

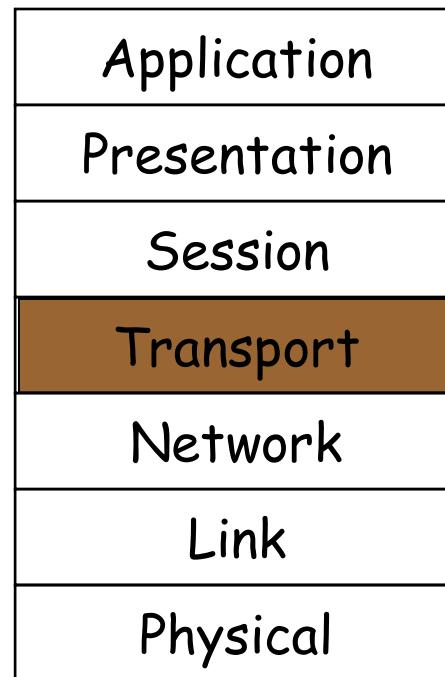
תקשורת מחשבים ואלגוריתמים מבזרים

קורס מס' 202-2-1131

מתרגל: ד"ר גיא לשם leshemg@cs.bgu.ac.il

הרצאה חמישית – שכבת התעבורה

נידוד סכמת תקשורת אוניברסלית ותקשורת



The 7-layer OSI Model

אנו פאכט הילגה

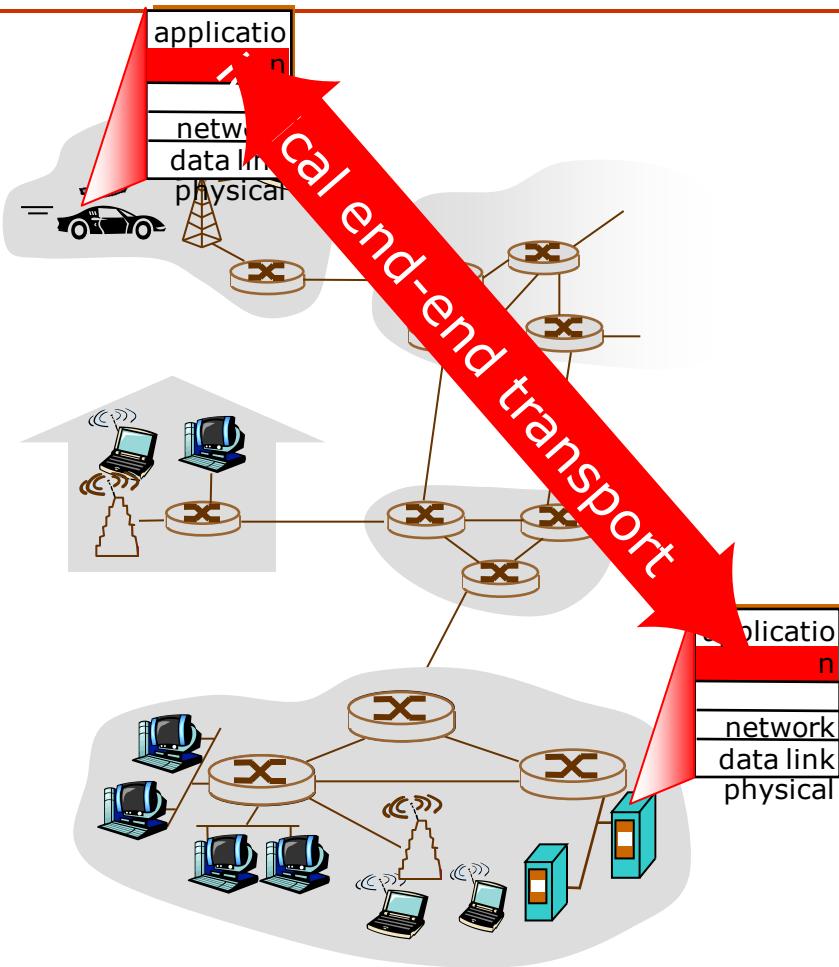
המטרה שלנו:

- **להבין את העקרונות העומדים מאחוריו השירותי שכבת הובלה.**
 - ריבוב (demultiplexing) / פילוג (multiplexing)
 - העברת אמינה של מידע.
 - בקרת זרימה.
 - בקרת עומסים.
- **ללמוד על פרוטוקלי שכבת הובלה באינטרנט**
 - UDP: תעבורה חסרת קשר (connectionless).
 - TCP: תעבורה מונחת קשר (connection).
 - TCP: בקרת עומסים (flow control).

עכמת התקשורת

- **שירותי שכבת ההובלה.**
- ריבוב/פילוג.
- עקרונות של בקרת עומסים.
- עקרונות תעבורה מידיע אמינה ברשת.
- **תעבורה מונחת קשר: TCP**
 - מבנה המקטע.
 - תעבורת מידע אמינה.
 - בקרת זרימה.
 - ניהול הקשר

שלומי סכמת התקשורת ופלוואוקויל



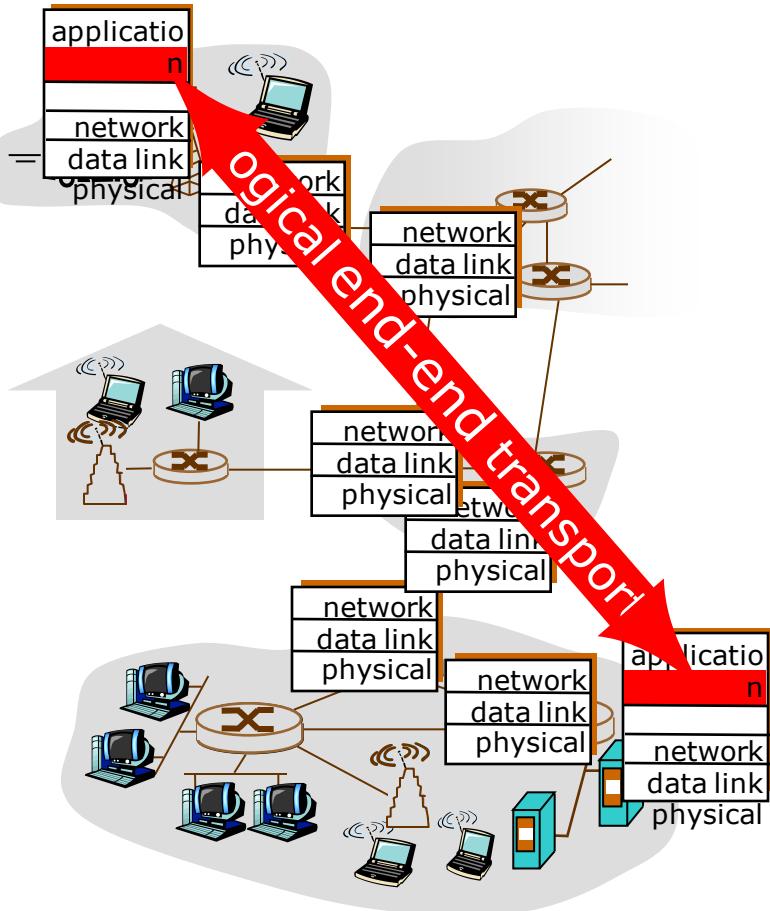
- ◻ **שירותי ופרוטוקולי התעבורה** מספקים חיבור לוגי בין שני מחשבים ברשת. שני יישומים מחוברים כביכול דרך ערוץ ישיר.
- ◻ **פרוטוקולי התעבורה** רצים במחשיبي הקצה: הצד השולח מחלק את המסרים לסגמנטים ו מעביר אותם לרמת הרשת. הצד מקבל מרכיב מחדש את הסגמנטים למסרים ו מעביר אותם לרמת היישום.
- ◻ **קיימים מספר פרוטוקולי תעבורה, העיקריים באינטרנט הם TCP ו UDP**

כאמת החלטת גזואם כאמת המתקשרות

- אנלוגיה: קחו לדוגמה שני בתים האחד בתל-אביב והשני בירושלים. בכל בית עשרה ילדים. כל ילד בכל בית שלוח מכתב לכל אחד מהילדים בבית השני, כל מכתב במעטפה נפרד, וכך מכל בית נשלחים 100 מכתבים לבית השני. בכל בית ישנו ילד שתפקידו לאסוף את המכתבים לשילחה אחיו, להעביר אותם לסניף הדואר ולחולק את המכתבים המגיעים. שירות הדואר מספק קשר לוגי בין בתים.
- מסרים = מכתבים במעטפות.
- תהליכיים = ילדים.
- מחשבי קצה (hosts) = בתים.
- פרוטוקול התעבורה = הילד שתפקידו לאסוף ולחולק מכתבים.
- פרוטוקול הרשת = שירות הדואר, כולל סניפי הדואר.

- רמת הרשות: העברת הודעות בין שני מחשבים.
- רמת התעבורה: העברת תקשורת בין ישומים, בין היישום השולח ליישום מקבל המתאים. רמת התעבורה היא מעל רמת הרשות.

ככים וכך, מזגואה אקיירלן



- **שירות תעבורת אמין TCP:** שירות שבו הודעות מועברות ומתקבלות בסדר הנכון מבלי לבלית לאיבוד. מאפייניהם:
 - בקרת עומס
 - בקרת זרימה
 - הקמת קשר במנגנון handshake
- **שירות תעبورה לא אמין UDP:** שירות **Best effort** - יתכן אי-איבוד הודעות, הגעתן שלא על פי הסדר והכפלתן. בפרוטוקול זה יש הרחבנה מינימלית של רמת ה IP.
- **שירותים שאינם קיימים באינטרנט:**
 - הבטחת רוחב פס מינימלי
 - הבטחת השהייה מינימלית.
 - כדי לתמוך ביישומי זמן אמיתי, כמו טלפוןינה, צריך לתמוך בשירותים האלו.

עכמת התקשורת

- שירותים שכבות הhonebhla.
- **ריבוב/פילוג.**
- עקרונות של בקורת עומסים.
- עקרונות תחבורה מידע
- תעבורת חסרת קשר: UDP.
- בקורת עומסים TCP.
- תעבורת מוגנת קשר: TCP
 - מבנה המקטע.
 - תעבורת מידע אמינה.
 - בקורת זרימה.
 - ניהול הקשר

כינור (de-multiplexing) / אינטראקס (Multiplexing)

□ אנלוגיה בין האינטרנט לרשות טלפון:

טלפון	אינטרנט
מספר טלפון	כתובת IP
שען ומיכיר טלפון	Socket
מספר שלוחה	מספר Port

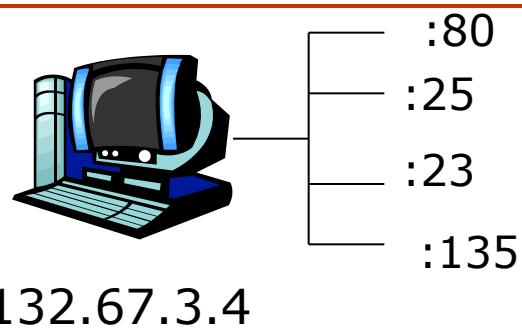
□ אפשר תמיינה במספר יישומים באותו מחשב קטן.

□ **Multiplexing**: המחשב השולח מוסיף למידע header. (זהו socket המקור והיעד לצורך demultiplexing המקלט).

□ **De-multiplexing**: המחשב מקבל מעביר את הסגמנטים המתתקבלים אל ה- socket הנכון, לפי המידע ב- header.

□ **סיכון**: התפקיד החשוב שמבצעת שכבת התקשורת הוא ריבוב (Multiplex) הנתונים השונים שמאגיעים מהיישומים השונים על גבי אפיק אחד, והיכולת להפרידם ⁹ בהתאם (DeMultiplex) בהתקן היעד.

сетותם קיררכז



■ מיעון בرمת הרשת (IP)

■ כתובת IP לכל תחנה ברשת

□ ארבעה בתים, סה"כ 32 ביט

□ אמור להיות ייחודי לכל תחנה

□ לדוגמה: 132.67.3.4

■ מיעון בرمת התחנה (Host)

■ Port

□ לכל תחנה K 64K פורט-ים

□ חלק מה포רטים שמורים לשימוש יישומים מיוחדים

▪ FTP – 21

▪ HTTP – 80 (גילהה)

▪ SMTP – 25 (אימייל)

Port Mechanism

- כל תהליך או אפליקציה המשדרים ב- TCP או ב- UDP מזוהים באמצעות מספר חד-חדרי (בכל Host בן 16 ביט), סה"כ - 64k פורט-ים.
- 1-1023 Ports מוגדרים Well Known Ports ומוגדרים ע"י IANA (רשות האינטרנט להקצתה מספרים (The Internet Assigned Numbers Authority))
 - ניתן למצאים ב: rfc 1700
 - FTP: 21, Telnet: 23, SMTP: 25, DNS: 53, HTTP: 80, POP-3: 110...
- Ports נקראים Registered Ports 1024-49151
- Ports נקראים Dynamic and/or Private ports 49152-65535
- מרבית האימפלמנטציות מקצתות ל Clients - פורטים 1024 - 5000.
- Socket: הכתובת של IP Address ושל ה- Port - נתן זהה חד-חדרי לאפליקציה ברשת.

Local IP: 111.22.3.4
Local Port: 2249



Remote IP: 123.45.6.78
Remote Port: 3726

SOCKET – סוקט

- Sockets מזוודה ע"י צמד Connection {TCP, 129.5.32.11, 1026, 129.5.1.1, 21} ■
יזהה קשר בין ל��וח FTP בכתובת 129.5.32.11 לבין שרת FTP בכתובת 129.5.1.1.
- קיימים 3 סוגי Sockets : Stream Socket ■
: Connectionless Datagram Socket ■
. (TCP) . (UDP)
- גישה ישירה לפרוטוקולים ברמות נמוכות Raw Socket ■
. יותר.

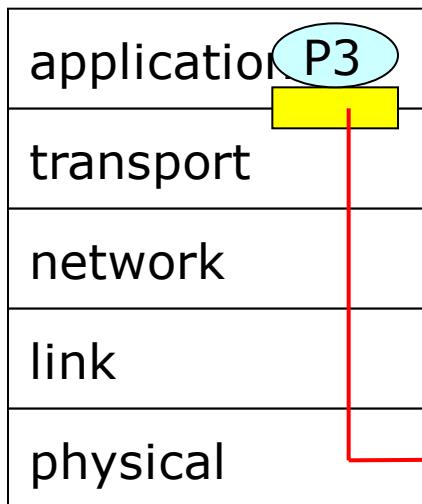
כינור (de-multiplexing) / פירוט (Multiplexing)

Demultiplexing at rcv host:

המחשב המקלט מעביר את הסגמנטים המתקבלים אל ה- socket הנכון, לפי המידע ב-.header.

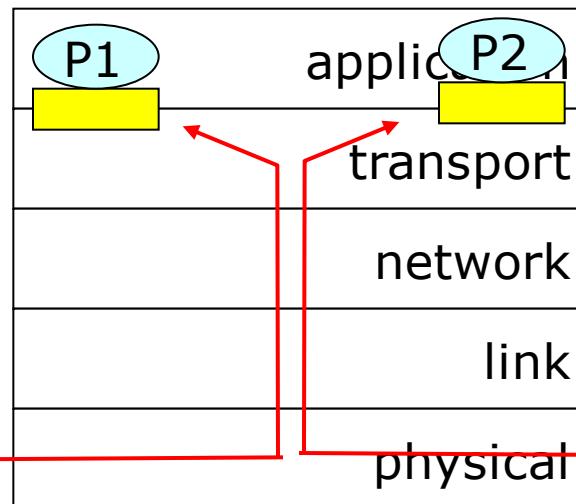
Multiplexing at send host:

המחשב השולח מוסיף למידע socket.header (זיהוי המource והיעד לצורכי demultiplexing במחשב המקלט).



host 1

תקשורות מחשבים ואלגוריתמים מבזרים
(חורף 2010)

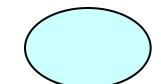


host 2

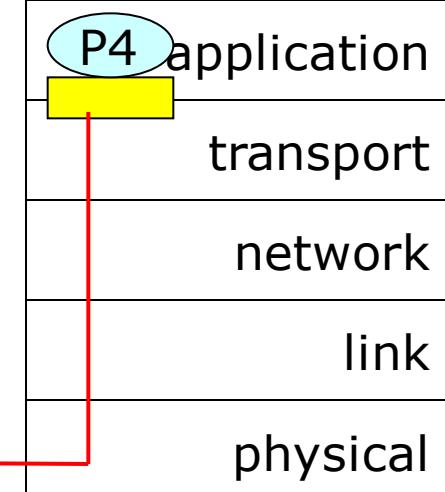
זיא לשם ©



= socket

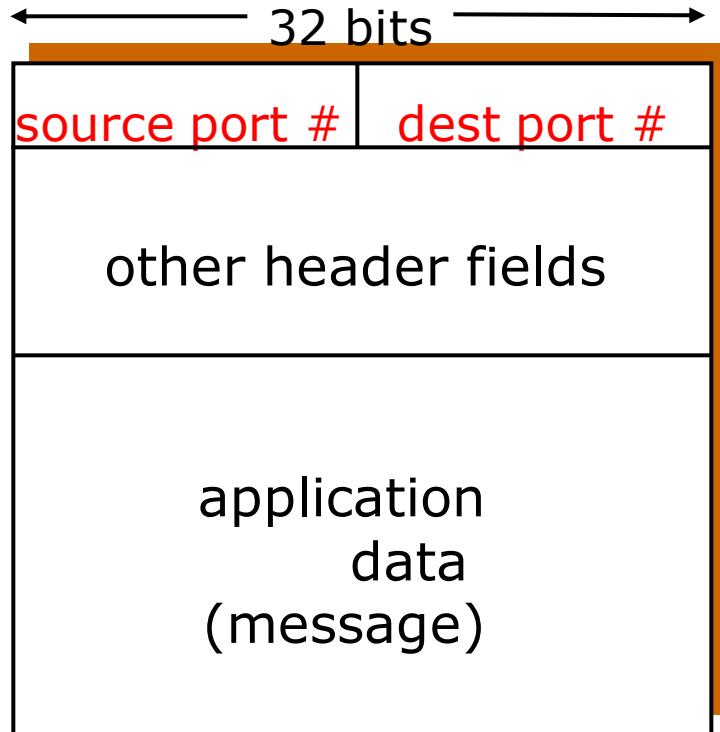


= process¹³



host 3

ק'ק א-סלאם? De-multiplexing



TCP/UDP segment format

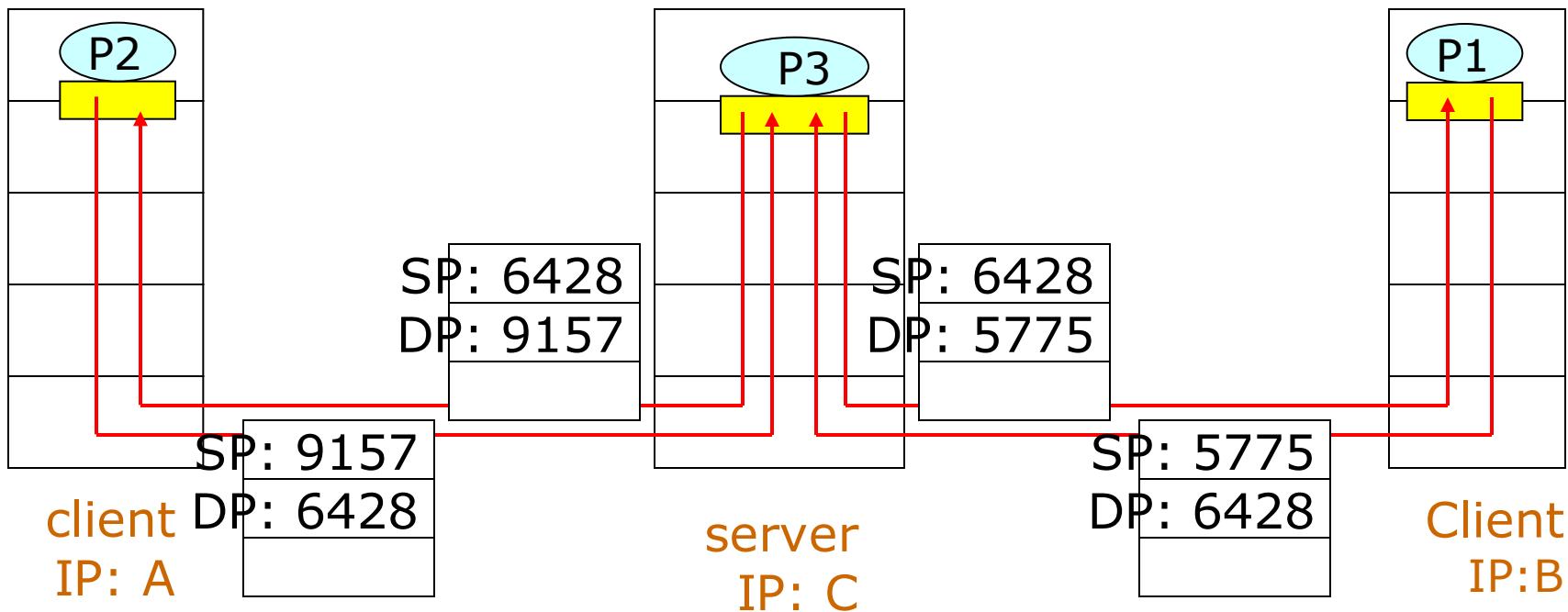
- למחשב מקבל מגיעות חבילות IP. לכל חבילה יש כתובת IP מקור ויעד.
- לכל חבילה יש port מקור ויעד.
- הנמען מתעניין בעיקר בשדה Port יעד. לפי כתובת ה IP ו-port היעד, רמת התמסורת במחשב יודעת להפנות את החבילה ל-socket המתאים.
- Port המקור משמש לצורכי שליחת תשובה.

UDP כפלאוקרי De-multiplexing

- כאשר הנמען מקבל את חבילת ה- UDP:
 - בודק את מס' port היעד.
 - מכoon את החביליה ל-socket עם מס' ה- port המתאים.
- לאוטו socket מופנות חבילות מחשבי מקור שונים בעלי כתובת IP שונה ומס' Port שונה.
- ייצרת socket עם מספרי port מתאימים: המחשב השולח צריך לדעת את מספר ה- port אליו יש לפנות במחשב מקבל כדי לדבר עם אפליקציה מסוימת.
- Socket ה- UDP מצויה ע"י שני שדות:
 - .dest IP address
 - .dest port number

UDP פירוט כפלה De-multiplexing

DatagramSocket serverSocket=new DatagramSocket(6428);

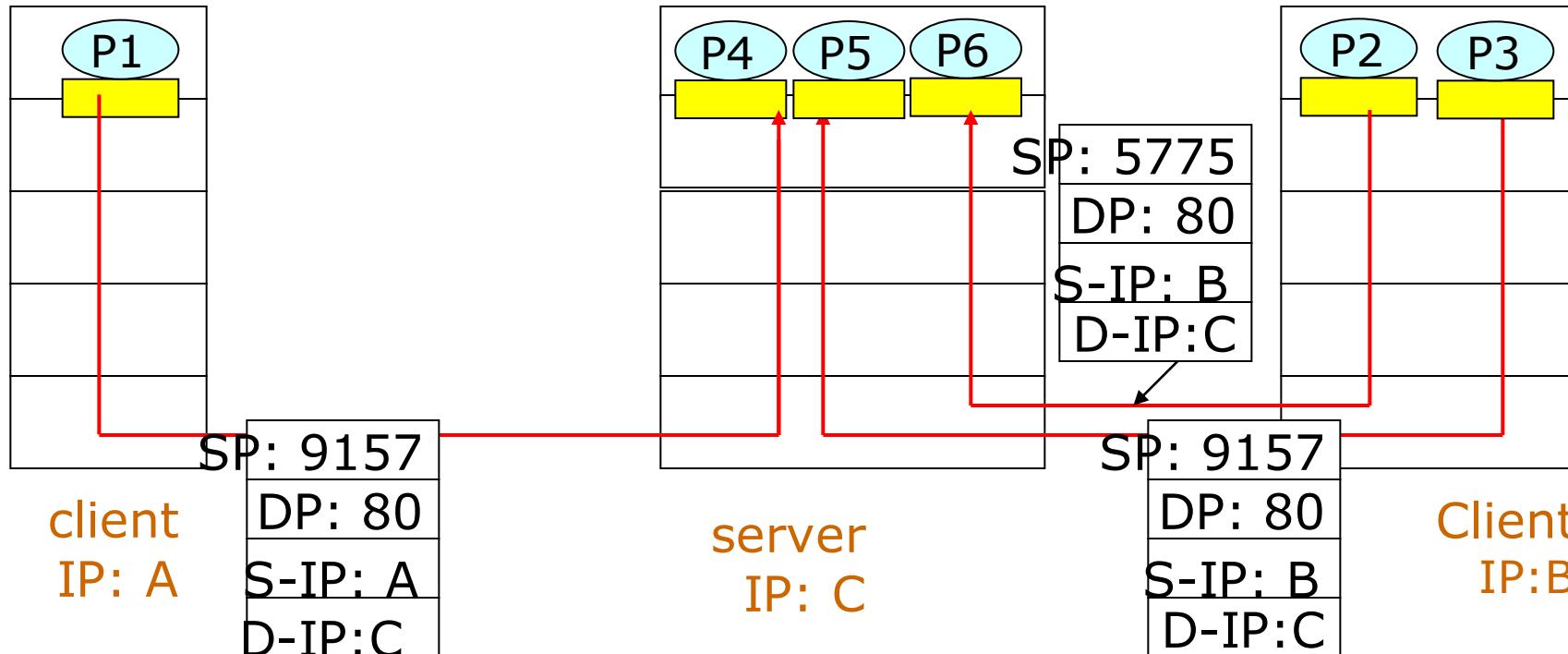


SP provides “return address”  = socket  = process

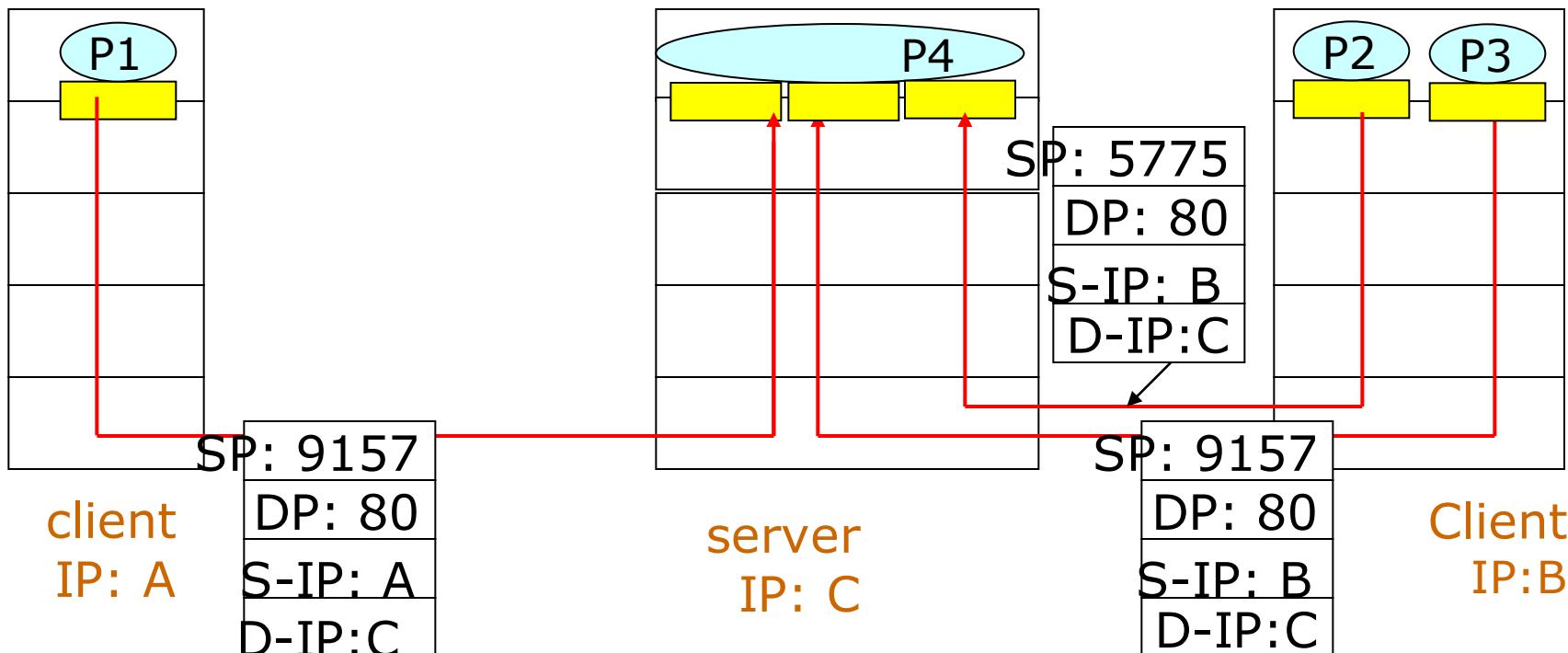
TCP פְּלִיאוֹקָי De-multiplexing

- ב TCP ה- socket מזוהה ע"י 4 שדות:
 - כתובת IP יעד.
 - מס' port יעד.
 - כתובת IP מקור.
 - מס' port מקור.
- המחשב מקבל משתמש בכל ארבעת השדות כדי להפנות את החבילה ל- socket המתאים.
- המחשב מקבל יכולות multiplexinguTCP. כל אחד מזוהה עם 4 שדות:
 - בריבוי sockets של TCP. כל socket יש בשרות Web נפרד לכל לקוח.
 - בחיבור HTTP שהוא non-persistent (לא נשמרת עקביות הקשר) ישsocket נפרד לכל בקשה.

TCP פיראטים De-multiplexing



WEB TCP פירוט אכטוי De-multiplexing



עכמת התקשורת

- עקרונות של בקרת עומסים.
- בקרת עומס TCP.
- שירותי שכבות ההובלה.
- ריבוב/פילוג.
- **העברה חסרת קשר: UDP.**
- עקרונות העברות מידע אמינים ברשות.
- **העברה מונחת קשר: TCP**
 - מבנה המקטע.
 - העברות מידע אמינים.
 - בקרת זרימה.
 - ניהול הקשר

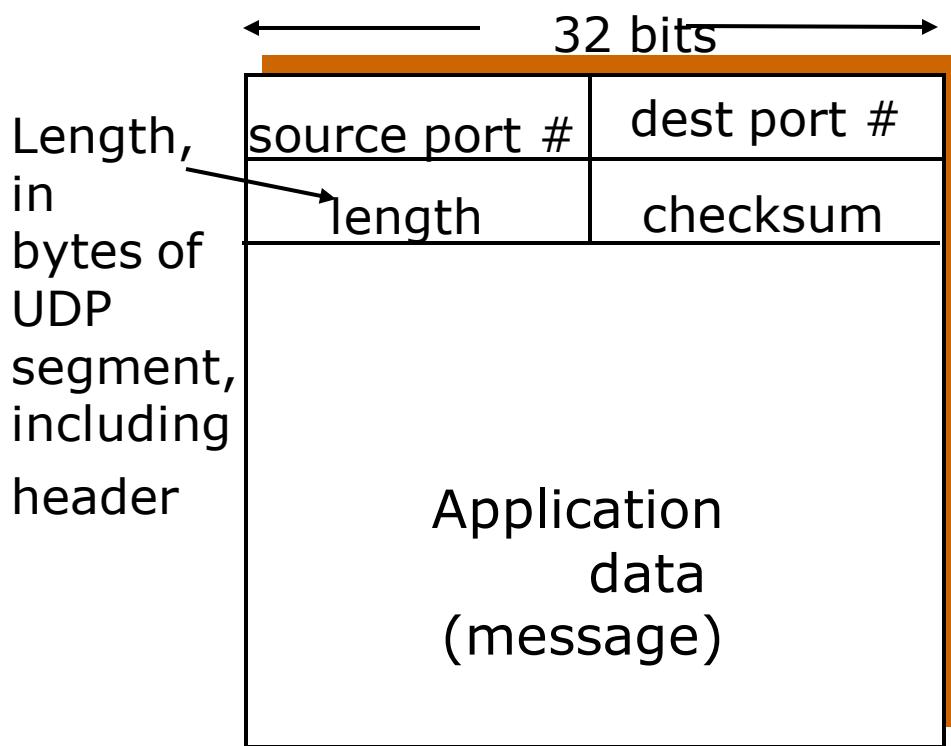
תאגידות מסכם קול (connectionless)

מדוע צריך את פרוטוקול UDP?

- חסכו התקורה ביצירת החיבור.
- פרוטוקול פשוט – חשוב בשבייל מכשירים פריימיטיביים, כמו מכשיר אזעקה.
- בעל header קטן – הודעות קצרות יותר.
- אין בקורת דחיסה ועומס – גורם להעמסת הרשות ואפשרות לניצול מקסימלית.

- פרוטוקול תעבורה פשוט.
- חברות UDP יכולות ללקת לאיבוד או להגעה של על פי הסדר (Best Effort).
:connectionless
- ללא ליחצת יד,
- כל חבילה UDP מטופלת בנפרד.
- חסכו ביצירת החיבור.

UDP א'ז ג'



UDP segment format

- ❑ משמש ליישומי מולטימדיה, בהם אפשרי לאבד חבילות ובהם חשוב קצב מהיר של העברה, יש עדיפות ל מהירות על פני אמינות.
- ❑ משמש גם בשילוח הודעות (**Domain Name System**): אלו הודעות קטנות ואין חשיבות לסדר בין חבילות שונות. וגם משמש בпротокол SNMP שהוא פרוטוקול לניהול רשת.

סיג'וי סכום UDP: סיג'וי סכום

מטרה: מציאת שגיאות פשוטות בלבד ללא מנגןון לתיקון שגיאות.

המקבל:

- מחשב את ה-
checksum של הסegment
שקיים.
- בודק אם המחשב שווה
לזה שהתקבל:
 - אם הוא שונה הtgtltha
טעות.
 - אם זהה לא הtgtltha טעות
אבל יתכן שהייתה טעות.

שלוח:

- מחלק את הסegment
לקטעים של 16 ביט.
- לחבר אותם ומבצע
השלמה ל-1.
- את התוצאה שם בשדה
ה- **checksum**.

Checksum-פְּקַדָּה

◻ שים לב:

- כאשר מחברים מספרים, אם יש נשא סופי לחבר אותו אל התוצאה (נשא מעגלי).

◻ דוגמא: חיבור של 2 מספרים בעלי 16 ביטים.

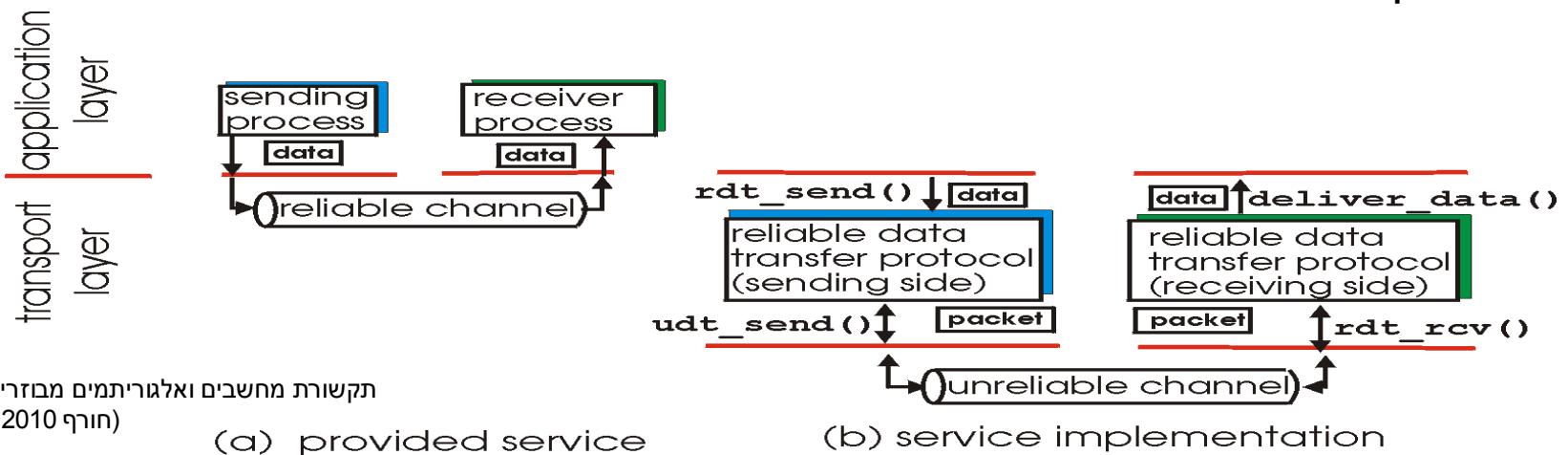
	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
נשא מעגלי				1	1	0	1	1	1	0	1	1	0	1	1	1
<hr/>																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

עכמת התקשורת

- עקרונות של בקרת עומסים.
- בקרת עומס TCP.
- שירותי שכבות ההובלה.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- **עקרונות תעبورת מידע אמינה ברשות.**
- תעبورת מונחת קשר: TCP
 - מבנה המקטע.
 - תעبورת מידע אמינה.
 - בקרת זרימה.
 - ניהול הקשר

ק'יך אגץ'ים הטענת נא'ז'ומ זרכ אלט'וי כהט סקי'רמ אגץ'ים נא'ז'ומ?

- ▢ חשבות האמינות היא בرمות שונות ולא רק ברמת התעבורה. יש לה חשבות גם בرمת הערז והיישום. שליחת מידע מתבצעת דרך רמה תחתונה שאינה אמינה, דבר שיכל לגרום לשיבוש הודעות.
- ▢ ניתן לזהות שגיאה באמצעות קוד ושדות הסתברותיים (כמו checksum). ישנו סיכוי ששגיאה לא תגלה, אך הוא זניח.
- ▢ ככל שהתווך יותר משובש כך על פרוטוקול התעבורה האמין, rdt protocol יתמודד עם יתר מצבים-דבר שמעלה את מורכבות הפרוטוקול.
- ▢ נבע סקירה של פרוטוקולים, כל אחד מהם מורכב יותר מקודמו, עד שנגיע לפרוטוקול האמין ביותר.



תְּלַקְּדָשׁ וְתִּמְךְּנָה

rdt_send() : called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

send side

reliable data transfer protocol (sending side)

udt_send() ↔ packet

udt_send() : called by rdt, to transfer packet over unreliable channel to receiver

deliver_data() : called by rdt to deliver data to upper

data ↑ deliver_data()

reliable data transfer protocol (receiving side)

packet ↔ rdt_rcv()

receive side

rdt_rcv() : called when packet arrives on rcv-side of channel

מצבים נקיים ואינם

- בסדר חשיבות עולה - נדרש לפתח את הצדדים השולח, מקבל של פרוטוקול להעברת נתונים אמינים (rdt).
- נתיחס רק להעברת נתונים חד כווני■ אר השיטה תהיה על זרימת המידע בשני הכיוונים!
- נשתמש במכונית מצבים סופית להציג השולח וה מקבל.

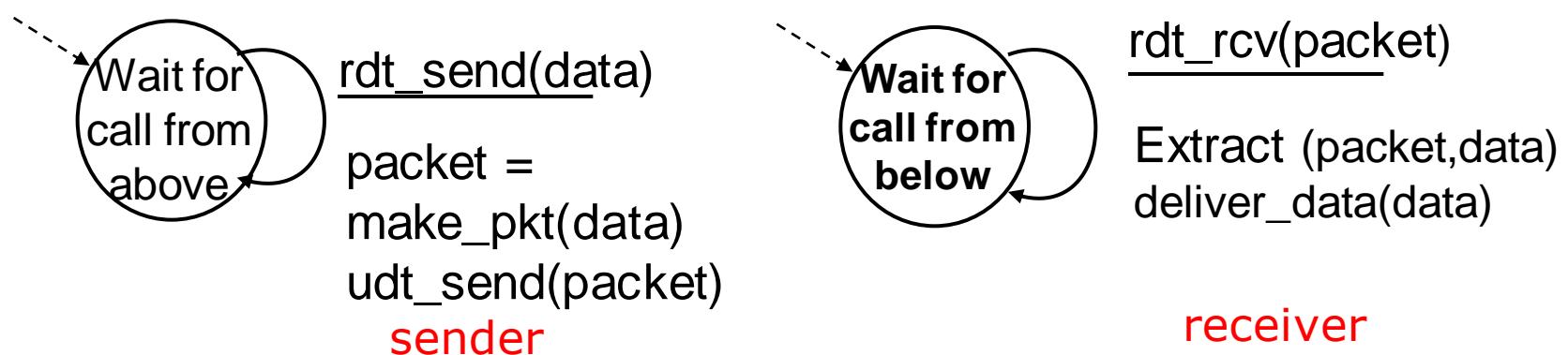


Rdt1.0: כלוקטן סלאני מהמאים נו-*Nk*

- ☒ לשולח ולמקבל יש רק מצב אחד - המתנה לקבלת הודהה.
- ☒ הצד השולח ממחכה לקבלת הודהה מהרימה העליונה. כאשר מתרחש מאורע של שליחת הודהה מתבצעת פעולה שליחת הודהה, תוך העברתה לרימה התחתונה.
- ☒ הצד המקלט ממחכה לקבלת הודהה מהרימה התחתונה. כאשר מתרחש מאורע של קבלת הודהה - המקלט מוציא מהחbillה את המידע.

סיכום:

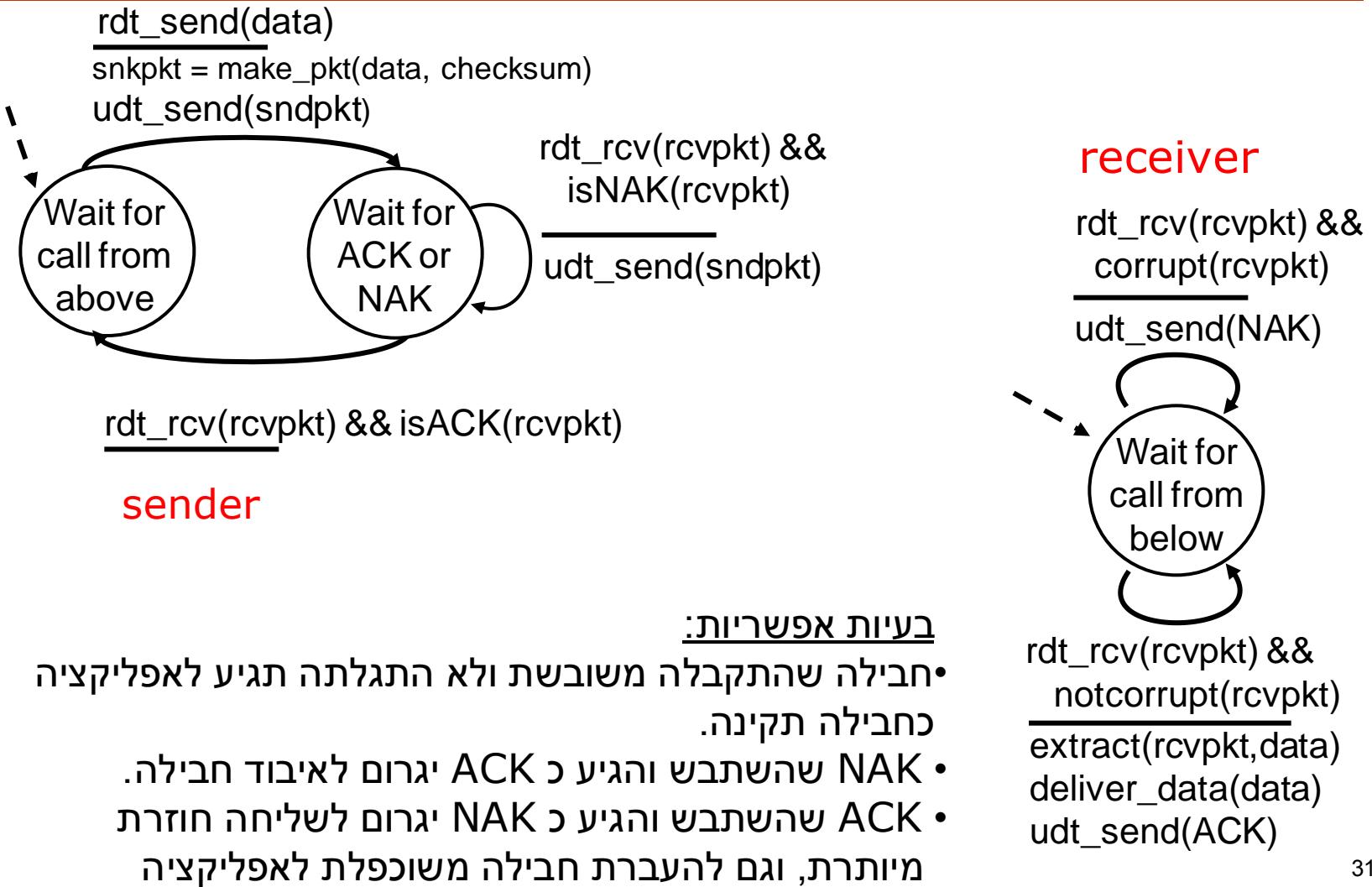
- העroz מוגדר להיות אמין ואין בדיקת corrupt.
- כל שגיאה שכן תתרחש תגיע ישיר לאפליקציה.
- אין אפשרות לתקינה של הפרוטוקול.



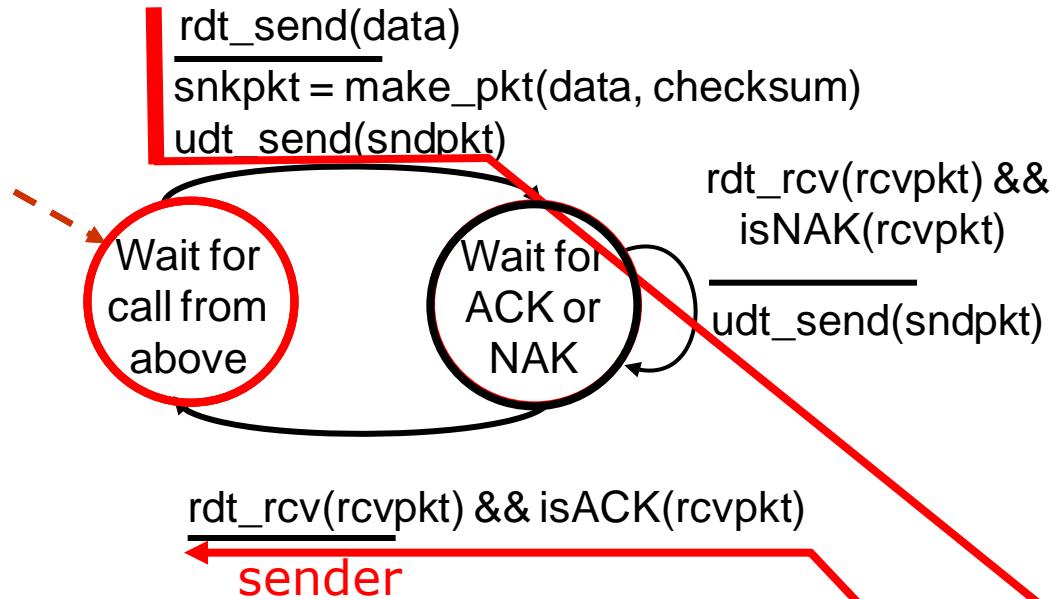
אלגוריתם מקורה Rdt2.0

- באופן עקוני ערוץ תקשורת יכול "להיעף" סיביות מהחbillות במהלך השידור.
 - מספר הנחות:
 - אין אובדן חbillות (אבל יתכן שיגיעו מושבות).
 - החbillות מגיעות בסדר הנכון.
 - כל הטעויות מתגלות על ידי מנגן ה-.checksum.
 - שאלה: איך מתאוששים מטעויות?
 - קיימן הייזן חוזר מהצד מקבל לצד השולח:
 - אישור על קבלת חbillה תקינה. (acknowledgment) **ACK**
 - אישור על קבלת הודעה מושבשת. (negative acknowledgment) **NAK**
 - חbillה שהגיעה מושבשת תשלח מחדש (retransmission).
 - תוספות שלא היו קיימות ב-Rdt1.0:
 - גילוי טעויות.
 - הייזן חוזר מהצד מקבל לשולח (ACK, NAK).

rdt2.0 באנזג'ים



כאיום nik'ce kff rdt2.0



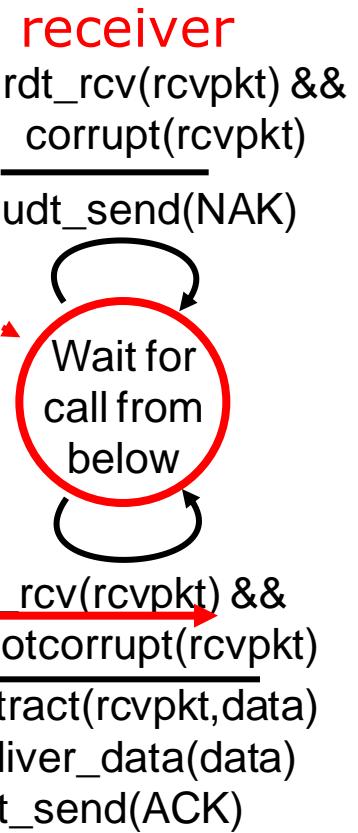
יש שני מצבים בצד השולח:

מצב 1: מחהקה לקבלת מידע לשליחת מהרימה העליונה. כאשר יתרחש המאורע של קבלת המידע השולח ייצור חבילת שמכילה את המידע ואת שדה ה **checksum**. השולח ישלח את החבילת דרך הערוץ הלא אמין.

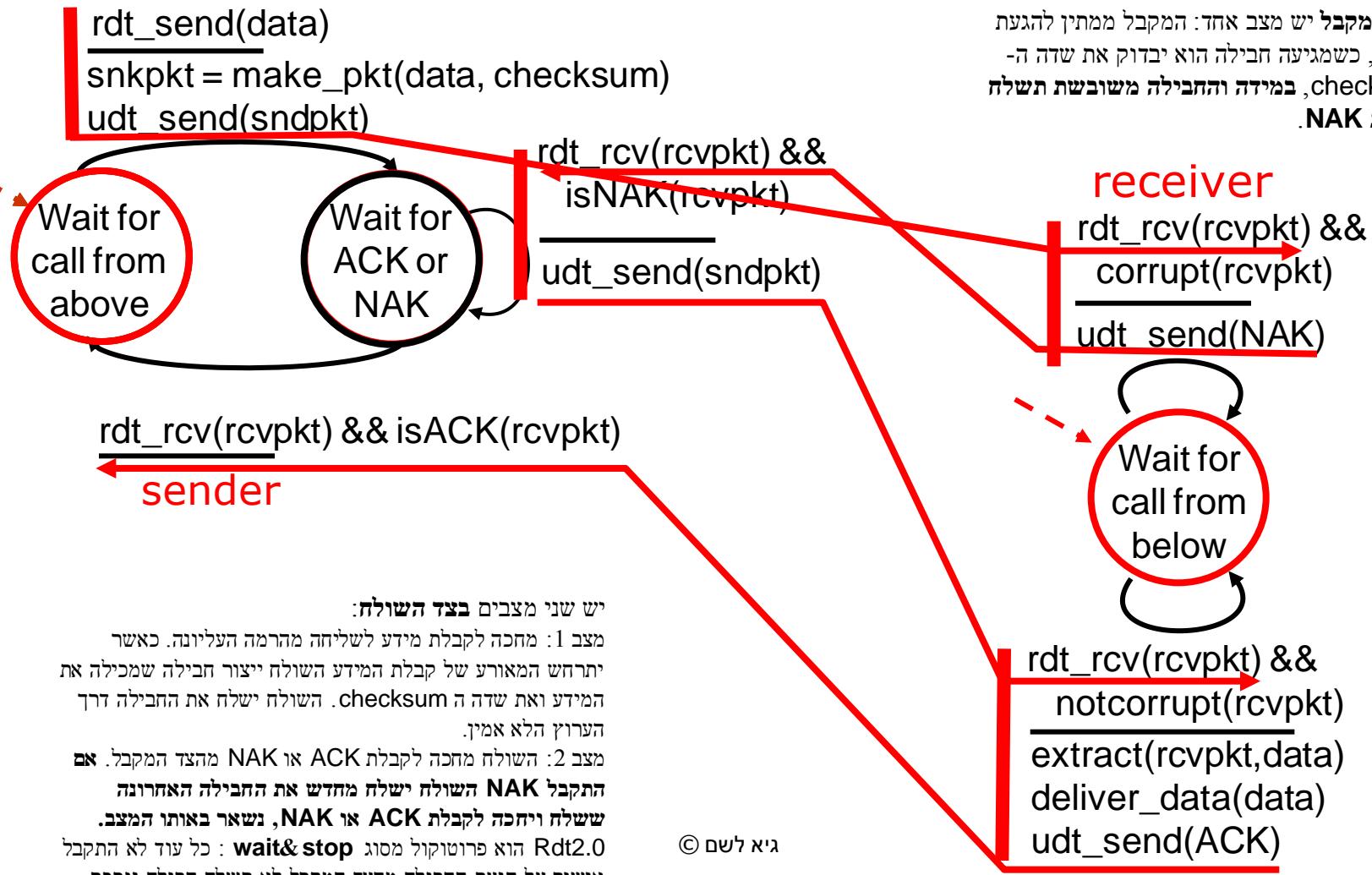
מצב 2: השולח מחהקה לקבלת ACK או NAK מ从此ד המתקבל. אם מתקבר **ACK** השולח יודע שהחבילת האחורונה נשנה להה הגייעה תקינה וחוור למספר 1.

Rdt2.0 הוא פרוטוקול מסוג **wait&stop**: כל עוד לא התקבל אישור על הגעת החבילת מ从此ד המתקבל לא תשלוח חבילת נוספת.

בצד מקבל יש מצב אחד: מקבל ממitin להגעת חבילת, כשהגיעו החבילות הוא יבדוק את שדה ה-**-ACK**, אם החבילת תקינה ישלח **ACK, checksum**.



כיצד מ작ה rdt2.0



rf2.0 – מתקני מאוץ!

על פניו נראה כי הפרטוקול מבטיח אמינות אבל יש בו פגם
חמור

טיפול בשיכפול:

- אם יהיה שיבוש הודעה NAK אז חבילה שהגיעה משובשת לא תשלח שוב כנדרש ויהי איבוד הודעה.
- אם יהיה שיבוש הודעה ACK אז תשלח אותה ההודעה פעמיים.

מה קורה אם ACK/NAK פגומים ?

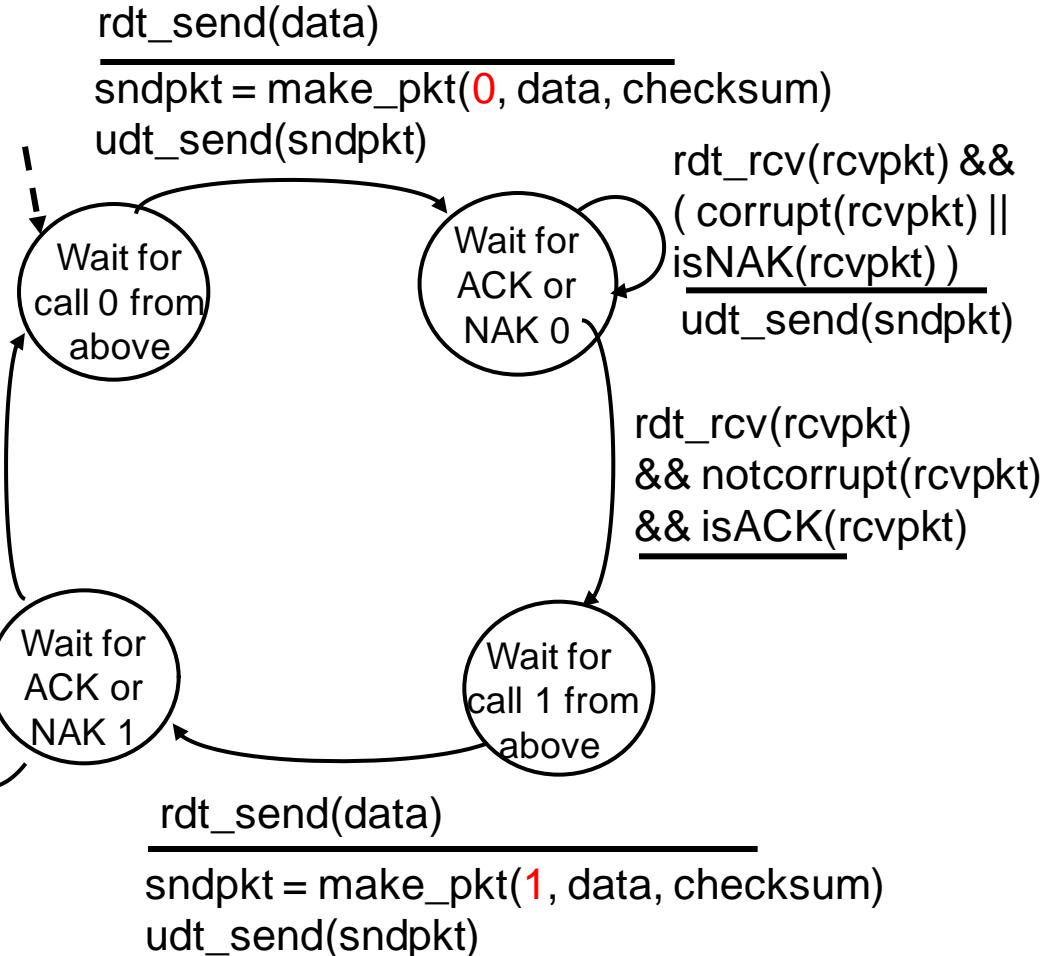
- השולח לא יודע מה קרה אצל המקלט.
- השולח אינו יכול פשוט לשלוח מחדש את החבילה כי אז יכול להיווצר מצב של שכפול

ACK/NAKs - מנגנון הילוף :rdt2.1

מונח של ביט אחד נוסף לחבילה (מספרו של 0 או 1) - מספיק בגל שחייבות מושבותות يتגלו ואין שינוי בסדר החבילות. הצד השולח צריך לבדוק האם הودעות קיימות 4 מצבים (כפול מ-2, Rdt2.0), מפני שמבצעים את אותן הפעולות עבור חבילות שמוספרות 0 או 1.

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt)

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isNAK(rcvpkt))
udt_send(sndpkt)



ACK/NAKs – נסיגת ה-ACK, רידט 2.1

צורך לבדוק האם החבילה שהתקבלת משוכפלת, על פי המספר הסידורי של ההודעה האחורונה שהתקבלת (0 או 1).

אם ההודעה הגיעה משוכפלת הפרוטוקול ידע לא להעביר אותה לרמת היישום. המקבל לא יודע אם הודעתה היא NAK או ACK האחורונה ששולחה התקבלו בצורה תקינה בצד השולח.

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

 sndpkt = make_pkt(NAK, checksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
 not corrupt(rcvpkt) &&
 has_seq1(rcvpkt)

 sndpkt = make_pkt(ACK, checksum)
 udt_send(sndpkt)

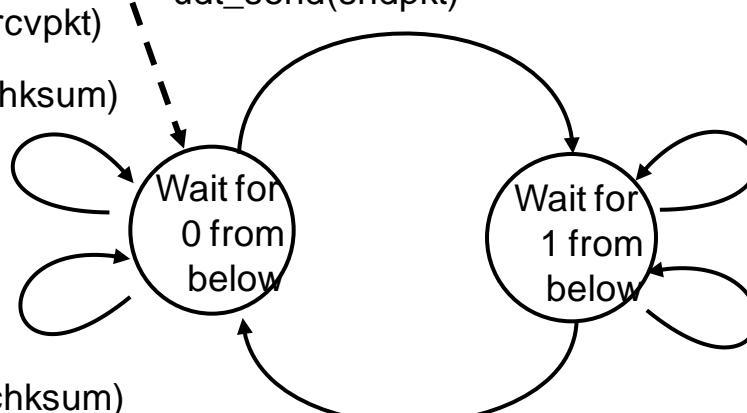
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
 && has_seq0(rcvpkt)

 extract(rcvpkt,data)
 deliver_data(data)
 sndpkt = make_pkt(ACK, checksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
 sndpkt = make_pkt(NAK, checksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
 not corrupt(rcvpkt) &&
 has_seq0(rcvpkt)

 sndpkt = make_pkt(ACK, checksum)
 udt_send(sndpkt)



אג'ר: RDT 2.1

- חבילה שבה ה- data התקבל משובש, ולא התגלתה, תגיע לאפליקציה כחבילה תקינה. (האפליקציה מקבל data שגו).
- מספר סידורי שהשתבש:
 - אם הייתה מוגנونة ע"י מנגן השגיאות, החבילה תטופל כ- corrupted, וישלח NAK.
 - אם החבילה לא מוגננה, החבילה לא תטופל כי מצפים למספר אחר. לפי הפרוטוקול אין שליחת NAK, אז הפרוטוקול ייתקע.
- NAK שהשתבש והגיע כ- ACK יגרום לתקיעת הפרוטוקול (רק הצד השולח יעבור למספר הבא).
- ACK שהשתבש והגיע כ- NAK יגרום לתקיעת הפרוטוקול (רק הצד המתקבל יעבור למספר הבא).

rdt2.1: קיון

המקבל:

- חייב לבדוק האם החבילה שהתקבלת משוכפלת.
- המצב מצביע האם המספר הסדרתי שהתקבל הוא 0 או 1.
- שים לב: המקלט אינו יכול לדעת האם ה-ACK/NAK/האחרון שהתקבל בשולח הוא תקין.

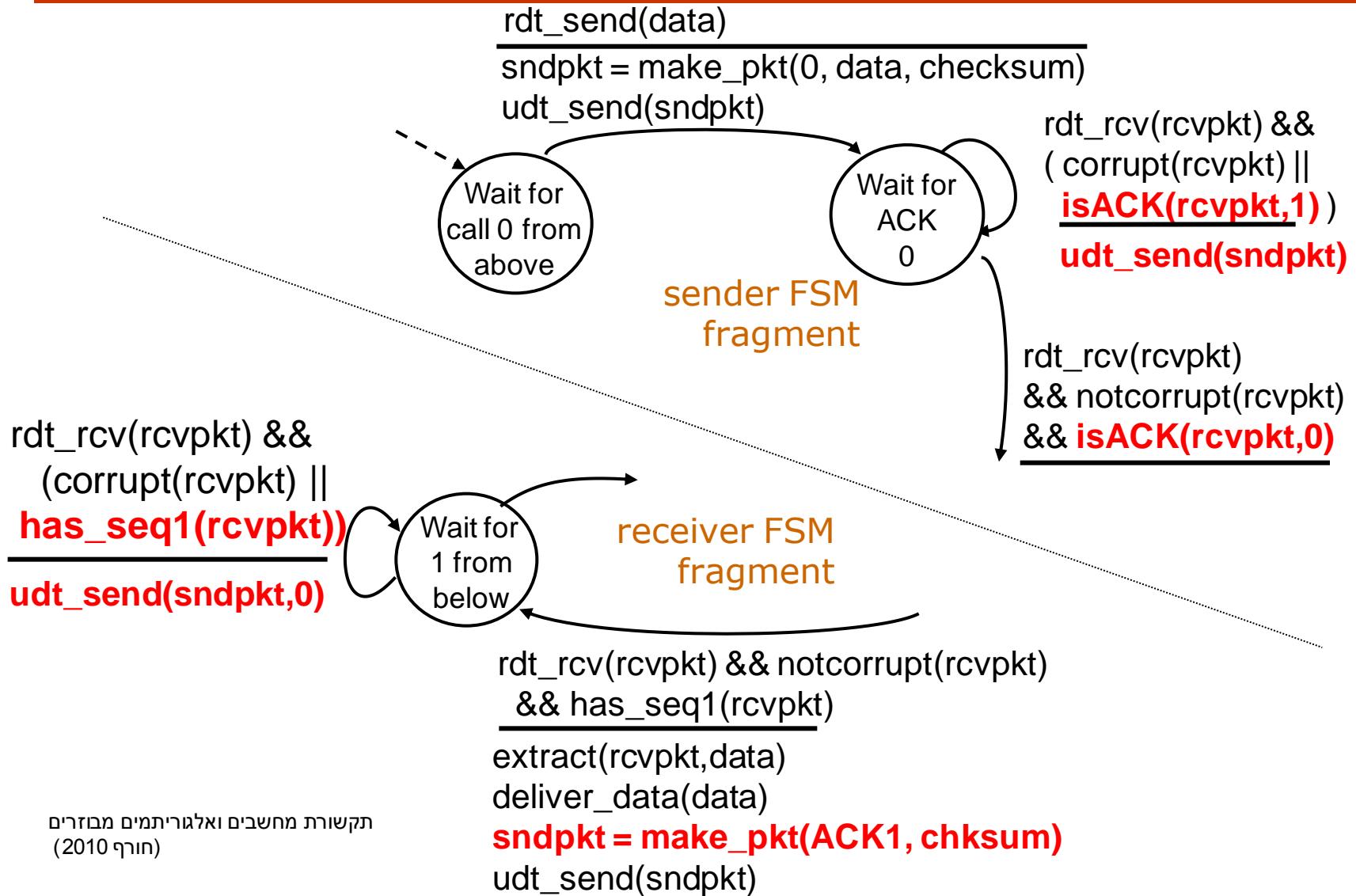
שלוח:

- מספר סידורי (מספר) נוסף לחבילה.
- **2 אפשרויות מספר 0 או 1 יופיעו.**
- חייבים לבדוק אם התקבל NAK / ACK פגום.
- פעמיים מספר המצבים:
 - המצב חייב "לזכור" אם לחבילה הנוכחית יש 0 או 1.

NAK \leftarrow : rdt2.2

- ה프וטוקול הוא כמו rdt2.1 אך ללא שימוש בהודעות NAK.
- במקום NAK מקבל שולח ACK להודעה האחרונה שהתקבלה בצורה תקינה. מקבל חייב לצרף את מספרה הסידורי של החבילה שעלייה הוא נותן ACK.
- שיפור לעומת rdt2.1: מונע שכפולי חבילות ו-ACK.
- אם יש שכפול של הודעת ACK אז השולח ישלח מחדש את החבילה הנוכחית ולא יהיה איבוד מידע.
- אין טיפול באובדן חבילות, יתרון מצב בו מקבל והשלוח יחכו זמן אינסופי.

rdt2.2: sender, receiver fragments



RDT 2.2 אג'ס

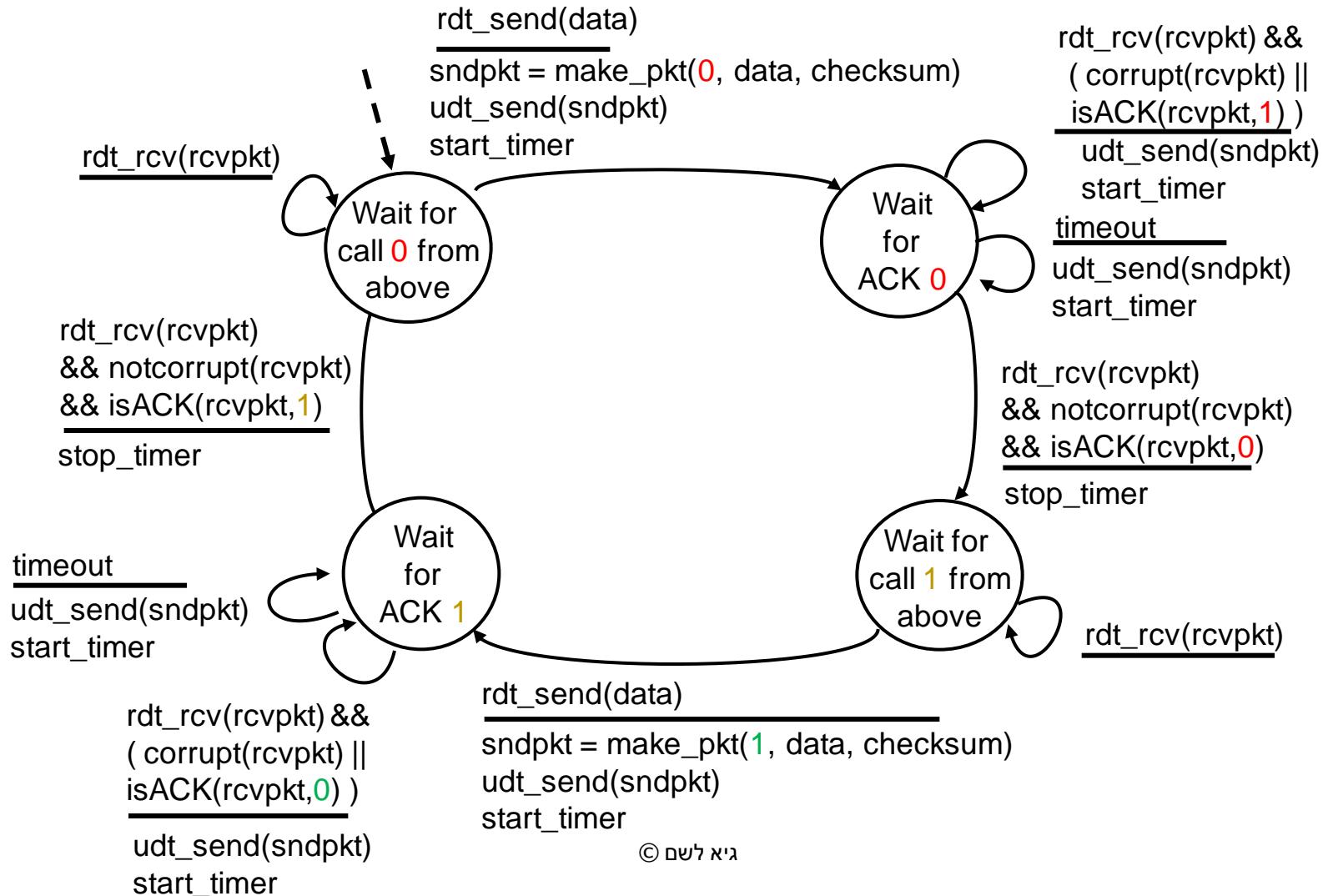
- חvíלה שבה ה-data התקבל משובש, ולא התגלתה, תגיע לאפליקציה כחvíלה תקינה (האפליקציה תקבל data שגו).
- מספר סידורי שהשתבש :
 - אם הייתה מוגנונן השגיאות, החvíלה תטופל כ-corrupted, וישלח ACK על החvíלה קודמת.
 - אם החvíלה לא מוגנונה, החvíלה לא תטופל כי מצפים למספר אחר. לפי הפרוטוקול אין שליחת ACK על החvíלה קודמת, אז הפרוטוקול יתקע.
- **ACK על חvíלה קודמת שהשתבש ל ACK על חvíלה נוכחית** יגרום לתקיעת הפרוטוקול (רק הצד השולח עברו למספר הבא)
- **ACK על חvíלה נוכחית שהשתבש ל ACK על חvíלה קודמת** יגרום לתקיעת הפרוטוקול (רק הצד המתקבל עברו למספר הבא)

rdt3.0: אכזב מכך ואגזר

- פרוטוקול שניtin להרץ מעל ערוצים אמיתיים. מתאים לערוצים פיזיים כמו רשת מקומית.
 - הפרוטוקול מטפל ב: שגיאות, שכפולים, אובדן, אין טיפול בשינוי סדר של הודעות.
 - הנחות:
 - הערוץ יכול לאבד חבילות (מידע או ACK).
 - קיימים המנגנונים מהפרוטוקולים הקודמים: sum, מס' סידורי של חבילה, ACK, שליחה מחודשת. אבל הם אינם מספיקים.
 - דרך ההתחממות עם אובדן חבילות:
 - **Timer**: השולח מחרכה לACK פרק זמן "סביר" אשר קטן מהעיכוב המקסימלי (Round Trip Time) RTT). במידה ולא התקבל ACK בזמן זהה תבצע שליחה מחדש. במידה והיה רק עיכוב ולא אובדן, יתבצע שכפול שבו הצד מקבל יודע לטפל (בעזרת מס' סידורי).
 - בעיות:
 - מניח שאין שינוי בסדר ההודעות (FIFO), למרות שרשנות datagram לא מבטיחות FIFO. לדוגמה- שתי חבילות שנשלחו זו אחר זו, לשניה מסלול קצר משל הראשונה ועל כן היא מגיעה לפניה. במיוחד יש שינוי סדר כאשר משתמשים ב프וטוקול ב-IP.
 - אין **pipelining**: לכל היותר שלוחים חבילה אחת ומחייבים לקבל עליה ACK.イトיות.
- תקשורת מחשבים ואלגוריתמים מבזרים
גיא לשם © (חורף 2010)
- 42

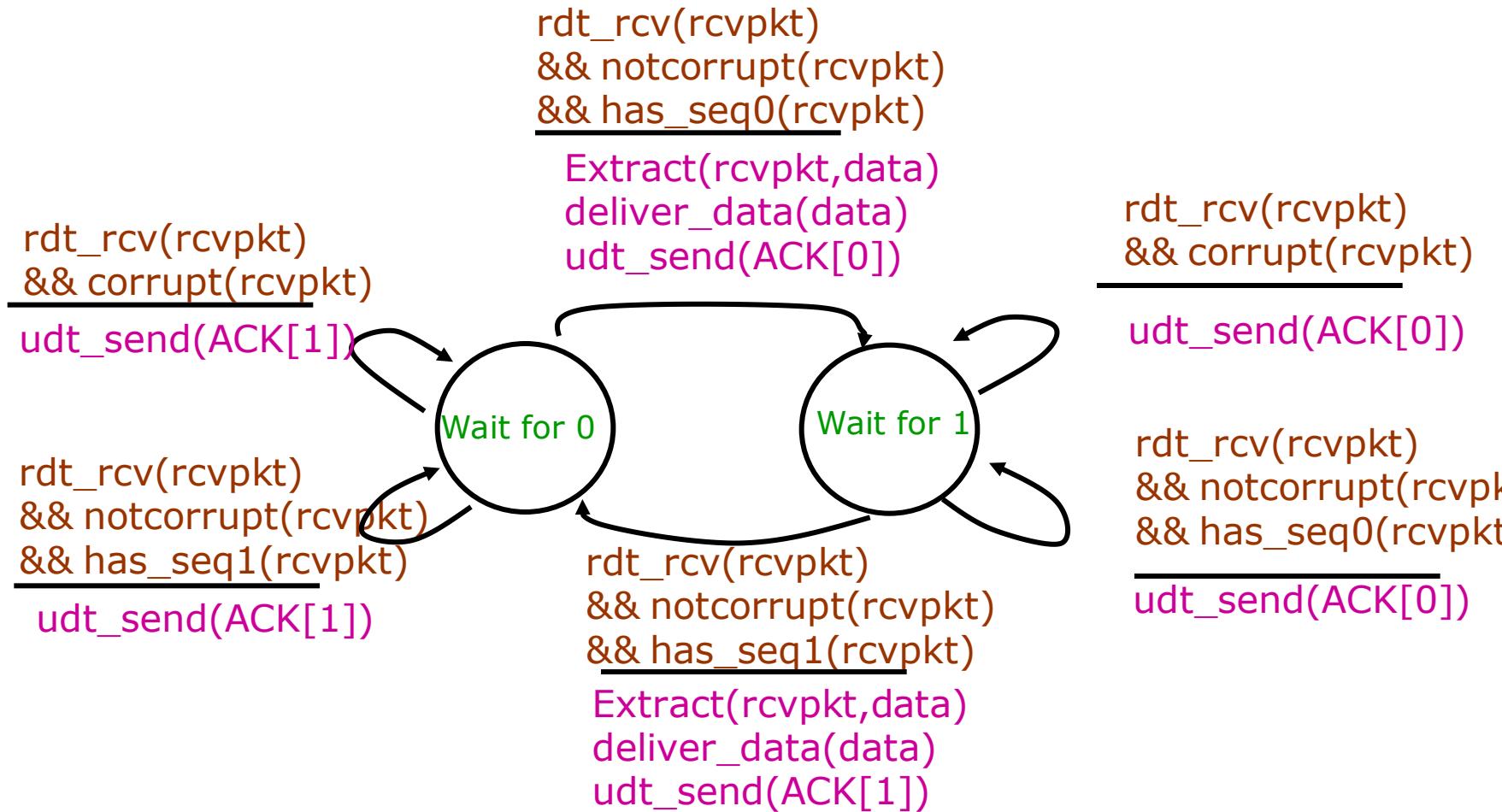
תֵּפְשָׁרָה גַּם הַמִּתְחָדֶשׂ : rdt3.0

הצד השולח:



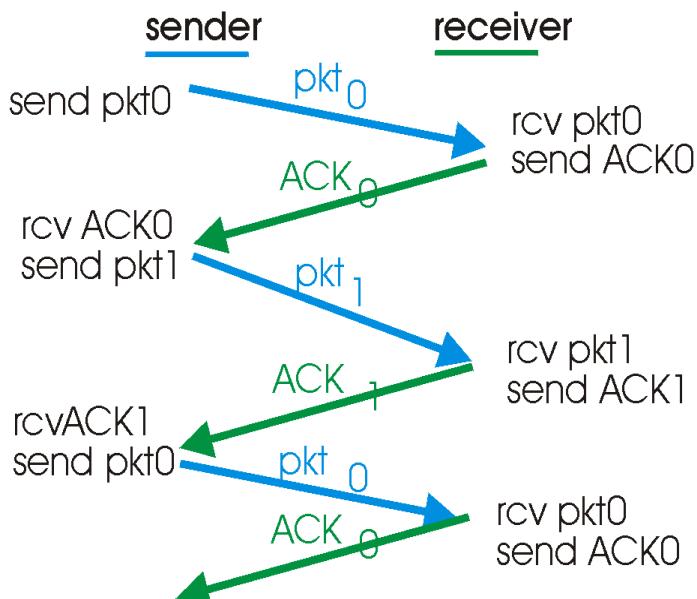
תפקידים באנטומיה של rdt3.0

הצד המקבל:

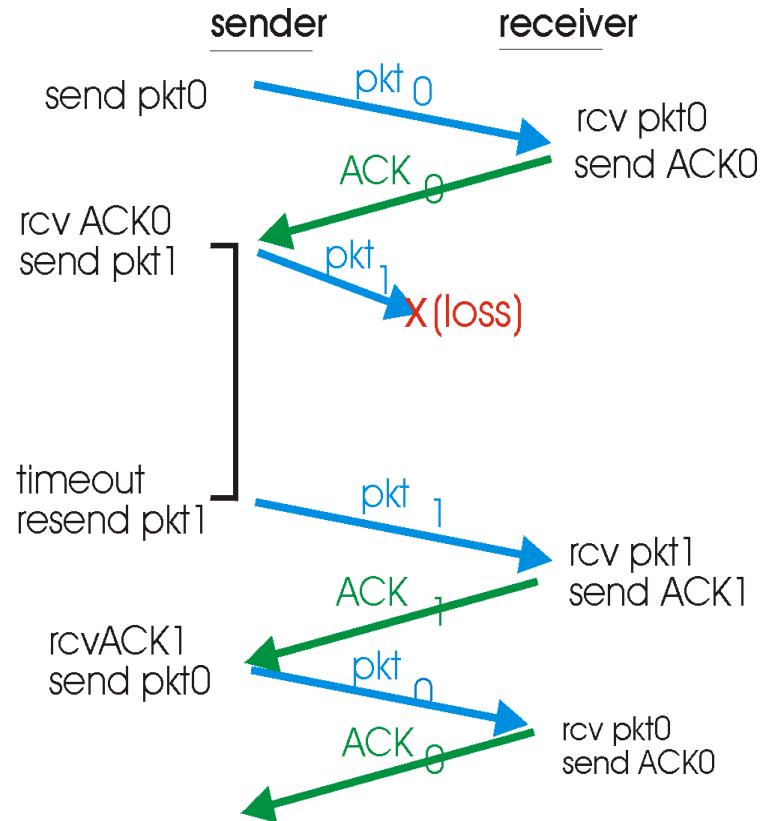


rdt3.0 כפולה

בתרשימים הבאים מתוארים מצבים של פעילות רגילה, אובדן חבילת ACK, ו-

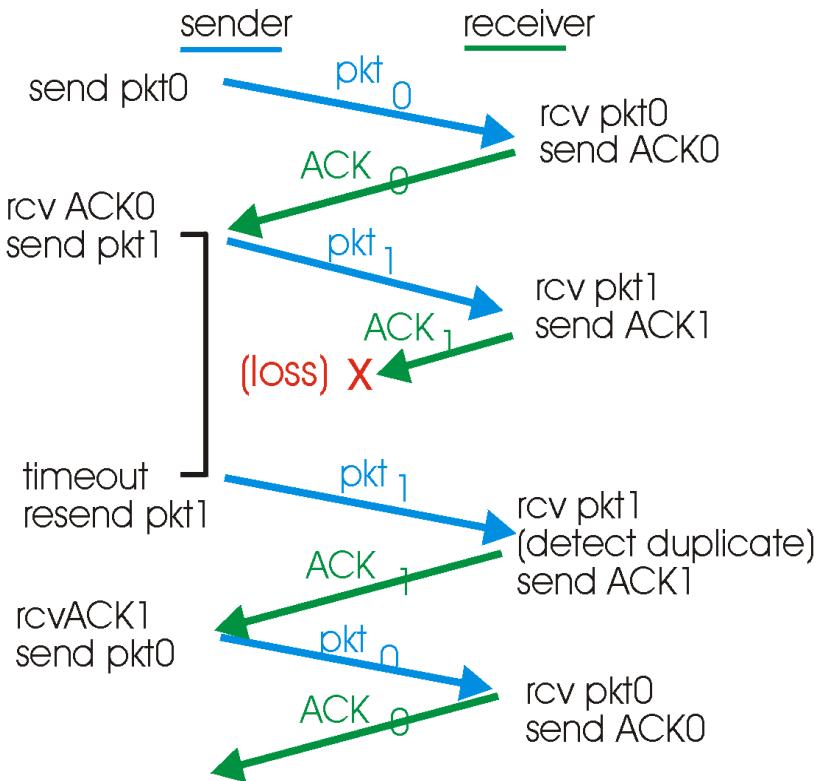


(a) operation with no loss

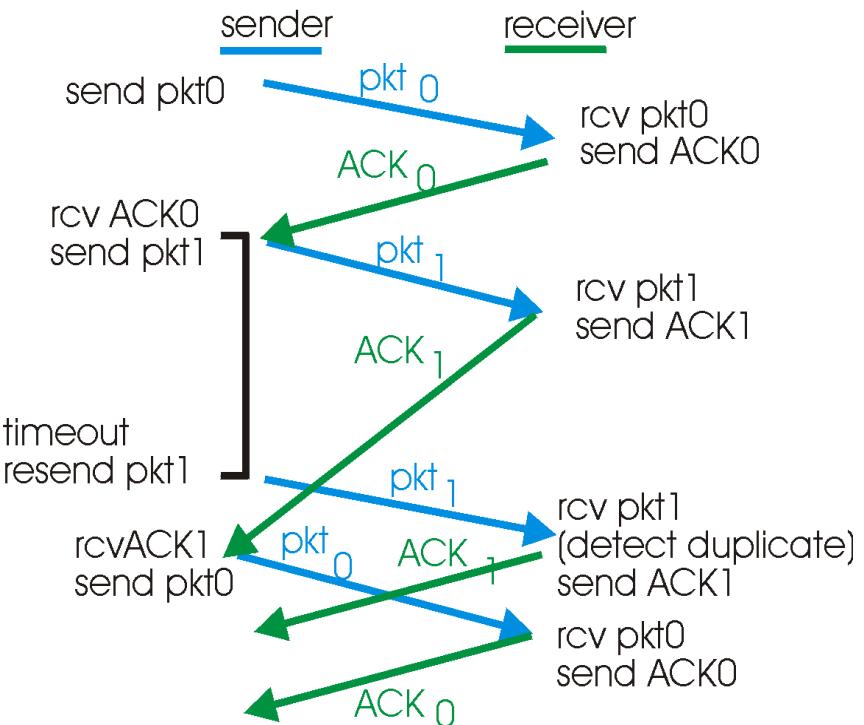


(b) lost packet

ନ୍ୟୂଆ ର୍ଡିଟ୍ 3.0



(c) lost ACK



(d) premature timeout

rdt3.0

- ☒ ביצועי הפרטוקול יכולים להיות גורעים בגלל RTT יכול להיות גדול עקב תורים בנתבים.
- ☒ זמן השידור (transmission delay) תלוי באורך החבילה המשודרת ובקצב השידור:
- ☒ דוגמא לחישוב נצילות:

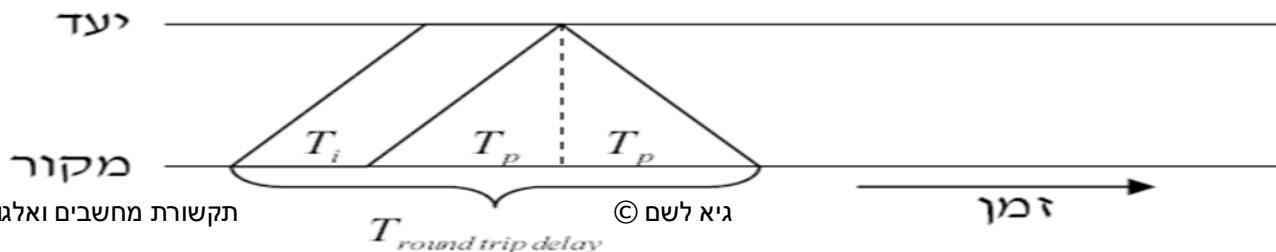
$R=1 \text{ Gbps link}$, $T_p=15 \text{ ms}$ (prop. delay), $L=8000 \text{ bit}$
packet

$$T_i = \frac{\text{אורך - חבילת}}{\text{קצב - שידור}} = \frac{L}{R} = \frac{8000 \text{ bits}}{1 * 10^9 \text{ bps}} = 8 \text{ microseconds}$$

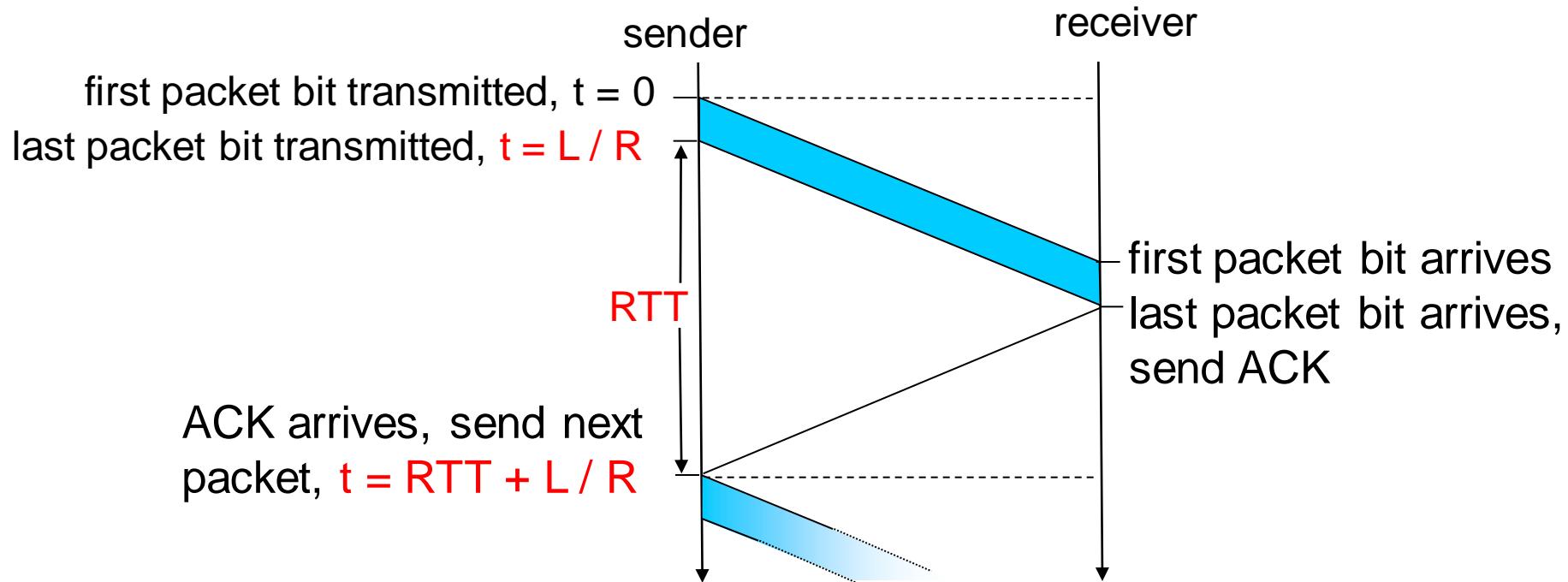
- ☒ נצילות הערוץ: זמן שידור מסגרת אחת ביחס לזמן הכלול למסגרת להישלח ולחזור

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{0.008}{30.008} = 0.00027$$

דיאגרמת זמנים:



ר'ז'ט מ ה-rtt3.0

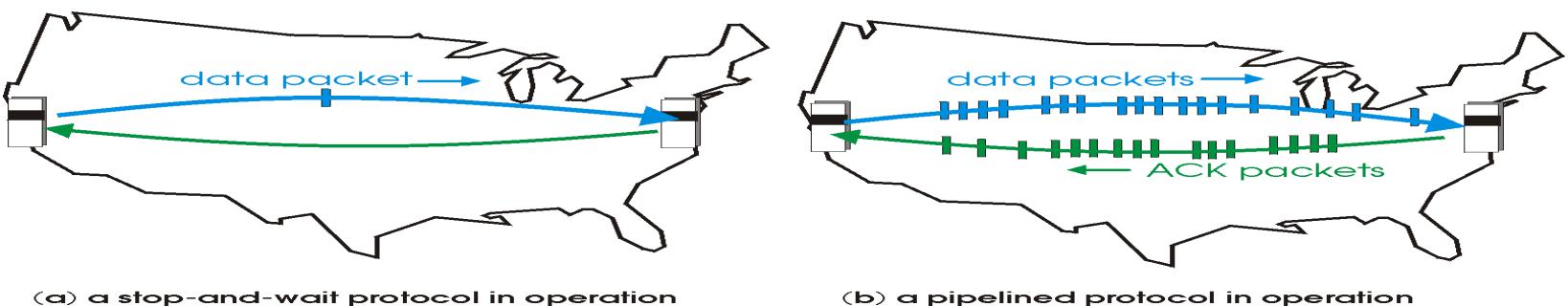


$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

הчисוב מדגיש שמעורץ רחב קיבלו ניצולת מאוד נמוכה.

פְּלִינְהַפִּי Pipeline

- בפרוטוקולים מסווג stop and wait נוצרה בעיה של ניצולת נמוכה, ניתן לשלוח הודעה אחת בכל פעם בלבד. כמו כן RTT יכול להיות גדול בגלל תור שנוצר בנתבים.
- פרוטוקולים מסווג pipeline מאפשרים שליחת הרבה הודעות במקביל מבעלי שקבענו עליהן ACK.
- הדבר מחייב הגדלת טווח המספר במנויים של החבילות. כמו כן נדרש חוצצים בשולח כדי לשמר הודעות שנשלחו ועדין לא התקבל עליהם ACK במקרה שнецטרך לבצע שליחה מחדש של ההודעה. לעיתים נדרש גם חוצצים במקביל.
- מתמודד גם עם שינוי סדר בחבילות ישנות.
- שימוש ב- pipelining גורם להגדלה של הניצולת, אבל תמיד תהיה מגבלה מסוימת שנובעת מגודל החוצצים שיש לנו. מספר המונה הוא סופי.

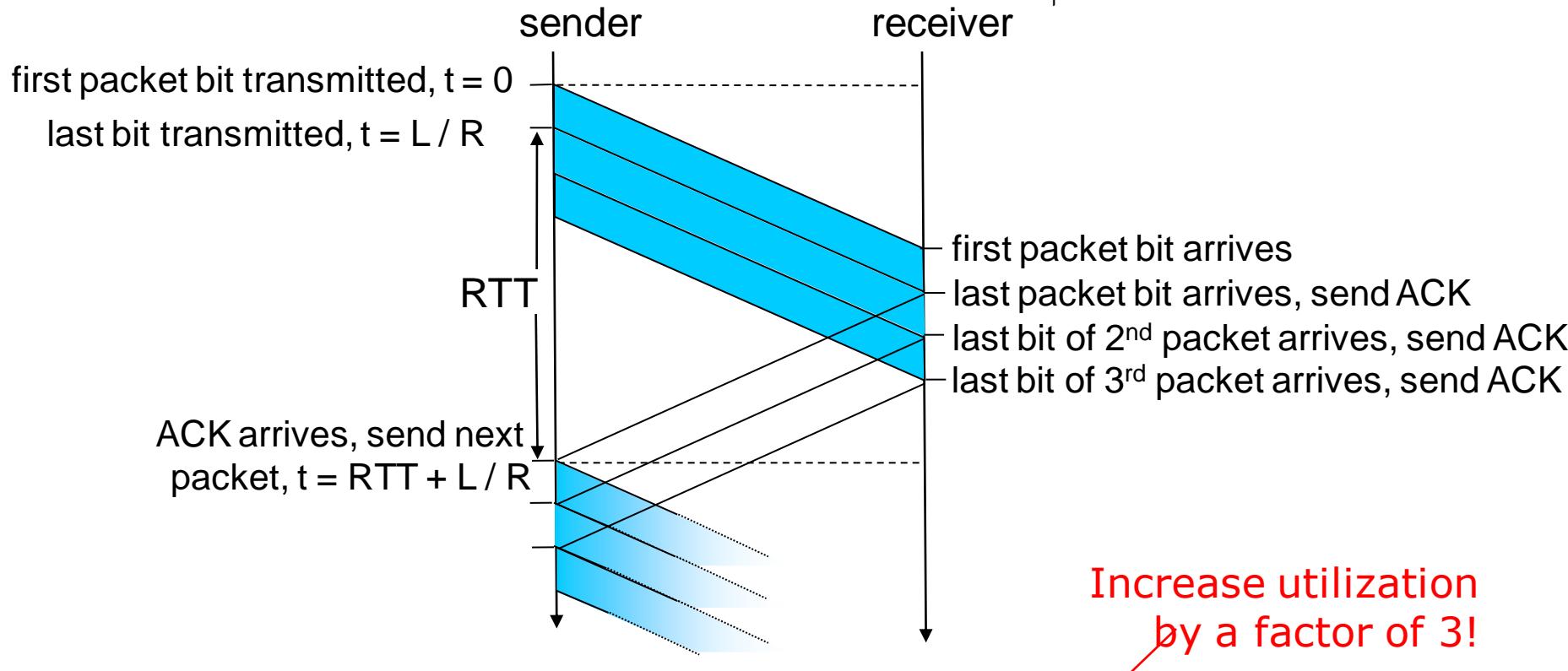


- יש שתי אסטרטגיות של pipeline : Selective Repeat - Go-Back-N

পাইপলাইning: প্রক্রিয়াজটি

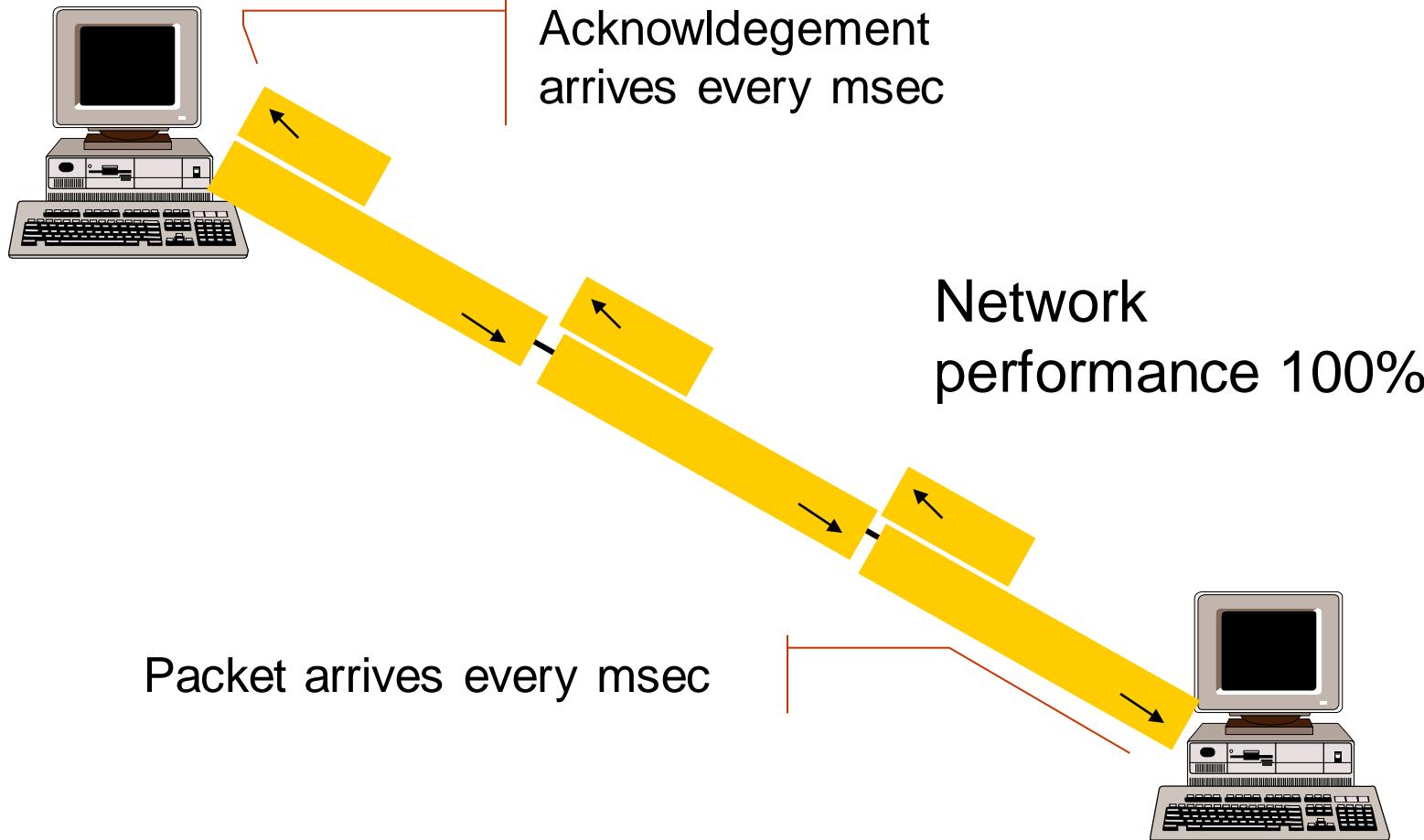
নথমে বন্ধনীগুলির মধ্যে কোডের হিসাব করা দ্বারা পাইপলাইning এর মাঝে নিয়ন্ত্রণ মেরুদণ্ড উন্নয়ন করা যায়। আবার এই ক্ষেত্রে পাইপলাইning এর মাঝে নিয়ন্ত্রণ মেরুদণ্ড উন্নয়ন করা যায়।

যদি 3 টি শোলার পাইপলাইning করা হয় তবে এটি একই সময়ে পাইপলাইning করা হচ্ছে।



$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

(פִּלְינִינְג) Pipelining



Sliding Window Protocols

- **חbillות ו-ACK-ים ממוספרות.**
- **חלון שליחה (אצל המשדר)**
 - מתחילה מהחbillה הראשונה שנשלחה ועדין לא אושרה.
 - המשדר יכול לשלוח עד S חbillות "קדימה".
 - תפקיד חלון השליחה: לאפשר "עבודה במקביל"
- **חלון קבלה (אצל המקלט)**
 - מתחילה מהחbillה הבאה שלא מצפה המקלט.
 - המקלט יכול לבאר עד R חbillות "קדימה" – כלומר שהגיעו שלא לפי הסדר המצוופה. חbillות מחוץ לחלון – נזקות.
 - תפקיד חלון הקבלה: לארגן את החbillות ולמסור אותן לשכבה שמעל – לפי הסדר הנכון.

כלאוקול הצעמת מילוי

□ נראה:

$W_R = 1, W_S = N$: **Go-Back-N** ■

$W_R = W_S = N$: **Selective Repeat** ■

□ פרוטוקולים "בטבע":

. Go-Back-N (שכבה שנייה) מיישם את N.

. TCP (שכבה רביעית) מיישם את SR.

□ **High-level Data Link Control** (Bit-level: פרוטוקול תלוי סיביות (Data encapsulation) Oriented) המגדיר שיטה לעטיפות נתונים (TCP/IP). בرمת שכבת קישור הנתונים ובמודל ה-IP.

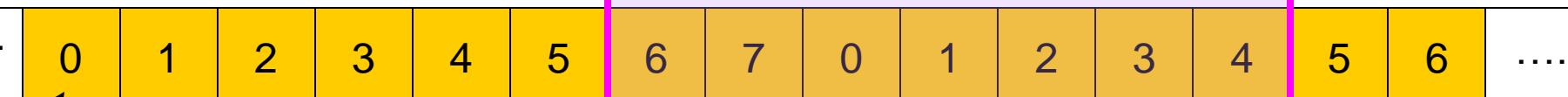
□ **TCP**: נלמד בהמשך.

לִינְקָגֶת – Sliding Window

Sender Perspective

Frames already acknowledged

Window of frames that may be transmitted



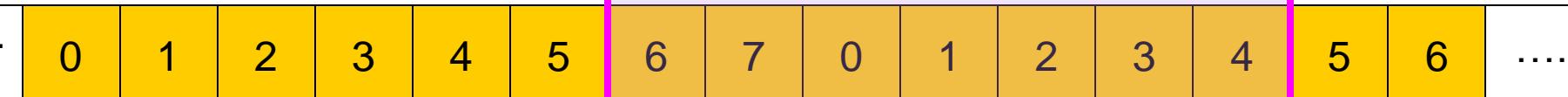
Window shrinks from trailing edge as frames are acknowledged

Window expands from leading edge as ACKs received

Receiver Perspective

Frames already received

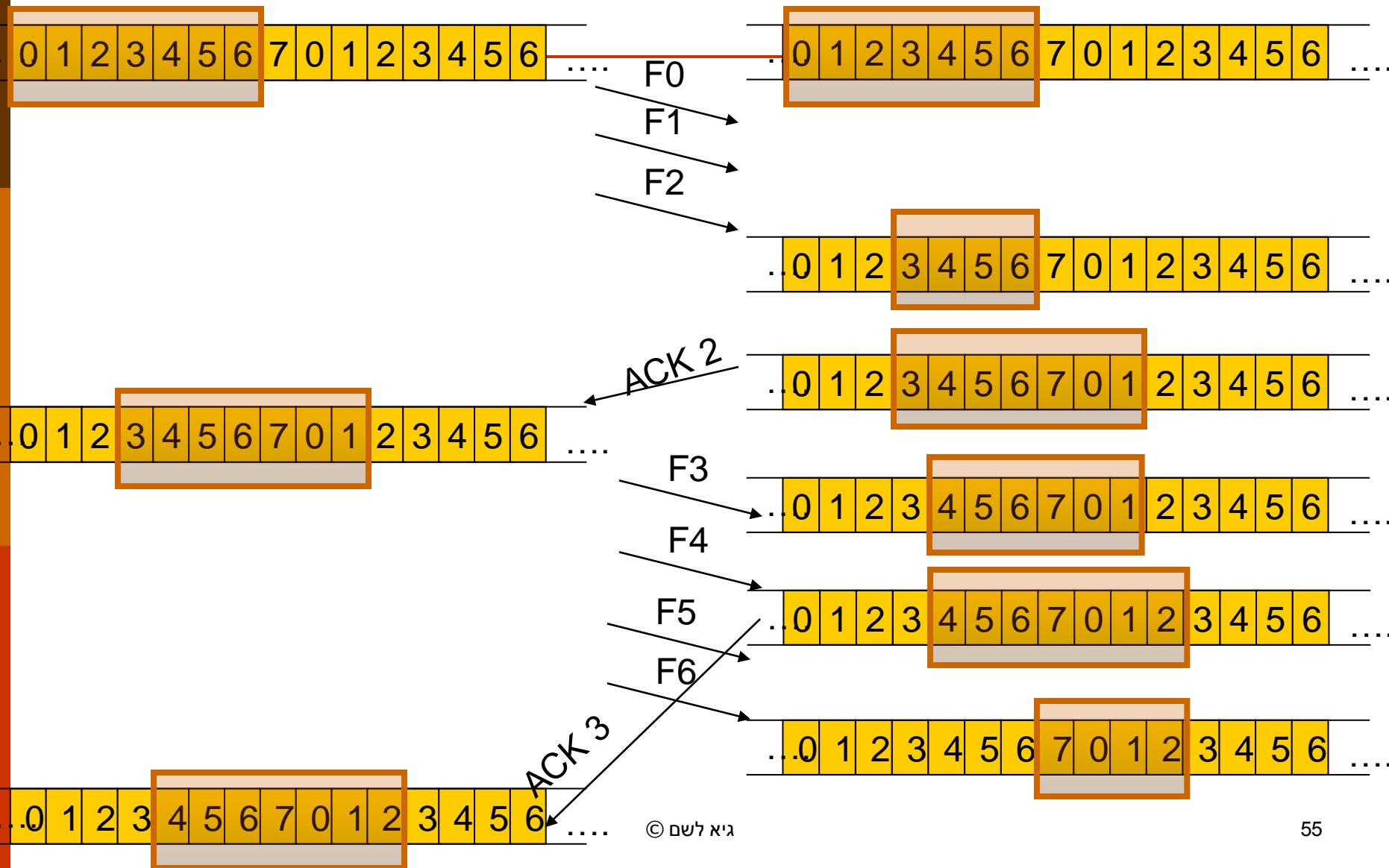
Window of frames that may be transmitted



Window shrinks from trailing edge as frames are received

Window expands from leading edge as ACKs are sent

JIfn תבננ f knclz



פלוֹקְוָן Go-Back-N

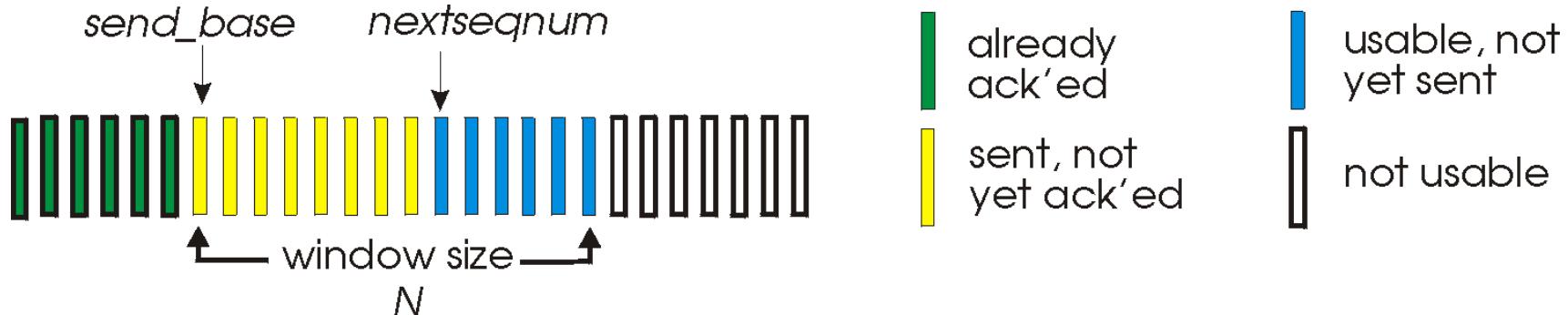
□ תחנה משדרת

- משדרת חבילות בזו אחר זו לפי חלון שגודלו N .
- כאשר עובר T_{out} חוזרת לחבילה האخונה שעדיין לא אושרה ומשיכה לשדר חבילות החל מחבילה זו.

□ תחנה קולטת

- מקבלת ומארשת חבילות אר רך לפי סדר (האישורים "מצטברים": אישור על חבילה ch מאשר את כל החבילות מהראשונה ועד ch , כולל)

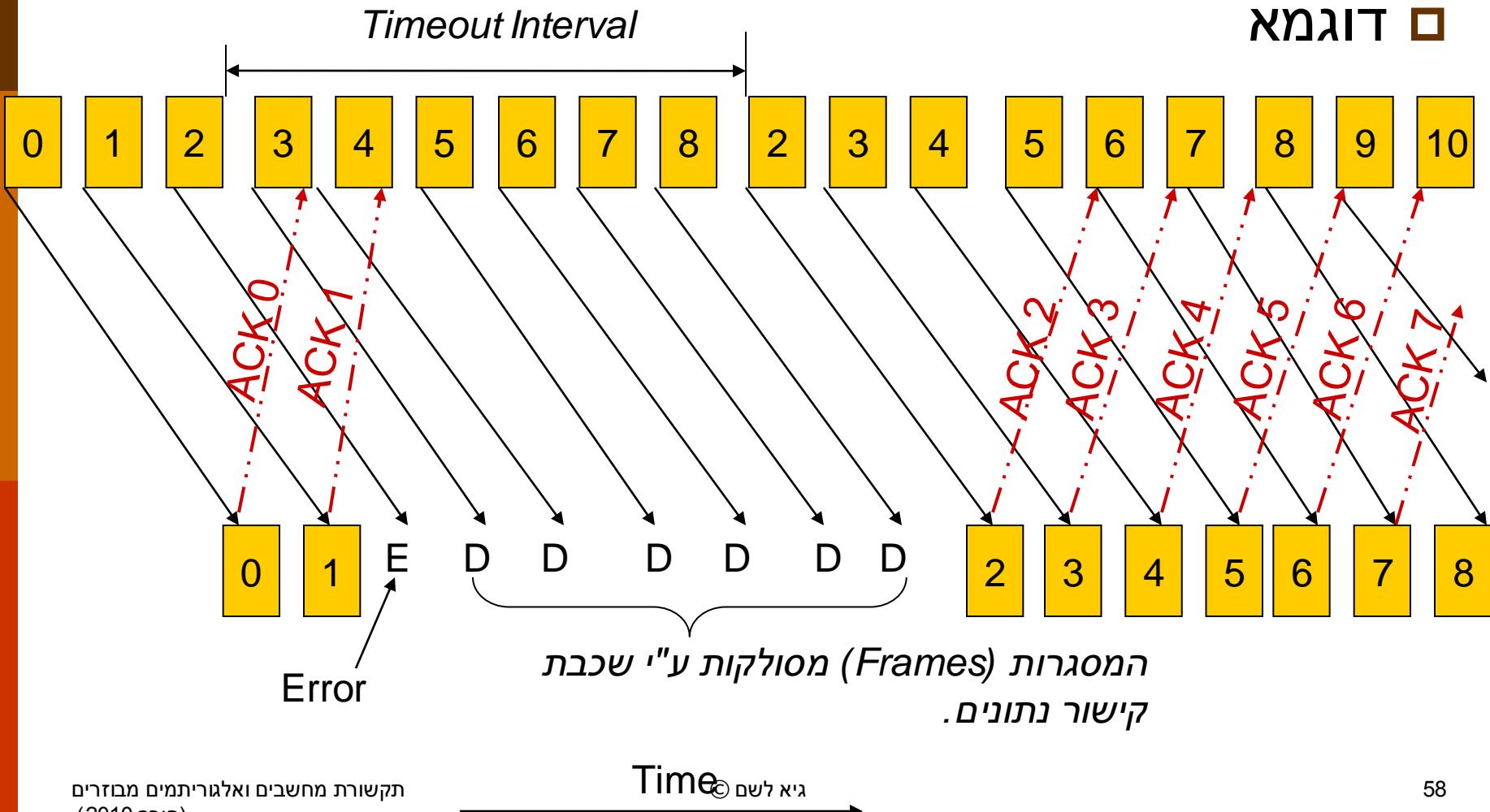
כלואקוי Go-Back-N



- ▢ בפרוטוקול GBN לשולח מותר לשלוח יותר מחבילה אחת בלי לחכות לאישור. קיימת הגבלה למספר הודעות מקסימלי N ב-pipeline. הטווח של מספרים סידוריים אפשריים לחבילות שנשלחו אבל עדין לא התקבל עליהם אישור מוגדר כ"חלון" בגודל N . תור כדי פועלת הפרוטוקול החלון זו קדימה.
- ▢ לכל חבילה יש timer, וכאשר יש אירע timeout לחבילה מסוימת יש שידור מחדש של אותה חבילה. **למქבל אין חוץ** ולכן כל החבילות שנשלחו אחרי חבילה זו ישלו מחדש גם הן. נוצרת בעיה - ככל שהערך פחות אמין יש יותר שליחות מחודשת של הודעות.
- ▢ העובדה שלמქבל אין חוץ היא יתרון עבור רכיבים קטנים.

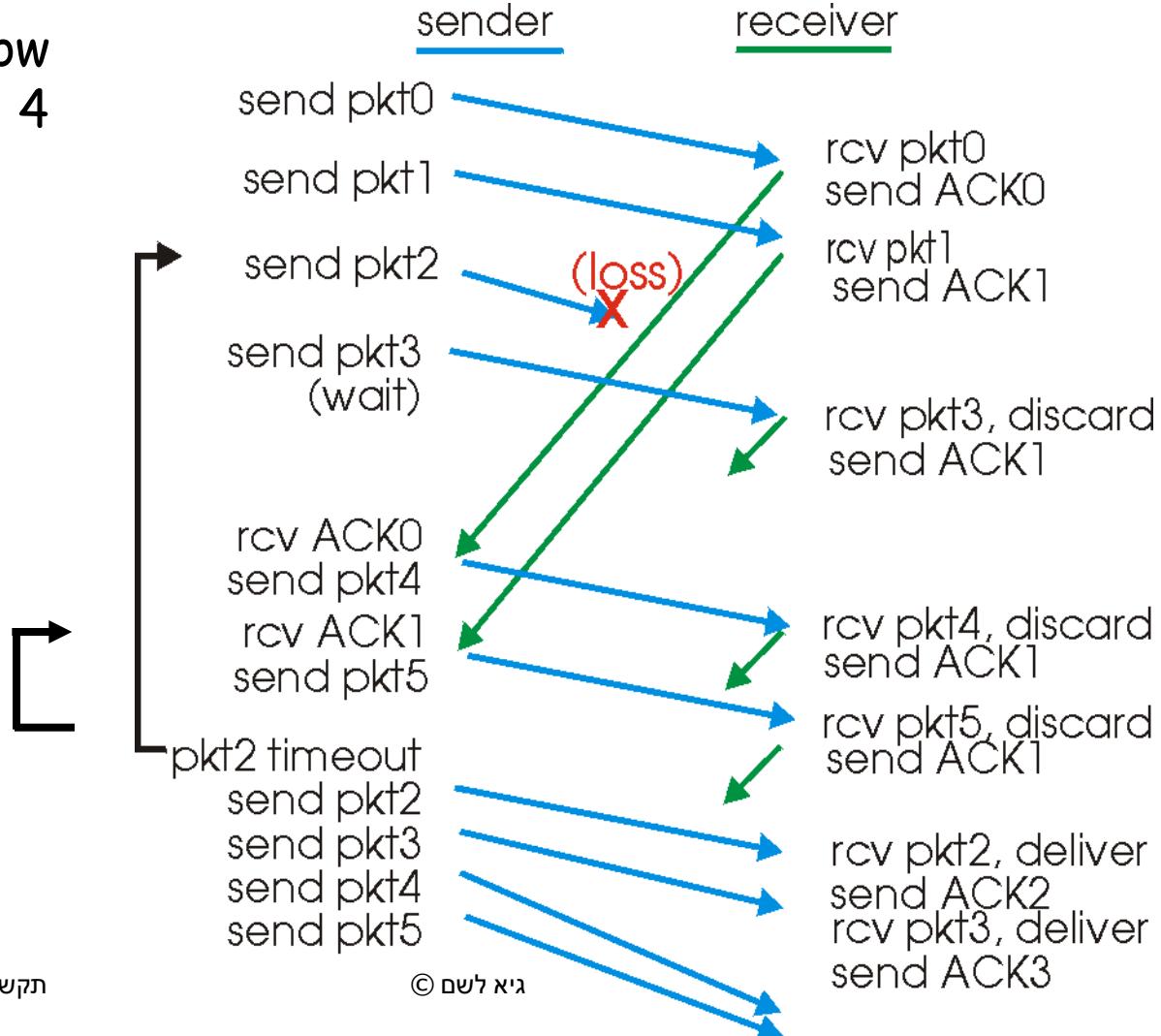
Go-Back-N פיראט

דוגמא

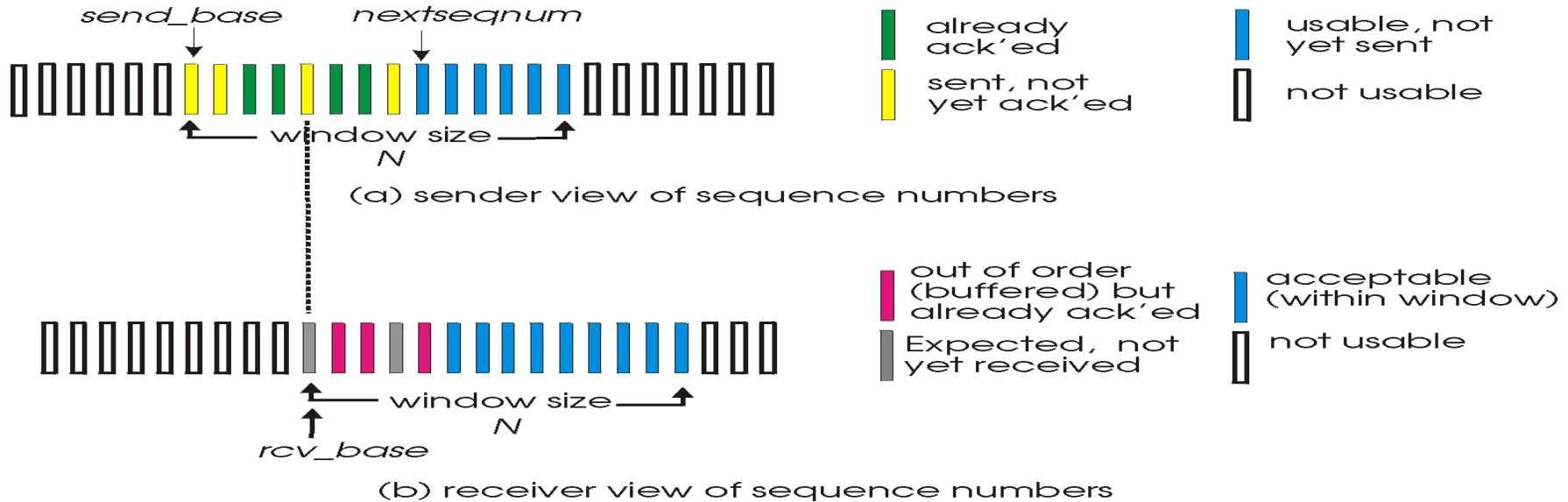


תעלוגיה Go-Back-N

window
size = 4



��לואק אט Selective Repeat



- הפרוטוקול מונע שלילה מיותרת בגל שלוח רק את החבילות שאבדו או הגיעו למקבל משובשות. לשם כך המקלט צריך לשЛОח ACK לכל חבילה שהגיעה תקינה בנפרד.
- נדרש כאן חלון בגודל N כדי להגביל את מספר החבילות הבלתי מאושזרות ב- *cholinepink*. שלא כמו ב프וטוקול GBN השולח כבר קיבל ACK למספר הודעות שנמצאות בתחום החלון. הצד מקבל שלוח ACK לחבילה תקינה שהתקבלה גם אם לא התקבלה בסדר הנכון.
- חבילות שמגיעות בסדר לא נכון נשמרות בחוץ אצל המקלט עד שהחבילות החסרות יתקבלו, ואז החבילות בחוץ יועברו בסדר הנכון לרמה העליונה.

��לט אס (SR) Selective Repeat (SR)

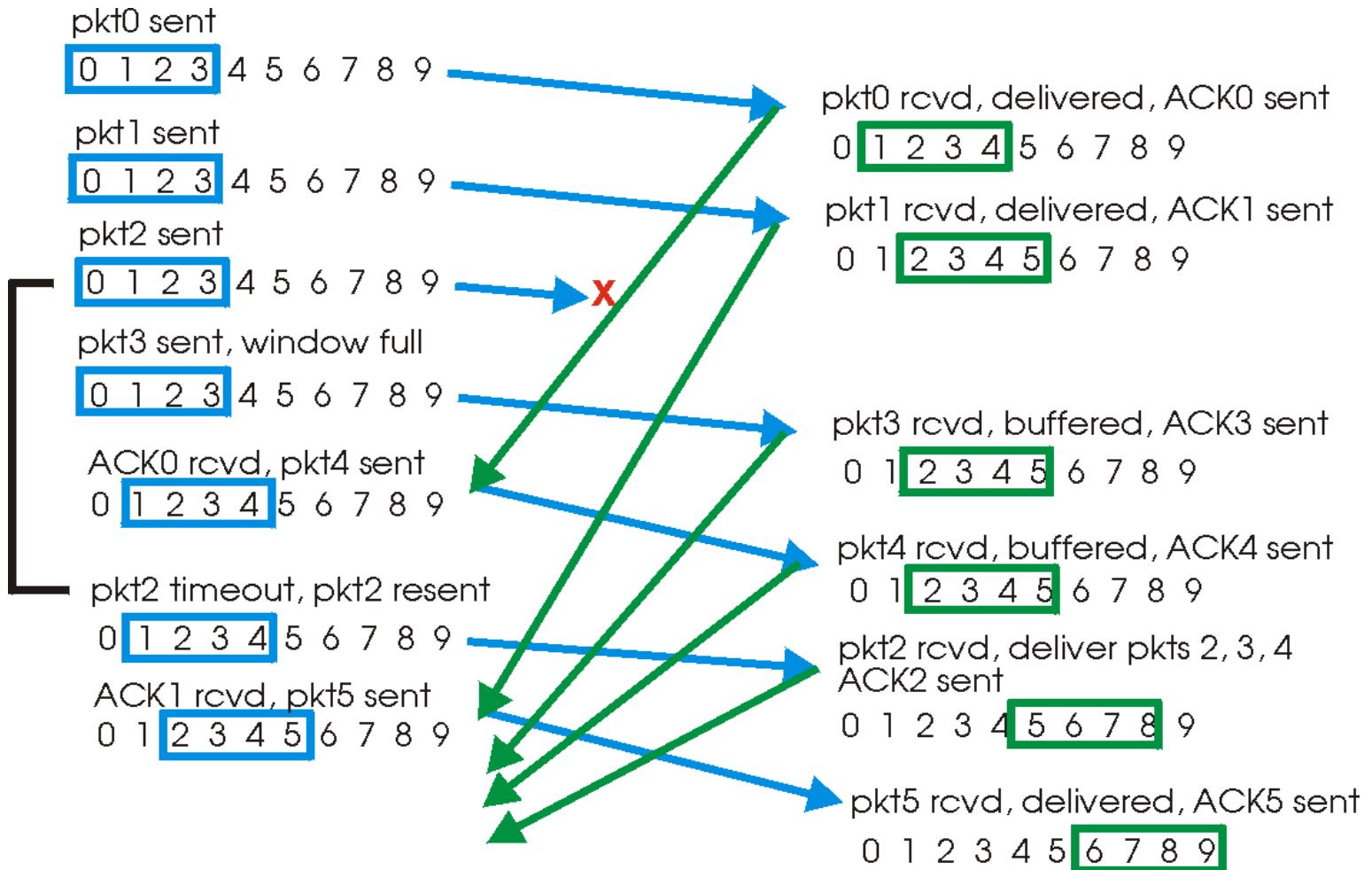
□ תחנה משדרת:

- משדרת חבילות ברצף זו אחרי זו כל עוד הן נמצאות בגבולות החלון.
- אם לא התקבל (i)ACK אחרי זמן (i)timeout משדרת שוב את חבילה i .
- (i)ACK שהתקבל מאשר את חבילה i . אם זו הייתה החבילה בעלת ה-SN הנמוך ביותר שעדיין לא אושר – היא מפונה מהחוצץ, והחלון מתקדם.

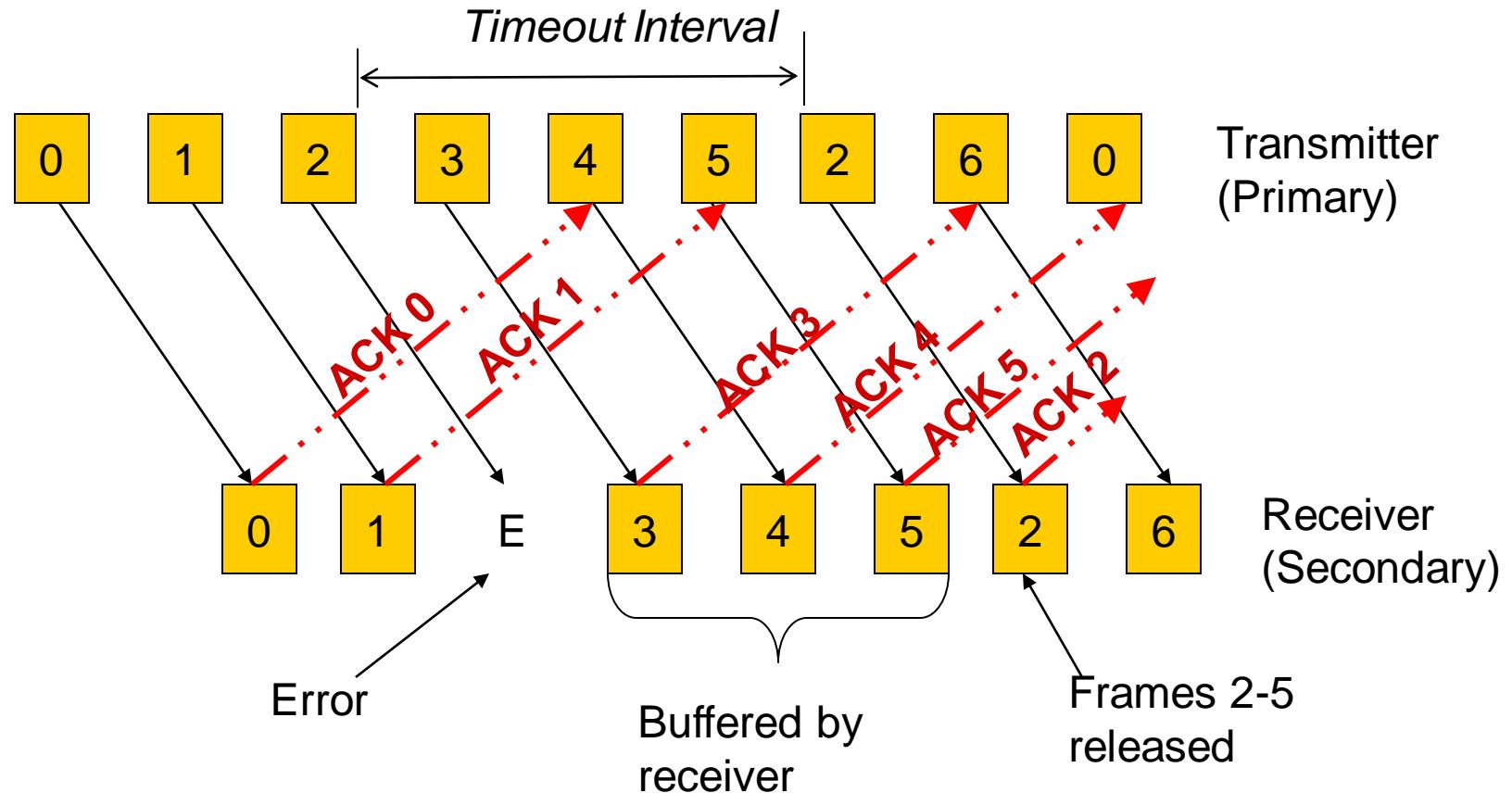
□ תחנה קולטת:

- קולטת את חבילה i ושולחת עליה (i).ACK.
- אם החבילה הגיעה לפי סדר – מעבירה אותה לשכבה מעל, ומקדמת את החלון בהתאם.

כליאקיף נאיף Selective Repeat



Selective Repeat סelektiv knodl?



עכמת כבאות

- שירותים שכבות הhonebhla.
 - ריבוב/פילוג.
 - עקרונות של בקרת עומסים.
 - עקרונות תעבורת מידע
 - תעבורת חסרת קשר: UDP.
 - בקרת עומסים TCP.
 - אמינה ברשת.
- תעבורת מונחת קשר: TCP**
- **מבוא**
 - מבנה המקטע.
 - תעבורת מידע אמינה.
 - בקרת זרימה.
 - ניהול הקשר

TCP – יסוד – תכנון

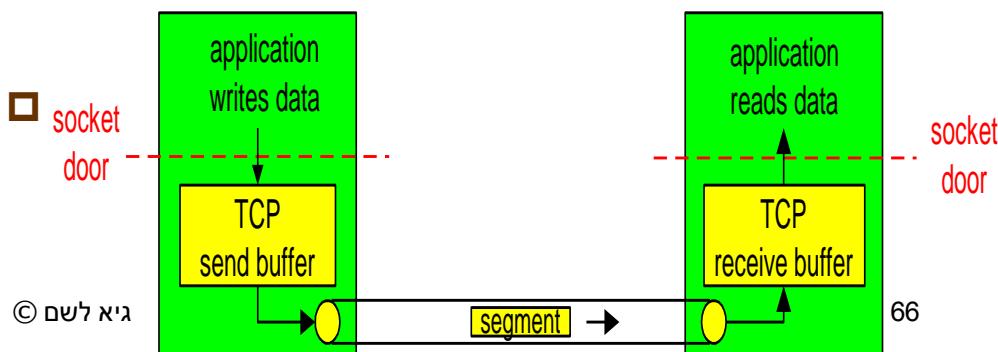
- פרוטוקול המממש מעגל וירטואלי (Virtual Circuit).
- מאפשר תקשורת אמינה מkcחה לkcחה.
- מעביר רצף (לא חסום) של בิตים בשני הכוונים
- משתמש בפרוטוקול IP ל קישוריות בלבד ומנהל את שאר העניינים בעצמו.
- רואה את כל הרשות כערוץ לא אמין המסוגל להפר סדר המשלוח.
- פרוטוקול משפחת GBN (המיישם גם את SR) כאשר המספר מתיחס לבתיים.
- בקרת הזרימה בעזרת חלון.
- הסבר על פועלות פרוטוקול TCP:

■ הקמת הקשר בין תחנת מקור לתחנת יעד זהו רק שלב אחד שבו שיבין שתי התחנות, הצעד הבא הוא ניהול מתמשך של תקשורת בין תחנת היעד לתחנת המקור, כדי לשדר את כל המידע הדרוש, כדי לוודא את הגעתו התקינה וכן לשלווח מחדש חבילות מידע שהגיעו אל היעד פגומות. בכיוון ההפוך, ציריך הפרוטוקול למציג חבילות מידע למועד השלם, תוך הרכבת החבילות בסדר הנכון ולאחר מכן שוויידא שהחבילות תקיןות.

■ פרוטוקול TCP מקבל מידע לשלוח משכבות היישום, מחלק את המידע למספרות שוות גודל ומעביר אותן אל שכבת הרשות להמשר טיפול, כמו כן מטפל ה프וטוקול בהרכבה מחדש של מספרות המגיעות אל התחנה, בונה את המידע השלם ומעבירו אל שכבת היישום. אם לא הגיעו כל חבילות המידע הדרשיות אל תחנת היעד או שחלקן הגיעו במצב לא תקין, תוך תקשורת בין פרוטוקולי TCP במחשב המקור ובמחשב היעד, יעדכנו הצדדים על חבילות שיש לשלווח מחדש ויטפלו בכך עד להשלמת שליחה וקבלת המידע כולו, או עד סיום של פרק הזמן שהוקצב לתהילה, אז תוצג הודעה שגיאיה מתאימה.

מקודמת אוכוּרט קאָר (Connection Oriented) TCP

- קיימ (full duplex data) - קיימ (maximum segment size) MSS שהוא גודל מקסימלי של סגמנט שניtin להעברה.
- מוכוֹנת קשֶׁר (connection oriented) קיימת לחיצת יד (handshake) שמאתחלת את מצב השלוח וה מקבל לפנֵי העברת מידע ביניהם.
- בקרת זרימה (flow control): השולח לא ישלח יותר מידע ממנו שהنعمן יכול לקבל.
- קיימים חוצצים בשני הצדדים.
- קשר נקודת נקודת point to point) - שלוח אחד ומקבל אחד. אין אפשרות ל- multicasting: העברת מידע משולח אחד להרבה מקבלים בפועל שליחת אחת אינה אפשרית.
- הפרוטוקול רץ רק במחשבים הקצה ולא בנתבים בדרך.
- אמינות: הגעת הודעה בסדר הנכון.
- תומך ב- sessions דו-כיווני, שני הצדדים שולחים ומקבלים מידע



סכמי TCP

- מכיוון שסגןטיים של TCP נשלחים כחבילות IP ומכיון שחבילות IP יכולות להגיע לא דוקא באותו הסדר שבו נשלחו, גם סגןטיים של TCP יכולים הגיעו לא באותו הסדר שבו נשלחו. TCP מקבל חבילות מסדר אותן מחדש לפי הסדר שבו נשלחו, אם יש צורך, ועביר את המידע מסודר לאפליקציה.
- TCP מארח וחבילות IP יכולות להשתכפל, TCP יודיע לזרוק חבילות משוכפלות.
- TCP נותן בקרת זרימה. כל צד בקשר TCP מכיל Buffer סופי. TCP שמקבל חבילות מהצד השני יאפשר לצד השני לשЛОוח מידע לפי יכולות-hBuffer שלו, ככלمر לא יאפשר לצד השני לשLOWוח חבילות ללא קבלת אישור מעבר לגודל h-buffer. דבר זה מונע מחשב מהיר לסתום את h-buffer של מחשב איטי.

TCP נותן אמינות ע"י התכונות הבאות:

- המידע של האפליקציה מחולק לחלקים בעלי גודל המוגדר ע"י ה-TCP כ-"גודל הטוב ביותר לשילחה". חלק המידע המועבר ע"י ה-TCP לרמת ה-IP נקרא סגןט.
- כאשר TCP שולח סגןט הוא מחזק Timer שמחכה לקבלת אישור על קבלת הסגןט מהצד השני. אם האישור אינו מגע תוך זמן ה-timeout הסגןט משודר שוב.
- כאשר TCP מקבל מידע מהצד השני הוא שולח אישור על קבלת המידע. האישור אינו נשלח מידית, בד"כ הוא מעקב חלקיק שנייה.
- TCP מחזק checksum של המידע ושל הרישא של חבילת TCP. מטרת ה-checksum למצוא למצוא האם ארהע שגיאה בחבילה בין השידור לקליטה. במידה ו-TCP מוצא checksum שגוי הוא יזרוק את החבילה ולא יודיע לצד השני על קבלתה.

עכמת התקשורת

- עקרונות של בקרת עומסים.
- בקרת עומסים TCP.
- שירותי שכבות ההובלה.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעبورת מידע אמיןہ ברשות.
- **תעبورת מונחת קשר: TCP**
 - מבוא
 - **מבנה המקטע (סגןט).**
 - תעبورת מידע אמיןہ.
 - בקרת זרימה.
 - ניהול הקשר

אקרת TCP מגדלת

Source Port	Destination Port				
Sequence Number					
Acknowledgment Number					
Data Offset	Reserved	Flag	Window		
Checksum		Urgent Pointer			
Option + Padding					
Data					
<p>מזהה את הנקודה בה שכבת היפרxisום בחתנת המקור ובחנתה היעד מקבלת שרotaTCP (SrcPort , DstPort) – 16 בית כל אחד).</p>					
מספר סידורי המזהה את השידור עבור מנוט השיכוכות להודעה.					
מספר סידורי המציין את מספר החבילה שתחנת המקור מצפה לקבל (32 בית).					
מצין את מספרן של המילים בנות 32 בית בគורת מסגרת ה-TCP.					
שמור לשימושים עתידיים.					
נושא מידע בקרה מגוון.					
מגדיר את גודל החוץ המוצע לקבלת נתונים בחתנת המקור.					
מצין האם החבילה נפגעה בתהליכי המשלוחה.					
מצביע אל שדה המידע בחבילה.					
מגדיר מגוון אפשרויות TCP.					
מכיל את המידע משכבה היפרxisום.					
Destination and Source Port					
Sequence Number					
Acknowledgment Number					
Data Offset					
Reserved					
Flag					
Window					
Checksum					
Urgent Pointer					
Option					
Data					

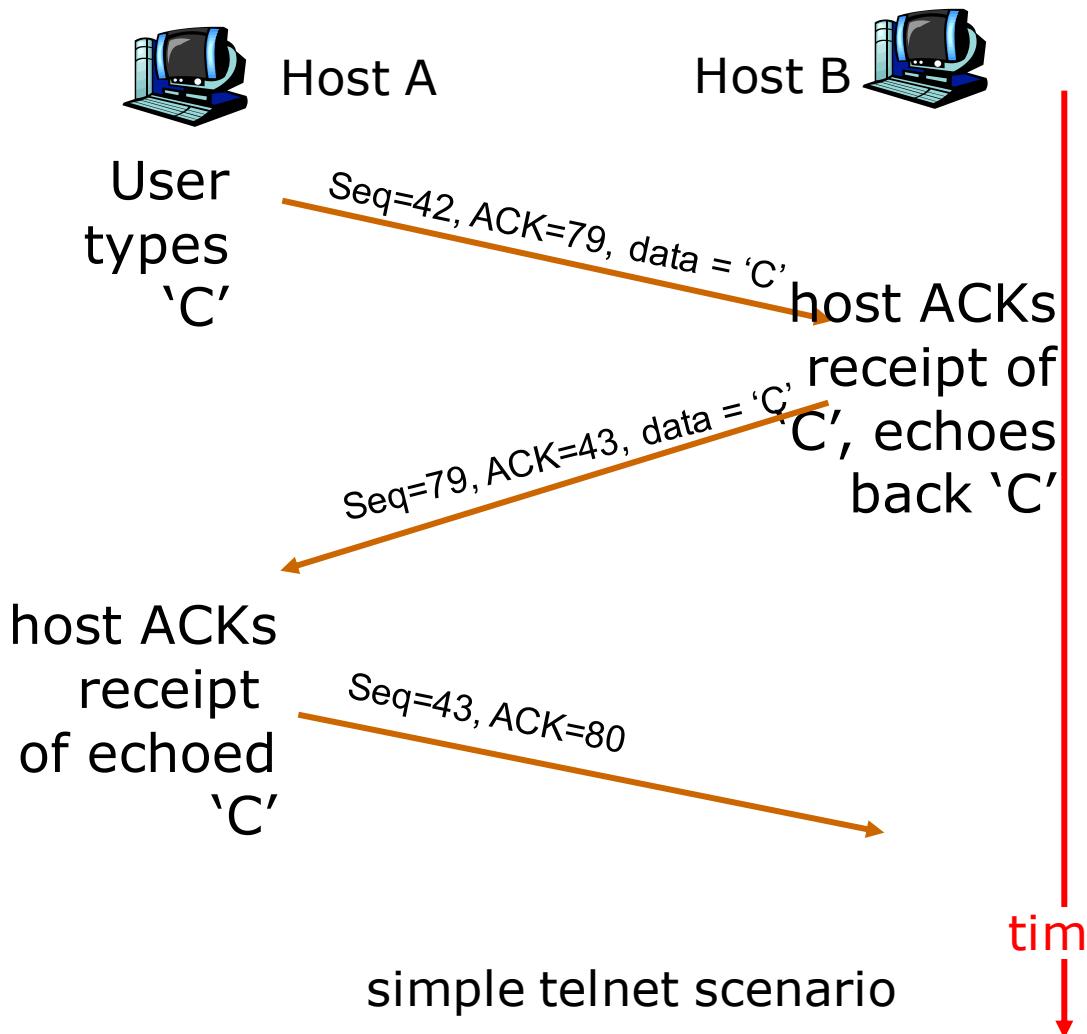
איזה כלאי סיגו/מיקו סדיות פה כאן 4 לפניהם 2

- **כיוון شبשכבות קישור נתוניים שירות משלוח אמין לא תמיד קיימים.**

הסבר:

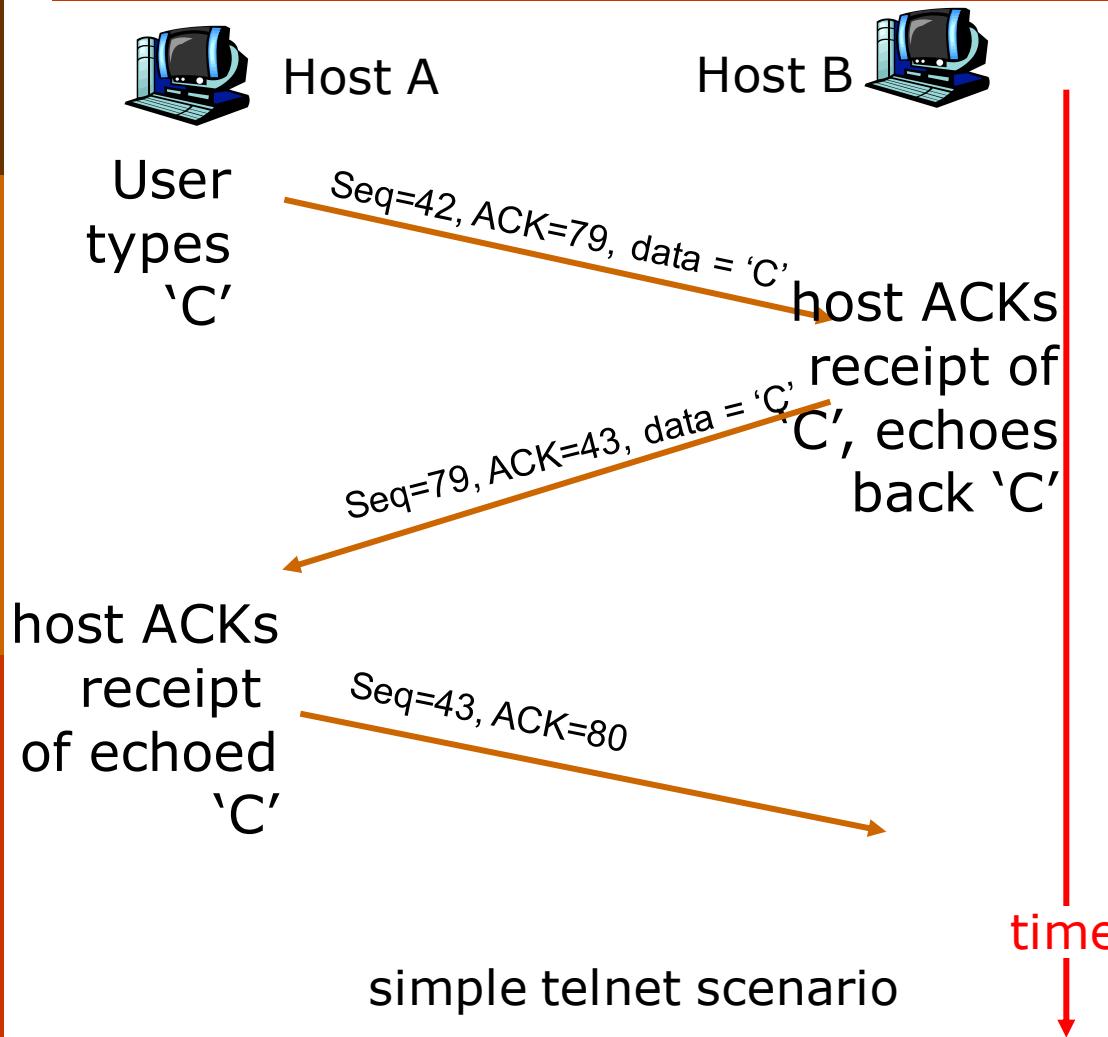
- משלוח אמין: אם שכבת קישור הנתוניים מספקת שירות משלוח אמין, אז זה מבטיח כי חבילת המידע תגעה בערוץ ללא שגיאות.
- נזכר שפרוטוקולי שכבת התעבורה (כגון TCP) יכולים גם הם לספק שירות של משלוח אמין. בדומה לשירות משלוח אמין של שכבת התעבורה, שירות משלוח אמין של שכבת קישור נתוניים מושג ע"י אישור על קבלה (acknowledgments) ומשלוח מחדש (retransmissions).
- שירות משלוח אמין של שכבת קישור נתוניים בד"כ משתמש עבור ערוצים שונים לקצב שגיאות גבוהה, כגון ערוצים אלחוטיים, עם מטרה של תיקון שגיאות מקומי בערוצים שבהם קוראות שגיאות במקומות לאלו שליחת נתוניים מקצת לkaza ע"י פרוטוקול שכבת התעבורה או האפליקציה.
- בכל מקרה, שירות משלוח אמין של שכבת קישור נתוניים לעיתים קרובות הוא נחשבת עם תקורה מיותרת עבור ערוץ עם קצב שגיאות נמוך, הכלל סיב אופטי, קבל קוואקסיאלי וערוצי זוג שזרור רבים. מסיבה זו, פרוטוקולי שכבת קישור נתוניים פופולרים רבים לא מספקים שירות מישלח אמין.

כלאוקים TCP נאכזות גז היפר



- כאשר מקבל מידע מרמת היישום:
 - נוצר סגמנט עם מס' סידורי שהוא מספרו הסידורי של ה-byte הראשון באותו סגמנט.
 - אתחול ה-timer אם אינו מופעל כבר. (ה-timer הוא עבור ההודעה הישנה ביותר שלא קיבל עליה ACK).
- Timeout: שליחה מחדש של סגמנט לאחר ל-timer ואתחול ה-timer.
- קבלת ACK: אם מקבל ACK על חבילות שעדיין לא הגיע להן ACK, אז מתבצע עדכון של הסגמנטים אשר עליהם צריך לעשות ACK.
- במקרה זה שלחhim סיגמנט עם 1 byte

כלואוקי TCP - ACK'ות נאכזות נזק באנקוף



- מ' סידורי: סגמנט עם מ' סידורי שהוא מספר הסידורי של ה-byte הראשון באוטו סגמנט.
- ACK: מ' סידורי של ה-byte הבא אותו מצפים לקבל מהצד השולח.
- הצד מקבל מתמודד עם סגמנטים שהגיעו לא לפ' הסדר בכר שהוא מתעלם מחבילות ישנות, ושם בחוצצים או מתעלם מחבילות עתידיות.

TCP Round Trip Time and Timeout

שאלה: איך לאמוד את RTT ?

תשובה:

- כדי לאמוד זמן RTT יש לקחת דגימות שהן הזמן משילוח סיגמנט ועד לקבלת ה-ACK. מעריכים שלוחות כפולות ומבצעים ממוצע בין הדגימות.
- יש שימוש בקדם ביטחון שגדל ככל שיש הבדלים יותר גדולים בין ערכי הדגימות. לבסוף נקבע ערך ה- timeout ע"פ זמן ה- RTT המשוער אליו נוסיף את מקדם הביטחון: (ערך טיפוסי של α הוא 0.125).

שאלה: איך נאמוד את הערך של ה- Timeout ב- TCP ?

תשובה:

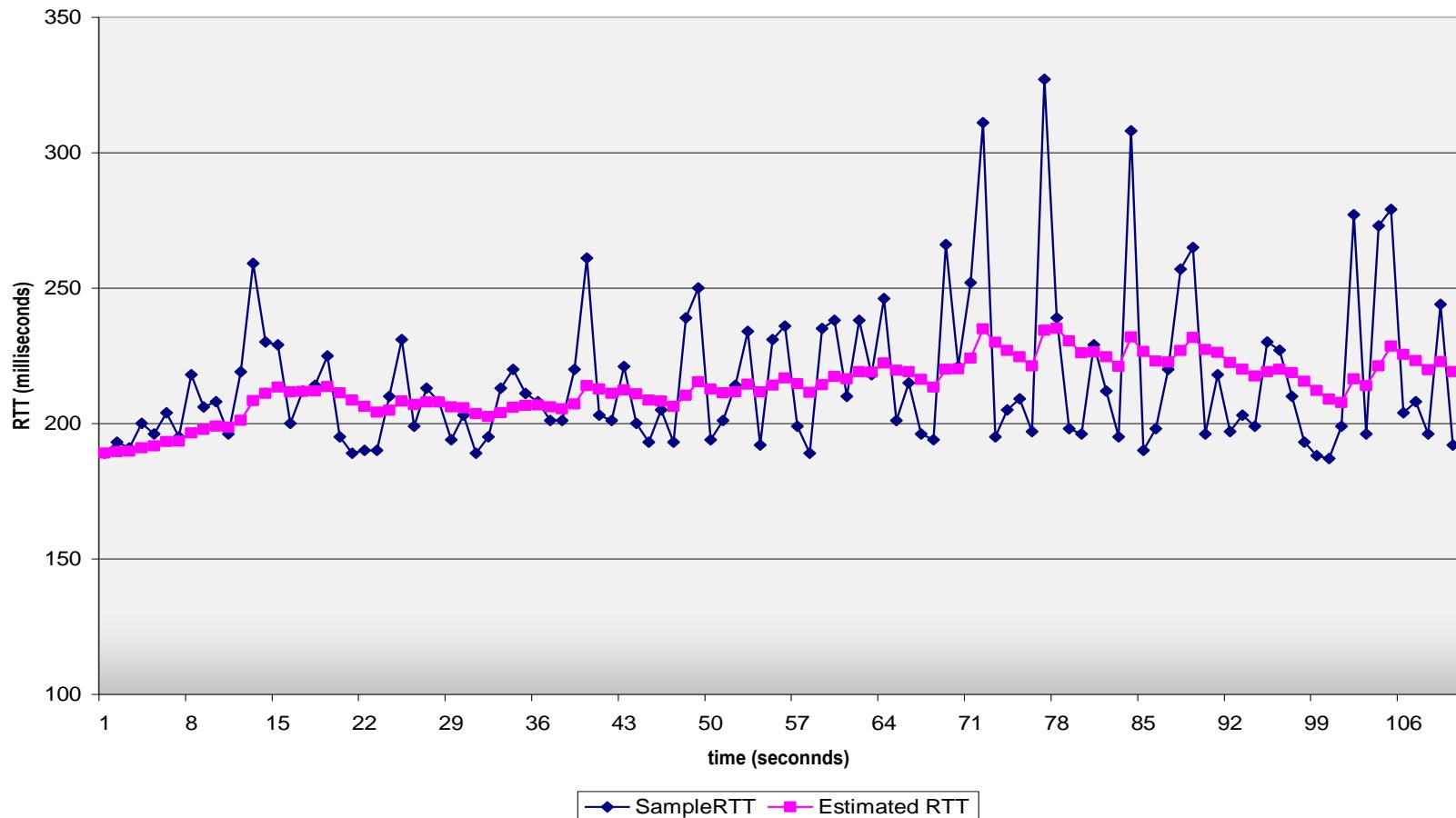
- ישנו קושי בקביעת ערך ה-Timeout ב- TCP. ערכו צריך להיות גדול מערך ה- RTT שימושה.
- علينا לבחור RTT שבמידה יהיה קצר מדי הדבר יגרום להרבה שלוחות כפולות (retransmission) ובכך הולמת התקורה (המידע המצורף להודעה ברשות כדי להבטיח העברת נקייה מטעויות ליעד הנכון), ובמידה יהיה ארוך מדי הדבר יגרום לאטיות ואובדן מידע.

$$\text{EstimatedRTT} = (1-\alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

– מודד את הזמן משילוח הסיגמנט ועד לקבלת אישור (ack). SampleRTT

RTT נקיינט

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



שכמת התקשורת

- עקרונות של בקרת עומסים.
- בקרת עומסים TCP.
- שירות שכבת ההובלה.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעבורת מידע אמינה ברשת.
- **תעبورה מונחת קשר: TCP**
 - מבנה המקטע.
 - **תעבורת מידע אמינה.**
 - בקרת זרימה.
 - ניהול הקשר

TCP מזג אמת איז איננה

- TCP יוצר שירות `rdt` על קצה של שירות ה-IP הלא אמין.
- מקטעי `Pipelined`.
- הצלבות של `acks`.
- TCP משתמש ב-`timer` (זמן) יחיד לשילחה מחדש.
- השילחה מחדש מתעוררת כתוצאה מ:
 - פסק זמן של אירועים.
 - שיכוף של `acks`.
- בתחילת הצורך הפשטות ה-TCP השולח:
 - מتعلم מ-`acks` משוכפים.
 - מتعلم מבקרת הזרימה ועומסי הזרימה.

TCP מנגנון של קדימה?

```
NextSeqNum = InitialSeqNum  
SendBase = InitialSeqNum
```

```
loop (forever) {  
    switch(event)
```

event: data received from application above
create TCP segment with sequence number NextSeqNum
if (timer currently not running)
 start timer
pass segment to IP
NextSeqNum = NextSeqNum + length(data)

event: timer timeout
retransmit not-yet-acknowledged segment with
smallest sequence number
start timer

event: ACK received, with ACK field value of y
if ($y > \text{SendBase}$)
{
 \text{SendBase} = y
 if (there are currently not-yet-acknowledged segments)
 start timer
}
/* end of loop forever */

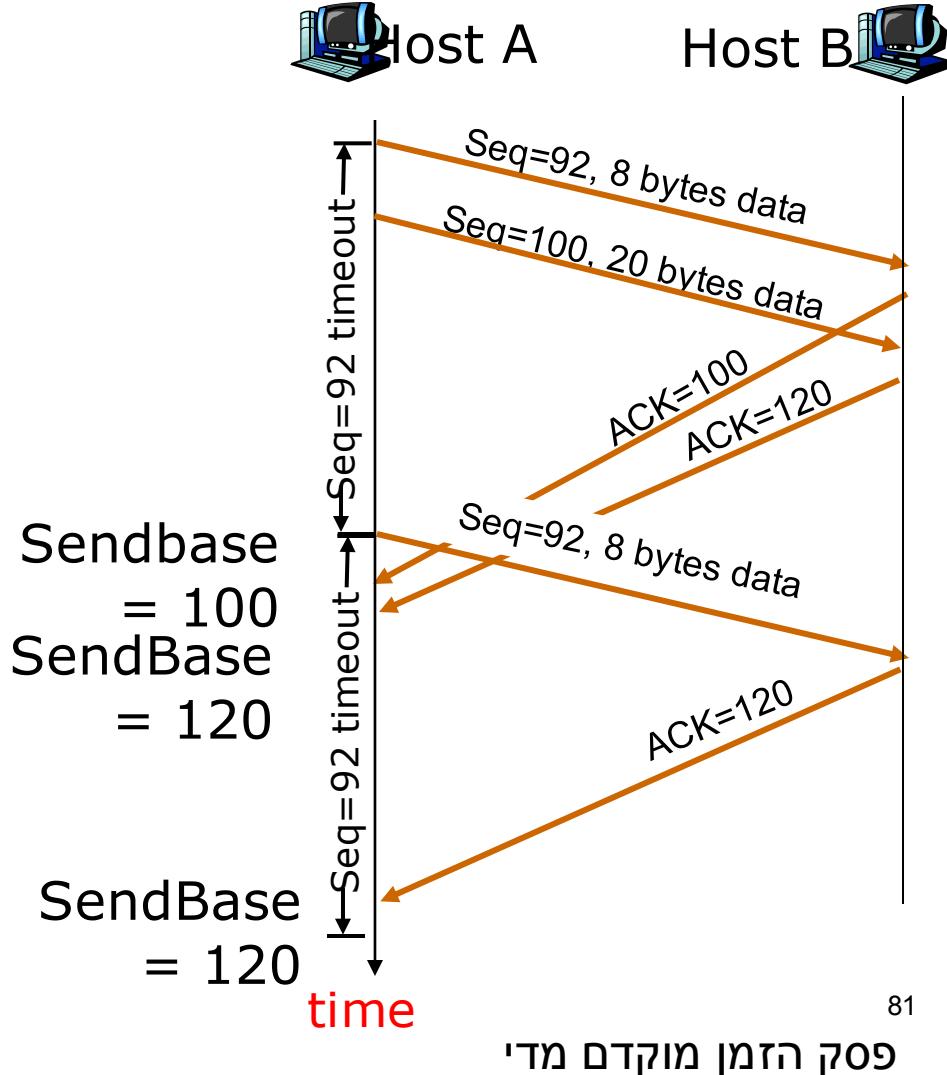
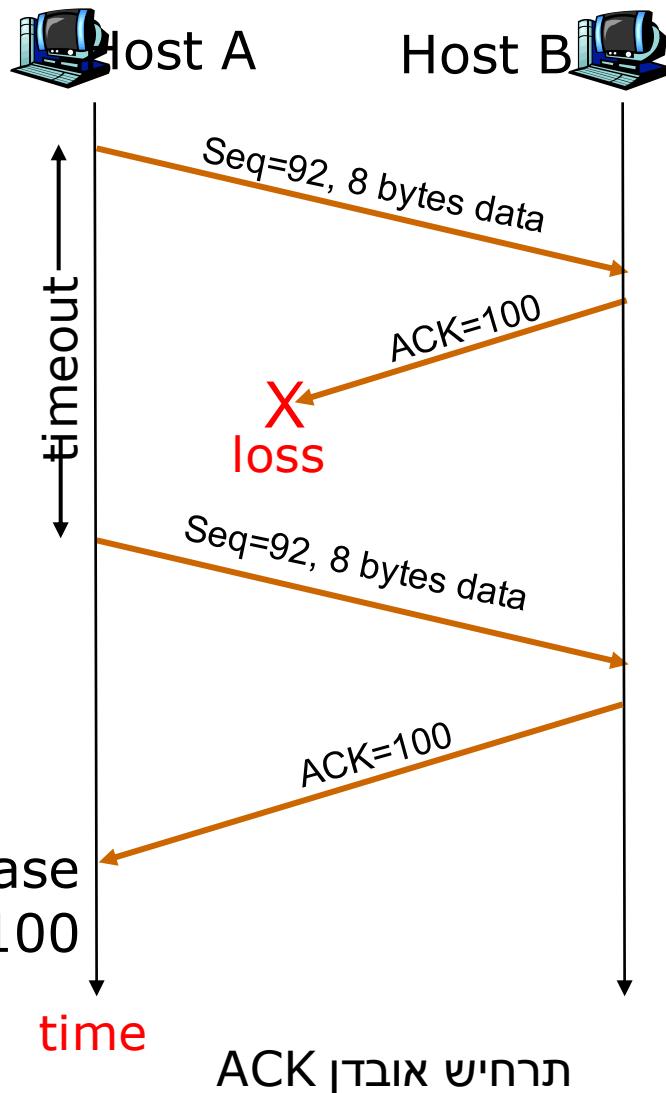
Comment:

- $\text{SendBase}-1$: last Cumulatively ack'ed byte

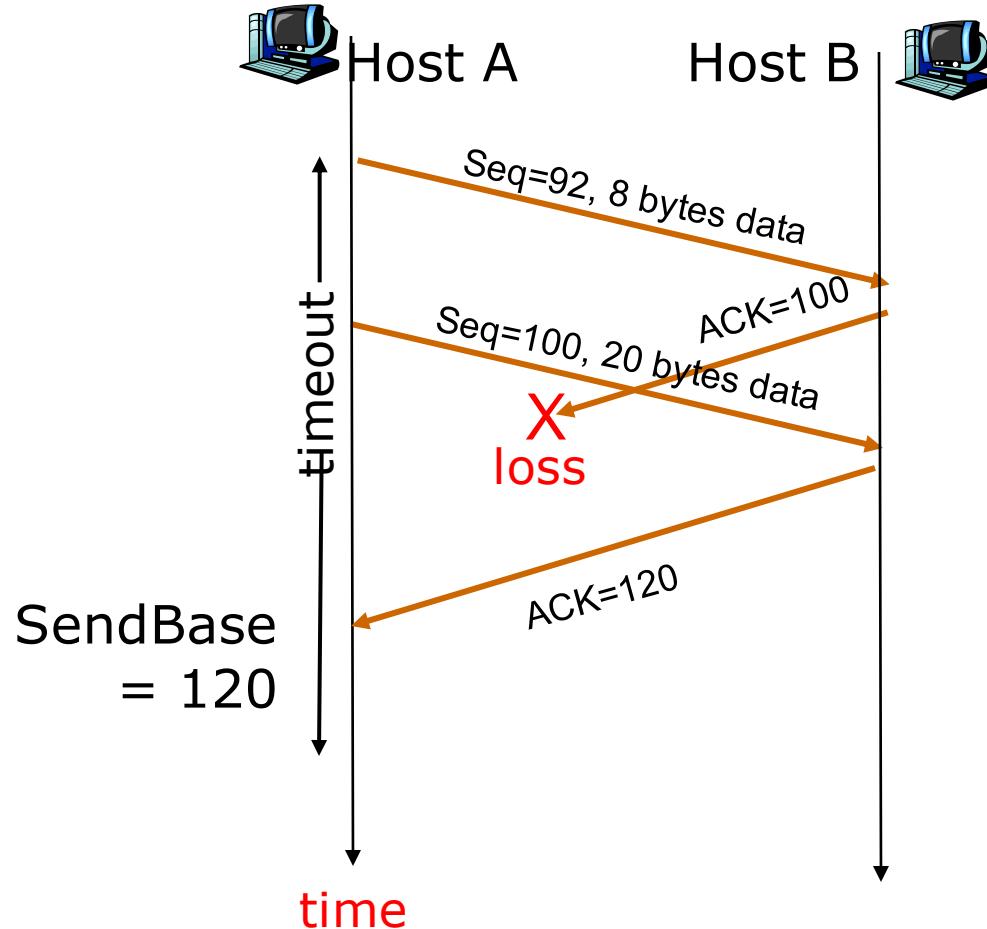
Example:

- $\text{SendBase}-1 = 71$;
 $y = 73$, so the rcvr wants $73+$;
 $y > \text{SendBase}$, so that new data is acked

תיכונת TCP-הפרוטוקול



תרכיש הצבורות ACK



תרכיש הצבורות ACK

TCP→ ACK ה'ג'

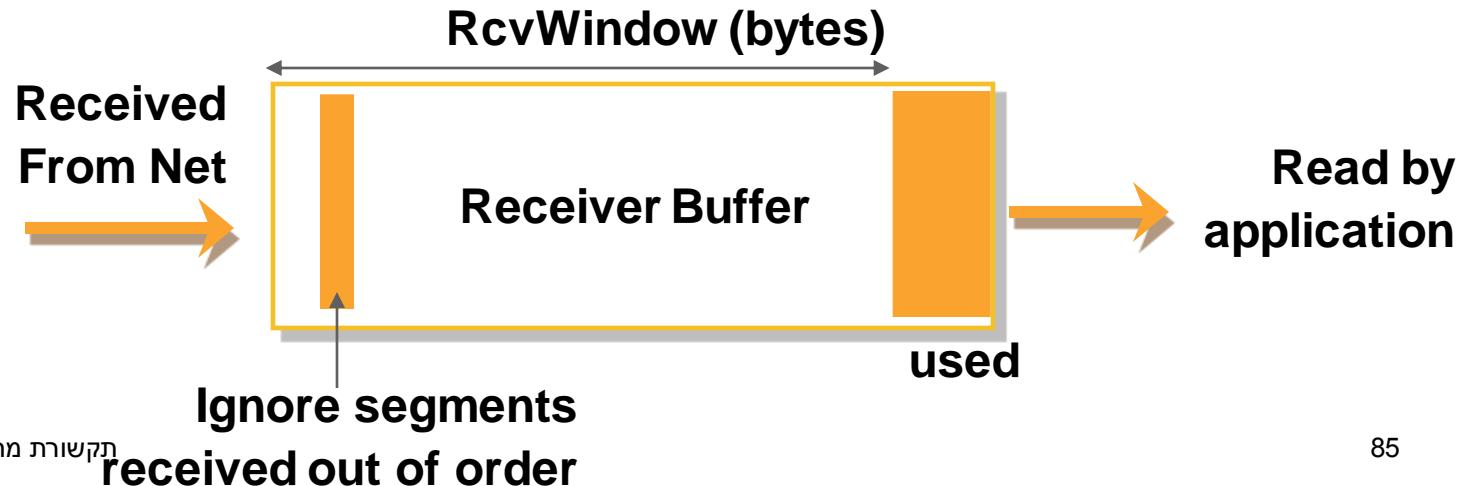
Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send single cumulative ACK
out-of-order segment arrival higher-than-expect seq. # gap detected	send duplicate ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate ACK if segment starts at lower end of gap

עכמת התקשורת

- שירותים שכבות ההובלה.
- עקרונות של בקרת עומסים.
- בקרת עומסים TCP.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעבורת מידע אמיןہ ברשות.
- **תעבורת מונחת קשר: TCP**
 - מבוא.
 - מבנה המקטע.
 - תעבורת מידע אמיןہ.
 - **בקרת זרימה.**
 - ניהול הקשר

נקמת לכיאת נ-TCP

- **הבעיה:** האפליקציה במחשב המקלט יכולה להיות איטית יותר מאשר הרשות \leftarrow לא יהיה מקום למידע המתקבל.
 - **בקרת הזרימה:** למנוע מהמחשב המקלט גלשת מידע בחוץ, כולם נחפש מנגנון שבאמצעותו דואגים שלא תשלח יותר אינפורמציה מאשר המקלט יוכל לשמר אצלם. ולא נגיע במצב שבו החוץ מלאים כאשר האפליקציה עוד לא קראה מהזיכרון.
 - נעשה זאת ע"י הגבלה של מקטעים ללא ACK במעבר.
 - המקלט מקצה מקום למידע שהוא אמרור לקבל מהשלוח – RcvBuffer ומסמן לשולח כמה מקום נותר לו פנוי לאורך זמן החיבור – Window: RcvWindow
- : $RcvWindow = (|Rcvbuffer| - |used|) / MSS$
- **MSS (Max Segment Size)** – גודל סגמנט מקסימלי בBITS.



נקמת לכיאת ה-TCP

- **פתרון:** השולח מגביל את נפח המנות שהוא יכול לשולח ללא קבלת ACK, נפח זה הוא כנפח ה-RcvWindow. זאת כדי למנוע הצפה של החוצצים של מקבל.
- ה-ACKים שמתקבלים בשולח, לא תמיד מגיעים לפि סדר השליחה המקורי.
- סיבות לכך שהחוצץ מלא:
 - חבילה ישנה לא התקבלה (בשלוח לא קיבל ACK עליה).
 - האפליקציה במקבל לא קראה עדין מידע מהחוצצים.

נקמת לכיהTCP-ה RcvWindow

Source Port	Destination Port				
Sequence Number					
Acknowledge Number					
Hdr Len	RS V	Flags	RcvWindow		
Checksum		Urgent pointer			
Options (variable length)					
Payload (Application Data)					

RcvWindow: □

- שדה Flow control ב-TCP header
- מספר מקסימלי של מקטעים ללא ACK למחשב עמית יכולים להשלח.

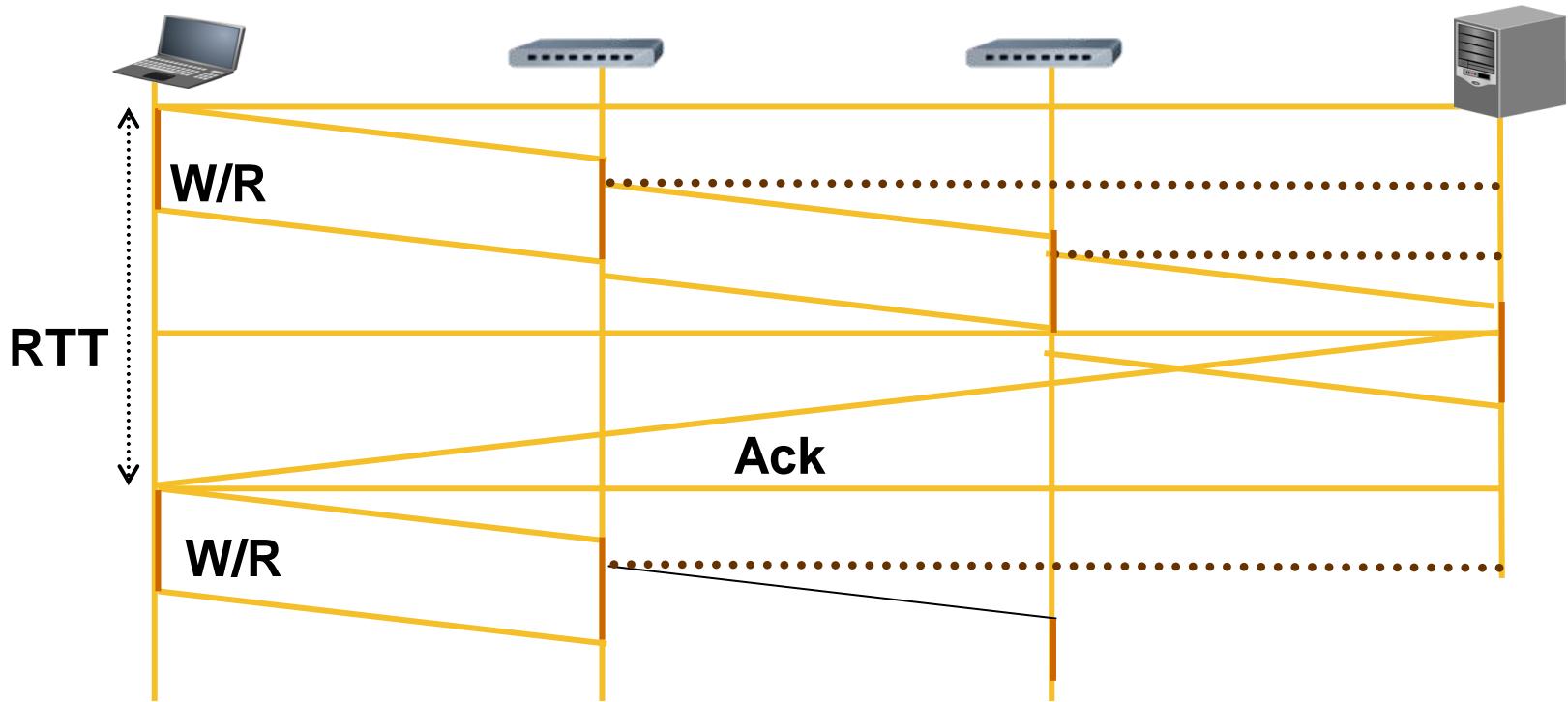
סבירו: שדה ה-RcvWindow הוא מנגנון בקרת זרימה המרכזית של TCP. הוא מצין את המספר המרבי של מקטעים שהשלח יכול לשולח לפני שהוא מקבל ack. שדה זה נשלח בדרך כלל מתחנת הנמען, כאשר מקבלים ack של מקטע, אבל אם צריך גם המחשב לקבל רשיי לשולח כדי להודיע לשולח על עלייה ב-RcvWindow.

פואט כהן גור (1)

- גודל החלון הוא אלמנט קריטי עבור הביצועים
 - אם יותר $W_{Rcv} < W_{Send}$ מידי קטן, הביצועים יפגעו.
 - אפילו אם הרשת זמינה, האפליקציות יפגעו.
- דוגמא: נסתכל על רשות התקשרות פשוטה הבא:
 - $W = \text{גודלו של חלון (בBITS)}$
 - $R = \text{קצב השליחה של כל הערכות}$.
 - $MSS = \text{גודלו סגמנט מקסימלי בBITS}$.
 - $RTT = \text{Round Trip Time (זמן המחזור)}$

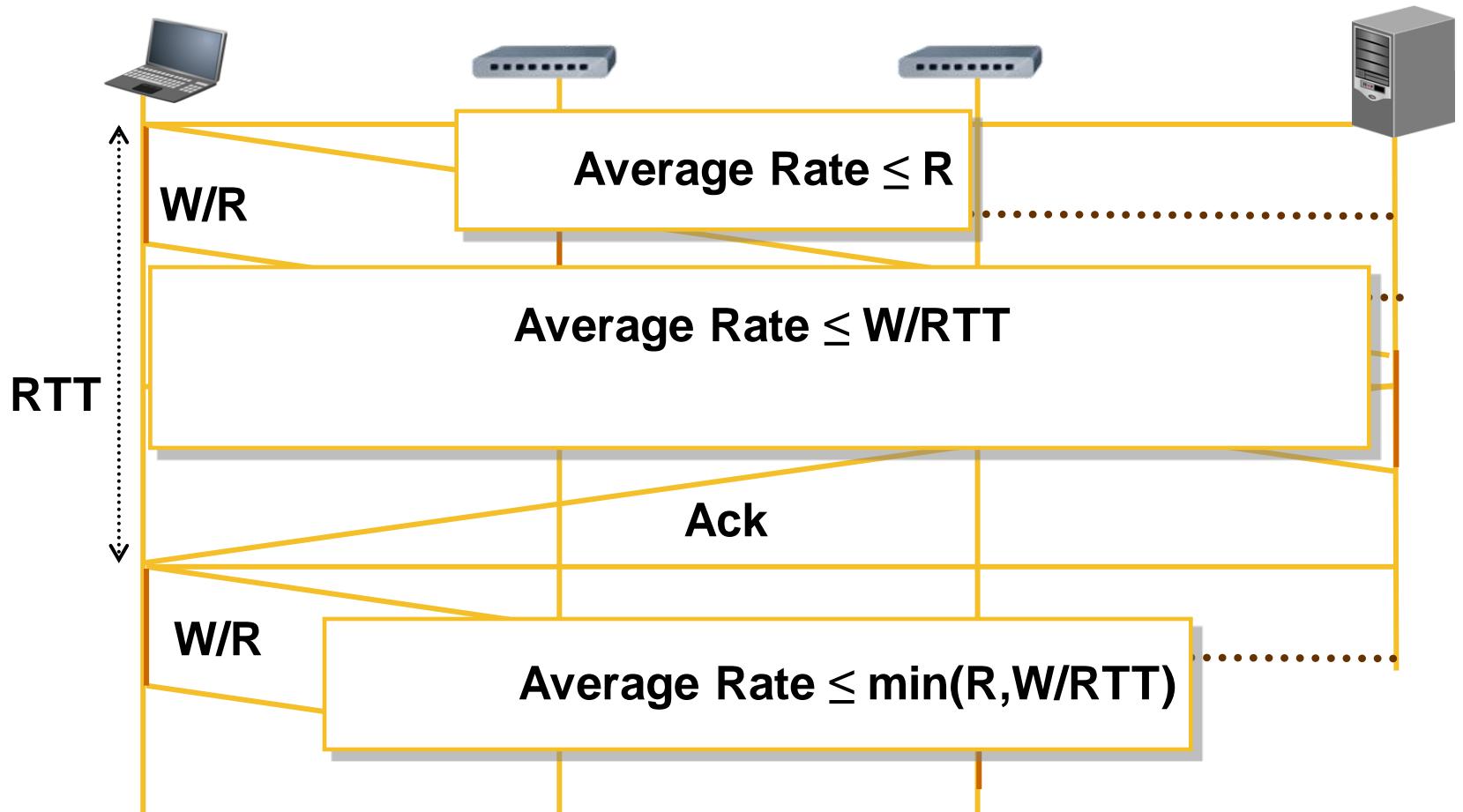


לעג תחמי (2)



כאן אנו רואים את ההשפעה של גודל החלון: נסמן את גודל החלון - W , ואת הקצב השידור R , מכאן זמן שידור הוא R/W . נניח, כמו באIOR, כי זמן המחזזר RTT (זמן העובר מתחילה שידור החבילה בצד השולח ועד קבלת הביט האחרון של חבילת האישור נשלחת ע"י מחשב היעד עם סיום קליטת חבילת המידע) גדול בהרבה R/W . מה שקרה בעצם הוא שאנו חנכו שולחים את כל חלון (W) ויש לנו ללחכות ל-ack (אשר יקח כמובן RTT). ואז במשך (R/W -RTT) העורץ לא היה בשימוש!

תקן CANNAI



הקצב הממוצע – פונקציה של זמן שימוש הערוץ מתוך זמן האפשרי לשימוש
ברצף נחפש את המינימום מבין הקצב השידור R והיחס שבין גודל החלון W
לזמן המחזורי RTT .

אנו קמץ לאטויין!

- █ קצב שידור ממוצע = $\min\{R, W/RTT\}$
- █ בקرت זרימה אינה צואր בקבוק כל עוד: $R * RTT \geq W$ (גודל החלון W חייב להיות גדול ממכפלת קצב השידור בזמן המחזור על מנת לא לגlossen מהחוץ)
- █ החוץ מקבל "ReceiverBuffer" שווה ל:

$ReceiverBuffer \geq (Delay \times Bandwidth) \times (Width \text{ (יעט)}$

- █ אם החלון הוא קטן מדי: רידה במריבות היחסור.
- █ ערך "שםן" - לדוגמה לוין רוחב פס גדול שווה ל:
Fat links: huge Bandwidth x Delay (e.g. satellite)
- █ בקרט זרימה אגרסיבית: נשלח RcvWindow אחר יותר מאשר החוץ האפשרי.
 - לפי הערכת קצב "הניקוז", והסיכון של "הצפת" החוץ
 - פועל היטב כאשר העיכוב אינו משנה הרבה

חומר TCP: נסעה קליינט

- חוץ קבלה TCP: קרייטי עבור **ביצועי המהירות**.
- אבל חוצצים גדולים מגבילים את מספר החיבורים
 - 10,000 חיבורים, 10KB לכל חיבור \leftarrow MB 100.
 - השלכות על העליות והביצועים.
- השירות חייב למצוור את כמות החיבורים הפתוחים
 - הלוקחות חייב לסגור חיבורים (ולהמתין 30 שניות).
 - שירותי רבים יזמו סגירה, אבל לא ימתינו 30 שניות
 - אם ה-Ack אבד, הלוקח עשוי לקבל פסק הזמן (לסגור באופן חריג)

מג'ור צוות...

- נדרש עבור הרשת חוץ גודל, המקבל
 - ברירת מחדל, באופן שמרני: חוץ המקבל $>$ חלון.
 - אגרסיבי, בקרת זרימה אופטימאלית: חוץ $<$ חלון (ע"י הערכת קצב הניקוז, סיכון של הצפת החוץ).
- ביטים רבים יכולים להישלח ל-'חברים שבורים' או רשת עמוסה.
- תורים גדולים בנתב \leftarrow עיכוב, עומסים, אובדן... לא טוב!
- פתרון: להגביל את גודל החלון גם ע"י בקרת עומסים.

עכמת התקשורת

- עקרונות של בקרת עומסים.
- בקרת עומסים TCP.
- שירותים שכבות ההובלה.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעבורת מידע אמיןہ ברשות.
- **תעבורת מונחת קשר: TCP**
 - מבוא
 - מבנה המקטע.
 - תעבורת מידע אמיןہ.
 - בקרת זרימה.
 - **ניהול הקשר.**

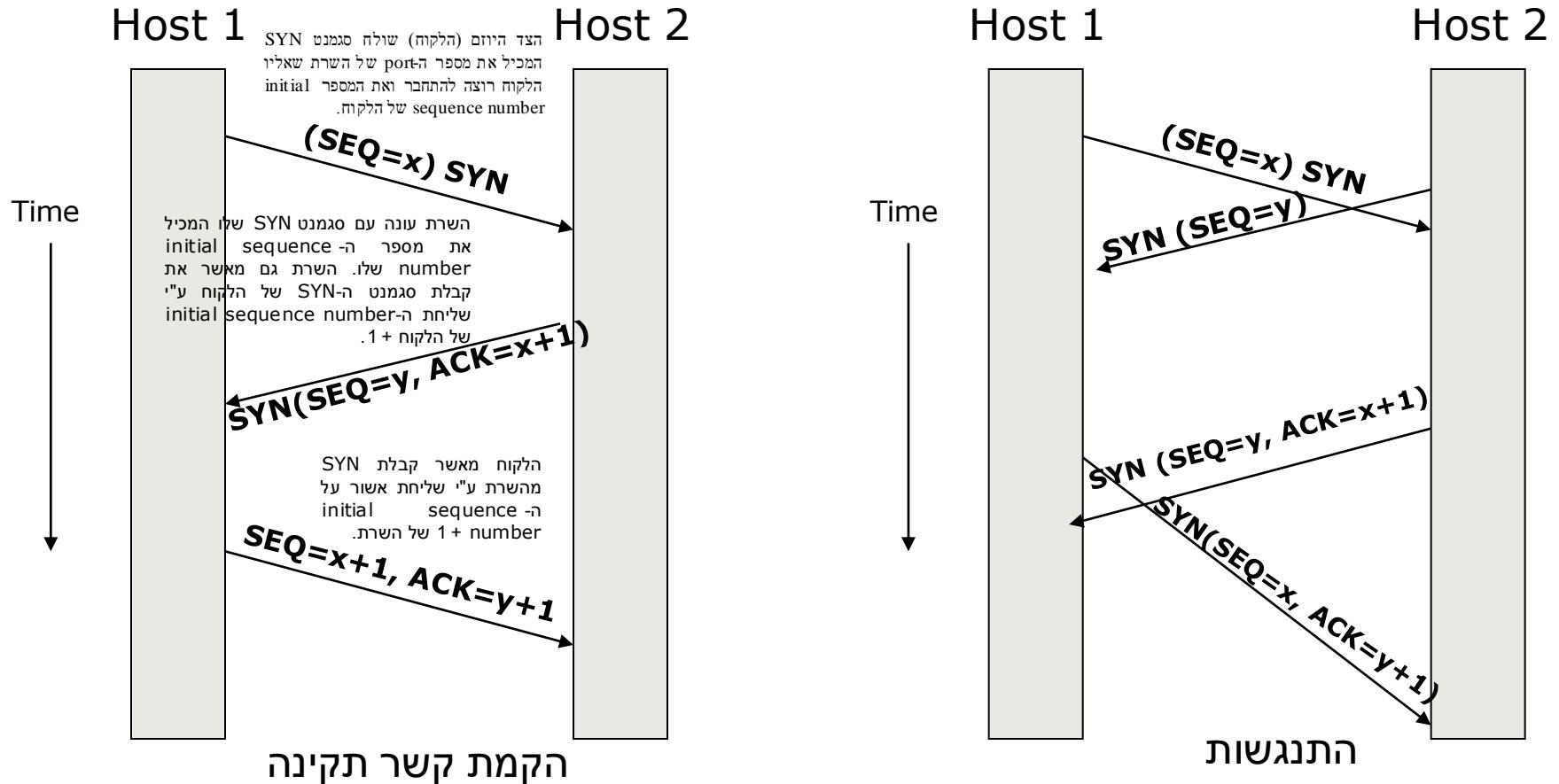
תקאת קאכ – TCP

- הקמת הקשר עובדת לפי לחיצת יד מושלשת והמספרים הסידוריים בנויים לפי שעון למקורה של הטרנסקוט, כפי שראינו קודם.
- כמו כן, בעת הטרנסקוט של תחנה, היא ממתינה זמן מחזור שלם של הודעה עד לשלב בו היא تتאושש.
- אלגוריתם התקשרות three way handshake פועל כך:
 1. הלקוח שלח הודעה ובה ה `sequenceNum` הראשון $= \text{sequenceNum} - \text{flag} = \text{SYN} = x$.
 2. השרת שמקבל את ההודעה מאשר את קבלת המשלוח, ושלח את המספר הסידורי של המשלוח שלו.
 $(\text{SYN+ACK}: \text{sequenceNum}=y, \text{acknowledgment}=x+1)$
 3. הלקוח מוחזיר אישור על הודעה השרת
 $(\text{ACK}: \text{acknowledgment}=y+1)$
- לשתי ההודעות הראשונות מתוך השלושה נלקח `timer`, ואם לא מגיעה תגובה בזמן המוגדר, ההודעה נשלחת שוב. ההודעה השלישית לא מגובה וגם אם לא תתקבל התקשרות תחילית.
- הסיבה לשילוח ה `sequenceNum` היא שהפרוטוקול דורש שכל צד יבחר בתור מספר הודעה ראשון מספר רנדומלי, ולא אפס או מספר מסוים אחר כיון שם במקרה עדין מגיעים דרך הרשות סגננטיים מהתקשרות קודמות – הפרוטוקול לא יתבלבל ביניהם ובין ההודעות מה-חסיס `sessions` הנוכחיים.

ריאיון מילאכ TCP

- מספר הבטים הנשלחים יעשה על פי המספר הסידורי. המספר עולה על פי מספר הבטים שנשלחו.
- את החיבור חשוב לסגור כיוון שצרי לשחרר את המשאים שהוקטו וכדי לדעת שכל ההודעות שנשלחו התקבלו.
- הסגירה יכולה להתחיל מהשרת או מה-client, ע"י שליחת הודעה FIN, אליה ישלח ACK. אם לא קיבל ה-ACK על ה-FIN אז ה-FIN נשלח שוב. יש המתנה לאורק זמן מסוים כדי לוודא שלא מקבל עוד FIN, ככלומר, שייהי בטוח שהתקבל ה-ACK על ה-FIN. המתנה נעשית ב-client ולא בשרת כיון שהוא מול הרבה לקוחות.

TCP – קום ו-ג

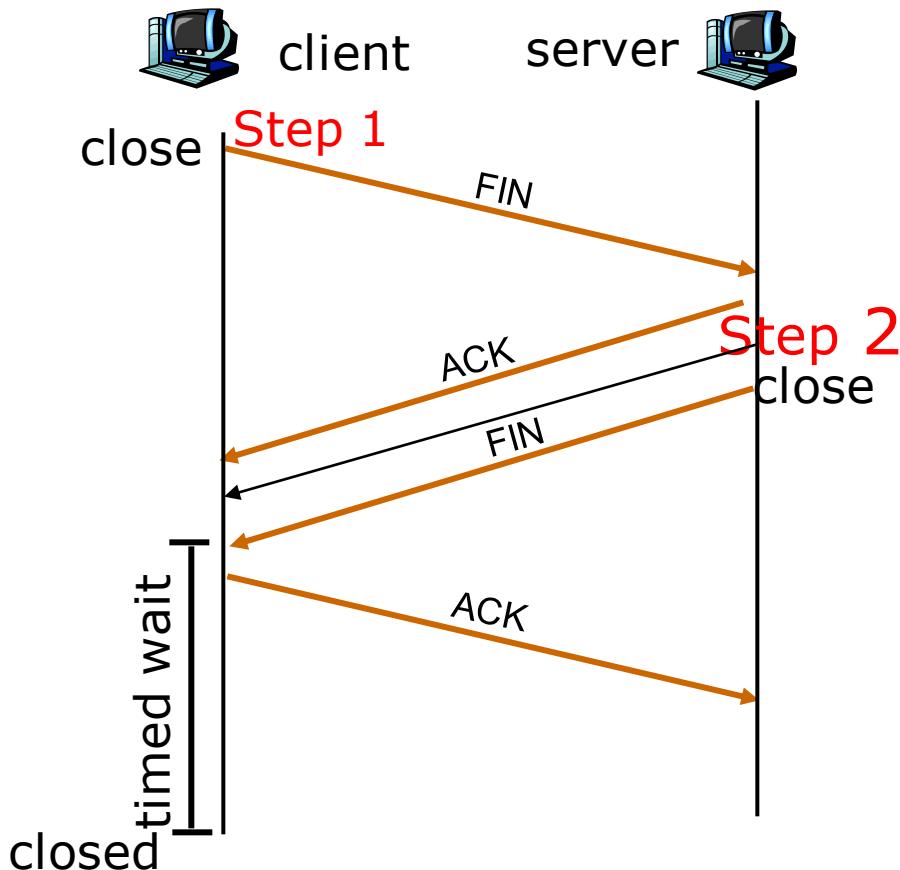


שליחת SYN: ה-Host שולח מספר סידורי אקראי.

שליחת ACK: ה-ACK הוא עם המספר הסידורי שהתקבל מה-Host. יכול גם להכיל כבר מידע.

98

ריכוז חיבור TCP



סיום הרשתות:

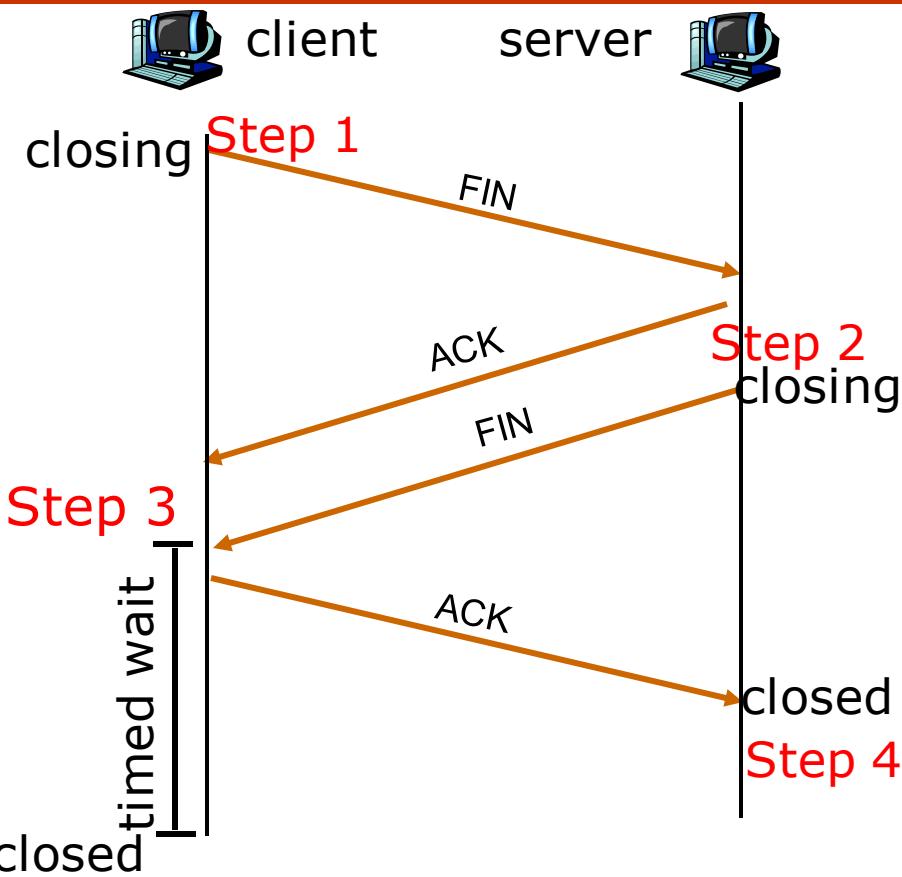
:client closes socket

`clientSocket.close();`

צעד 1: המחשב ל风俗 שליח
"TCP FIN control" מקטע לשרת.

צעד 2: השרת מקבל FIN,
מגיב עם ACK, סגור
תקשורת ושליח FIN.

רִיכוֹר מִינָה כ-TCP



צעד 3: הלקוח מקבל FIN ומחזיר ACK.

- הזנת "timed wait" תענה עם ACK לקלט .FINs

צעד 4: השרת מקבל ACK, תקשורת נסגרת.

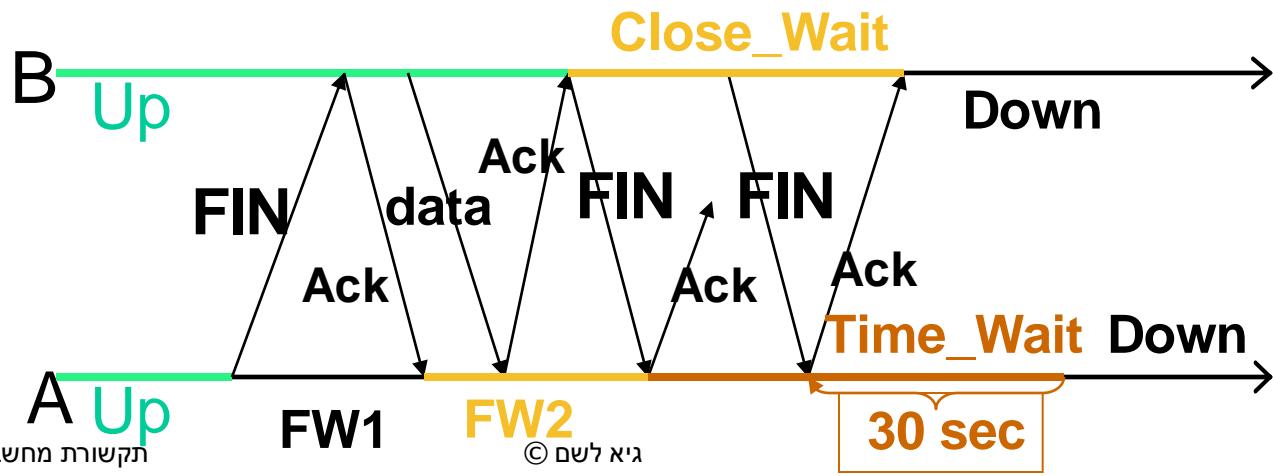
סיכום: כדי לסיים קשר TCP יש צורך בשילוח של 4 סגמנטים. הסיבה לכך נועוצה בתכונה של TCP הנקראת "half-close". מאחר וקשר TCP הוא full-duplex, כלומר מידע יכול לזרום בשני הכוונים ללא תלות הצד השני, כל צד צריך לסייע את הקשר שלו ללא תלות הצד השני. הכלל אומר שככל צד יוכל לשולח FIN ברגע שהוא סיים להעביר את המידע. כאשר TCP מקבל FIN, הוא חוסם מזרים ¹⁰⁰ קשות מהחשבון אליו הגיעו לאפליקציה על כך שהצד השני סיים את הקשר בכוונו שלו. כמו ביצירת קשר אומרים שהשלחה את ה-FIN **"active close"**. הצד השני שמקבל את ה-FIN **"passive close"**.

ריגול מיאור TCP

- כדי לסגור את החיבור, אנו חייבים לוודא שני הצדדים נסגרו (כל ההודעות שנשלחו).
- המחשב המגיב מקבל את הודעה שהמחשב היוזם סגר את התקשרות בסדר, אבל היוזם יכול רק להמתין לראות אם הוא לא קיבל שוב FIN.
- יש לנו אפשרות קלה שהיוזם יסגור עם שגיאה ואילו המגיב יסגור היבט וכל ההודעות משני הצדדים יועברו בצדקה נכונה

ריכוז מיגור TCP

- מצבים הלקוח:
 - אין יותר שליחות, מחייב ACK על ה-FIN ששלח. (FW1)
 - כנ"ל, עדין מחייב, ל-FIN. (FW2)
 - מחייב ACK לשולח FIN אם אנו קיבל FIN חדש. (Time_Wait)
- היוזם: שולח FIN, מחייב ACK ל-FIN וLN-FIN.
- המגיב: ACK ל-FIN, ההודעה الأخيرة, FIN.
- מתבצע כאשר מקבלים ACK ל-FIN.



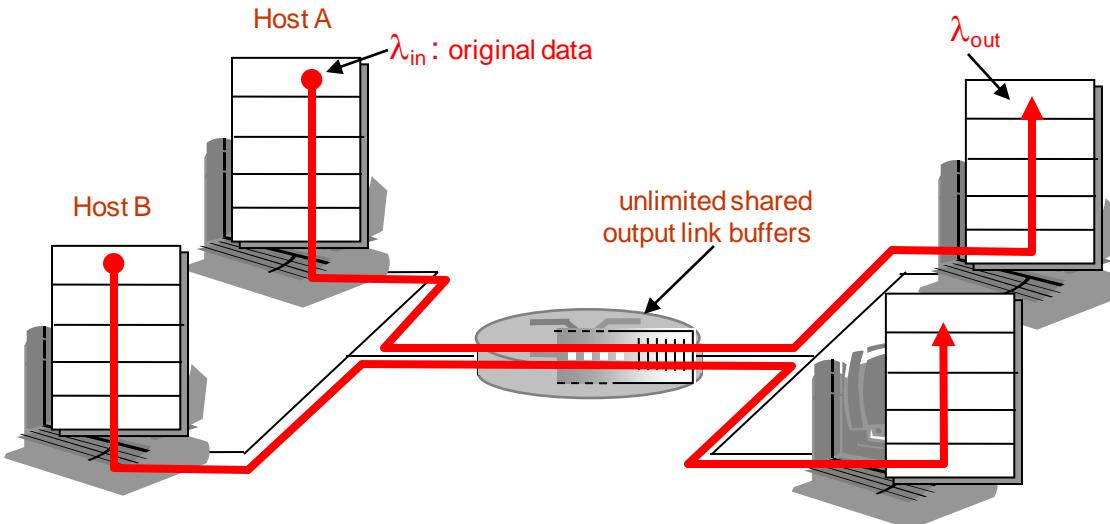
עכמת כוכב

- עקרונות של בקרת עומסים.
- בקרת עומסים TCP.
- שירותי שכבות הhonebhla.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעבורת מידע אמיןה ברשות.
- תעבורת מונחת קשר: TCP
 - מבנה המקטע.
 - תעבורת מידע אמיןה.
 - בקרת זרימה.
 - ניהול הקשר

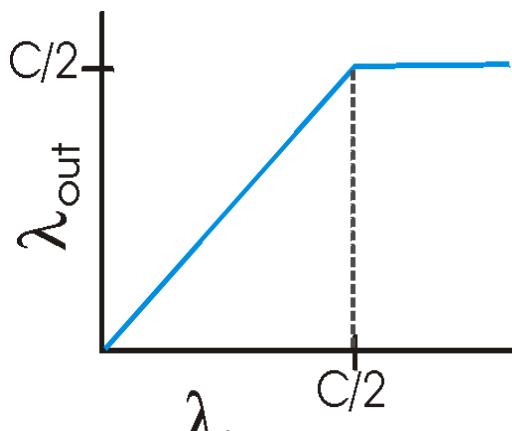
אפקטור אפקט האז

- בעומס יתר תורי המתנה בנתבים נשים ארוכים
וכך נוצרת השהיות ארוכות.
- כמו כן, מנוגות הולכות לאיבוד.
- ההבדל בין **בקרת עומס לזרימה** הוא שבעומס מתמקד בחולצים של הנתבים ובזרימה בחולצים של מחשיبي הקצה.
- אין שליחת אישורים מהנתבים, תפקידם היחיד הוא העברת החבילות ליעד הבא.

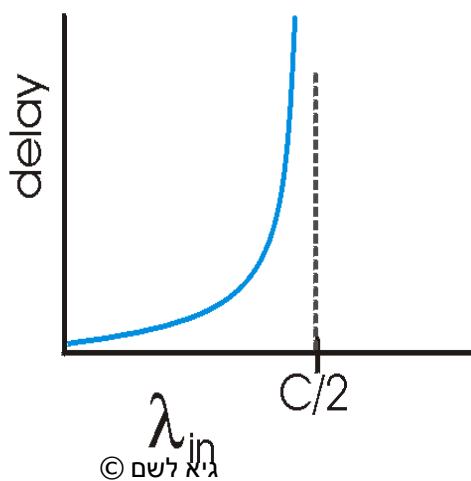
סיגומט ועפויוּם של צוֹאנָס – אקְכָה 1



- ◻ ישנים 2 שלוחים ו- 2 יעדים וביניהם נתב עם חוץ אינסופי.
- ◻ אין שליחה חוזרת.
- ◻ לא – קצב שידור ממוצע של בתים לשניה.
- ◻ עיכוב גדול כאשר יש עומס.
- ◻ התפקוקה מקסימלית



תקשורות מחשבים ואלגוריתמים מבזרים
(הורף 2010)



סיכום ועקרונות של צוואם – אקלחת 1 (הגסכה)

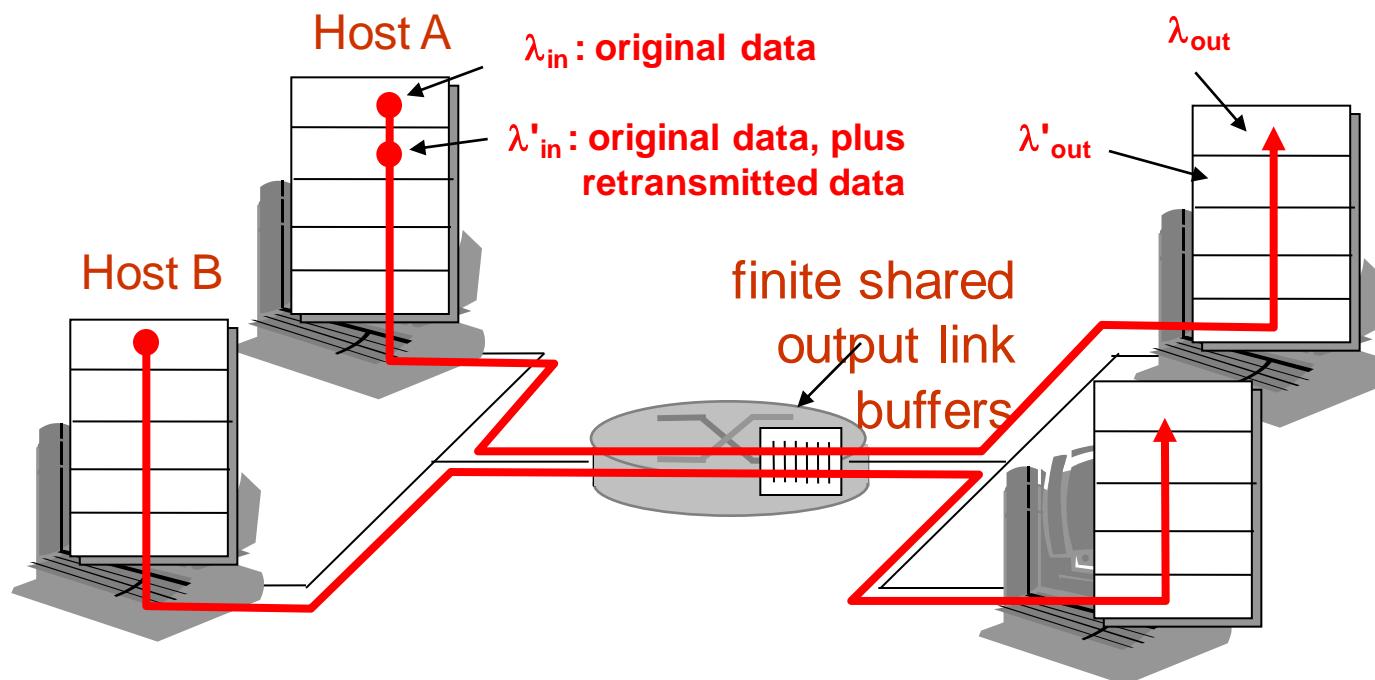
- ישנים 2 שלוחים ו-2 יעדים וביניהם נתב.
- בנתב יש חוץ אינסופי. גודל הערוץ (רוחב פס) הוא C.
- לנtab נדרש זמן כדי לקבוע לאן לשלוח את המנה שהוא מקבל.
- אם נתב מקבל שני השולחים בו-זמןית אז קצב השילחה שלו לכל יעד יהיה קטן מ- $\frac{1}{2}C$. זאת כיון שכל פעם ב ממוצע הוא שלוח מנה שהתקבלה משלוח אחר + הזמן שלוקח לו להחליט לאן לשלוח את המנה. לכן כאשר קצב קבלת המנות אצל הנתב הוא $\frac{1}{2}C$ או יותר החוצצים ימלאו עד אינסוף (קצב הקבלה גדול מקצב השילחה) וכך גם ההשניה תתרחן ותגדל.
- לא – קצב שידור ממוצע של בתים לשניה.
- הסבר לגרף שמאל: קצב השילחה מנקודת הקצה i_{out} הולך וגדל לעומת קצב הקבלה אצל נקודת היעד i_{out} שלא עולה על $\frac{1}{2}C$ בגלל קצב השילחה מהנתב.
- הסבר לגרף ימין: ככל שקצב השילחה מנקודת הקצה i_{in} מתקרב ל- $\frac{1}{2}C$ כך ההשניה נעשית גדולה יותר (עד אינסוף).

סיגומט ועפויות של צואנס - אקציה 2

◻ ישנים 2 שולחים ו-2 יעדים

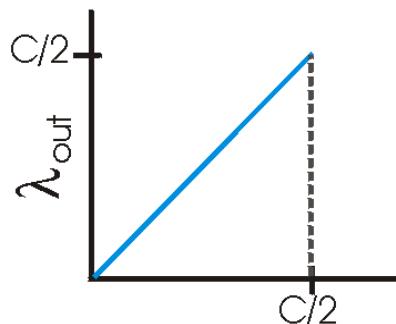
◻ נתב אחד עם מספר חוצצים סופי.

◻ שליחה חוזרת של חבילות אבודות.

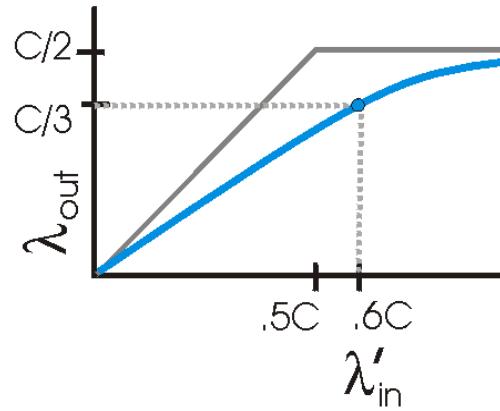


סיכום ועקבות של צוואם – אקכה 2 (הסכם)

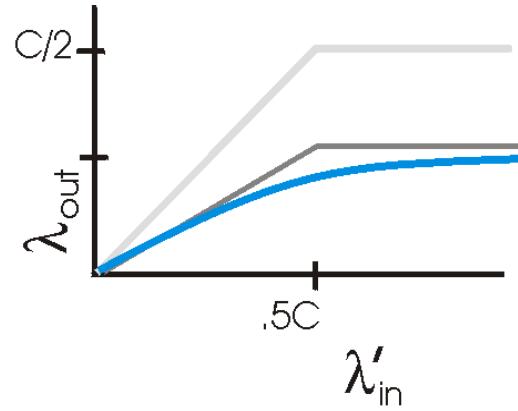
- תמיד $\lambda_{in} = \lambda_{out}$ (good - השידור אמיתי).
- שליחה חוזרת מושלמת רק באובדן $\lambda_{in} > \lambda_{out}$.
- שליחה חוזרת של העיכוב (לא אובדן) החבילה מבצעת λ_{in}' אחר יותר (מהמקרה המושלם) עברו אותו λ_{out} .



(a)



(b)



(c)

λ_{in} – קצב השידור כולל השידור החוזר, גדול מ- λ_{in} .

"מחיר" של העומס:

- יותר עבודה (חזרת) עברו goodput נתון.

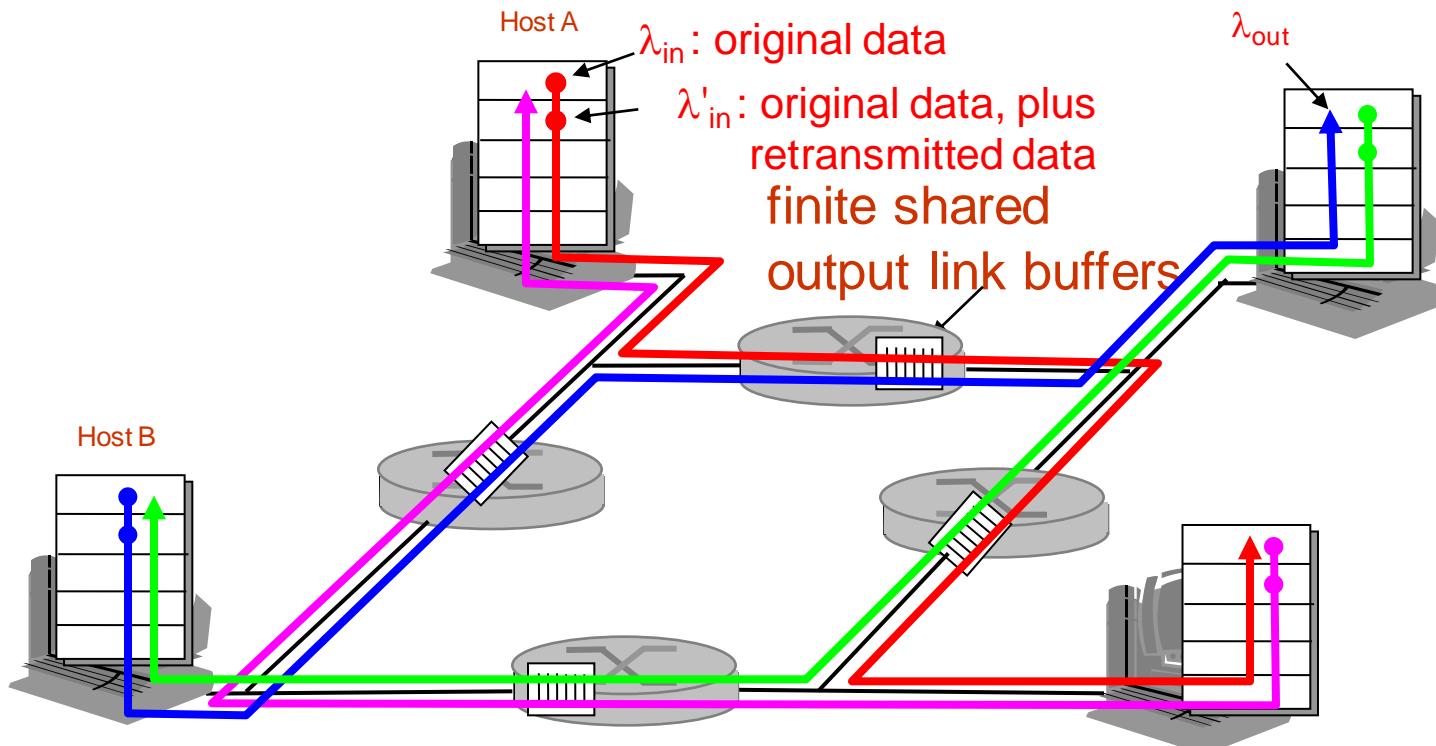
סיכום ועקבות של אקסה 2 (הגסכה)

- כמו בפרק 1 אך הפעם יש מספר חוצצים סופי. מספר חוצצים סופי גורם לכך ש- $t_{out} = t_{in}$ כלומר, השידור אמין, כל חבילת ח"ב תגיע. יש גם שידור חוזר במקרה של גלישה בחוץ או רعش או שיתבצע $timeout$.
- 'א' - קצב השידור כולל השידור חוזר, גדול מ- t_{in} .
- 'א' - קצב קבלת ההודעות בرمת ה-TCP כולל הודעות חוזרות, ההודעות שmagiyot משובשות לא עולות לרמה האחורנית لكن הוא גדול מ- t_{out} .
- הסבר לגרף שמאל: כאשר אין שידור חוזר, ככלומר אין עומס והטור של הנטב ריק או לא מלא אז $t_{in} = t_{out}$. מצב זה נוצר כאשר קצב השליחה קטן מ- $\frac{1}{2}/C$ וכל השולחים מקפידים לבצע את חוקי ה-TCP.
- הסבר לגרפים האמצעי והימני: כאשר יש עומס ונוצר מצב של איבוד מנת הדורש שידור חוזר אז קצב קבלת המנות t_{out} קטן כיוון שחלק מהמנות שהוא אמר לקלט ונשלחו, הוא לא קיבל.
- שימוש ב-NACK GoBack אשר דורש לשלוח את כל המנות שהתקבלו אחרי המנה שלא קיבלה ACK מחדש יחמיר את העומס.
- ההשניה תהיה ארוכה יותר כי יהיו יותר הודעות להעביר (המקורות + המשובשות).
תקשורת מחשבים ואלגוריתמים מבזרים
גיא לשם ©
(חורף 2010)

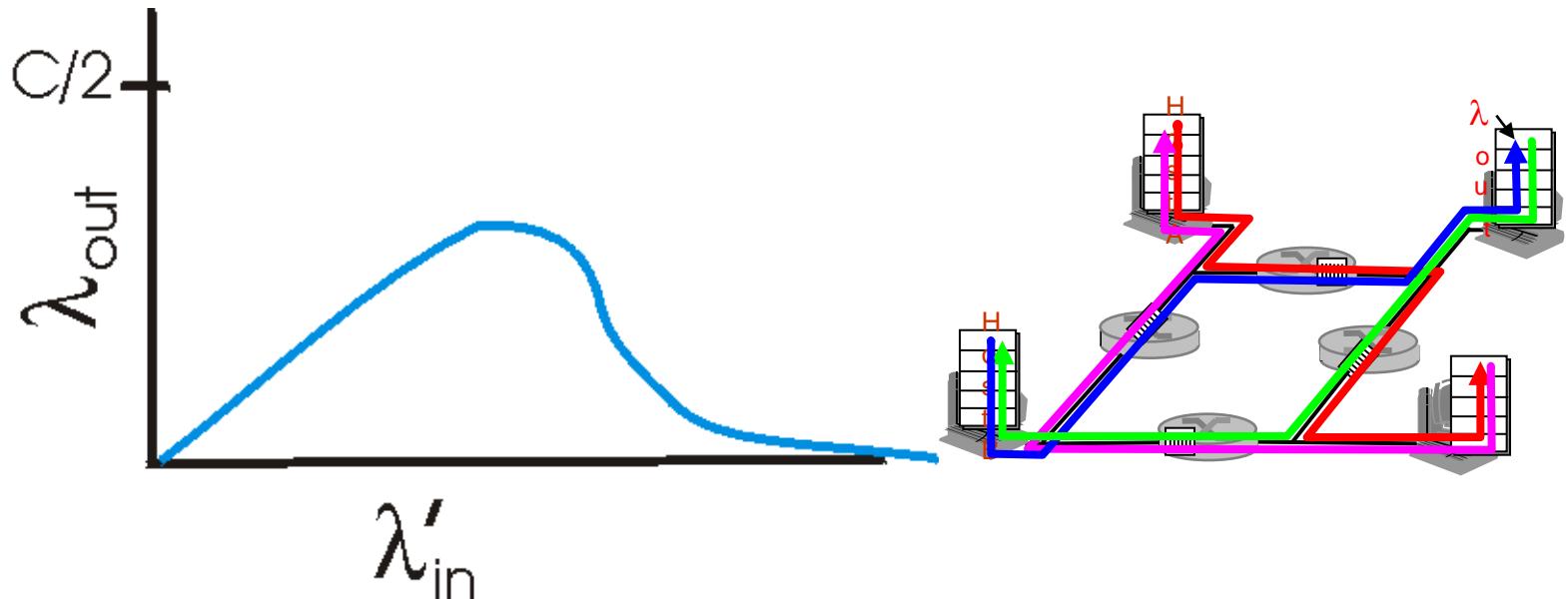
סינון ותפזרות fe צוואם - אקציה 3

שאלה: מה קורה כאשר λ_{in} גדלים?

- ארבע שלוחים.
- נתיב רב-HOP Multi-hop paths
- פוק זמן / שליחה מחדש



סיגומט ועפויום של ציאנס - אקציה 3



מחיר נוסף של העומס:

- כאשר חבילה אובדת, כל יכולת השליחה של זרם נתונים המשמש לחבילה זו יהיה בזבוז !

סיגומ ועפויום של צוואם – אקכת 3 (הגסכה)

- הוספה עוד ערכאים ונتابים ועוד נתיבי הגעה לכל יעד.
- הסבר לגרף: אם שולח אחד יתחיל לשדר מ Każב מהיר יותר הוא יגרום בסופו של דבר למצב של deadlock. זאת כיוון שע"י שליחה מוגברת הוא יגרום לעומס בנتابים, מה שיביא לאיבוד מננות ושידור חוזר שלהם.
- השידור החזר של המנות יגרום לעוד עליה בעומס וחוצצי הנتابים יתמלאו, כל זה יגרום לקריסה של הרשת.

սכמת התקשורת

- עקרונות של בקרת עומסים.
- **בקרת עומסים TCP.**
- שרותי שכבת ההובלה.
- ריבוב/פילוג.
- תעבורת חסרת קשר: UDP.
- עקרונות תעבורת מידע אמיןہ ברשות.
- **תעבורת מונחת קשר: TCP**
 - מבנה המקטע.
 - תעבורת מידע אמיןہ.
 - בקורת זרימה.
 - ניהול הקשר

קכמת TCP-אום

- **בקרת עומס מוקצת לקשה** – מחשבי הקצה מבקרים את העומס ע"י בדיקה של כמות ההודעות המתקבלות משידור חוזר ורמת ההשניה. פרוטוקולים חדשים חווים להיות "ידיוטיים" ל-TCP, כלומר לא יכולים לפגוע בתפקוד ה-TCP.
- **בקרת עומס בעזרת הרשות** – הנטים מספקים מידע באורך בית יחיד – יש או אין עומס. כמו כן, מסמנים את קצב שידור השילחה האפשרי.
- במידה וישנם נתבים /או חיבורים איטיים ברשות בין השולח למקבל עלולות להיווצר בעיות כאשר חבילות יאבדו או התעכבו כתוצאה מעומס זמן. בכך לוווסת את קצב כתיבת המידע לרשות ע"י השולח כך שכל המידע הגיע למקבל, משתמשים בין השאר, באלגוריתם "Slow start".
Congestion Window – אלגוריתם "Slow start" מוסיף עוד חלון ל-TCP של השולח, **CongWin** או **cwnd**.
- השולח מגביל את הביטים לשילחה לא רק ע"י **בקרת הזרימה** (למנוע מהמחשב מקבל גליישת מידע בחוץ) אלא גם ע"י חלון העומס – **CongWin**. **ה-CongWin** הוא מספר הביטים שנשלחו שעדין לא קיבלו עליהם **ACK**. **ה-CongWin** נותן אינדיקציה טובה על העומס ברשות ובעזרתו ניתן לדעת את הקצב הרצוי לשילוח המנות.
- ערכו של **ה-CongWin** משתנה בהתאם לזהות אובדן חבילות ברשות. ניתן לזהות אובדן חבילות ע"י המאפיינות:
 1. **Timeout**
 2. **שלוש הודעות ACK זהות רצופות (3 duplicated ACK)**

קכמת TCP-ג-וֹס

- CongWin נתן אינדיקציה טובה על העומס ברשת ובעזרתו ניתן לדעת את הקצב הרצוי לשילוח המנות.
- ערכו של CongWin משתנה בהתאם לזיהוי אובדן חבילות ברשת. ניתן להזעות אובדן חבילות ע"י המאורעות:
 - ארוע אובדן: **timeout**, או שלוש הודעות **ACK** זהות רצופות.
 - השולח מקטין את הקצב (CongWin) אחר אובדן.
- שני אופנים לבקרת עומס:
 - **Slow Start** (להתחל לאט) – עם בניה מעריכית.
 - **Congestion Avoidance** – להימנע מגודש.

- בקרת העומס היא מקצתה לקצתה.
- השולח מגביל את הביטאים לשילחה לא רק ע"י בקרת הזרימה – אלה גם ע"י חלון העומס – CongWin.
- **LastByteSent - LastByteAcked**
 $\leq \min(\text{CongWin}, \text{RcvWin})$
- CongWin הוא מספר הבטים שנשלחו שעדין לא קיבלו עליהם ACK.
- מווין RcvWin הוא מקום שנותר בפנוי בחוץ לארך זמן החיבור נעריר:

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

TCP אמצעי ckf

ה-pwdc משמש כבקרה זרימה של השולח, בעוד ה-*Advertised window* (גודל החלון המרבי אותו התחנה מקבלת מוכנה לעבוד) משמש כבקרה זרימה של מקבל.

השלוח מתחילה ע"י שידור של סגמנט אחד והמתנה ל-*ack*. כאשר ה-*ack* מתקיים גדל ה-pwdc לגודל של שני סגמנטים ואז השולח משדר שני סגמנטים.

כאשר מקבלים *ack* על הסגמנטים גדל ה-pwdc להיות בגודל 4 סגמנטים וכך הלאה. **התנחות זאת מתארת גידול אקספוננציאלי של גודל חלון ה-pwdc.**



במידה וישן נתבים ו/או חיבורים איטיים בראשת בין השולח למקבל עלולות להיווצר בעיות כאשר חבילות יאבדו או התעכבות כתוצאה מעומס זמן.



בכדי לוויסת את קצב כתיבת המידע לרשות ע"י השולח כך שככל המידע יגיע למקבל משתמש, בין השאר באלגוריתם "Slow start".



אלגוריתם "Slow start" מוסיף עוד חלון ל-TCP של השולח, *Congestion Window*, הנקרא גם *pwdc*. כאשר נוצר קשר TCP חדש עם מחשב בראשת אחרת, ה-*pwdc* מאוטחל לגודל של סגמנט אחד.

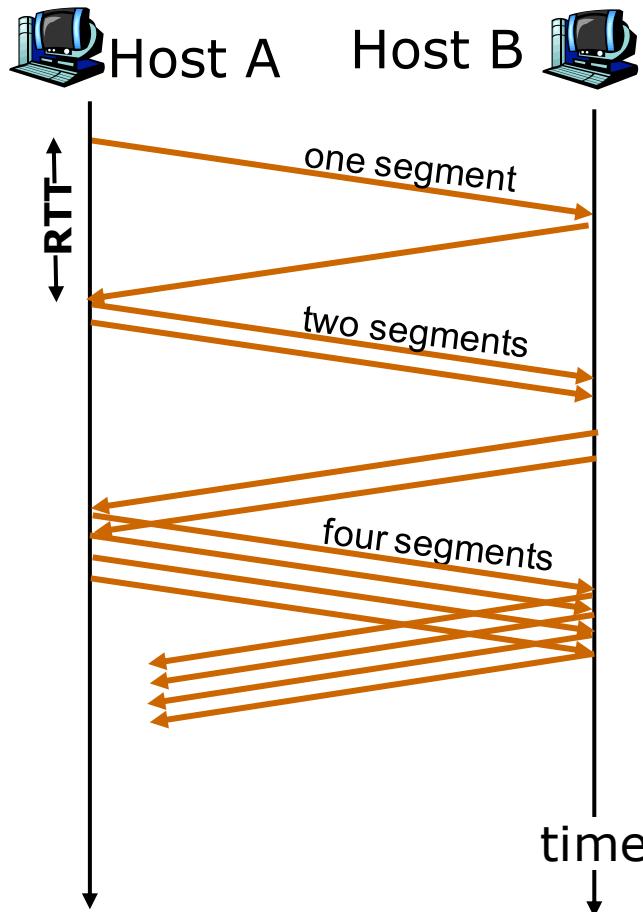


בכל פעם שמתקיים *ack*, ה-*pwdc* גדל בסגמנט אחד.



השלוח יכול לשדר עד שהוא מגיע למינימום בין ב-*pwdc* ל-*window*.

TCP Slow Start



- **קצב בניה מעריצי**
- הקצב הראשון איטי ($\text{CongWin} = 1\text{MSS}$) (CongWin = 1MSS).
- אבל הוא עולה במהירות. MSS=Maximal Segment
- עד ארוע איבוד ראשון, הקצב עולה ע"י: ■ הכפלת CongWin לכל RTT.
- ע"י הגדלה של CongWin עברו כל קבלת ACK.
- במקרה של אובדן: נשנה את הקצב על מנת למנוע עומסים (ע"י פרוטוקול - AIMD - Additive Increase, Multiplicative Decrease).

סיכום: בתחילת החיבור ערך CongWin הוא מינימלי. כל עוד אין אובדן מנות מכפילים את גודל CongWin פי 2. כאשר מתחילה איבוד מנות מקטניות את CongWin פי 2 אנו מתייחסים לאותם אובדים אשר מופיעים בפער תזמון בין אוביידיון ורply. מוגדרים מבודדים (Additive Increase, Multiplicative Decrease) AIMD (וחיף 2010).

תעלוקה 3dupACKs – \neq TimeOut – \neq

רעיון

- הרעיון מאחורי האלגוריתם הוא הבדלה בין שני המאורעות שנגרמת עקב עומס בראשת.
- האלגוריתם מבידיל בין חומרתן של שתי המאורעות. שלוש ACK זרים רצופים מעודים על עומס גדול אך חמורים פחות מ- $timeout$. לכן, כאשר מקבלים שלוש ACK מפחיתים לחצי את ה-CongWin וכאשר מקבלים $timeout$ מפחיתים למינימום האפשרי.
- אחרי שתרחש $timeout$ וגודל ה-CongWin קטן למינימום, ה-CongWin מתחילה לגדול אקספוננציאלית עד שהוא מגיע לחצי גודל ה-CongWin שבו התרחש ה- $timeout$ (נקודת הסוף) ואז הוא גדל לינארית.

- **אחרי 3 ACK זרים:**
 - CongWin קטן בחצי (multiplicative decrease)
 - החלו لكن גדול באופן לינארי (additive increase)
- **אחרי Timeout:**
 - התחל מחדש חיבור.
 - CongWin נקבע ל-1 MSS.
 - גידול מעריכי עד לערך סף, ואח"כ גידול לינארי.

ואפ"א התחלה איטית

- התחלה איטית מתחילה עם חלון בגודל מקטע 1
 - אבל במהירות מסוימת: גדל ב-מקטע 1 לכל ACK (כלומר פי 2).
 - מהחvíלה השניה, בהתחלה האיטית: ניתן לשלוח 2 מקטעים על ACK (זוג).
 - אחרי קבלת ה-ACK עבור W מקטעים: $W+W = \text{Cong. Window}$
- לאחר אובדן האינדיקציה (3 dup acks או timeout)
 - $\lfloor W/2 \rfloor = W_{th}, 1=W$, נשלח מחדש חvíלה ללא ACK ראשונה.
 - מאחר $1=W$: מchnerה ל-ACK (על חvíלה אחת או יותר) המשיר ממשם.
- כאשר, $|window| \geq W_{th}$ (עריך הסוף של התחלה האיטית):
 - משנה לモפע 'מניעת עומס'.

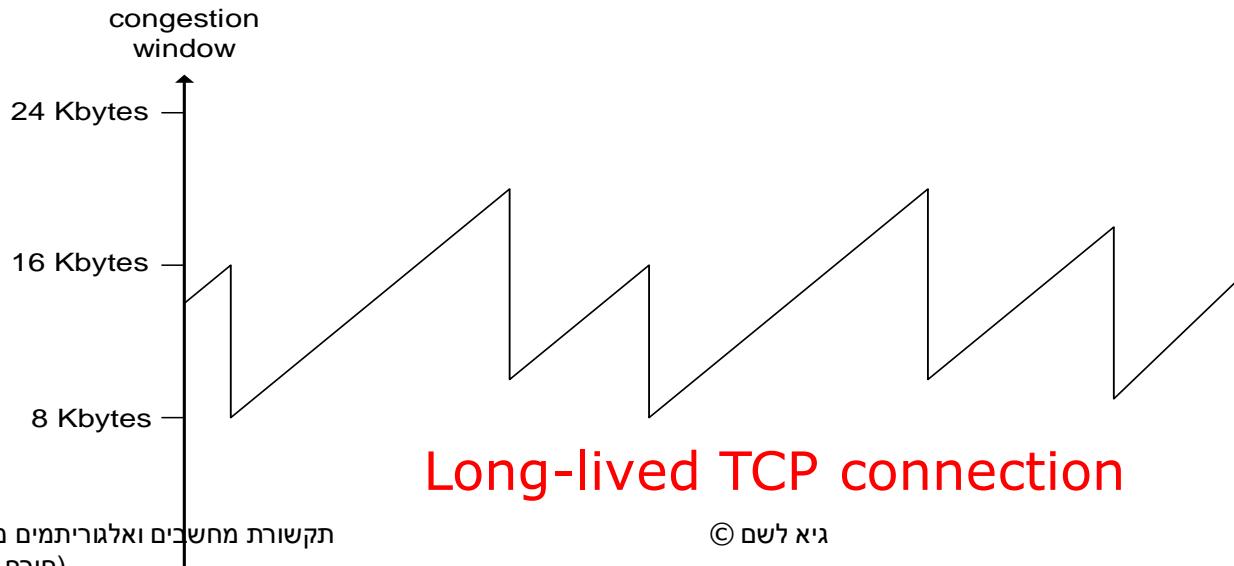
TCP אוניברסיטט

(AIMD: Additive Increase, Multiplicative Decrease)

ה- AIMD מגדיר כיצד הпрוטוקול ישנה את גודל החלון של השילוח בתנאים של עומס וחוסר עומס. השיטה מגדירה שכאשר אין עומס, החלון יגדל בצורה ליניארית, וכאשר יש עומס (כלומר, איבוד חבילות) החלון יקטן ע"י חלוקה ב- 2. משתמשים בשיטה זו בпротокול TCP במצב של "congestion avoidance" כדי לווסת את גודל החלון, ומכךן את קצב השילוח לפי מצב הרשות. בנוסף, שיטה זו מבטיחה שלאורך זמן, כל המשדרים יקבלו קצב שידור שווה – כלומר מבטיחה הוגנות.

RTT CongWin במקסימום סגמנט אחד בכל (מגבירים על פי מה שנקבע ב-handshake).
Additive Increase: כאשר אין איבוד מנות מגדילים את CongWin

Multiplicative Decrease: לעומת זאת, אם יש איבוד מנות או מקטינים את CongWin בחצי אחרי 3 ACK משוכפלים. ככלומר, העליה בקצב היא מתונה מאוד והירידה חדה מאוד



סיכום קכמת צ'ואס

- כאשר CongWin נמצא מתחת לנקודת הסוף השולח המצב start slow – גידול אקספוננציאלי.
- כאשר CongWin מעל נקודת הסוף השולח נמצא במצב של המנוות מעומו (Congestion Avoidance) – גידול לינארי.
- כאשר מקבלים שלושה ACK, נקודת הסוף נקבעת ל-CongWin וה-CongWin/2.
- כאשר יש timeout נקודת הסוף נקבעת ל-CongWin/2 ו-CongWin נקבע למינימום.

אַקְגִּיטָמָן

Congestion Avoidance

האלגוריתם המשותף (Congestion Avoidance) נקרא גם **Tahoe**:

אתחול קשור קובע את $cwnd$ לגודל סגמנט אחד, ו- $ssthresh$ לגודל bytes 65535.

פונקציית הפלט של TCP אף פעם לא שולחת יותר מהמינימום בין $cwnd$ והחלון שפורסם ע"י המქבל (Advertised Window). Congestion Avoidance זהו מנגנון בקרת זרימה של השולח. לעומת פורסום החלון זהו מנגנון בקרת זרימה של המქבל. ראשון מבוסס על הרכבת השולח את ה- $cwnd$ ברשות בעוד השני (Advertised Window) קשור לגודל החוצצים של המქבל.

כאשר חסוך Congestion קורה (ע"י פקיעת $timeout$) או קיבלת מספר ack-ים זרים, חצי מגודל החלון הנוכחי (המינימום בין $cwnd$ והחלון שפורסם ע"י המქבל (Advertised Window), אך לפחות 2 סגמנטים) נשמר ב- $ssthresh$. בנוסף, אם ה- $cwnd$ נגמר כתוצאה מפקיעת $timeout$, $cwnd$ נקבע להיות סגמנט אחד (כלומר Slow Start).

כאשר מקבל ack על מידע חדש, נגדיל את $cwnd$. הדרך בה $cwnd$ גדל תלוי באם אנחנו מבצעים Slow Congestion Avoidance Start או Slow Start threshold size ($ssthresh$).
תודה למחשבים ואלגוריתמים מבזרים
גיא לשם © (חורף 2010)

- Slow Start משמש לאותה תקופה זרימת המידע בקשר. בנקודה כלשהו נגיע לקלול המקסימלי של Router, וחבילות "ירקו". Congestion Avoidance שabaydo. השיטה מתוארת ב-1988 Jacobson.

הנחה של האלגוריתם היא שאיבוד חבילות הנגרם מנזק הוא מאד קטן (הרבה פחות מ-1%), וכך איבוד חבילה מאותת על Congestion בשרות בין המקור ליעד. ישנן שתי אינדיקציות לאיבוד חבילה: פקיעת $timeout$, וקבלת מספר ack-ים זרים (Duplicate Ack).

- Slow start Congestion Avoidance אלגוריתמים בלתי תלויים בעלי מטרות שונות. אבל, כאשר קורה חסוך Congestion אנחנו רוצים להאט את קצב שידור החבילות לרשות, אז עברו ל- Slow Start כדי לחזור קצב השידור.

Slow Start Congestion Avoidance צרכי שני משתנים לכל קשר:

- **1. חלון-h- $cwnd$ Congestion Avoidance.**
- **2. Slow Start threshold size ($ssthresh$)**

fast & slow

Fast Retransmit and Fast Recovery

האלגוריתמים בד"כ ממומשים ביחד בפניהם **Fast Retransmit** ו- **Fast Recovery** גם :

כאשר Duplicate Ack שלishi מתקבל, נקבע ssthresh למספרם בין החזי ה-pwdws הנוכחי לבין גודל החולון המפורסום ע"י המקבל. שידור חוזר של הסגמנט החסר. Cwnd נקבע להיות $3 * \text{ssthresh}$ החדש (3 כפול גודל סגמנט).

בכל פעם שעוד Duplicate Ack מתקבל, נגדיל את pwnd ב-1 ונשדר חビלה באמצעות הערך החדש של pwnd מאפשר זאת.

כאשר מתקבל Ack שמאמר מידע חדש, נקבע את Cwnd לערך של ssthresh. זהה ציריך להיות ה-Ack-h-Ack. בנוסוף ה-Ack-h-Ack זהה אמרו לאשר את כל בזמן RTT לאחר השידור החזר. הסגמנטים שנשלחו בין שליחת הסגמנט שאבד ועד לקבלת ה-Ack-h-Ack השלישי. זהו Congestion Avoidance. מכיוון שאנו מאטיטים לקבל שידור של חזי ממה שהוא כאמור הסגמנט אבד.

בהמשך לאלגוריתם המעודכן יצא :

New Reno הוא שינוי קטן ב-Reno. הוא שונה מ-Reno בכך שהוא לא מפסיק את Fast Recovery עד אשר כל המידע שהוא שלח לפני שהוא נכנס למצב Fast Recovery, מאשר.

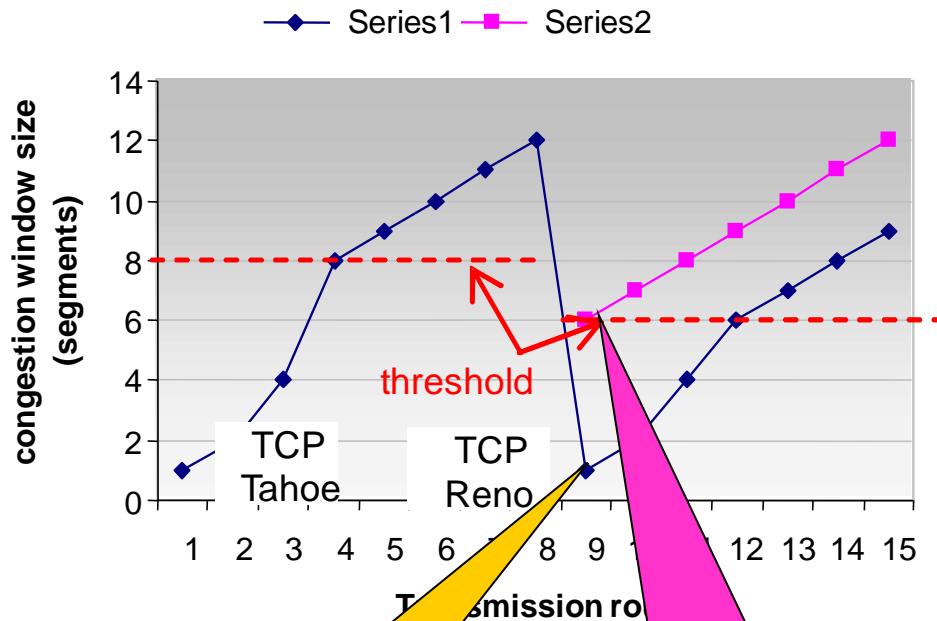
בכך הוא מתגבר על הבעיה ש-Reno מקטין את ה-pwdws מספר פעמים. בשלב ה-Reno Fast Retransmit הוא זהה ל-Reno. ה-Reno-Fast Recovery מתחילה כמו Reno, אבל כאשר Ack חדש מגיע אליו ישנים שתי אפשרויות:

אם ה-Ack מאשר את כל החבילות ששודרו לפני שנכנסנו למצב Fast Recovery אז הוא יוציא מצב Fast Recovery וקובע את ה-pwdws כ-ssthresh להיות ssthresh וועבור למצב Congestion Avoidance כמו ב-Tahoe.

אם ה-Ack מאשר חלק מהחבילות ששודרו לפני שנכנסנו למצב Fast Recovery אז אנחנו מניחים שהסגמנט הבא אחריו האמור אבד גם כן וכן גיא לשות[©] אנחנו מבצעים שידור חוזר שלו. וקובע את מספר ה-AcksFastRetransmit (הקלטת ACKS) לשיקבלם ל-0.

- שינויים באלגוריתמי Congestion Avoidance הוצעו ב-1990 ע"י Jacobson. TCP צריך לשדר מיד (Duplicate Ack) כאשר מגיע סגמנט לא לפוי הסדר.
 - אטור שה-Ack ייעוק.
- המטרה של ה-Ack היא להודיע לכך שהולח שהגיע סגמנט לא על-פי הסדר, וגם להודיע לאיזה Sequence number (כלומר איזה סגמנט) הצד מקבל מצפה.
- מכיוון שלא יודעים אם ה-Ack הגיע או בגלל הגעת חבילות בסדר לא נכון, מחכים לקבל מספר קטן של Duplicate Acks.
- ההנחה היא שאם הבעה היא רק בהגעת החבילות לא Duplicate Acks הנכון אז יהיו רק אחד או שניים Acks לפני שתגיע החבילה החסורה. אם שלושה או יותר Duplicate Acks מגיעים ברכף זהו סימן שכנראה החבילה אבדה.
- במקרה זה מבצעים שידור חוזר של החבילה SCNREAHA אבדה, מבלי לחכות לפקיעת timeout. Fast Retransmit-algorithm זה נקרא Slow Congestion Avoidance Fast Recovery. Start.

Multiplicative Decrease Refinement



TCP Series 1
(Tahoe):
Restart after
3 Dup Acks

TCP Series 2
(Reno) – 3 Dup
Acks ‘only’
halves CongWin

- תחילה, גירסת של TCP Tahoe, גם 3 ACKs זהים וגם ל-timeout גורמים להתחלה מחדש.
- CongWin קבוע ל-1 MSS.
- חלון גדול מעריצי.
- לערך הסוף, אז גדול ליניארי.
- חדש יותר, גירסת TCP Reno, מגיב ל-3 ACKs זהים והוא יותר נוח: רק multiplicative decrease