

Communication Networks (0368-3030) / Spring 2011

The Blavatnik School of Computer Science,
Tel-Aviv University

Allon Wagner

A decorative graphic consisting of several horizontal lines of varying lengths and colors (teal, light blue, white) extending from the right side of the slide towards the center.

TCP Flow & Congestion Control

Kurose & Ross, Chapter 3.5.5, 3.7 (5th ed.)

Many slides adapted from:

J. Kurose & K. Ross \

Computer Networking: A Top Down Approach (5th ed.)

Addison-Wesley, April 2009.

Copyright 1996-2010, J.F Kurose and K.W. Ross, All Rights Reserved.

TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver

TCP Receiver action

Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed

Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK

Arrival of in-order segment with expected seq #. One other segment has ACK pending

Immediately send single cumulative ACK, ACKing both in-order segments

Arrival of out-of-order segment higher-than-expected seq. # . Gap detected

Immediately send *duplicate ACK*, indicating seq. # of next expected byte

Arrival of segment that partially or completely fills gap

Immediate send ACK, provided that segment starts at lower end of gap

Fast Retransmit

- ❖ time-out period often relatively long:
 - long delay before resending lost packet
- ❖ detect lost segments via duplicate ACKs.
 - sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs.
- ❖ if sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
 - fast retransmit: resend segment before timer expires

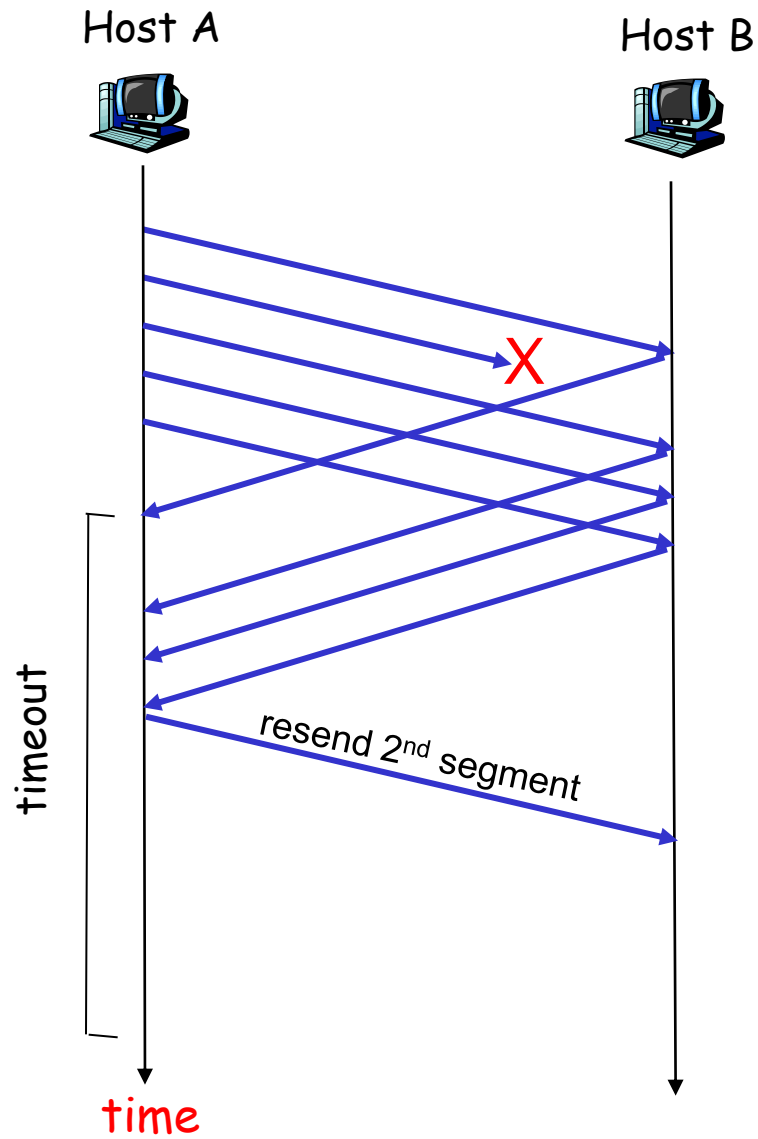


Figure 3.37 Resending a segment after triple duplicate ACK

Fast retransmit algorithm:

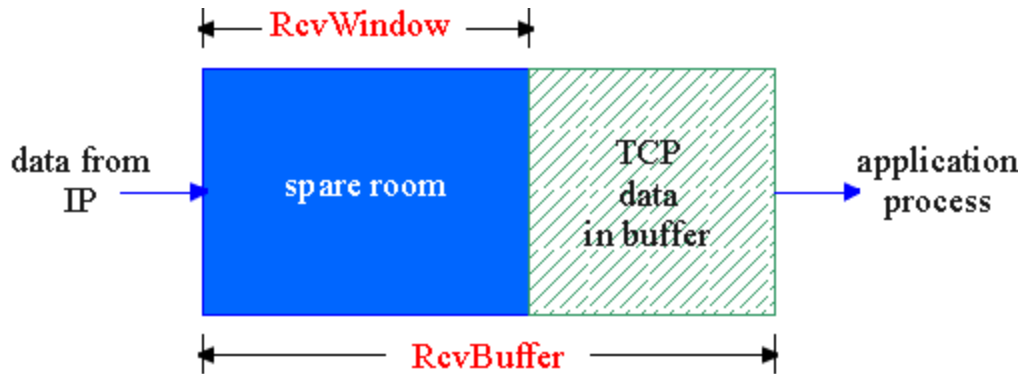
```
event: ACK received, with ACK field value of y
    if (y > SendBase) {
        SendBase = y
        if (there are currently not-yet-acknowledged segments)
            start timer
    }
    else {
        increment count of dup ACKs received for y
        if (count of dup ACKs received for y = 3) {
            resend segment with sequence number y
        }
    }
```

a duplicate ACK for
already ACKed segment

fast retransmit

TCP Flow Control

- ❖ receive side of TCP connection has a receive buffer:



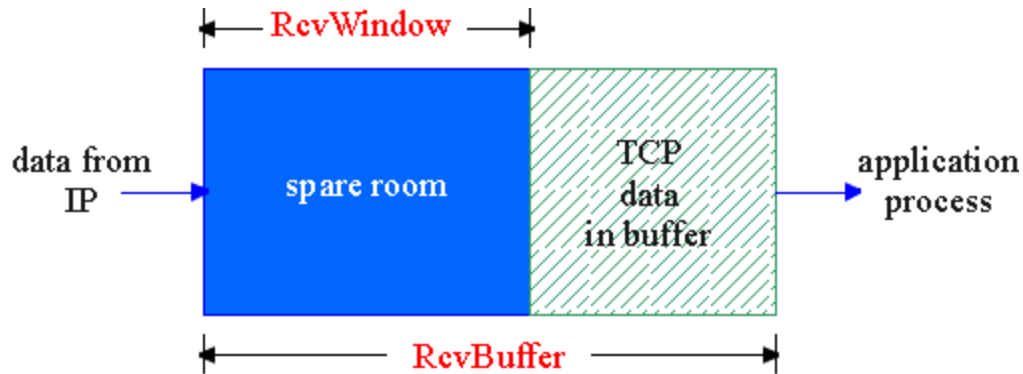
- ❖ app process may be slow at reading from buffer

flow control

sender won't overflow receiver's buffer by transmitting too much, too fast

- ❖ speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



(suppose TCP receiver discards out-of-order segments)

❖ spare room in buffer

= RcvWindow

= $\text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$

- ❖ rcvr advertises spare room by including value of RcvWindow in segments
- ❖ sender limits unACKed data to RcvWindow
 - guarantees receive buffer doesn't overflow

Approaches towards congestion control

Two broad approaches towards congestion control:

end-end congestion control:

- ❖ no explicit feedback from network
- ❖ congestion inferred from end-system observed loss, delay
- ❖ approach taken by TCP

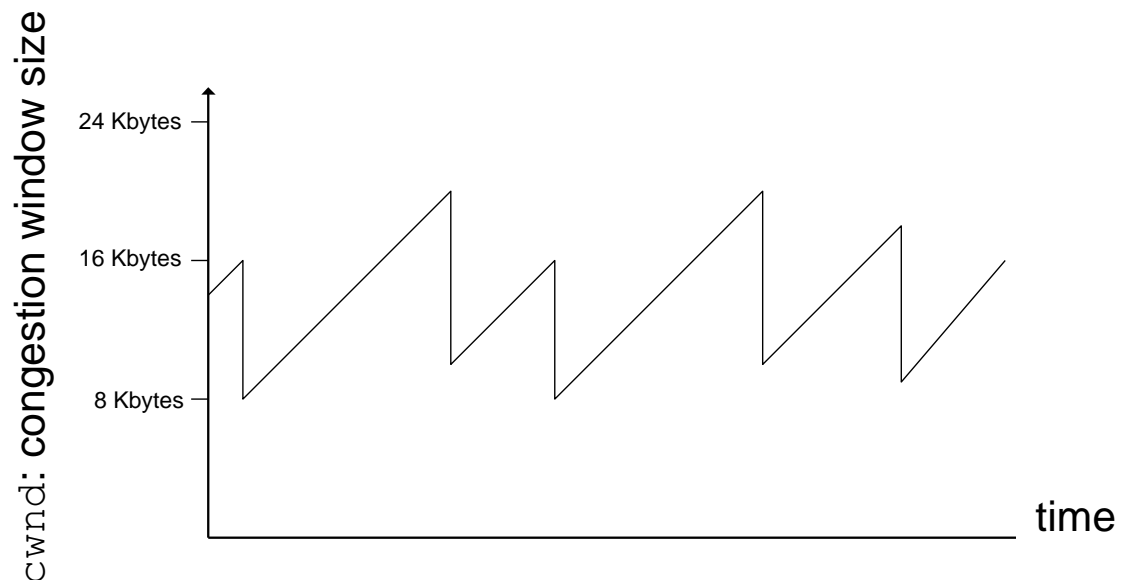
network-assisted congestion control:

- ❖ routers provide feedback to end systems
 - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
 - explicit rate sender should send at

TCP congestion control: additive increase, multiplicative decrease

- ❖ *approach*: increase transmission rate (window size), probing for usable bandwidth, until loss occurs
 - *additive increase*: increase cwnd by 1 MSS every RTT until loss detected
 - *multiplicative decrease*: cut cwnd in half after loss

saw tooth
behavior: probing
for bandwidth



TCP Congestion Control: details

- ❖ sender limits transmission:
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$

- ❖ roughly,

$$\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- ❖ cwnd is dynamic, function of perceived network congestion

How does sender perceive congestion?

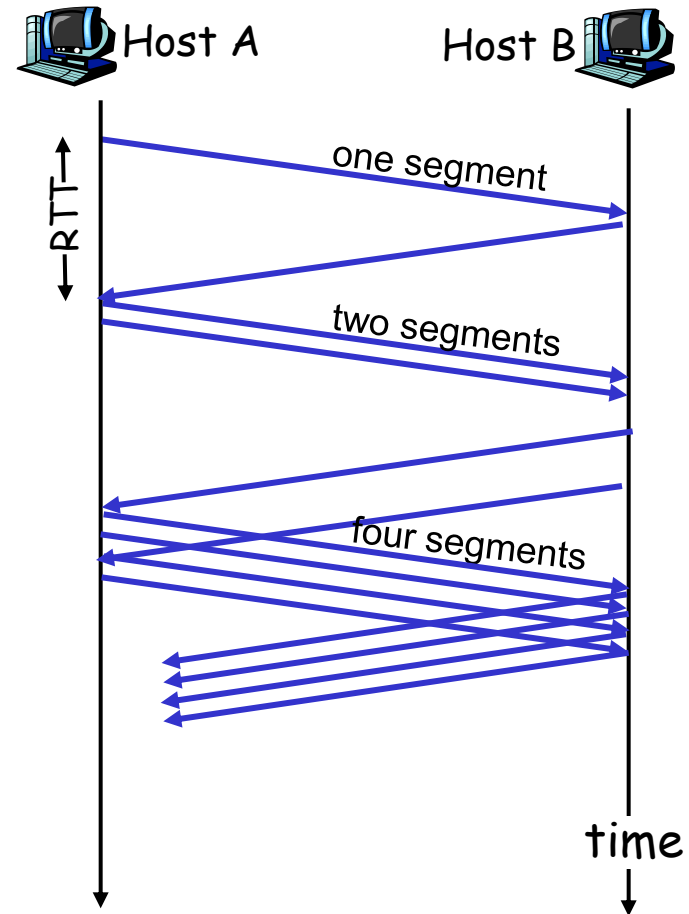
- ❖ loss event = timeout *or* 3 duplicate acks
- ❖ TCP sender reduces rate (cwnd) after loss event

three mechanisms:

- AIMD
- slow start
- conservative after timeout events

TCP Slow Start

- ❖ when connection begins, increase rate exponentially until first loss event:
 - initially `cwnd = 1 MSS`
 - double `cwnd` every RTT
 - done by incrementing `cwnd` for every ACK received
- ❖ summary: initial rate is slow but ramps up exponentially fast



Refinement: inferring loss

- ❖ after 3 dup ACKs:
 - cwnd is cut in half
 - window then grows linearly
- ❖ but after timeout event:
 - cwnd instead set to 1 MSS;
 - window then grows exponentially
 - to a threshold, then grows linearly

Philosophy:

- ❖ 3 dup ACKs indicates network capable of delivering some segments
- ❖ timeout indicates a "more alarming" congestion scenario

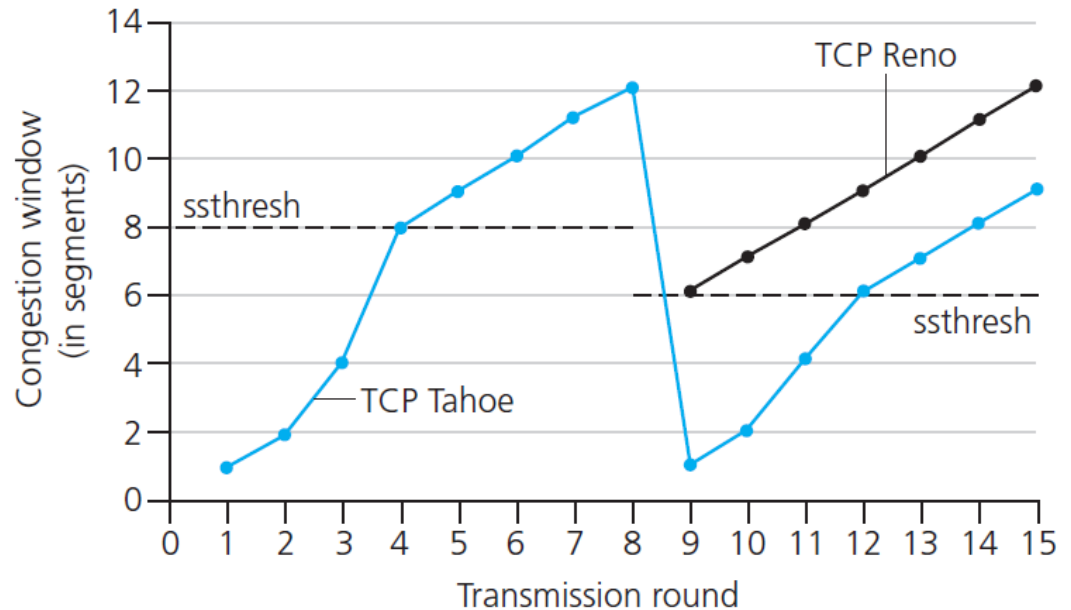
Refinement

Q: when should the exponential increase switch to linear?

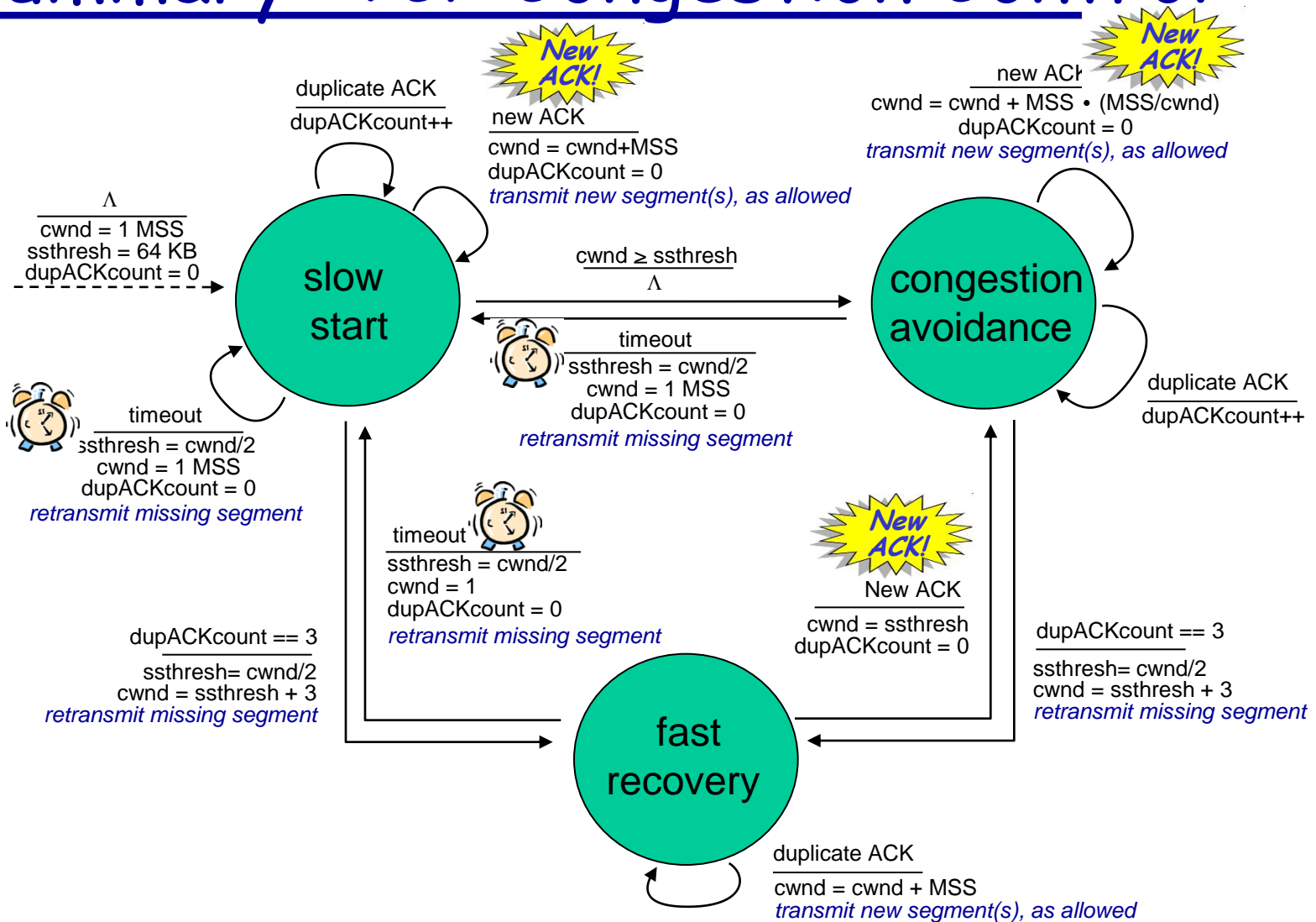
A: when `cwnd` gets to 1/2 of its value before timeout.

Implementation:

- ❖ variable `ssthresh`
- ❖ on loss event, `ssthresh` is set to 1/2 of `cwnd` just before loss event



Summary: TCP Congestion Control



שאלה ממבחן (2009/10)

- נתון לקוח שמריץ אלגוריתם TCP Reno
- בזמן t_1 קרה אירוע ששינה את $cwnd$ מ-62KB ל-34KB.
- מה המאורע?
 - $MSS = 1KB$
 - התקבלו 3 dup acks. כאשר TCP נמצא ב-CA או ב-SS ומתקבלים 3 dup acks הוא נכנס ל-Fast Recovery וקובע:

$$sshresh = \frac{cwnd}{2}$$

$$cwnd = ssthres + 3MSS$$
- מה ערך $ssthresh$ לאחר המאורע?
 - 31KB, לפי האמור למעלה

שאלה ממבחן (2009/10)

- נתון לקוח שמריץ אלגוריתם TCP Reno
 - בזמן t_2 קרה אירוע ששינה את cwnd מ 34KB ל 1KB.
 - מה המאורע?
 - $MSS = 1KB$
 - אירע timeout. כאשר TCP נמצא ב - CA או ב - SS ויש timeout הוא חוזר ל - SS וקובע:
- $$sshresh = \frac{cwnd}{2}$$
- $$cwnd = 1MSS$$
- מה ערך ssthresh לאחר המאורע?
 - 17KB, לפי האמור למעלה

שאלה ממבחן (מועד א' 2010/11)

- נתבונן בקישור TCP ובו Host A שולח נתונים ל – Host B.
- הנתונים המוצגים נלקחו אחרי שהערוץ כבר פעיל זמן מה.
- אנו עוקבים אחר התנהגות ה – Congestion Control של A לפי המודל המפושט של TCP Reno שהוצג בהרצאה ובו:
 - חלון העומס cwnd משתנה רק אחרי השלמת השליחה של חלון cwnd שלם וקבלת כל ה – acks שלו.
 - אם באמצע סיבוב cwnd עובר את ssthresh, אזי בסוף הסיבוב ערכו של cwnd הוא בדיוק ssthresh.
 - אם היה בסיבוב גם 3 dup acks וגם timeout, תהיה מדיניות הירידה לפי timeout.
 - אין שימוש ביוריסטיקה שמאפשרת להגדיל את cwnd עבור dup acks שהתקבלו כבר ב – fast retransmit (בפרט, לא מגדילים את cwnd להיות ssthresh+3 אלא פשוט קובעים אותו להיות ssthresh).

שאלה ממבחן (מועד א' 2010/11)

ש ל ב	cc State	ssthresh	cwnd	ארוע מיוחד בשלב זה
1				
2		33,000		
3				
4			4,000	
5			8,000	
6			16,000	
7			18,000	
8			20,000	

- מהו ה - MSS של הקישור?
 □ 2,000, לפי שלבי ה - CA.

שאלה ממבחן (מועד א' 2010/11)

ש ל ב	cc State	ssthresh	cwnd	ארוע מיוחד בשלב זה
1	SSt / CA	33,000	16,000 / 34,000	אין
2	SSt / CA	33,000	32,000 / 36,000	Timeout
3	SSt	16,000 / 18,000	2000	אין
4	SSt	"	4,000	אין
5	"	"	8,000	אין
6	CA / S.St	"	16,000	אין
7	CA	"	18,000	אין
8	"	"	20,000	לא ידוע

- מלאו את החלקים החסרים בטבלה עפ"י הנתונים המופיעים בה.
- בטור "ארוע" יש לרשום אם ארע Timeout או 3Dup עבור אחד הסגמנטים ששודרו במסגרת החלון המתואר בשורה זאת. אם לא ארע ארוע כזה, יש לרשום "אין".
- קיבלנו גם תשובות שאמרו ש – ssthresh בשלב 3 אינו יכול להיות 18,000 (עם כל השינויים שנגזרים מכך), מפני שאז בשלב 0 (השלב שלפני שלב 1), נקבל ש – cwnd בדיוק עובר את ssthresh. לפי חוקי המינימום שהוגדרו לגרסת TCP הפשוטה שבשאלה, זה אומר שבשלב 1 צריך להיות $cwnd = 33,000$, מה שלא ייתכן מנתוני השאלה.

ש ל ב	ארוע מיוחד בשלב זה	CC State	ssthresh	cwnd
1	אין	SS / CA	33,000	16,000 / 34,000
2	Timeout	SS / CA	33,000	32,000 / 36,000
3	אין	SS	16,000 / 18,000	2000
4	אין	SS	"	4,000
5	אין	"	"	8,000
6	אין	SS / CA	"	16,000
7	אין	CA	"	18,000
8	Timeout / 3 dup acks	"	"	20,000
9	לא ידוע	SS / CA	10,000	2,000 / 10,000

שאלה ממבחן (מועד א' 2010/11)

- מלאו את שורה 9 כהמשך לטבלה הקודמת בהנחה שבשלב זה ערך cwnd ירד.

שאלה ממבחן (מועד א' 2010/11)

שלב	cwnd
7.0	18,000
7.1	18,222
7.2	18,442

- נניח עתה כי בשלב 7 הערוץ פועל לפי TCP Reno "אמיתי", אשר מכונת המצבים שלו הוצגה בהרצאה.
- כלומר cwnd מתעדכן אחרי קבלת האישור של כל סגמנט. רשמו את cwnd אחרי קבלת האישורים של שני הסגמנטים הראשונים.

- זכור את הנוסחה ממכונת המצבים

$$cwnd = cwnd + MSS \cdot \left(\frac{MSS}{cwnd} \right)$$

שאלה ממבחן (מועד א' 2010/11)

- הסבירו מדוע מדיניות TCP Reno בתגובה ל- Triple Duplicate שונה מהמדיניות במקרה של Timeout.
- כאשר יש 3 dup acks סימן שאמנם היה loss, אבל זה גם סימן שהרשת העבירה סגמנטים כשורה לצד השני, כך שהיא עדיין מתפקדת, ולכן אפשר לנקוט בצעדים פחות דרסטיים להתמודדות עם ה- congestion.
- כשיש Timeout אין לנו אינדיקציה שהרשת בכלל מתפקדת, ולכן אנחנו מניחים congestion חמור יותר.