

# 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, white, and light blue) extending from the right side of the slide towards the center.

# 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.

# 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

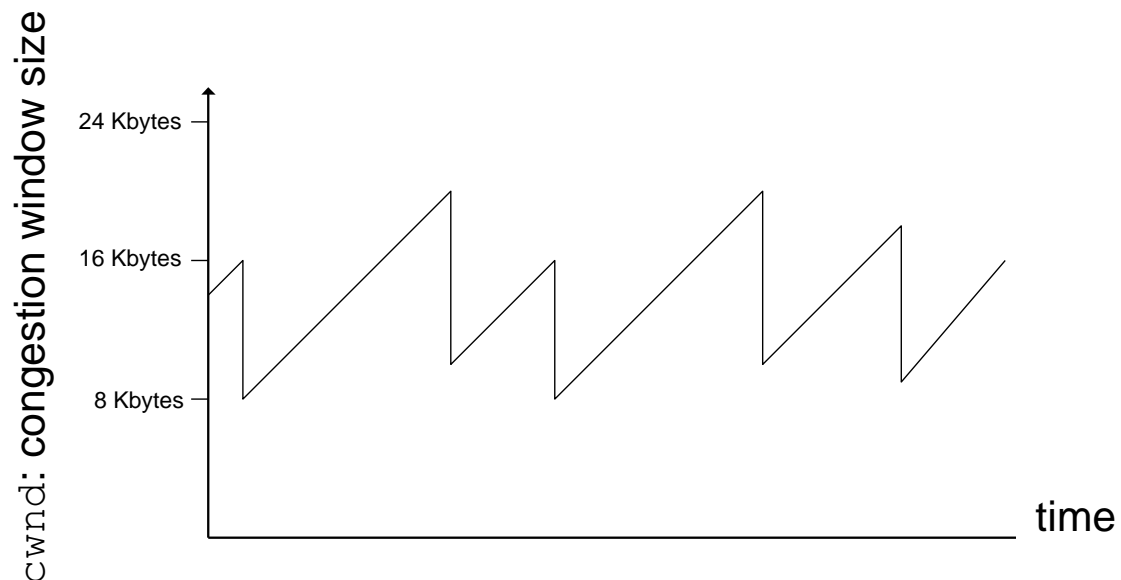
## network-assisted congestion control:

- ❖ 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?

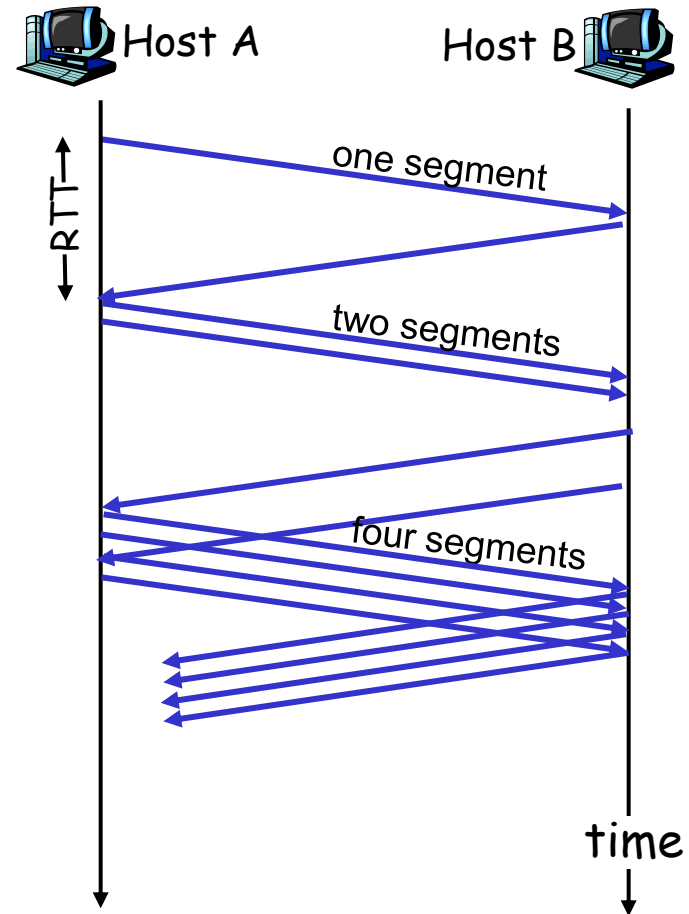
- ❖ loss event = timeout or 3 duplicate acks
- ❖ TCP sender reduces rate (cwnd) after loss event

## 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

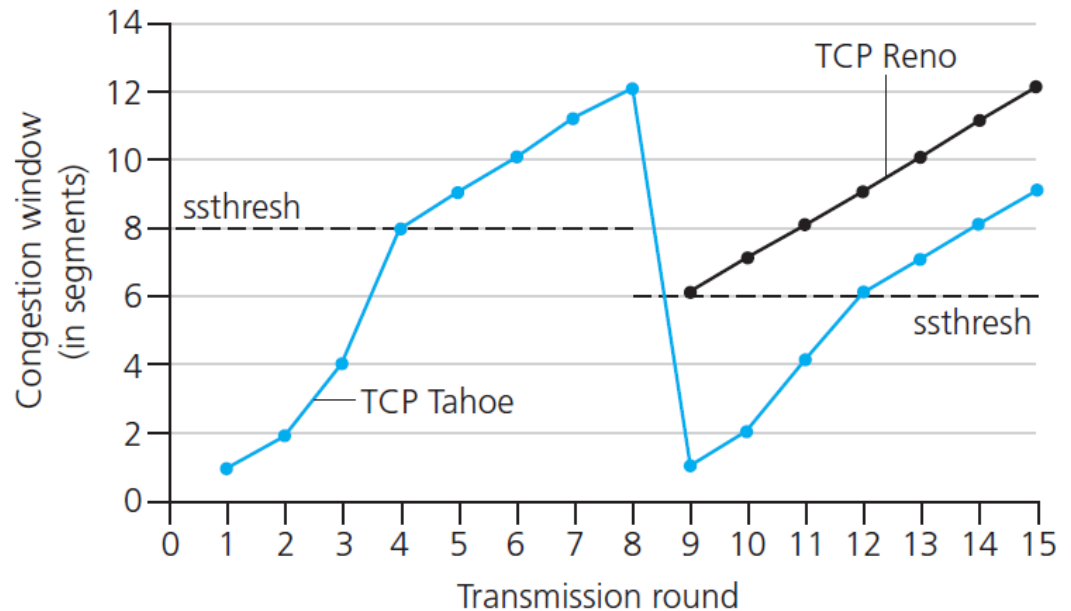
# Refinement

**Q:** when should the exponential increase switch to linear?

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

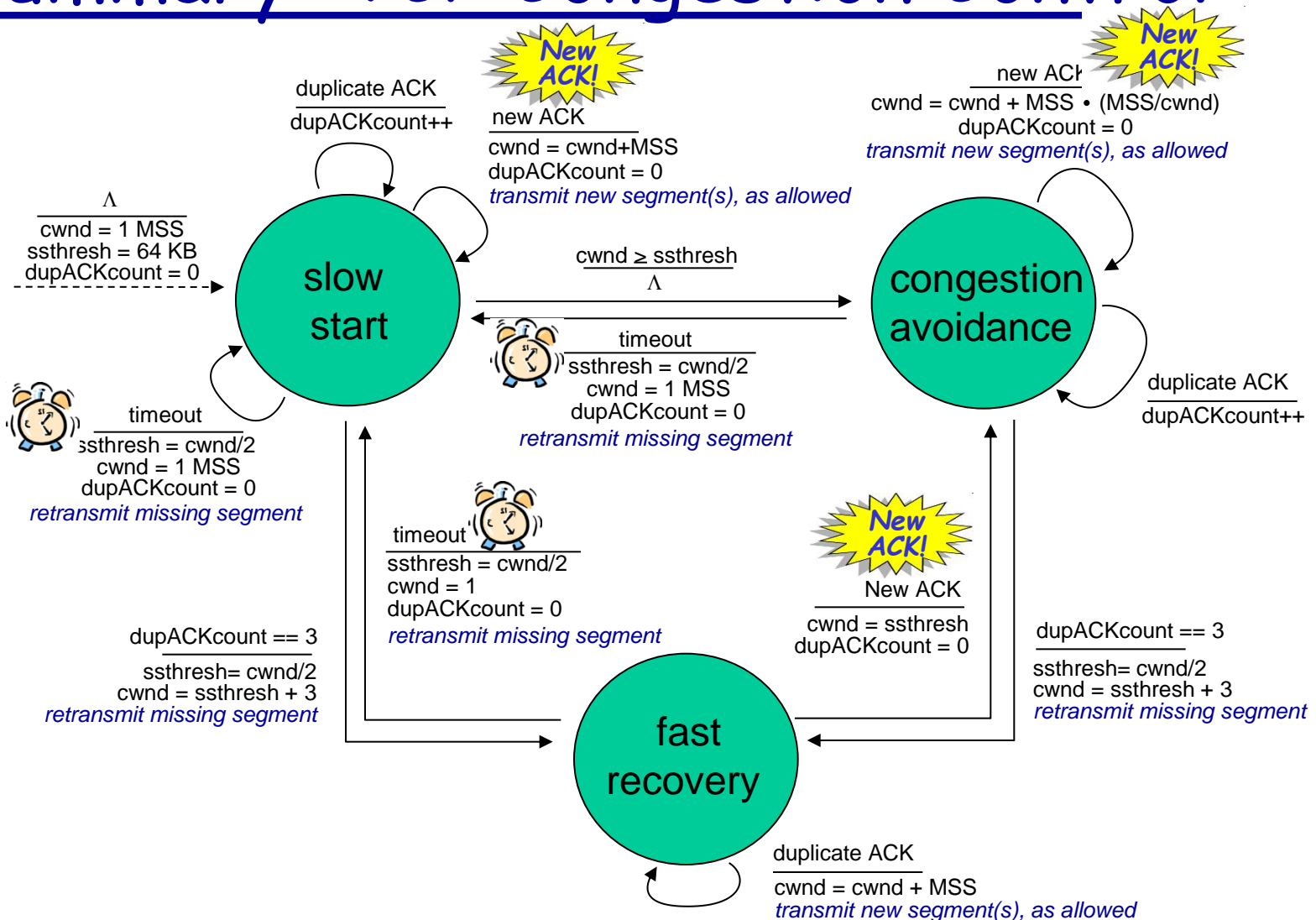
## Implementation:

- ❖ 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:
 
$$ssthresh = \frac{cwnd}{2}$$

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

## שאלה ממבחן (2009/10)

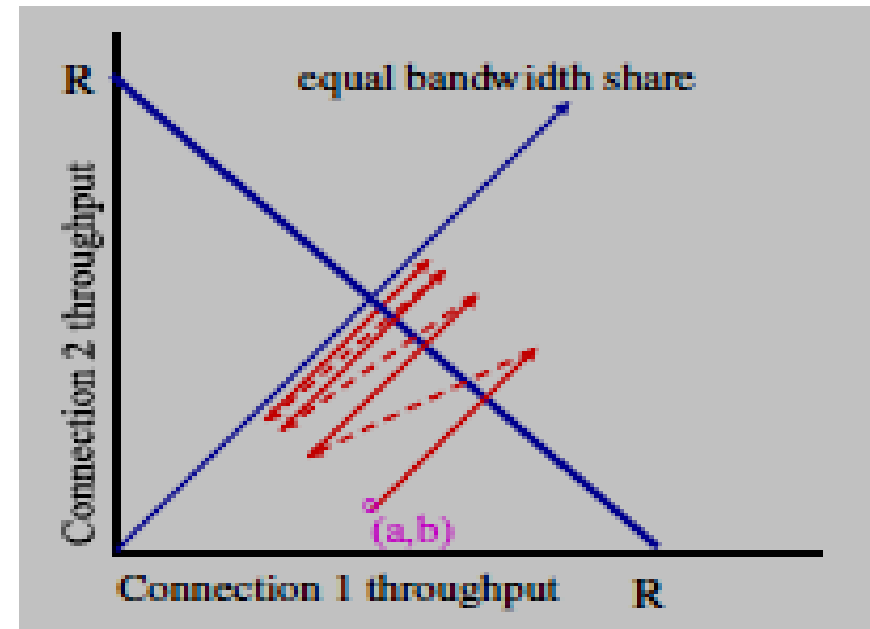
- נתון לקוח שמריץ אלגוריתם TCP Reno
  - בזמן  $t_2$  קרה אירוע ששינה את cwnd מ 34KB ל 1KB.
  - מה המאורע?
    - $MSS = 1KB$
    - אירע timeout. כאשר TCP נמצא ב - CA או ב - SS ויש timeout הוא חוזר ל - SS וקובע:
- $$sshresh = \frac{cwnd}{2}$$
- $$cwnd = 1MSS$$
- מה ערך ssthresh לאחר המאורע?
    - 17KB,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

## Exam question (2011/12)

- We learnt that the congestion control of TCP is based on the AIMD (Additive Increase, Multiplicative Decrease) principle. In which phase of TCP's congestion control is this principle implemented precisely?
- In the CA phase when there are only dup acks and not timeouts.
- Every RTT cwnd grows in 1MSS (approximately) → additive increase
- When there is a dup ack, cwnd drops to half its former value ( $\pm 3$  because of the fast retransmit phase) → multiplicative decrease.

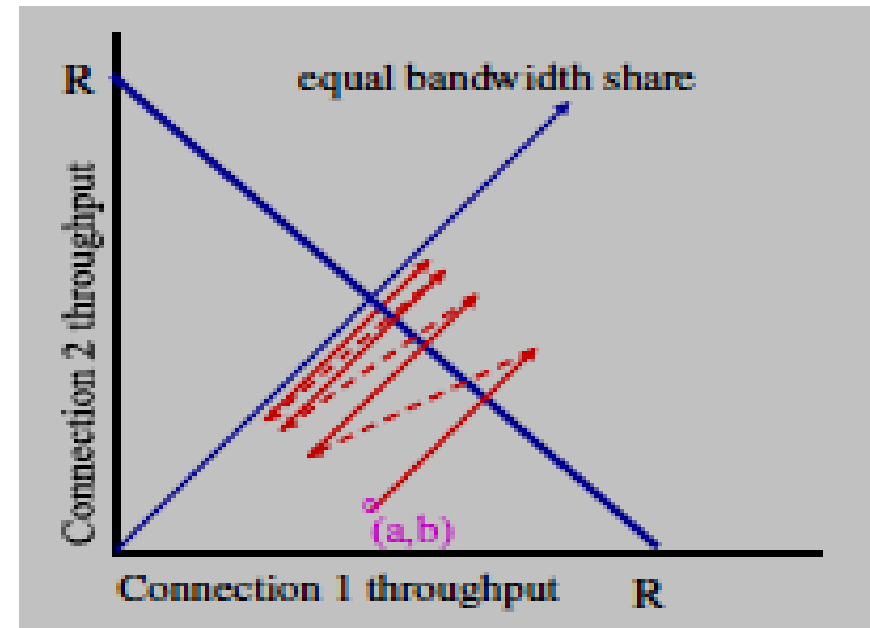
# Exam question (2011/12)

- We saw that the AIMD principle allocates network resources optimally.
- Explain what does the figure represents, and how does it supports AIMD fairness
- Each axis: the throughput of one of the clients.
- Left-to-right diagonal: fair allocation. Each client gets the same share



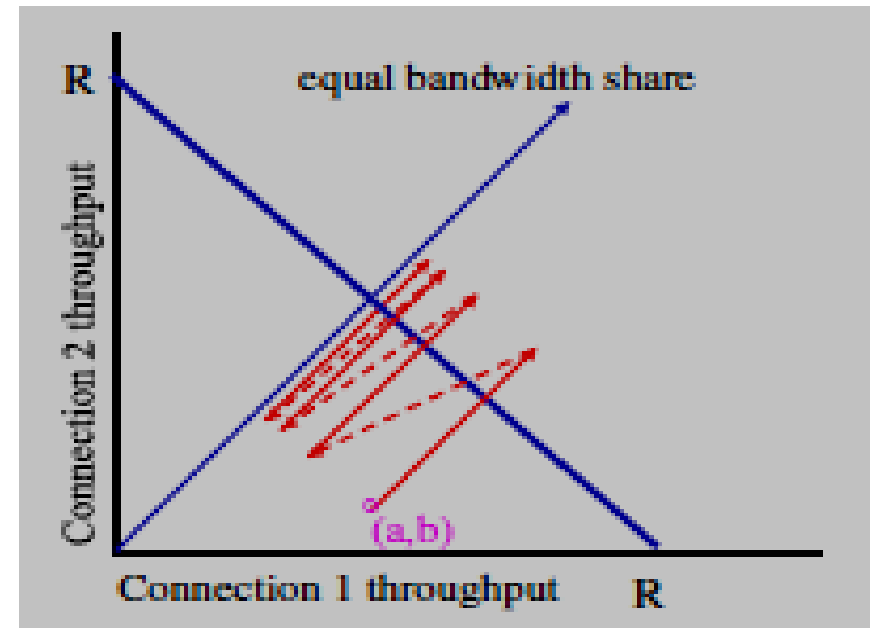
# Exam question (2011/12)

- Right-to-left diagonal: efficient allocation. The bandwidth is fully utilized.
- Each axis: the throughput of one of the clients.
- Their intersection: optimal allocation (fair and efficient).
- (a,b): arbitrary starting point for the two connections



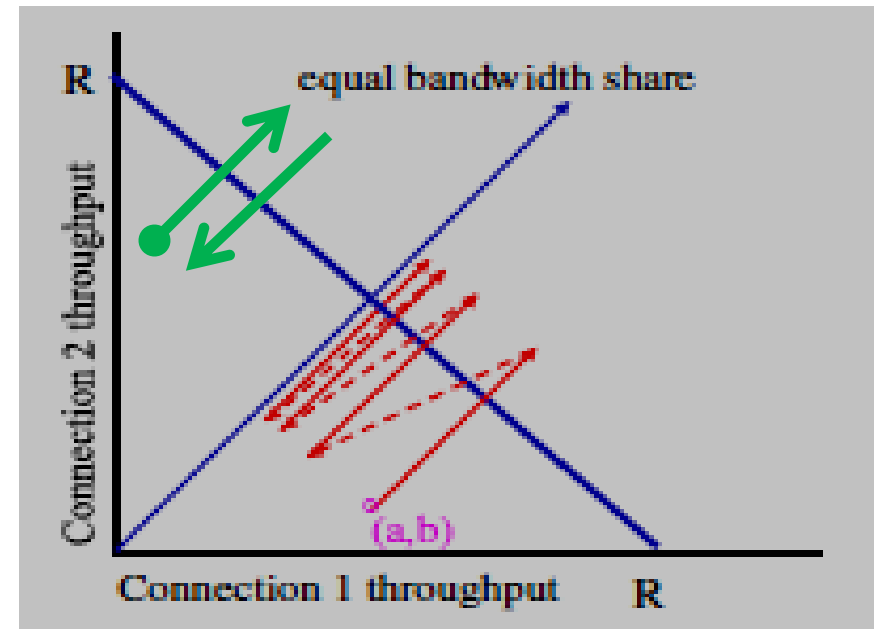
# Exam question (2011/12)

- Solid lines: additive increase phases
- Dashed lines: multiplicative decrease events
- We can see the connections converge to the optimal point
- This happens no matter where the starting point  $(a,b)$  was.



# Exam question (2011/12)

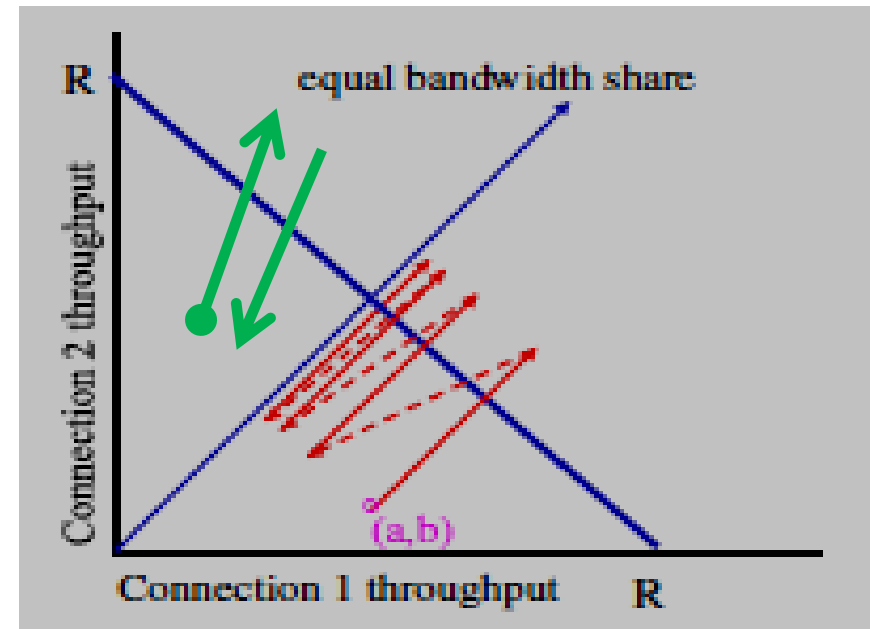
- What will happen if on any loss event we decrease cwnd by 2 MSS?
- (i.e., additive decrease instead of multiplicative decrease)
- In every step a constant factor is added to the current point (actually one of two constant factors - either because of an increase or a decrease).
- Thus, all intermediate points are of the form  $(a + c, b + c)$  with  $c = c_1 + c_2 + \dots + c_n$ .
- Oscillation along an additive line:  
 $y = x + b - a$ .
- (45° angle, i.e., slope = 1 and intersects the y-axis according to the starting point)





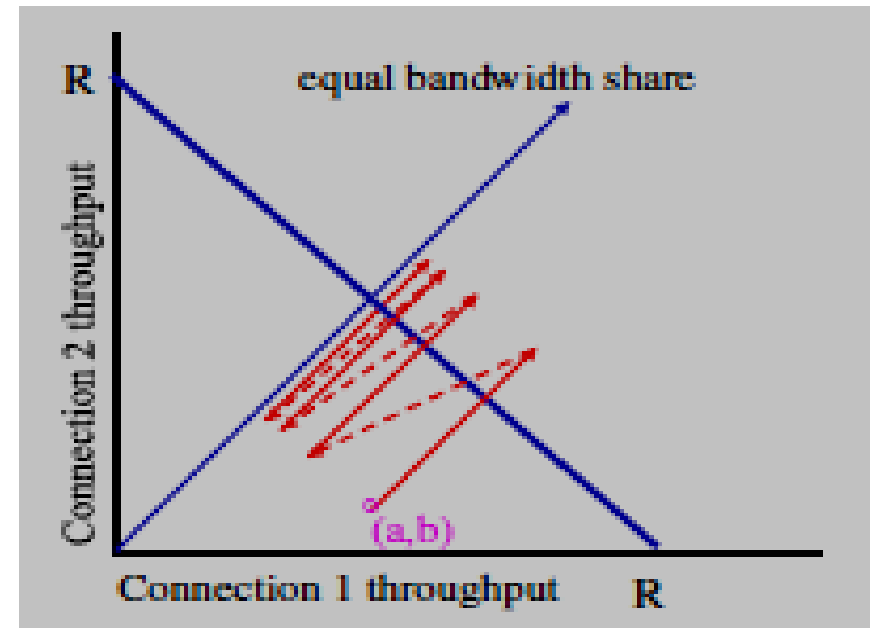
# Exam question (2011/12)

- What will happen if in every RTT we increase cwnd by 10%?
- (i.e., multiplicative increase instead of additive increase)
- In every step the current point is multiplied by a constant factor (actually one of two constant factors - either because of an increase or a decrease).
- Thus, all intermediate points are of the form  $(ca, cb)$  with  $c = c_1 c_2 \cdots c_n$ .
- Oscillation along a multiplicative line:  $y = \frac{b}{a} x$
- (points to the origin, slope determined by the starting point).



# Exam question (2011/12)

- What will happen if we do multiplicative increase additive decrease?!
- Divergence from optimality by exactly the same plot as the original
- Simply traverse the lines backwards.

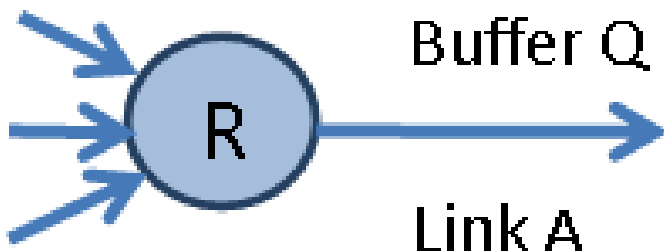


# Exam question (2011/12)

- Does AIMD ensures fairness “in the wild”?
- No. For example:
  - Applications that use UDP do not reduce their sending rate on loss, so they will eventually choke TCP connections.
  - An application that initiates multiple parallel TCP sessions will receive an unfair allocation.
  - TCP does additive increase on every RTT (and not on every global time unit), and so the allocation is biased in favor of short-distance sessions.

## שאלה ממבחן (2011/12)

- בשאלה זו נדון בשיטת Random Early Detection (RED) לטיפול בעומס ע"י הנתבים (routers) ברשת.
- נתון נתב R עם מספר כניסות. נניח שכל הלקוחות העוברים דרכו משתמשים ב-TCP Reno. כל הלקוחות שולחים מידע היוצא מ-link A ולכולם יש כל הזמן מידע חדש לשלוח.
- הקצב של הלינק מוגבל כמובן, ולכן יש תור (Buffer) שנסמן Q ובו מאוחסנות פקטות הממתינות לתורן להשלח.
- אם הלינק לא מספיק לשלוח את הפקטות החוצה בקצב שבו הן מצטברות בתור, התור ילך ויגדל עד שיתמלא.



## שאלה ממבחן (2011/12)

### פעולת הנתב ללא RED

- נניח שהתור מלא. כשמגיעה הפקטה הבאה, הנתב זורק אותה (tail-drop policy). תארו את התהליך שיוביל להקלה בעומס ברשת כתוצאה מכך. התייחסו בתשובתכם לפעולות של TCP אצל ה-host אליו מיועדת הפקטה שנזרקה, ואצל ה-host ששלח אותה
- כשהפקטות הבאות אחרי זו שנזרקה יגיעו ליעד יוצרו duplicate acks שיביאו להפעלת מנגנון ה-congestion control אצל השולח: הקטנת ה-threshold לחצי מ-cwnd בעת ה-loss event ומעבר ל-FR ואח"כ אל CA.
- לחילופין, אם נזרקות הרבה פקטות של השולח ולא יוצרו מספיק dup-acks יקרה אצל השולח timeout והוא יוריד את cwnd ל-1 ואת thresh כמתואר קודם ויעבור ל-SS.

## שאלה ממבחן (2011/12)

### פעולת הנתב ללא RED

- תוך כמה זמן מרגע שמתחילות להזרק פקטות צפוי הראוטר להרגיש תחילת הקלה בעומס המגיע אליו? (רמז: נסחו את התשובה במונחי RTT).
- לפי הנחות השאלה, מייד אחרי הפקטה שנזרקה ממשיכות להשלח עוד פקטות שעוברות דרך הראוטר. מהרגע שבו הפקטות הללו עוברות דרך R, עד שהן מגיעות ליעד (ויוצרות dup acks), ה – acks חוזרים מהיעד אל המקור, המקור מוריד את הקצב שלו ולראוטר מתחילות להגיע פקטות בקצב נמוך יותר עובר בערך RTT אחד.
- לחילופין, אם אצל השולח קורה timeout הוא גם קורה בערך אחרי RTT אחד (מרגע שהפקטה נשלחה) ולכן השינוי מורגש אצל הראוטר כעבור RTT מהזמן שבו הראוטר זרק את החבילה.

## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

RED מנסה לאתר עומס עוד לפני שהוא נוצר ולמנוע אותו מראש. נסמן ב-  $Q.Count$  את מספר הפקטות הממתכות כרגע בתור  $Q$ . יהיו  $MAX\_THRESH$  ו-  $MIN\_THRESH$  שני קבועים שחוסמים את גודל התור.

אלגוריתם RED הוא:

- כשמגיעה פקטה חדשה שרוצה להכנס לתור:

- אם  $Q.Count < MIN\_THRESH$ , קבל את הפקטה לתור.
- אחרת, אם  $Q.Count > MAX\_THRESH$ , זרוק את הפקטה.
- אחרת, זרוק את הפקטה בהסתברות  $p$  וקבל אותה לתור בהסתברות  $1 - p$ .

לשם פשטות, נניח ש-  $p \in (0,1)$  הוא קבוע (במציאות זה לא כן).

## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

- כיצד RED מסייע להקל על העומס ברשת?
- אבדן הפקטות קורה עוד לפני שמגיעים למצב של קיבולת מלאה. באופן הזה הרשת מאותתת ל – hosts שיש עומס ומאפשרת להם להוריד את הקצב לפני שהעומס אינו מאפשר יותר לרשת לתפקד.



## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

- נניח שחלק מהלקוחות שולחים תעבורה שהיא bursty – כלומר, יש פרקי זמן קצרים שבהם הם שולחים כמות גדולה של מידע, ובשאר הזמן הם משדרים בקצב נמוך. מה הבעיה המתעוררת?
- במקרה כזה יכול להיות שנקבל פרץ (burst) של מידע שיעלה אותנו מעבר ל – MIN\_THRESH ולכן נזרוק חלק מהפקטות. לאחר מכן הרשת תהיה שקטה לזמן מסויים ובזמן הזה היינו יכולים לרוקן את התור בשקט – מסתבר שזרקנו את המידע לחינם. הבעיה היא בעצם שאומדן רגעי של גודל התור אינו מספק באמת מדד טוב לעומס ברשת בנוכחות bursts.

## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

- על מנת להתמודד עם הבעיה הוצע להשתמש במשתנה חדש  $WA$ . האלגוריתם החדש הוא:
  - $WA \leftarrow 0$
  - כשמגיעה פקטה חדשה שרוצה להכנס לתור:
    - אם  $WA < MIN\_THRESH$ , קבל את הפקטה לתור.
    - אחרת, אם  $WA > MAX\_THRESH$ , זרוק את הפקטה.
    - אחרת, זרוק את הפקטה בהסתברות  $p$  וקבל אותה לתור בהסתברות  $1 - p$ .
  - $WA \leftarrow (1 - \alpha)WA + \alpha \cdot (Q.Count)$
- כאשר  $\alpha \in (0,1)$  הוא קבוע כלשהו, וכמו קודם  $p \in (0,1)$  קבוע.

## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

- נניח שכרגע התחלנו את האלגוריתם. מגיעות  $n$  פקטות. נסמן את גדלי התור בעת הגעתן  $x_1, \dots, x_n$ . תנו נוסחה סגורה (כלומר, ללא רקורסיה) לערך של  $WA$ . הסבירו כיצד השינוי פותר את הבעיה.

- $WA = \alpha(1 - \alpha)^{n-1}x_1 + \alpha(1 - \alpha)^{n-2}x_2 + \dots + \alpha x_n$   
 $WA$  הוא exponentially weighted moving average כפי שראינו בהרצאה עבור TCP RTT estimation. הוא ממצע ומחליק לאורך הזמן את הערך של Q.Count וככה מאפשר לנו להעריך יותר טוב מתי הרשת באמת עמוסה.

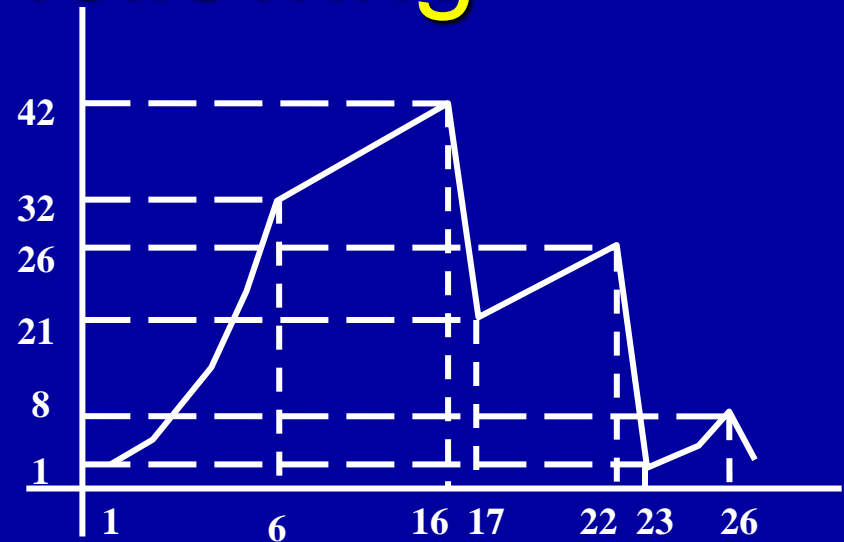
## שאלה ממבחן (2011/12)

### אלגוריתם RED (Random Early Detection)

- האם היה עדיף להחליף את שורה (iv) בשורה:  
 $WA \leftarrow \frac{1}{n} ((n - 1) \cdot WA + Q.Count)$  ? נמקו.  
 לא.

- נשים לב שמשמעות הנוסחה החדשה היא:  $WA = \frac{1}{n} \sum_{i=1}^n x_i$
- בנוסחה המקורית הערכים הישנים של  $x_i$  דועכים אקספוננציאלית כך שהם מאבדים את משמעותם מהר. משמעות השינוי המוצע היא ש-  $WA$  הוא ממוצע חשבוני רגיל של כל הערכים שנמדדו אי פעם  $x_i$ . כלומר, ממוצע שנותן משקל שווה לערכים הישנים ולחדשים ולכן משקף פחות טוב את גודל התור הממוצע ברגע הנוכחי.

# Answer the following



- Identify Slow Start times.
- Identify Congestion Avoidance times.
- At time 16 is it timeout or triple duplicate ACK?
- At time 22, is it timeout or triple duplicate?
- What is the Slow Start Threshold at time 1?
- What is the Slow Start Threshold at time 24?
- If a packet loss is detected after time 26 by the receipt of a triple duplicate ACK, what will be the value of the congestion-window size?

# Answers

- First note: since we see here a loss event followed by CA state (times 16-17) we conclude this is not the Tahoe variant.
- On the other hand, the behavior given in the chart conforms to Reno's behavior (which is the only variant we studied other than Tahoe).
- Identify Slow Start rounds.
  - 1-6 and 23-26: where we see exponential increase of cwnd
- Identify Congestion Avoidance rounds
  - 6-16, 17-22: where we see linear increase of cwnd

# Answers

- What is the Slow Start Threshold at time 1?
  - 32, since this is the threshold at which SS becomes CA
- What is the Slow Start Threshold at time 24?
  - Every loss event causes ssthresh to become half the size of cwnd on the time the loss event was detected (by timeout or 3<sup>rd</sup> dup ack).
  - Thus, ssthresh at time 23 is  $26 / 2 = 13$ .
  - There is no loss in time 23 so this is also the thresh of round 24.
- If a packet loss is detected after time 26 by the receipt of a triple duplicate ACK, what will be the value of the congestion-window size?
  - Again, we set ssthresh to half the size of cwnd on the time the loss event was detected, namely  $8 / 2 = 4$ .