

## TCP Flow & Congestion Control

Kurose & Ross, Chapter 3.5.5, 3.7 (5<sup>th</sup> ed.)

Many slides adapted from:  
 J. Kurose & K. Ross \ Computer Networking: A Top Down Approach (5<sup>th</sup> ed.) Addison-Wesley, April 2009.  
 Copyright 1996-2010, J.F Kurose and K.W. Ross, All Rights Reserved.

**Communication Networks**  
**(0368-3030) / Spring 2011**  
 The Blavatnik School of Computer Science,  
 Tel-Aviv University

Allon Wagner

### 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

### 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-expect seq. # . Gap detected	Immediately send <b>duplicate ACK</b> , 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

Transport Layer 3-4

Transport Layer 3-3

### 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

Transport Layer 3-6

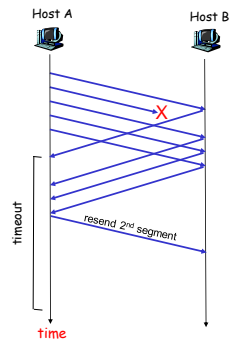
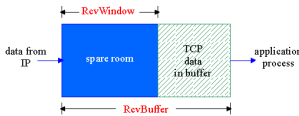


Figure 3.37 Resending a segment after triple duplicate ACK

Transport Layer 3-5

## TCP Flow control: how it works



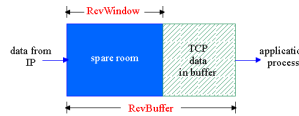
(suppose TCP receiver discards out-of-order segments)

- ❖ spare room in buffer = RcvWindow
- = RcvBuffer - [LastByteRcvd - LastByteRead]

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

Transport Layer 3-8

## TCP Flow Control



- ❖ receive side of TCP connection has a receive buffer:
- ❖ app process may be slow at reading from buffer

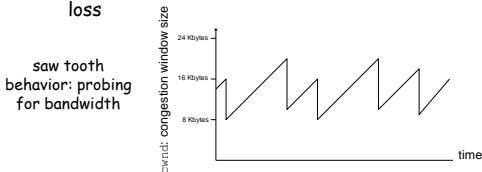
**flow control**  
sender won't overflow receiver's buffer by transmitting too much, too fast

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

Transport Layer 3-7

## 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



Transport Layer 3-10

## 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

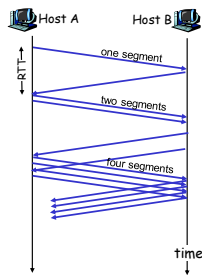
### network-assisted congestion control:

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

Transport Layer 3-9

## 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



Transport Layer 3-12

## 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

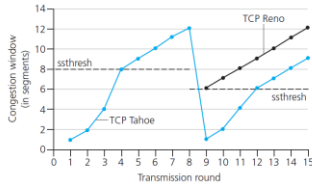
### three mechanisms:

- AIMD
- slow start
- conservative after timeout events

Transport Layer 3-11

## 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

Transport Layer 3-14

## 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

Transport Layer 3-13

3-16

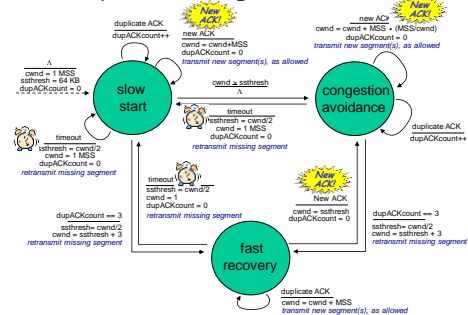
## שאלה ממבחן (2009/10)

- נתון לקוח שמריץ אלגוריתם TCP Reno
- בזמן  $t_1$  קרה אירוע ששינה את *cwnd* מ-34KB ל-62KB
- מה המאורע?
  - MSS = 1KB
  - התקבלו 3 dup acks נמצא ב-CA או ב-SS
  - ומתקבלים 3 dup acks הוא נכנס ל-Fast Recovery וקובע:
 
$$ssthresh = \frac{cwnd}{2}$$

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

Transport Layer

## Summary: TCP Congestion Control



Transport Layer 3-15

3-18

## שאלה ממבחן (מועד א' 2010/11)

- נתבונן בקישור TCP ובו Host A שולח נתונים ל-Host B.
- הנתונים המוצגים נלקחו אחרי שהערוך 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 האcks 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*).

Transport Layer

## 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:
 
$$ssthresh = \frac{cwnd}{2}$$

$$cwnd = 1MSS$$
- מה ערך *ssthresh* לאחר המאורע?
  - 17KB לפי האמור למעלה

Transport Layer

### שאלה ממבחן (מועד א' 2010/11)

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

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

### שאלה ממבחן (מועד א' 2010/11)

ש ל ב	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)

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.
- זכרו את הנוסחה ממכונת המצבים  $cwnd = cwnd + MSS \cdot \left(\frac{MSS}{cwnd}\right)$

### שאלה ממבחן (מועד א' 2010/11)

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

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

### 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.