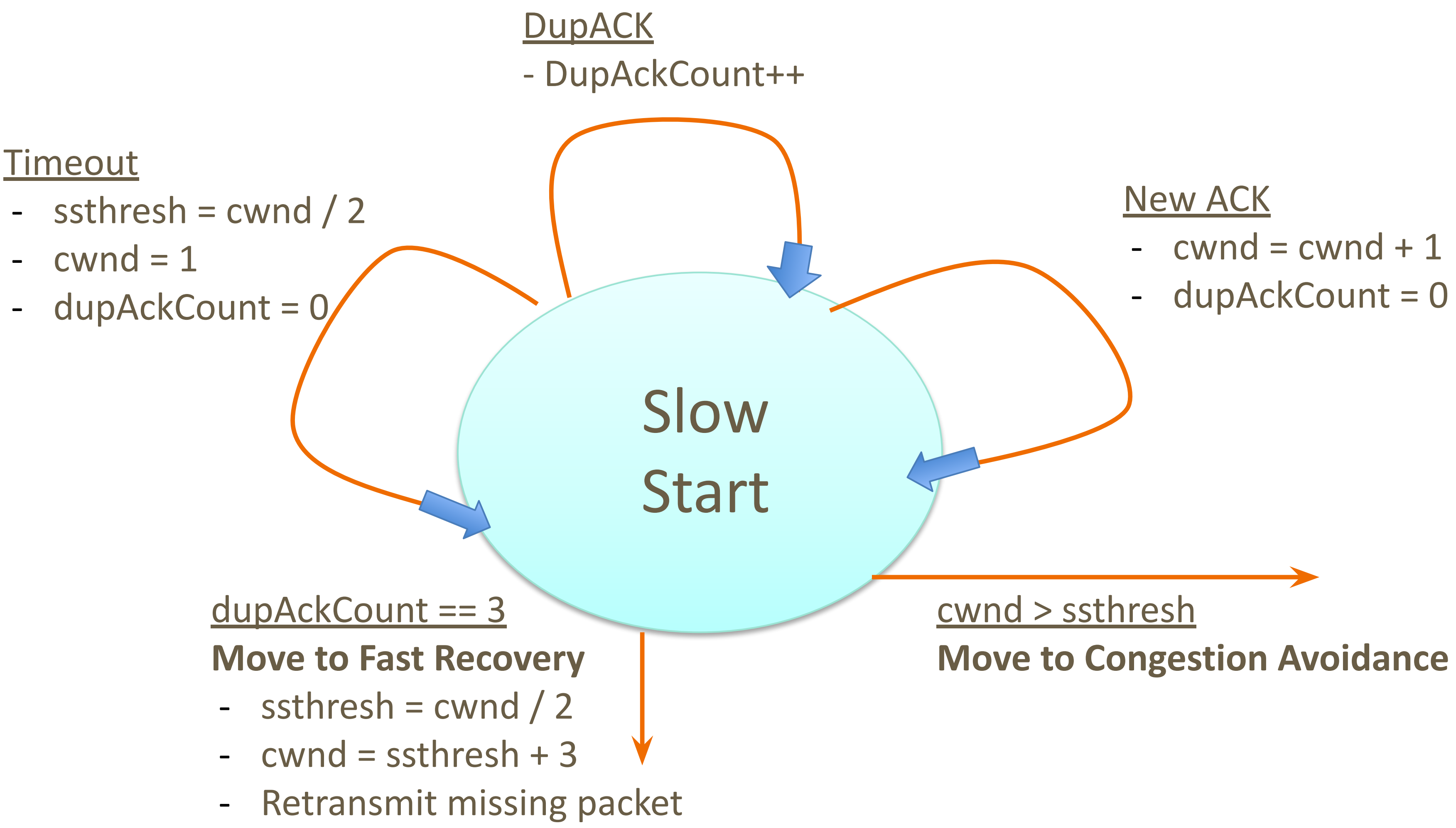
Slow Start -- Intuition

- Instead of blasting packets based on the receive window
- Build up initial transmission rate *slowly*
- Back off when we’ve exceeded the capacity

Slow Start – When Does It End?

- 2 Ways
  1) If CWND > ssthresh
     - Enter congestion avoidance
  2) If we get 3 duplicate ACKs
     - Enter Fast Recovery

- If timeout:
  - Restart slow start, ssthresh = cwnd/2, CWND = 1

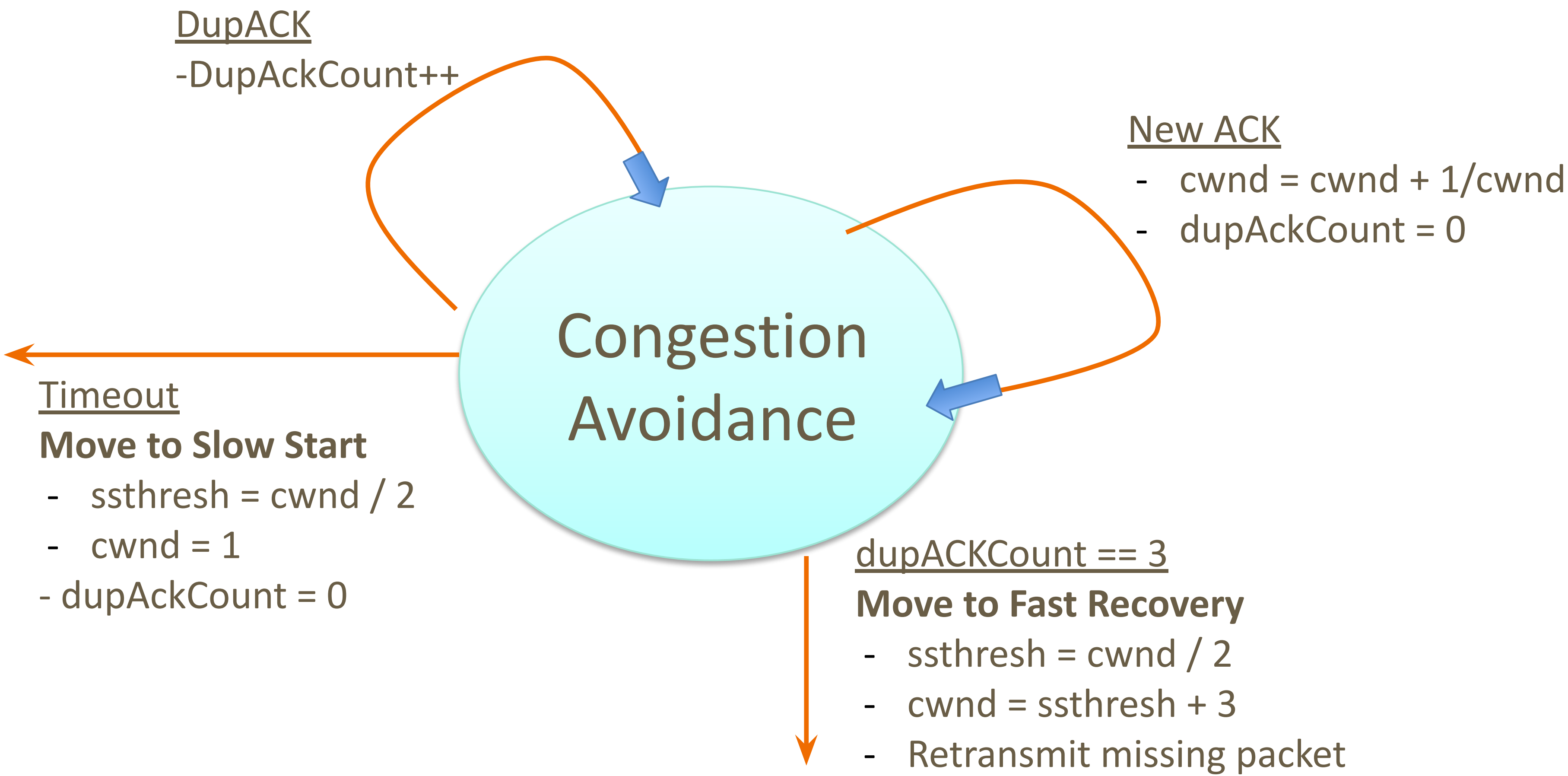

DupACK
- DupAckCount++
Timeout
- ssthresh = cwnd / 2
- cwnd = 1
- dupAckCount = 0
New ACK
- cwnd = cwnd + 1
- dupAckCount = 0
Slow Start
dupAckCount == 3
Move to Fast Recovery
- ssthresh = cwnd / 2
- cwnd = ssthresh + 3
- Retransmit missing packet
cwnd > ssthresh
Move to Congestion Avoidance

Congestion Avoidance -- Intuition

- In the steady state
- Constantly probe for more bandwidth
- When we’ve exceeded – back off aggressively

Congestion Avoidance

- Growth is more conservative than slow start
- After each **new** ACK, increase CWND by 1 / CWND
  - After one **RTT**, CWND will have **increased by ~1**
- When does it stop?
  1) Timeout → back to slow start
  2) 3 duplicate ACKS → Fast recovery


DupACK
-DupAckCount++
New ACK
- cwnd = cwnd + 1/cwnd
- dupAckCount = 0
Congestion Avoidance
Timeout
Move to Slow Start
- ssthresh = cwnd / 2
- cwnd = 1
- dupAckCount = 0
dupACKCount == 3
Move to Fast Recovery
- ssthresh = cwnd / 2
- cwnd = ssthresh + 3
- Retransmit missing packet

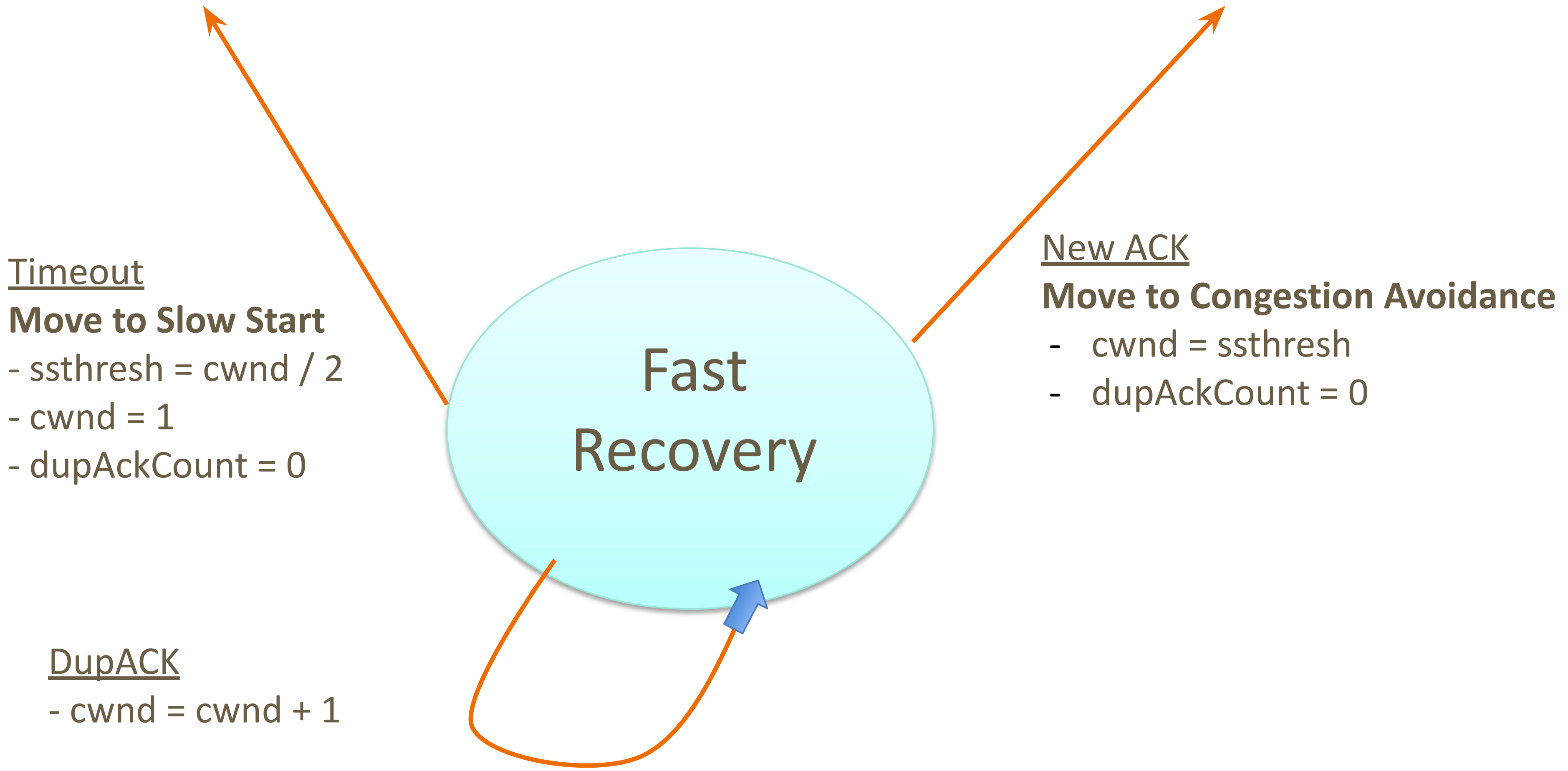
Fast Recovery – Intuition

- A single lost packet
  - May just be a fluke
- Resetting CWND may be too aggressive
- Instead just retransmit that single packet
  - And continue as if nothing happened

Fast Recovery

- Every **duplicate** ACK increases the window by 1

- When does it stop?
  1) Timeout → back to Slow Start
  2) New ACK → back to Congestion Avoidance


Timeout
Move to Slow Start
- ssthresh = cwnd / 2
- cwnd = 1
- dupAckCount = 0
Fast Recovery
New ACK
Move to Congestion Avoidance
- cwnd = ssthresh
- dupAckCount = 0
DupACK
- cwnd = cwnd + 1

Big Ideas

- Fundamental concepts:
  - Slow Start
  - AIMD
- Hack
  - Fast Recovery
- Lesson
  - Sometimes, BAND-AIDs scale remarkably well!

End of section slides

Worksheet

Question 1

1. UDP uses congestion control.
2. Flow control slows down the sender when the network is congested.
3. For TCP timer implementations, every time the sender receives an ACK for a previously unACKed packet, it will recalculate ETO.
4. CWND (congestion window) is usually smaller than RWND (receiver window).
5. AIMD is the only "fair" option among MIMD, AIAD, MIAD, and AIMD.

1) Without Fast Recovery

- *On new ACK,* ***CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs,* ***SSTHRESH = Floor(CWND/2)****, then* ***CWND = SSTHRESH***

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/
1.8	102 (109)	5	/
1.9	102 (110)	5	/
2.0	102 (111)	5	/
2.4	112 (102)	5+1/Floor(5)=5.2	112 – 116 (No)

1) Without Fast Recovery

- *On new ACK, **CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs, **SSTHRESH = Floor(CWND/2)**, then **CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)

1) Without Fast Recovery

- *On new ACK,* ***CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs,* ***SSTHRESH = Floor(CWND/2)****, then* ***CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/

1) Without Fast Recovery

- *On new ACK, **CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs, **SSTHRESH = Floor(CWND/2)**, then **CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/

1) Without Fast Recovery

- *On new ACK,* ***CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs,* ***SSTHRESH = Floor(CWND/2)****, then* ***CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/

1) Without Fast Recovery

- *On new ACK, **CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs, **SSTHRESH = Floor(CWND/2)**, then **CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/
1.8	102 (109)	5	/

1) Without Fast Recovery

- *On new ACK,* ***CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs,* ***SSTHRESH = Floor(CWND/2)****, then* ***CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/
1.8	102 (109)	5	/
1.9	102 (110)	5	/

1) Without Fast Recovery

- *On new ACK, **CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs, **SSTHRESH = Floor(CWND/2)**, then **CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/
1.8	102 (109)	5	/
1.9	102 (110)	5	/
2.0	102 (111)	5	/

1) Without Fast Recovery

- *On new ACK, **CWND = CWND + 1/Floor(CWND)***
- *On triple duplicate ACKs, **SSTHRESH = Floor(CWND/2)**, then **CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)=5	102 (Yes)
1.5	102 (106)	5	/
1.6	102 (107)	5	/
1.7	102 (108)	5	/
1.8	102 (109)	5	/
1.9	102 (110)	5	/
2.0	102 (111)	5	/
2.4	112 (102)	5+1/Floor(5)=5.2	112 – 116 (No)

2) With Fast Recovery

- *On triple duplicate ACK, **SSTHRESH = CWND/2,** then **CWND = SSTHRESH + 3**, and enter fast recovery*
- *In fast recovery, **CWND += 1** on every duplicate ACK*
- *Exit fast recovery on new ACK, setting **CWND = SSTHRESH***

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/

2) With Fast Recovery

- *On triple duplicate ACK, **SSTHRESH = CWND/2,** then **CWND = SSTHRESH + 3**, and enter fast recovery*
- *In fast recovery, **CWND += 1** on every duplicate ACK*
- *Exit fast recovery on new ACK, setting **CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)

2) With Fast Recovery

- *On triple duplicate ACK,* ***SSTHRESH = CWND/2,*** *then* ***CWND = SSTHRESH + 3****, and enter fast recovery*
- *In fast recovery,* ***CWND += 1*** *on every duplicate ACK*
- *Exit fast recovery on new ACK, setting* ***CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/

2) With Fast Recovery

- *On triple duplicate ACK,* ***SSTHRESH = CWND/2,*** *then* ***CWND = SSTHRESH + 3****, and enter fast recovery*
- *In fast recovery,* ***CWND += 1*** *on every duplicate ACK*
- *Exit fast recovery on new ACK, setting* ***CWND = SSTHRESH***

Ssthresh=5

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
1.5	102 (106)	9	/
1.6	102 (107)	10	/

2) With Fast Recovery

- *On triple duplicate ACK,* ***SSTHRESH = CWND/2,*** *then* ***CWND = SSTHRESH + 3****, and enter fast recovery*
- *In fast recovery,* ***CWND += 1*** *on every duplicate ACK*
- *Exit fast recovery on new ACK, setting* ***CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/
	1.6	102 (107)	10	/
	1.7	102 (108)	11	112 (No)

2) With Fast Recovery

- *On triple duplicate ACK, **SSTHRESH = CWND/2,** then **CWND = SSTHRESH + 3**, and enter fast recovery*
- *In fast recovery, **CWND += 1** on every duplicate ACK*
- *Exit fast recovery on new ACK, setting **CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/
	1.6	102 (107)	10	/
	1.7	102 (108)	11	112 (No)
	1.8	102 (109)	12	113 (No)

2) With Fast Recovery

- *On triple duplicate ACK, **SSTHRESH = CWND/2,** then **CWND = SSTHRESH + 3**, and enter fast recovery*
- *In fast recovery, **CWND += 1** on every duplicate ACK*
- *Exit fast recovery on new ACK, setting **CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/
	1.6	102 (107)	10	/
	1.7	102 (108)	11	112 (No)
	1.8	102 (109)	12	113 (No)
	1.9	102 (110)	13	114 (No)

2) With Fast Recovery

- *On triple duplicate ACK,* ***SSTHRESH = CWND/2,*** *then* ***CWND = SSTHRESH + 3****, and enter fast recovery*
- *In fast recovery,* ***CWND += 1*** *on every duplicate ACK*
- *Exit fast recovery on new ACK, setting* ***CWND = SSTHRESH***

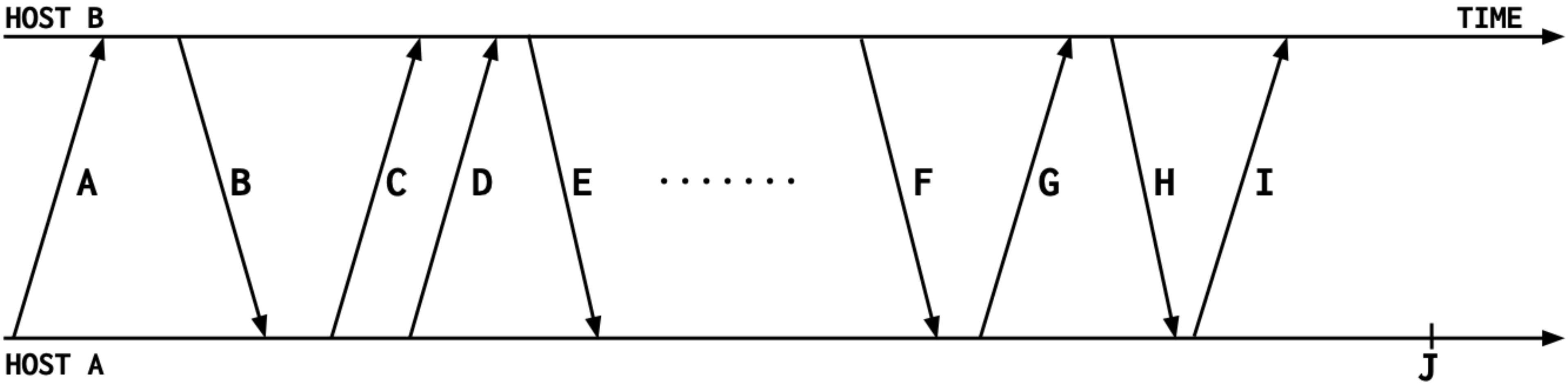
	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/
	1.6	102 (107)	10	/
	1.7	102 (108)	11	112 (No)
	1.8	102 (109)	12	113 (No)
	1.9	102 (110)	13	114 (No)
	2.0	102 (111)	14	115 (No)

2) With Fast Recovery

- *On triple duplicate ACK,* ***SSTHRESH = CWND/2,*** *then* ***CWND = SSTHRESH + 3****, and enter fast recovery*
- *In fast recovery,* ***CWND += 1*** *on every duplicate ACK*
- *Exit fast recovery on new ACK, setting* ***CWND = SSTHRESH***

	Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (**yes** for retransmit)
	1.0	102 (101)	10+1/Floor(10)=10.1	111 (No)
	1.2	102 (103)	10.1	/
	1.3	102 (104)	10.1	/
Ssthresh=5	1.4	102 (105)	Floor(10.1/2)+3=8	102 (Yes)
	1.5	102 (106)	9	/
	1.6	102 (107)	10	/
	1.7	102 (108)	11	112 (No)
	1.8	102 (109)	12	113 (No)
	1.9	102 (110)	13	114 (No)
	2.0	102 (111)	14	115 (No)
	2.4	112 (102)	SSTHRESH = 5	116 (No)

Question 3



A:

B:

C:

D:

E:

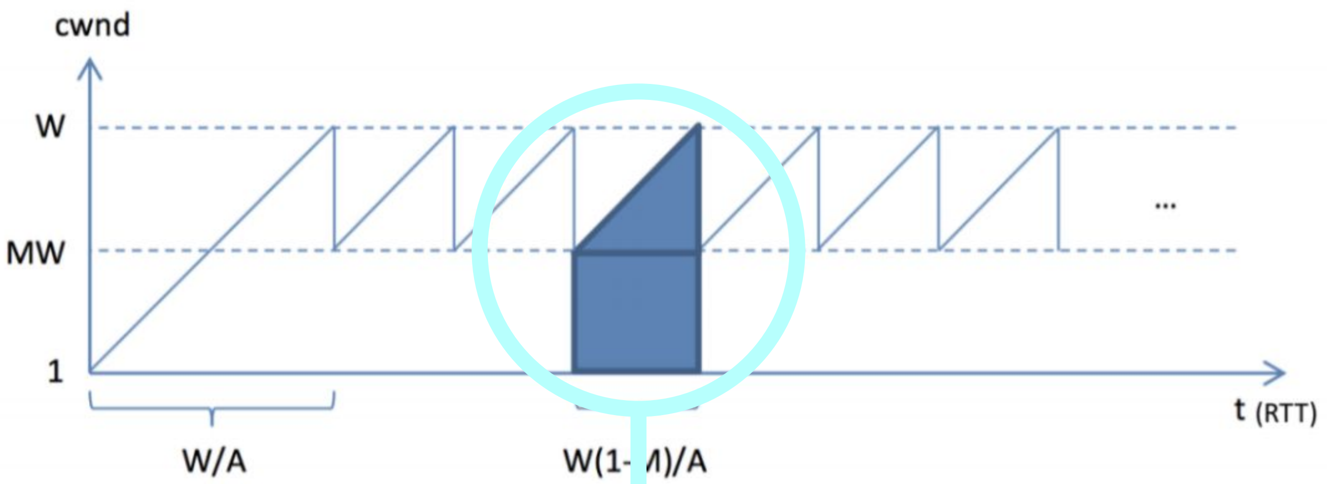
F:

G:

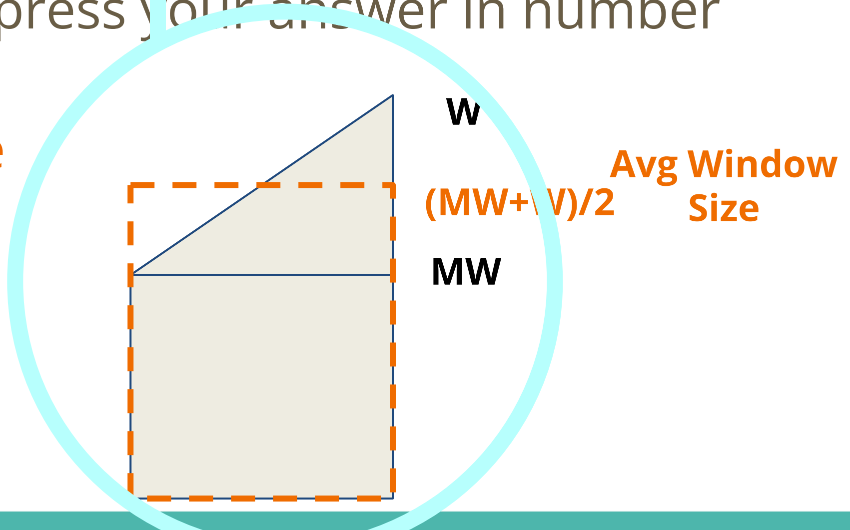
H:

I:

Question 3

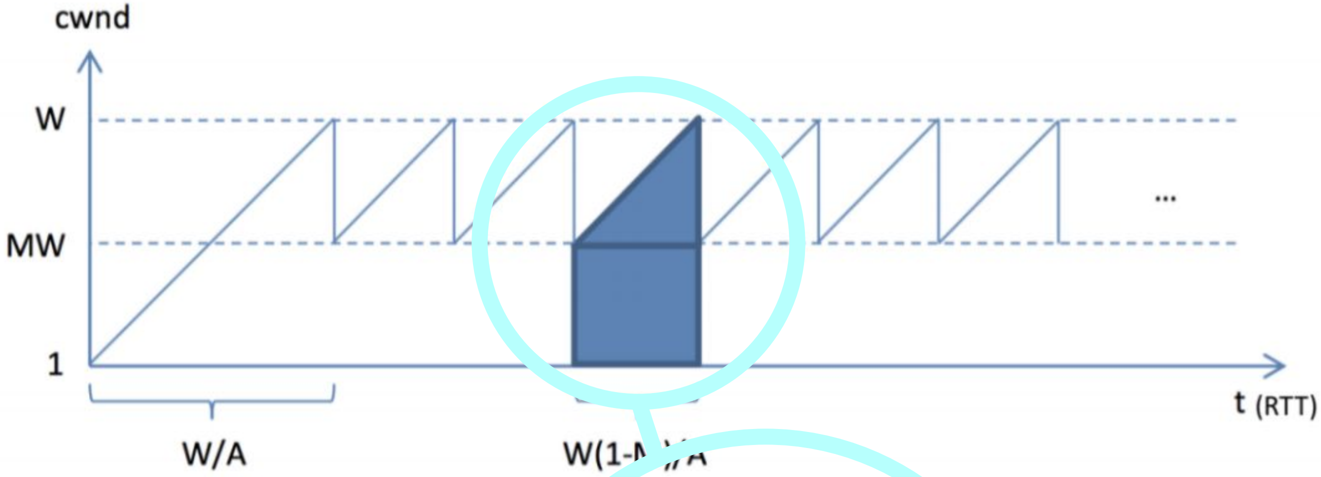


1. What is the average throughput? Express your answer in number of packets, not MSS.
   - Hint: Think of average window size



Window/RTT = Throughput
(MW+W)/(2*RTT)

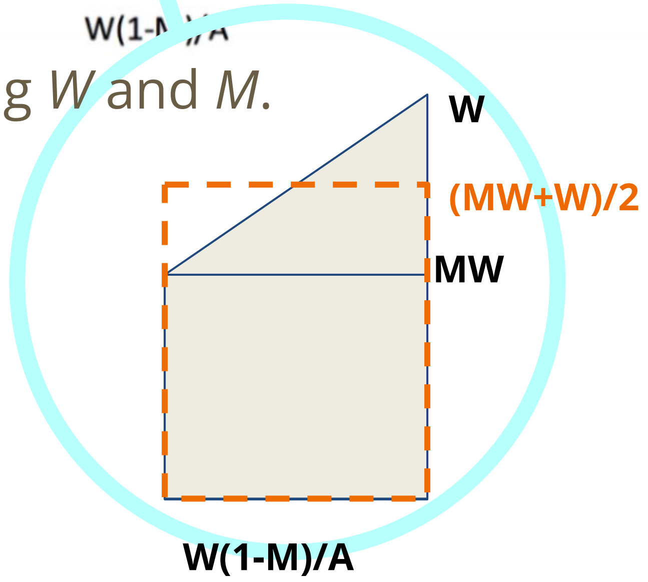
Question 3



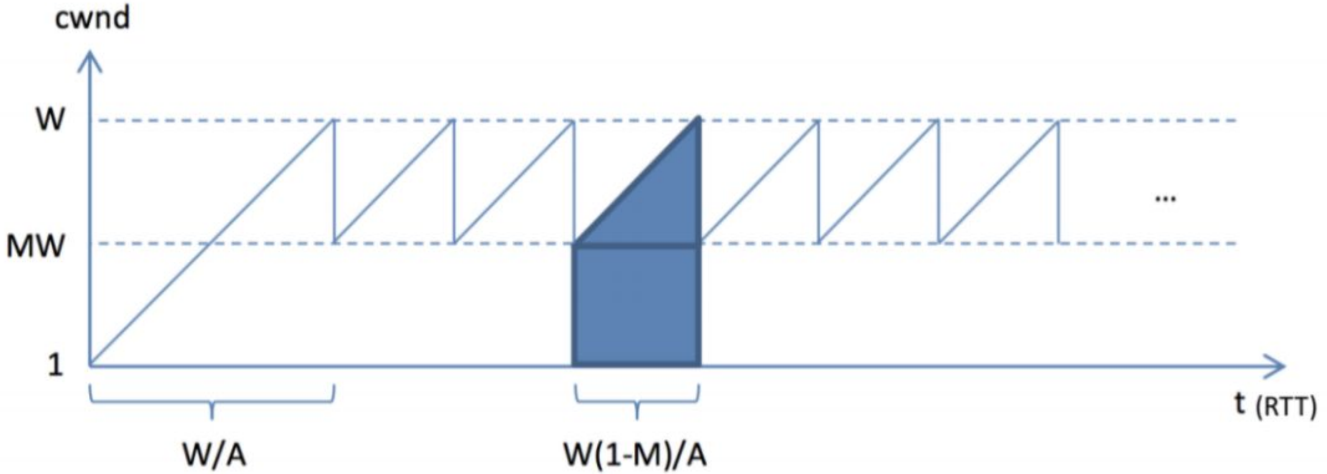
2. Calculate the loss probability *p*, using *W* and *M*.

- Hint: Use your result from Q1.

(MW+W)/2 * W(1-M)/A



Question 3



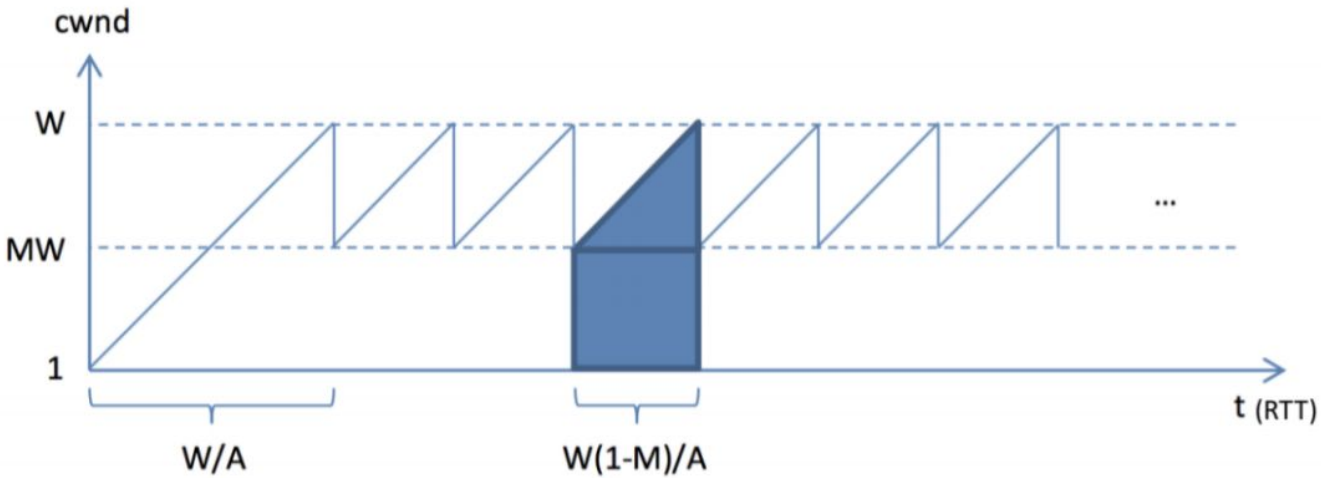
2. Calculate the loss probability *p*, using *W* and *M*.

- Hint: Use your result from Q1.

Total Packets:
$W^2(1-M^2)/2A$

1 drop per total packets:
p = 1/ [Total Packets] = $\boldsymbol{2A \,/\, [\, W^2 (1 - M^2)\,]}$

Question 3



3. Derive the formula for throughput in part 1 when *M = 0.5* and *A = 1* (try to only use *p* and *RTT*).

$$Throughput = \frac{1.5W}{2RTT}$$

$$p = \frac{2}{.75W^2}$$

$$W = \sqrt{\frac{8}{3p}}$$

$$Throughput = \frac{1.5\sqrt{\frac{8}{3p}}}{2RTT} = \frac{.5\sqrt{\frac{3*8}{p}}}{2RTT} = \frac{\sqrt{\frac{3}{2p}}}{RTT}$$