

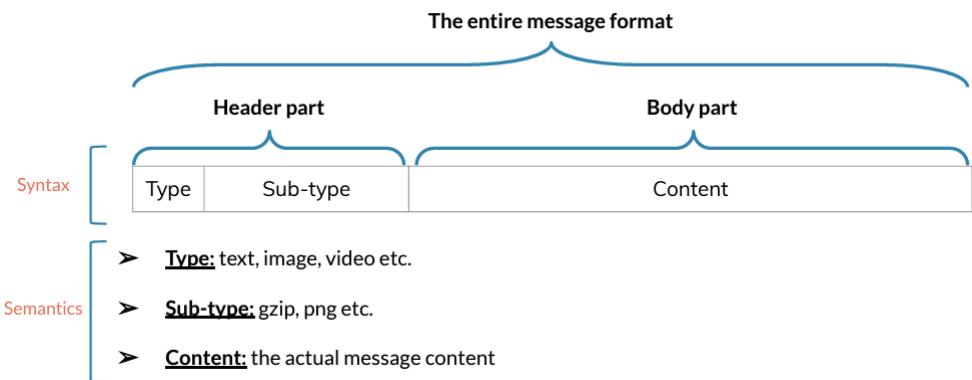
רשתות תקשורת – סיכום

הרצאה 1:

מודיבציה - אליס וbob רוצים לתקשר ולכון מחברים למחשב שלהם כבל דרך כרטיס הרשת. נניח שאנו נמצאים ב"עולם מושלם":

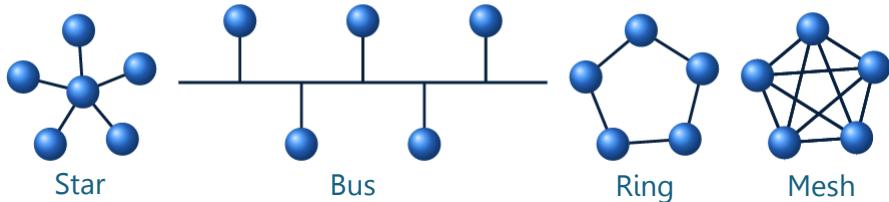
1. אליס וbob הם היחידים שמנסים לתקשר ביניהם.
2. הם קרובים יחסית האחד לשני.
3. המחשבים שלהם מרכיבים אך ווק תוכנה אחת שיכולה לעשות רק שני דברים: לשЛОח הודעות חדשות ולהציג על המסך הודעות שהתקבלו.

אולם מה יקרה אם הם רוצים לשЛОח מידע מורכב יותר מטקסט? עלינו להוסיף מידע על המידע – פרוטוקול.



לדוגמה אם אליס תשלח לבוב תמונה המבנה של ההודעה יורכב מ, type: image, sub-type: png, content: H4sIAAf...AAAAAh7Fmsdhg657AAAA

כעת נניח שנרצה לתמוך בתקשורת עם יותר מבן אדם אחד, נדרש להוסיף עוד כבליים בין כל אחד לכל אחד – לא ריאלי. וכך נוצר רשת – כל משתמש יתחבר אל הרשת והוא מודג לנתב את המסרים בין המחשבים המוחברים אליה.



hub: לחבר בין המחשבים ברשת – שלוחים אליו את ההודעה עם הנמען, והוא מעביר לכל המחשבים ברשת עם הודעה שמדובר במידע לנמען –ומי שהמידע לא מיועד אליו מתעלם.

שכבות TCP:

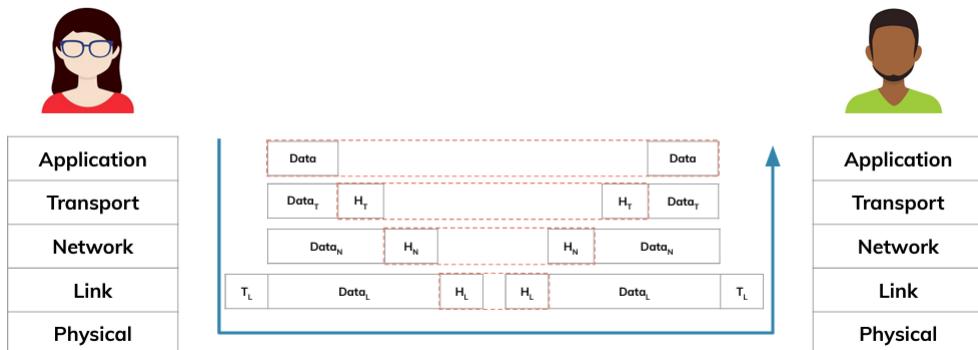
1. שכבת האפליקציה – מודג שתי אפליקציות שרצו לתקשר ביניהן י賓ו אחת את השניה, איך האפליקציה השולחת בונה את המידע כך שהאפליקציה מקבלת מבינה מה היא קיבלה.
2. שכבת התעבורה – אחד מתפקידיה הוא לדאוג שתהליך מהמחשב השולח יעביר את ההודעה לתהיליך אצל המქבל.
3. שכבת הרשת – צריכה לקבוע את המסלול בו יעבור המידע מהמחשב השולח למחשב המქבל – קצת כמו waze – צריכה לקבוע את מסלול הניות עד למחשב.
4. שכבת הערוץ – צריכה להחליט על התחנה הבאה שאליה יעבור המידע בדרך ליעדו.

5. השכבה הפיזית – כיצד עוברים הביטים של המידע פיזית.

בעת, אנחנו כבר לא צריכים להניח שאנו נמצאים בעולם מושלם, שכן:

1. שכבת הערוץ תוכל לטפל במקרה של כמה משתמשים שכן היא תדע לקבוע עבור כל משתמש את "התנה" הבאה שלו, כך שלא יהיו התנגשויות.
2. שכבת הרשת תוכל לקבוע את המסלול שיעבור המידע גם אם המשתמשים נמצאים רחוק מאוד אחד מהשני.
3. כל משתמש יוכל להריץ כמה תהליכי שהוא רוצה, שכן שכבת התעבורה תדאג להעביר את המידע מהתהליך הרלוונטי.
4. התוכנה של המשתמשים יכולה לעבוד ולשלוח מידע מורכב יותר מטקסט בלבד בזכות שכבת האפליקציה.

פרוטוקול TCP סימולציה -



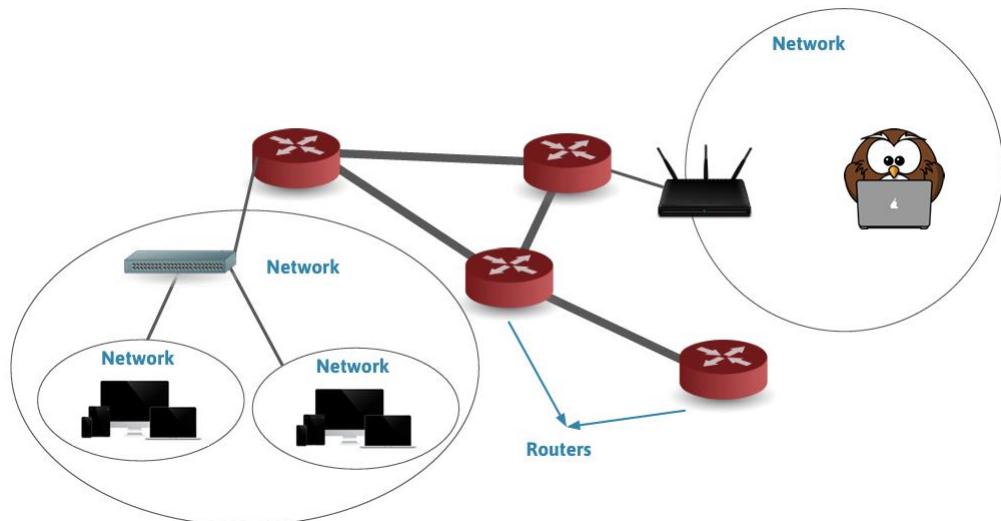
בעת שליחה – כל שכבה מלמעלה למיטה מוסיפה את המידע שלו והשכבה הבאה מקבלת את headers של השכבה מעליה כkoposa שחורה ומוסיפה עליו את המידע שלו. בעת קבלה – מתבצע תהליך הפירוש של כל headers רק בסדר הפוך.

Port – 16 ביטים, מהה של מערכת הפעלה לתחליק מסויים לטובת שימוש חיצוני. Port 0-1023 מספורים שמורות. בשכבה התעבורה header יכול את ports source וdestination, כלומר לאייה תחליק במחשבון הנמען ברכונו להעביר את המידע.

MAC address – לכל כרטיס רשת יש כתובת פיזית, שהמחשב קיבל בעת יצורו במפעל. הכתובת מורכבת מ-48 ביטים, בדר"כ כשהאנחנו נראה אותה היא תהיה כתובת במבנה 12 תווים hexadecimal המופרדים בנקודות, לדוגמה: 01:23:45:67:89:AB. ששת התווים הראשונים הריאנסים מתייחסים ליצן. בcut, כשנשלח הודעה מחשב אחד למחשב אחר, המידע יעבור דרך שכבת הערוץ לש switch עם source mac והזיהה destination mac וכן הswitch יידע לאיזה מחשב יעד עליו להעביר את המידע.

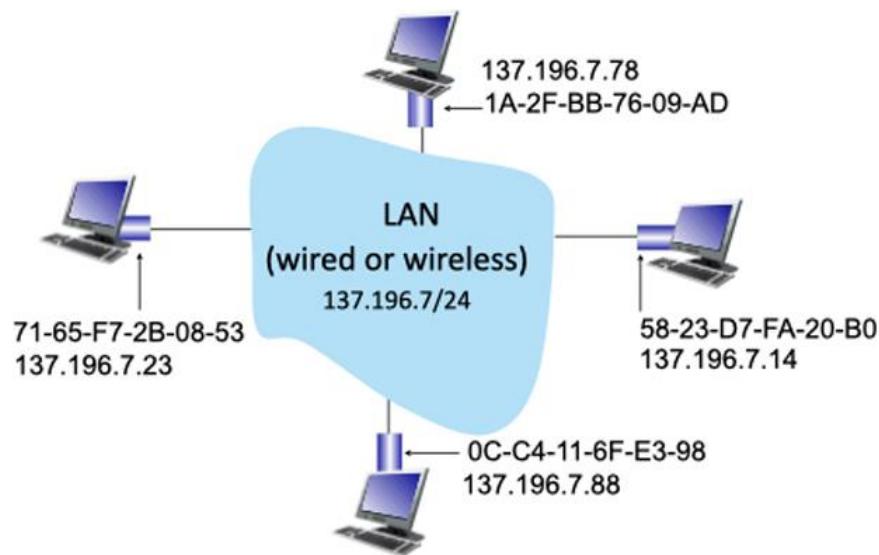
איך switch mac כתובות של המחשבים שמחברים אליו? switch מתחזק טבלה שמקילה את השדות הבאים – כתובת mac, הממשק שמננו המחשב מחובר אליו פיזית, TTL (time to live) – כמה זמן עליו לשמור את המידע של הרשומה הנ"ל בטבלה. בתחילת הטבלה ריקה, כאשר נשלחת הודעה ראשונה ממחשב A למחשב B – switch B שומר את המידע על A, ועביר את ההודעה לכל המחשבים ברשת. בcut, כשהמחשב B רצה להציג לא, הוא מעביר את ההודעה דרך switch – שמעדכן את הרשומה של B, והswitch מעביר ישרות לא את ההודעה כי הרשומה שלו שמורה אצל בטבלה.

כתובות IP (internet protocol) – מהה את הרשות אליו מחובר המחשב, וכן אפשרות לנtb מידע בין רשתות ברחבי העולם. בשכבה הרשת header יכול source IP וdestination IP.



המידע עובר דרך נתב הרשת שלי, לנטים נוספים עד הגיעו לנתב הרשת הרלוונטי, והוא כבר מעביר את המידע למחשב הגרפי.

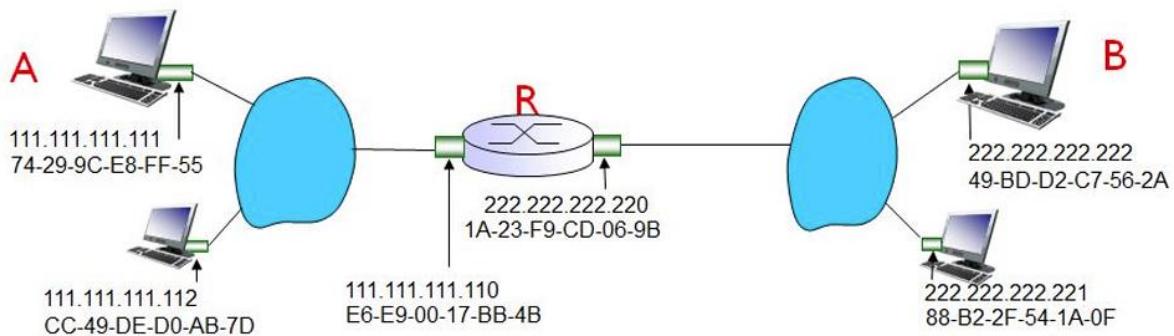
LAN (local area network): באյור מטה ניתן לראות רשת של ארבעה מחשבים מחוברים יחד ברכיב רשת כלשהו (יכול להיות HUB, switch וכו') כאשר לכל מחשב יש כתובת MAC ייחודית, וכן כתובת IP לוקאלית ייחודית – מתחילה עם אותהקידומת.



: – ניתן לקבוע כתובת MAC באמצעות כתובת IP: (Address Resolution Protocol) ARP

1. לכל מחשב יש טבלת ARP שהרשומות שלה הן: IP, MAC, TTL.
2. מחשב A שורצוה לשЛОוח למחשב B ויודע רק את כתובת IP של B ישלח הודעה לרכיב התקשרות אליו הוא מחובר ויצין כי על רכיב התקשרות לבצע broadcast ולשאול - מי זה IP ?B ?
3. מחשב B שיקבל את הודעהו מרכיב התקשרות יגיב בunicast לרכיב התקשרות, רכיב התקשרות יחזיר תשובה למחשב A עם כתובת MAC של B.
4. מחשב A שומר את MAC בטבלת ARP שלו (וסביר יותר ש גם מחשב B ישמור את MAC של A בטבלת ARP שלו).

דוגמיה חשובה:



באיור ישנן שתי רשתות שמחוברות ביניהן באמצעות router. אם A רוצה לשלוח הודעה למחשב שנמצא ברשת הולוקאלית שלו התהילך עובד בדיקוק כפי שהסביר (עם טבלת ARP). בעת, אם מחשב A רוצה לשלוח הודעה למחשב B – תחילתה הוא יסתכל על כתובת ה-IP של מחשב B ויבין שמחשב B נמצא ברשת אחרת (לפי קידומת ה-IP). וכך בשבב הרשות מחשב A יציין את ה-IP destination של מחשב B וכן את ה-IP source של מחשב A. לעומת זאת בשבב הארץן, כתובות MAC של המקור והיעד ישתו בכל פעם בהתאם לתחנה הקונקרטית שבה אנו נמצאים והתחנה הבאה במסלול אליה נגיעה. כאשר הודעה מגיעה/router הוא מתחליל אותה ומגליה כי אמם ה-destination MAC מתאים לכתובת ה-IP – אך ה-IP destination לא, וכך הוא מבין כי עליו לנטר את הודעה הלאה (במה שירן נלמד איך(router יידע דרך כתובות MAC לנטר את הודעה)). להלן פירוט כתובות IP ו-MAC בהתאם לאיור:

Source IP	Destination IP	Source MAC	Destination MAC
111.111.111.111	222.222.222.222	72-29-9C-E8-FF-55	E6-E9-00-17-BB-4B
111.111.111.111	222.222.222.222	E6-E9-00-17-BB-4B	1A-23-F9-CD-06-9B
111.111.111.111	222.222.222.222	1A-23-F9-CD-06-9B	49-BD-D2-C7-56-2A

תרגולים 2+1:

כאשר מתחילה VM ומגדירים attached to: NAT – בעצם מגדירים למחשב שהוא המחשב היחיד בראשת שלו. בנוסף ניתן להגדיר את MAC ואת IP.

הפקודה בלינוקס a קי – פולטת מידע על כרטיס הרשת, כתובות הIP.
Loop back – כרטיס למחשב עצמו, כשהמחשב רוצה להתיחס לעצמו – 127.0.0.1.

Ping (Packet Internet Groper) – פקודה בלינוקס שבוחקת את החיבור בין המחשב לשרת/המחשב עצמו. מקבלת כקלט את IP שאיתנו אנחנו רוצים לבדוק את החיבור, ומחזירה את תגובת הIP. אם נסנן לפיקט ICMP נקבל תוצאות Ping.

Wire shark: ניתן להסניף באמצעותה את התעבורה של כל הכרטיסים שיש למחשב. כמו כן מאפשרת לסנן את התוצאות לפי פרוטוקול מסוים, כתובות IP וכו'. ראיינו שנשלחה הודעה למחשב שאינו בראשת שלנו (ניתן להבדיל לפי כתובות הIP) נראה את IP של המחשב שלנו בפונקציית source IP ואת IP של המחשב המיועד בפונקציית destination IP, אולם כתובות MAC של הhost יהיה default gateway.

ARP – פקודה בלינוקס להדפסת טבלת ARP.
שרות UDP:

```
import socket  
IPV4                         UDP מסוג Socket  
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
קובעים באיזה PORT השירות מАЗין, וכן שהוא מАЗין לכל  
ברטיסי הרשות של המחשב (המחרחת הריקה)
```

```
while True:  
    data, addr = s.recvfrom(1024)  
    print(str(data), addr)  
    s.sendto(data.upper(), addr)
```

מגדירים כי המידע שיגיע יכנס
לdata (באורך של לפחות 1024
בתים) וכן addr יכיל את
הIP ואת הPORT של השולח.

בכדי שלא נדרש להתעסק עם כרטיס הרשות ורכיבי חומרה כאלה ואחרים, ומכוון שמערכת הפעלה כבר יודעת כיצד לתקשר עם רכיבי החומרה אנו מבקשים מערכת הפעלה שתיצור לנו socket שבאמצעוינו ניתן לתקשר עם מחשבים שונים. מערכת הפעלה צריכה לקבל מאייתנו את כל הפרטים כדי שתוכל לתרגם את הבקשה שלנו כמו שציר.

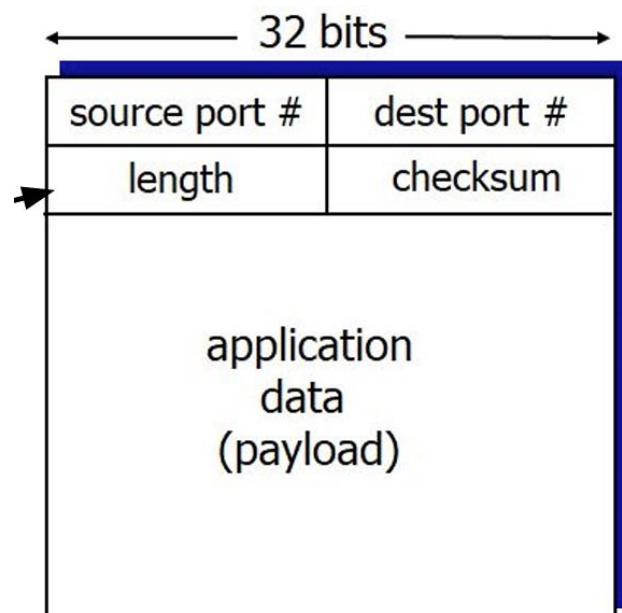
תזכורת: UDP הוא ראש קטן – אם היינו מגדירים data שמקבל מעט מדי בתים ביחס לגודל הודעה שנשלחה, היי מתקבלים רק כמהות הבתים שdata יכול להכיל והשאר יהיה מפרק לפחות זאת אם היינו מריםים זאת בדואו windows היינו מקבלים הודעה שגיאת שההודעה שנשלחה גדולה מדי עבור הbuffern.

הרצאות 3+2:

UDP – עושה את המינימום האפשרי, לא מבטיח אמינות, לדוגמה אם נשלח חבילה גדולה שחוורגת מהגדירות buffer שלו, הוא לא יחלק אותה לכמה חבילות וישלח כל אחת – אלא ישלח רק את הגודל המוגדר מראש, לא מתחזק חיבור רציף אלא פותח כל פעם חיבור חדש, שלוח חבילות לעומת TCP ששולח בתים. ניתן לדמות את הпрוטוקול הנ"ל לשילוח מכתבים – בכל פעם נדרש לכתוב למי המכתב מיועד, מכתבים יכולים לכתת לאיבוד וכו'. יתרונותיו – מהירות. ניתן למשת את הפונקציונליות שנוטן TCP כדי להבטיח אמינות ובה יותר בשכבות האפליקציה תוך שימוש בפרוטוקול UDP וכן לתקשר בצורה מהירה יותר.

TCP – מבטיח אמינות, לדוגמה אם חבילה שנשלחה לא הגיעו ליעדה TCP ישלח אותה שוב, מתחזק חיבור רציף. ניתן לדמות את הпрוטוקול הנ"ל לשיחת טלפון – תחיליה מציגים את הצדדים הדוברים ולאחר מכן השיחה זורמת. יתרונותיו – אמינות.

בשל פשוטותו, header של קטע יחסית: Header UDP

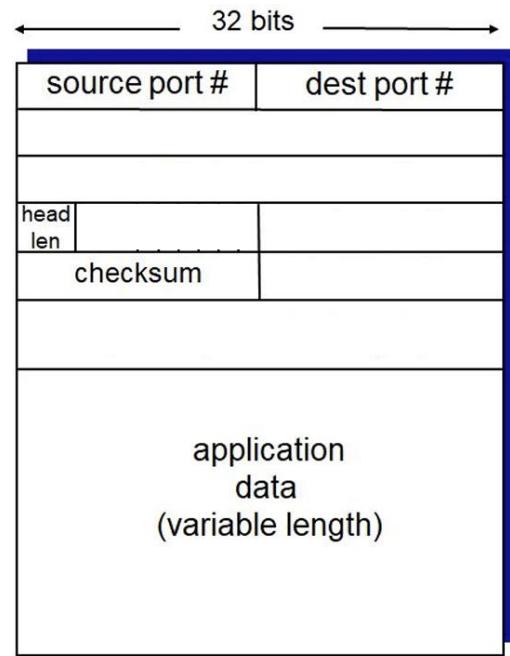


UDP segment format

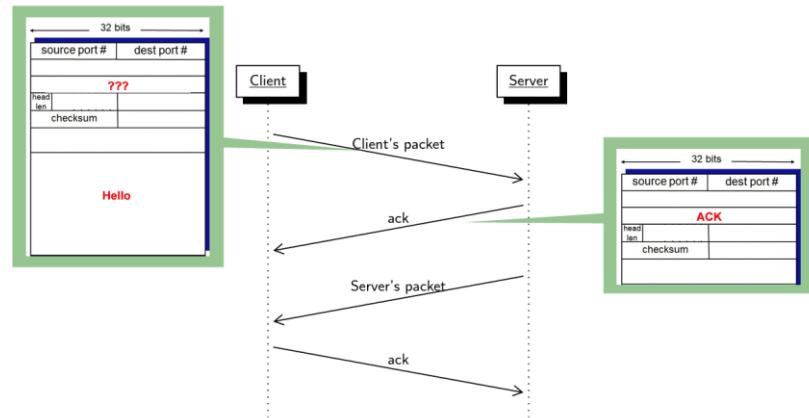
כעת נראה איך ניתן לבנות פרוטוקול דומה לTCP על גבי שכבת האפליקציה תוך שימוש בפרוטוקול UDP. תחילת נניח עולם מושלם:

1. ביטים לא משתבשים.
2. חבילות לא הולכות לאיבוד.
3. חבילות לא יעקרו את השניה לא לפי הסדר בהן שנשלחו.

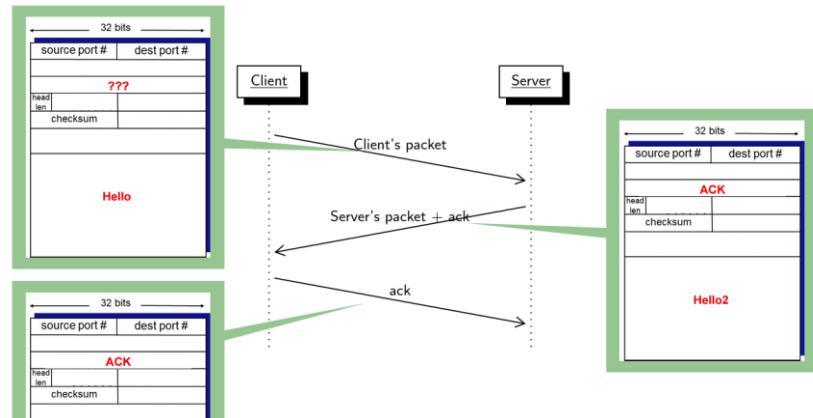
תחת הנחות אלו, פרוטוקול UDP לא צריך שום תוספות. להלן הheader שלנו עד כה, בהמשך נוסיף את השדות הריקים:



כעת נניח כי ביטים יוכלים להשתבש. עליה על השגיאה ולכן נשלח בתגובה NACK (negative acknowledgement) על מנת שהשולח ישלח את המידע שוב, נשים לב שהNACK נשלח גם הוא עם כל headers שהגרנו מוקדם. – המנגנון שמנע מהחבילות הבאות להישלח עד אשר מקבלים ACK על החבילה שנשלחה (אם ישלח NACK, נשלח שוב את החבילה ששלחנו מוקדם). להלן המחשה לשילוח הودעה וקבלת ACK:

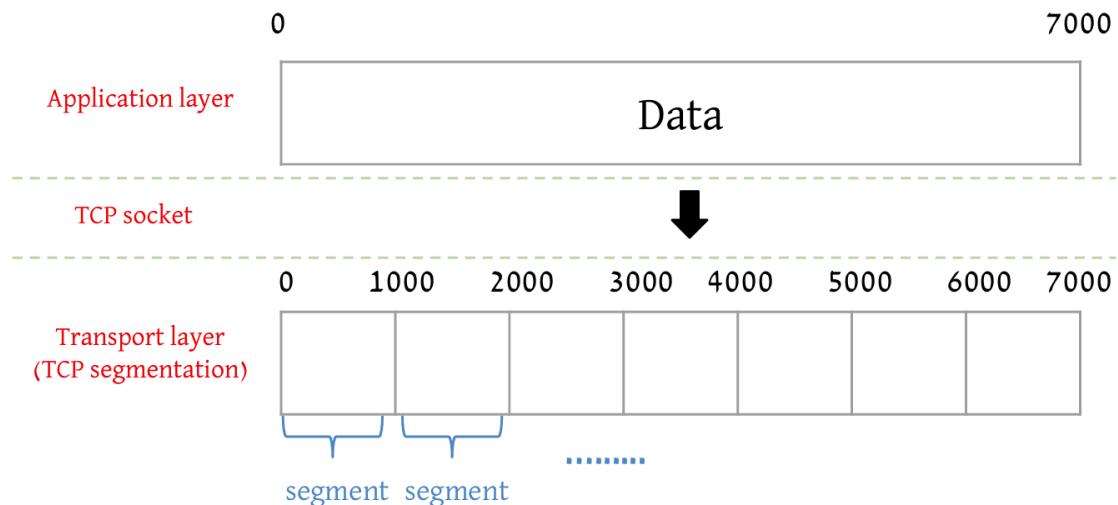


להלן המחשה לשילוח ACK + מידע שהשרת רצה לשלוח ללקוח:



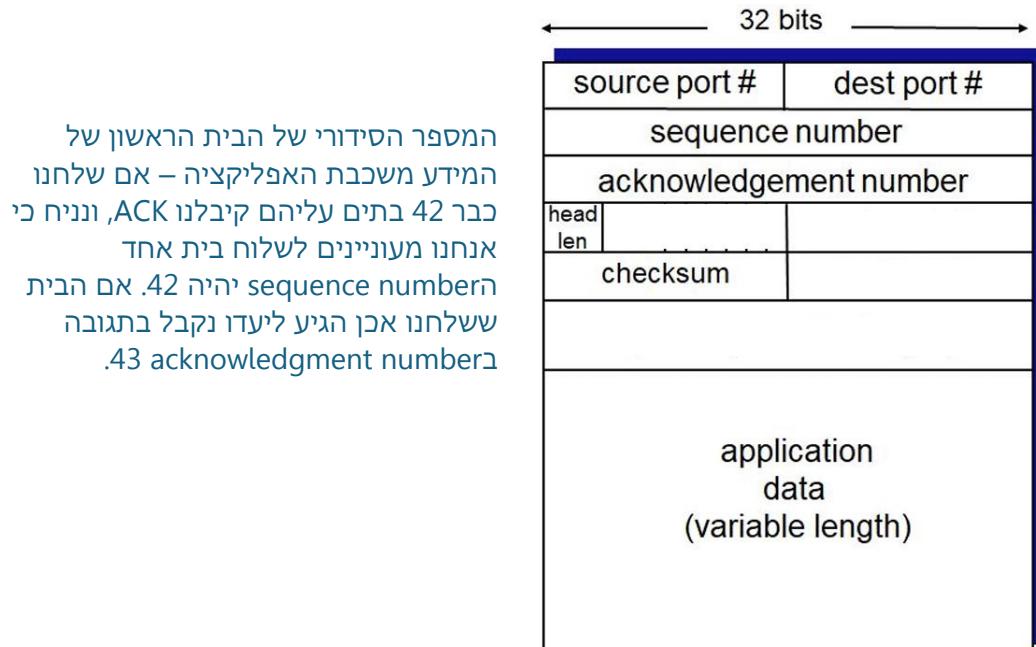
(maximum transmission unit) MTU – הגדל המקסימלי של חבילה שניתן להעבר על גבי הערוץ.

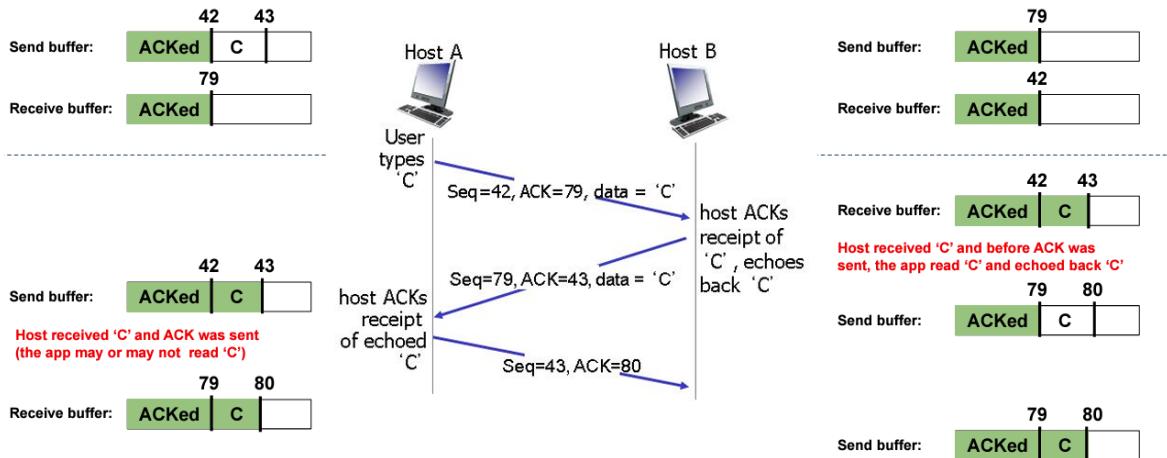
(maximum segment size) MSS – הגדל המקסימלי של מידע משכבות האפליקציה שיכל להכנס חבילה אחת כר' שאחרי שהוספנו את כל headers גודלה קטן מ-MTU (כלומר לא יהיה עליה להתפרק לחבילות קטנות יותר כדי להשלח בשלםותה).



באיר למעלה ניתן לראות כיצד חבילה בגודל 7000 בתים משכבות האפליקציה מפוצלת לחבילות בגודל MSS – 1000 בתים כל אחת, ע"י פרוטוקול TCP. כמו כן, מכיוון שההודעה הבאה בתור לא תשלח עד לקבלת ה-ACK, TCP יתחזק buffer send בו ימצאו כל הבטים לשילוח עד שנקבל ACK, ואז נשלח את ההודעה הבאה בbuffer.

אחרי שהוספנו את ה-ACKים לתוכנית שלנו, header החדש יראה כך:





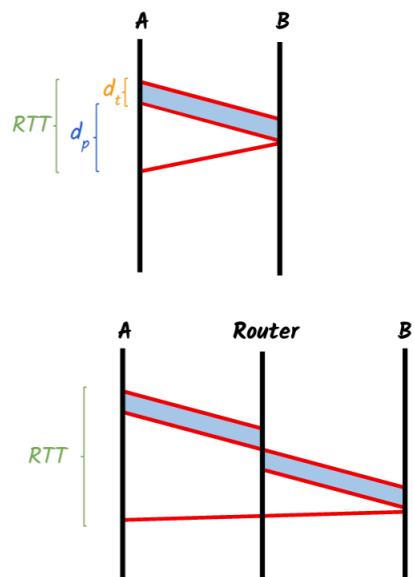
cut נניח כי חבילה הולכת לאיבוד. כיצד נדע שחבילה הולכת לאיבוד? נחכה פרק זמן מסוים שהוגדר מראש ואם הוא עבר – נשלח NACK.

– הזמן הכלול שלוקח מהרגע שהמגיד נשלח עד שקיבלו עליו ACK. RTT (round trip time)

d_t – הייתה שידור, פונקציה שמחשבת את הזמן שלוקח למידע להישלח על בסיס קצב השידור וכמות המידע שנשלח (לדוגמה אם קצב השידור הוא 100 ביט לשנייה ואנו רוצים לשלוח 100 ביטים d_t הוא שנייה). הזמן שלוקח לשילוח הבתים מראשון עד האחרון.

d_p – הייתה התפשטות, פונקציה שמחשבת את הזמן שלוקח למידע לעבור מרחק מסוים על בסיס קצב ההתפשטות וכמות המידע שנשלח (לדוגמה אם קצר ההתפשטות הוא 100 מטר לשנייה ואנו רוצים לשלוח ביט למחרך מאיינו 100 מטר d_p הוא שנייה). הזמן שלוקח לביט בוודuct לעבור את המרחק עד התחנה הבאה.

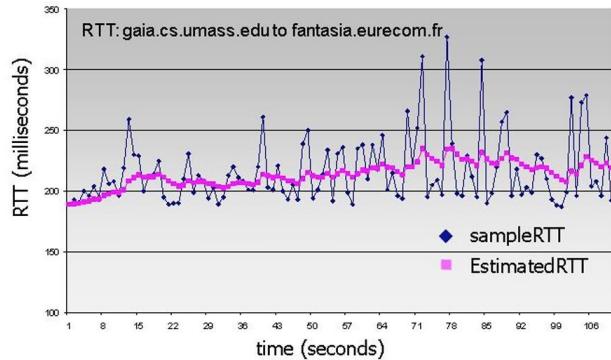
Store and forward – מנגןון של רכבי התקשרות, כל עוד המידע לא הגיע בשלמותו מאחסנים אותו בbuffer כלשהו ורק לאחר שכל המידע הגיע מעבירים אותו לתחנה הבאה.



איך נעריך את RTT על מנת שנקבע timeout טוב? כלומר לוקחים את הממוצע שקיבלו $EstimatedRTT = (1 - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$

עד כה ומכפילים ב- α – 1 ומוסיפים לו את הדגימה של RTT האחרון שקיבלו (זמן RTT האחרון – RTT הקודם). נשים לב ששלוחת הודעה אחרונה עד קבלת ACK עליה). ככל ש- α גבוה יותר ניתן יותר משקל לדגימה האחרונה, בדר"כ משתמשים ב- $\alpha = 0.125$.

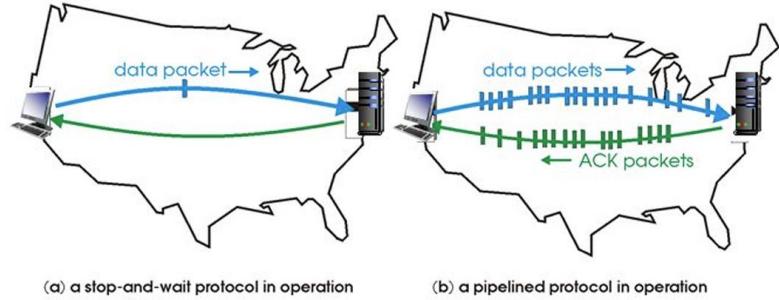
ניתן לראות בדוגמה הנ"ל שהsampleRTT הרבה יותר תנוודתי, ואילו הממוצע הרבה יותר יציב.



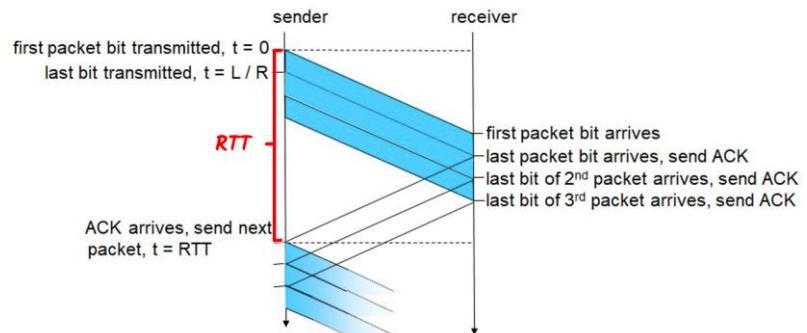
כעת לאחר שיחסבנו את הממוצע של RTT, נרצה לחשב את סטיות בין מה הממוצע שייצא לנו לבין הדגימות בפועל, שכן גדרה: $|DevRTT| = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$. בדר"כ משתמשים ב- $\beta = 0.25$.

ולבסוף גדריר את הזמן timeout שלנו להיות: $TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$

כעת לבעה אחרת – מכיוון שאנו עובדים על מנגנון stop & wait, אנחנו "מבזבזים" הרבה זמן, שכן בזמן שהודעה נשלחה יכולנו לשולח הודעות נוספות נספנות, אולם אנחנו מחייבים לקבלת ACK עליה ולען לא עושים זאת. הפתרון הוא שימוש בpipelining: ניתן לשולח כמה הודעות במקביל, אך ACKים ישמרו על הסדר שבו השולחות נשלחו.

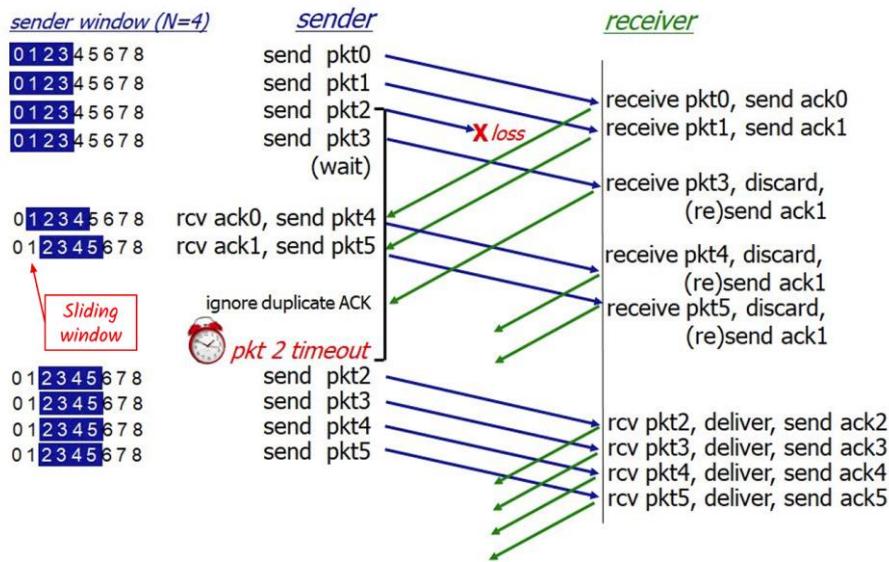


חולון – כמות החבילות שניתנת לשולח אחרி השניה בpipelining, מוביל לקבל ACK על החבילה הראשונה. כאשר קיבל ACK על החבילה הראשונה ששלחנו, החולון יכול "לגלוש" להלאה ולשלוח את החבילה הבאה בתור שעוד לא שלחנו וכן להלאה.

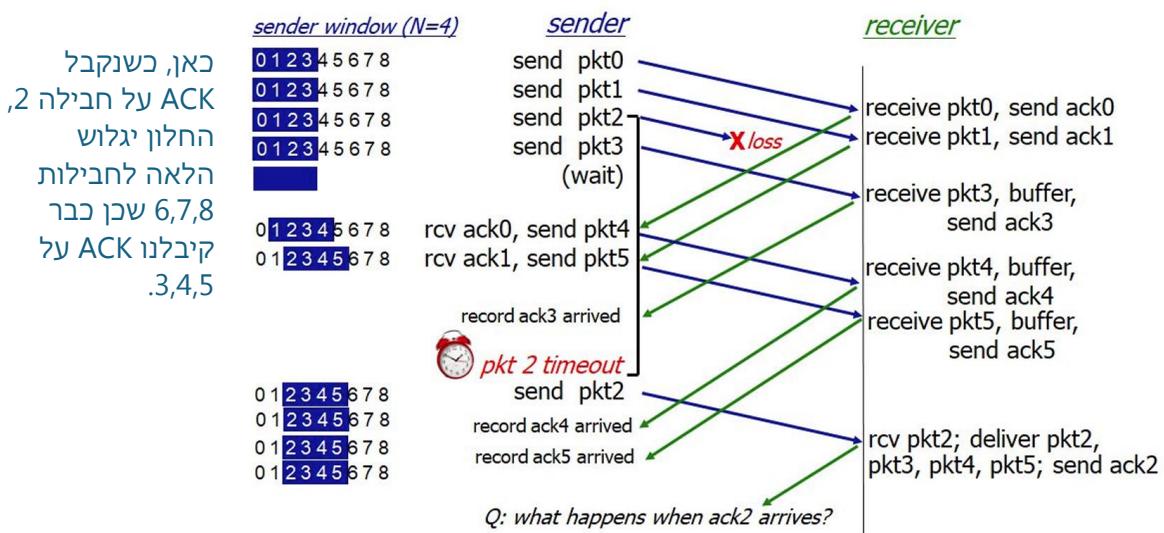


איך נתמודד עם חבילות שmagiuot לא לפי הסדר? נראה CUT 2 שיטות – TCP לא משתמש בהן (אלא מבצע מעין הכלאה בין שתייה):

- .1 GBN (go back) – שלוחים מספר חבילות כגודל החלון, ח. אם במהלך הדרך חבילה לא הגיעה ליעדה, המקלט לא שומר את שאר החבילות שיגעו אחריה עד שהיא תשלח שוב עם שאר החבילות שנמצאות אחריה בחלון. מכאן גם שם השיטה – ברגע שחבילה לא הגיעה ליעדה נצטרך לחזור אחריה ולשלוח מחדש את חבילות שנמצאותCut בחלון. החלון ימשיך לגלוש רק כאשר קיבל ACK על חביבה ראשונה שנמצאת בחלון.

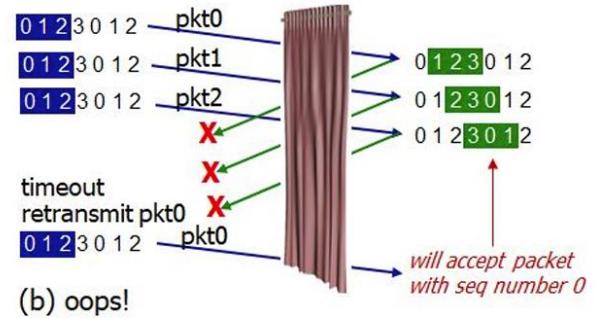


- .2 Selective repeat – שלוחים מספר חבילות כגודל החלון, ח. אם במהלך הדרך חבילה לא הגיעה ליעדה, המקלט שומר את החבילות שכן הגיעו ושולח עלייה ACK. אצל השולח החלון לא ימשיך מעבר לחבילה שטרם התקבל עלייה ACK, וכאשר יחול timeout לחביבה שנאבדה הוא ישלח רק אותה סלקטיבית שוב. ברגע שיתקבל עלייה ACK – החלון יחלש לחביבה הבאה שעדיין לא קיבלנו עלייה ACK.



נשים לב שנוצרת לנו בעיה בשימוש בשיטות אלו – שכן אם היחס בין גודל החלון לגודל הsequence number לא מספיק גדול (כלומר הsequence number צריך להיות גדול ממספרות מוגדל החלון), והACKים על החבילות לא הגיעו, השולח ישלח אותן מחדש – אך המקלט יחשב שמדובר בחבילות חדשות.

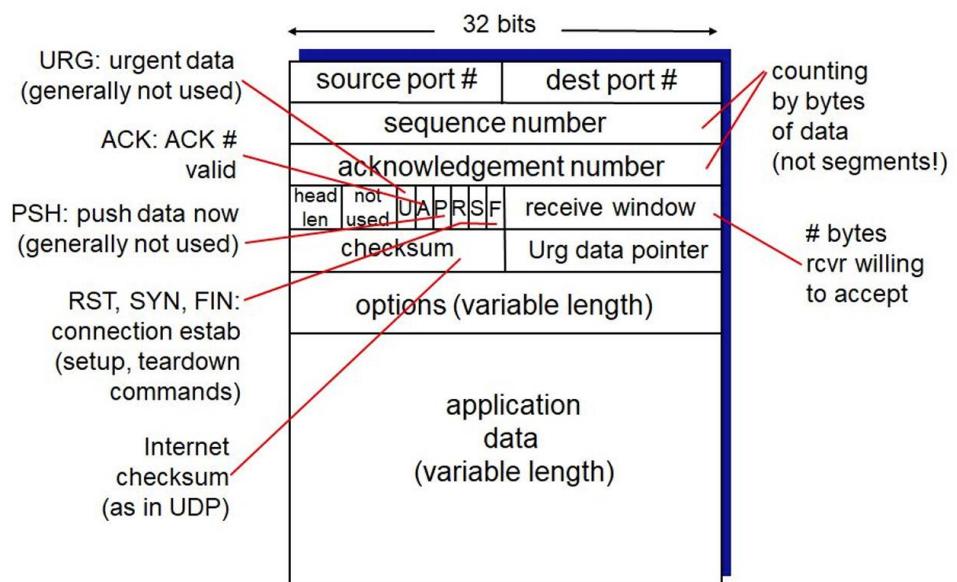
היחס בין גודל החלון לגודל ה sequence number כמעט זהה, וכך כאשר השולח לא קיבל את הACKים על 0,1,2 הוא שולח אותן מחדש, אולם המקלט בטוח שמדובר בחבילה חדשה שכן הוא מצפה לקבל חביבה עם sequence number 0 בהמשך.



כעת, אחרי שהראנו איך ניתן להתאים פרוטוקול TCP על גבי שכבת האפליקציה, נדגים איך TCP עובד בפועל:

- .1 Point to point – ישנו שולח אחד ומקבל אחד, כמו שיחת טלפון, ה socket מיצג גם את השולח וגם את המקלט (שלא כמו UDP ששרת אחד יכול היה לתקשר עם כמה לקוחות על גבי אותו socket).
- .2 אמין, דואג שהบทים שנשלחו אלינו יגיעו לפי הסדר ושיהיה מדובר ב stream של בתים ולא בהודעות.
- .3 Pipelined – משתמש בpipelining על מנת להגבר את היעילות, אך יודא באמצעות מגנונים לבקרת עומס ובקרת זרימה שאנו לא שולחים מהר מדי או יותר מדי כך שהיעילות תפגע.
- .4 התקשרות היא דו-כיוונית, כלומר כל אחד מהצדדים יכול גם לשולח בתים וגם לקבל בתים כל הזמן (בכפוף לעמידה ב MSS).
- .5 התקשרות היא מוכוננת חיבור – ככלומר TCP מקיים חיבור בכל פעם ומסיים אותו בסוף ההתקשרות בין המחשבים.

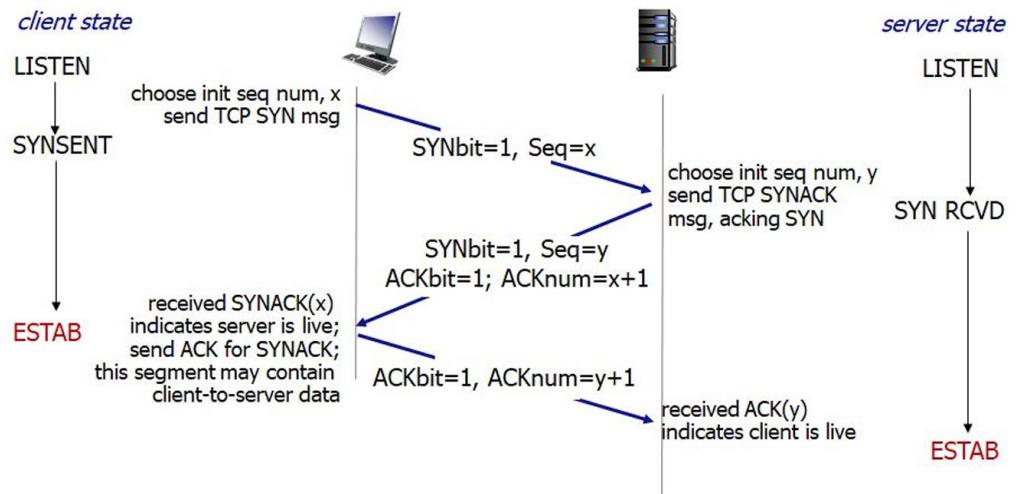
וכך נראה TCP header:



בכל פעם当我们初始化 TCP תקשורת אנחנו מקיימים חיבור בין שני הצדדים, החיבור נוצר ע"י hand shake שלושה שלבים:

- .1 SYN – תחילת שלוחים חביבה עם header TCP כאשר אין מידע בשכבת האפליקציה, ורק דגל SYN דולק. כך מקבל מוביל שהשולח רוצה להקים עמו חיבור.

- .2 – מחייב לשולח בחזרה header TCP שגם הוא ריק בשכבות האפליקציה עם שני דגלים Dolkim belid: SYN – מבקש מהשולח להסתنصرן אותו, ACK – מאשר את קבלת הבקשה לSYN.
- .3 – ACK – מאשר את קבלת הבקשה לSYN מהמקבל. יתכן שהודעה זו תכלול כבר מידע שמיועד למקבל.



תרגול 3:

בתרגול נראה איך מחשבים ייעילות של פרוטוקולים.

קצב שידור – כמה ביטים בשנייה ניתן לשדר.

קצב התפשטות – כמה ביטים בשנייה אותן מצליח לעברו.

הש HOLDERSHT שידור מקצתו מקצתו מחושבת ע"י: $d_{E2E} = d_t + d_p + d_{proc}$ (בהמשך נתיחס לעוד גורמים, בין היתר זה יספק לנו).

d_t הייתה שידור – משך הזמן שմדרדים את הביט הראשון עד הביט האחרון. כדי לחשב אותה נחשב את $\frac{L}{R}$ כאשר: L – כמות הביטים שרוצים לשדר, R – קצת השידור של R ביטים לשנייה.

d_p הייתה התפשטות – משך הזמן שלוקח לביט לעברו מהמקור ליעד. כדי לחשב אותה נחשב את $\frac{d}{s}$ כאשר: d – המרחק שעל הביט לעברו, s – קצת התפשטות של s ביטים לשנייה.

d_q הייתה תור – כמה זמן חביבה ממתיינה בתור לפני שהיא משוחררת להלאה. השהייה זו היא פונקציה של קצב השידור וכמות החביבות בתור.

d_{proc} הייתה העיבוד – כמה זמן לוקח לטפל בחביבה ולהעביר אותה לתור המתאים.
תזכורת:

$$1B (\text{Byte}) = 8b (\text{bits}) \quad .1$$

$$1KB = 10^3B = 8 \cdot 10^3b \quad .2$$

$$1MB = 10^6B = 8 \cdot 10^6b \quad .3$$

$$1GB = 10^9B = 8 \cdot 10^9b \quad .4$$

שאלה: לקוח הממוקם בניו יורק ניגש לשרת הממוקם בלונדון כדי להוריד קובץ. נתון:

.1. המרחק מהלכו לשרת הוא 2500km

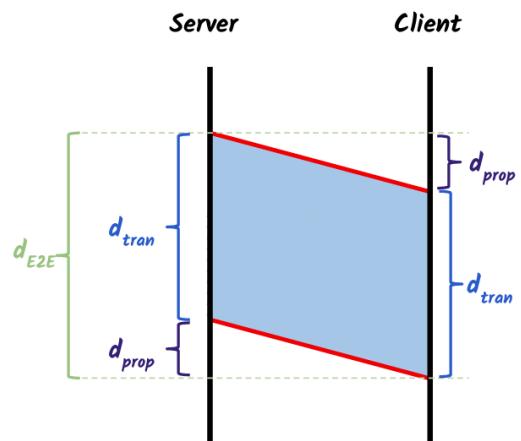
.2. קצב התפשטות הוא 250,000km/s

.3. קצב השידור הוא 8Mbps

.4. גודל הקובץ הוא 1MB

כמה זמן ייקח להעביר את הקובץ מהשרת לקווק, בהנחה שאין עוד תעבורה ברשות ואין השהייה עיבוד ותור?

תשובה: על מנת להוריד קובץ צריך לחשב את השהייה השידור של כל הקובץ, פלוס השהייה התפשטות של הביט האחרון.



נחשב:

$$L = 1MB$$

$$R = 8MBps$$

$$d_t = \frac{L}{R} = \frac{8 \cdot 10^6 b}{8 \cdot 10^6 bps} = 1sec$$

$$d = 2500km = 2.5 \cdot 10^6 m$$

$$s = 250,000kmps = 2.5 \cdot 10^8 mps$$

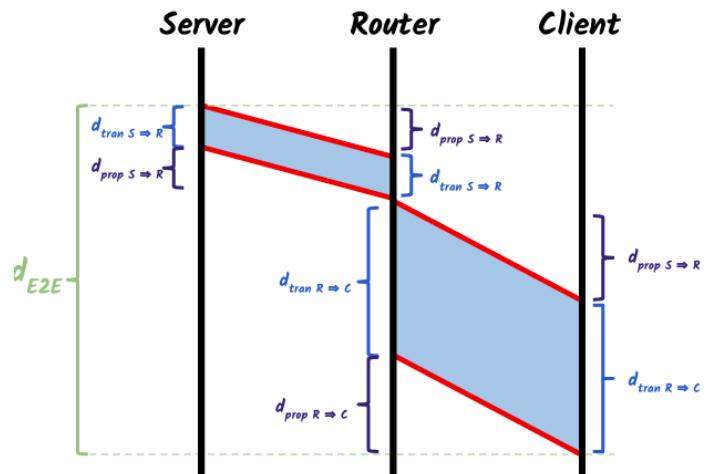
$$d_p = \frac{d}{s} = \frac{2.5 \cdot 10^6 m}{2.5 \cdot 10^8 mps} = 10^{-2} sec = 0.01 sec$$

$$d_{E2E} = d_t + t_p = 1 + 0.01 = 1.01sec$$

שאלה: בהמשך לשאלת הקודמת, כמה זמן ייקח להעביר את הקובץ בהינתן הנתונים הבאים:

- .1 גודל הקובץ הוא 1MB
- .2 קצב ההתפשטות הוא 250,000kmps
- .3 בין הלקוח לשרת ישנו רואוטר.
- .4 המרחק בין הלקוח לרואוטר הוא 250km
- .5 המרחק בין הרואוטר לשרת הוא 5000km
- .6 קצב השידור בין הלקוח לרואוטר הוא 800kbps
- .7 קצב השידור בין הרואוטר לשרת הוא $10^6 bps$
- .8 קצב השידור בין הרואוטר ללקוח הוא $10^6 bps$

תשובה:



מכיוון שהנתב עובד במנגנון store & forward, علينا לחשב את השהייהו של השידור מהשרת לרואוטר, ועוד השהייהו ההתקשות של הביט האחרון מהשרת לרואוטר. ולאחר מכן להוסיף את השהייהו של השרת ללקוח ועוד השהייהו ההתקשות של הביט האחרון מהשרת ללקוח.

נחשב:

$$d_{t server \rightarrow router} = \frac{L}{R} = \frac{8 \cdot 10^6 b}{8 \cdot 10^6 bps} = 1sec$$

$$d_{p \text{ server} \rightarrow \text{router}} = \frac{d}{s} = \frac{5 \cdot 10^6 \text{m}}{2.5 \cdot 10^8 \text{mps}} = 2 \cdot 10^{-2} \text{sec} = 0.02 \text{sec}$$

$$d_{t \text{ router} \rightarrow \text{client}} = \frac{L}{R} = \frac{8 \cdot 10^6 \text{b}}{8 \cdot 10^5 \text{bps}} = 10 \text{sec}$$

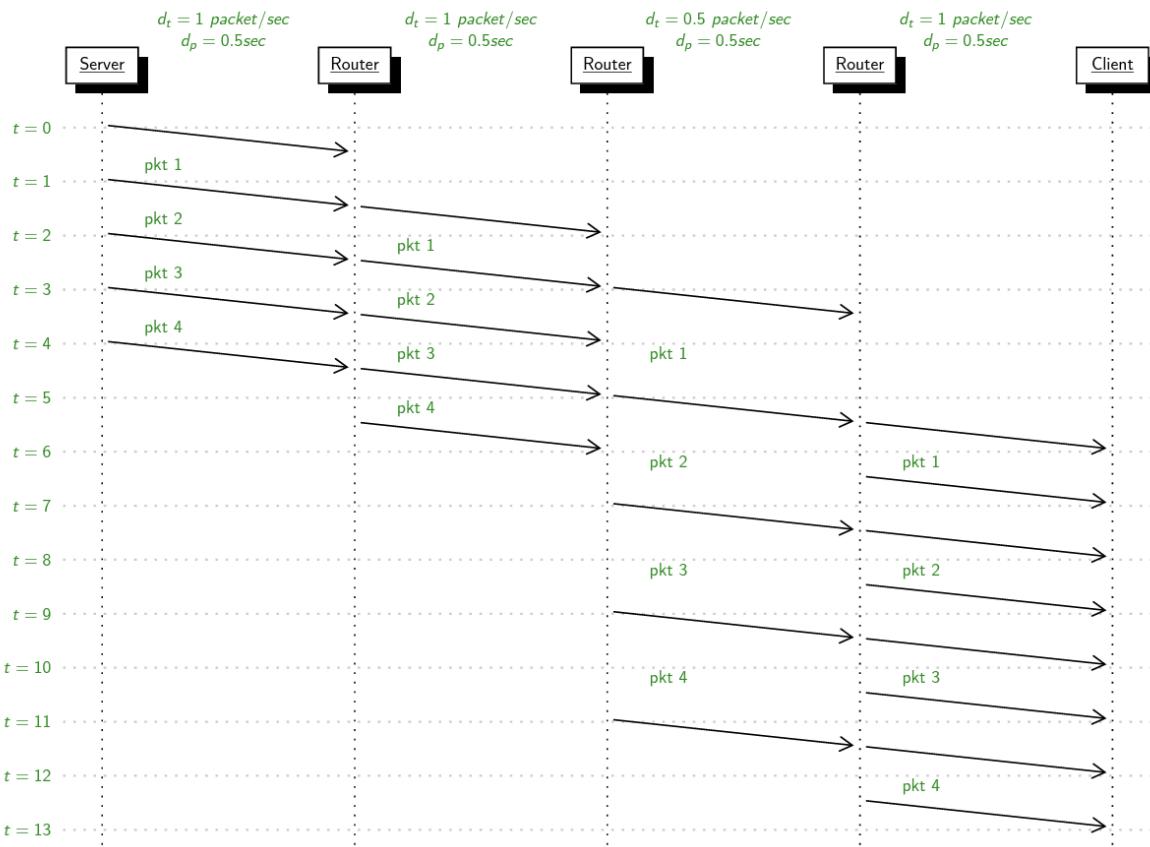
$$d_{p \text{ router} \rightarrow \text{client}} = \frac{d}{s} = \frac{2.5 \cdot 10^5 \text{m}}{2.5 \cdot 10^8 \text{mps}} = 10^{-3} \text{sec} = 0.001 \text{sec}$$

$$\therefore d_{E2E} = 1 + 0.02 + 10 + 0.001 = 11.021 \text{sec}$$

נשים לב שזמן רב התבזבז בגל מנגנון `stop & forward` של הרואוטר. אם היינו מחלקים את החבילה מראש לחבילות קטנות יותר נוכל לשלוח את הקובץ "במקביל", מה שנקרא **pipelining**.

לכן, נשדרג במעט את החישוב: $d_{E2E} = d_1 + d_2 + d_3$ כאשר:

1. d_1 – סך כל ההשיות שלוקח לחבילה הראשונה עד שהיא מגיעה לעורוץ האיטי ביותר.
2. d_2 – סך כל ההשיות בעורוץ האיטי ביותר.
3. d_3 – סך כל ההשיות שלוקח לחבילה האחרונות עד שהיא מגיעה ליעד.



שאלה: כמה זמן ייקח להעביר את הקובץ בהינתן הנתונים הבאים:

1. גודל הקובץ הוא 1MB ומשוחלך לחבילות בגודל 10,000B כל אחת.
2. קצב ההתפשטות הוא 250,000km/sec.
3. בין הלקוח לשרת ישנו רואוטר.
4. המרחק בין הלקוח לרואוטר הוא 250km.
5. המרחק בין הרואוטר לשרת הוא 5,000km.
6. קצב השידור בין הלקוח לרואוטר הוא 800kbps.
7. קצב השידור בין הרואוטר לשרת הוא 10^6bps .

תשובה: נחשב כמה חבילות שלוחים בסה"כ: $\frac{1MB}{10,000B} = \frac{8 \cdot 10^6 b}{8 \cdot 10^4 b} = 100$.zeit נחיש כמה זמן לוקח לחבילה הראשונה עד שהגיעה לערוץ האיטי, ככלומר לראוטר:

$$d_{t \text{ server} \rightarrow \text{router}} = \frac{8 \cdot 10^4 b}{8 \cdot 10^6 bps} = 10^{-2} sec = 0.01 sec$$

$$\cdot d_{p \text{ server} \rightarrow \text{router}} = \frac{5 \cdot 10^6 m}{2.5 \cdot 10^8 mps} = 2 \cdot 10^{-2} sec = 0.02 sec$$

$$\text{ולכן נקבל } d_1 = 0.01 + 0.02 = 0.03 sec$$

נחשב כמה זמן לוקח לשדר את 100 החבילות בערוץ האיטי ביותר:

$$\cdot d_{t \text{ router} \rightarrow \text{client}} = 100 \cdot \frac{8 \cdot 10^4 b}{8 \cdot 10^5 bps} = 100 \cdot 10^{-1} sec = 10 sec$$

נחשב את ההתפשטות בערוץ האיטי ביותר: $d_{p \text{ router} \rightarrow \text{client}} = \frac{2.5 \cdot 10^5 m}{2.5 \cdot 10^8 mps} = 10^{-3} sec = 0.001 sec$

$$\text{ונקבל כי } d_2 = 10 + 0.001 = 10.001 sec$$

מכיוון שהערות האיטי ביותר הוא גם הערוץ האחרון d_3 ולכן נקבל:

$$d_{E2E} = 0.03 + 10.001 = 10.031 sec$$

לכארה, לפי התוצאות שקיבלנו – שכך שמחקרים את המידע ליותר חבילות הזמן הכלול קטן, כדי פשוט לשЛОוח את המידע בחבילות בגודל בית 1. אולם לא לקחנו בחשבון את headerים שנשלחים עם כל חבילה, וכך יש מחיר לכך שנשתמש בהרבה חבילות.

שאלה: מחשבים A ו B שולחים למחשב D דרך נתב R. כאשר חבילה MA מגיעה לר' יש כבר 3 חבילות MB שממתינות בבאפר ועוד חצי חבילה שימושית ל-R. התוור נתב עובד בתצורת FIFO. כל החבילות הן בגודל 1000B 1Kb השידור הוא 1Mbps. מהי השהייה התוור עבור חבילה A?

תשובה: יש לחשב את הזמן שלוקח לנtab לשדר את 3 החבילות שממתינות בבאפר ועוד זמן השידור של

$$d_q = \frac{3.5 \cdot 8 \cdot 10^3 b}{10^6 bps} = 0.028 sec$$

שאלה: A, B שולחים כל אחד לד' קובץ בגודל 10Mb 1000B דרך נתב C.

$$R = 10 Mbps .1$$

$$d_p = \frac{d}{s} \text{ המרחק חלקי קצב ההתפשטות}.2$$

$$1. \text{ גודל חבילה מקסימלי הוא } 1KB$$

$$2. \text{ התעלמו מתחיליות.}$$

כמה זמן לוקח עד ש D יקבל את שני הקבצים? מהו גודל הבאפר המינימלי של C כל שלא יהיה אובדן חבילות?

תשובה: נבדוק כמה חבילות יש לשולוח: $1250 = \frac{10^7 b}{8 \cdot 10^3 b}$. כמו כן, נשים לב כי על שתי חבילות שמאזינעות לנtab C, משודרת רק אחת, ככלומר ב-C יתחל להיווצר תור שגדל כל פעם ב-1.

ראשית נבחן כי הערוץ האיטי ביותר (צואր הבקבוק) הוא השידור MC. וכך עליינו לחשב את השהייה השידור של חבילה אחת ולהכפיל אותו ב-2501 (סה"כ כל החבילות שעוברות דרך C לד', ועוד השהייה השידור של הגעת החבילה הראשונה LC) ולהוסיף את השהייה ההתפשטות של החבילה הראשונה MB/A LC ועוד השהייה ההתפשטות של החבילה الأخيرة MC לד'. נחיש:

$$d_t = \frac{L}{R} = \frac{8 \cdot 10^3 b}{10^7 bps} = 0.0008 sec$$

$$\text{ובסה"כ: } d_{E2E} = 2501 \cdot d_t + 2 \cdot d_p = 2501 \cdot 0.0008 + 2 \cdot 10^{-3} = 2.0028sec$$

לABI גודל הבארפר – מכיוון שעל כל שתי חבילות שמגיעות לC, C משדר רק אחת – נקלט שעל מנת שלא יאבדו חבילות גודל הבארפר צריך להיות לפחות חצי מגודל המידע שנשלח אליו, כלומר 10Mb (סך המידע שנשלח ל C היה 20Mb – כפול שני הקבצים).

שאלה:

1. נתון נתב עם תור בגודל 100KB.
2. התור עובד בתצורת FIFO.
3. הנתב מחבר בין המחשב של אליס לאינטרנט.
4. קצב השידור של הערוץ בין הנתב לאינטרנט הוא 125Kbps.

כאשר התור בננתב מלא, כמה זמן ייקח לתור להתרוקן?

$$\text{תשובה: } \frac{100KB}{125KB} = 0.8sec$$

שאלה: בהמשך לשאלת הקודמת, נתון כי בזמן 0 אין שולחת חבילות בגודל 1KB בקצב של 150 חבילות בשנייה. מתי תיזרק החבילה הראשונה ע"י הנתב?

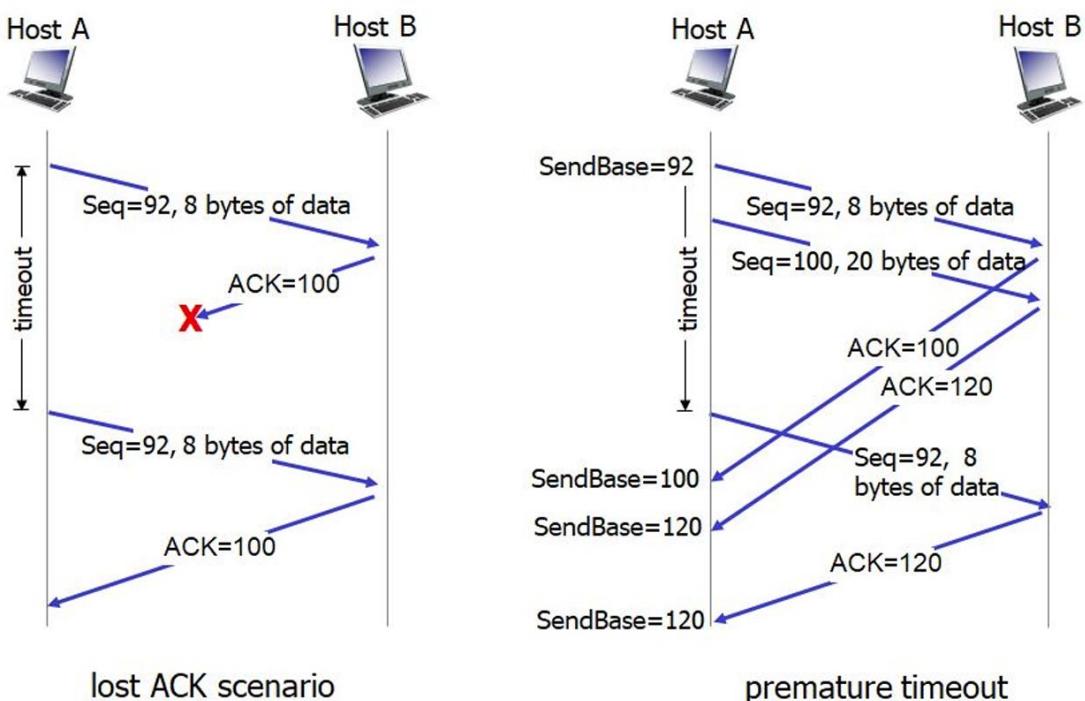
תשובה: תחילת נשים לב שאלה שולחת לננתב 150KBps וailo הנתב "מרוקן" מעצמו כל שנייה KB, כלומר בסה"כ בכל שנייה נאגרים בננתב 25KB, ולכן לאחר 4 **שניות** נקלט שתור הנתב מלא וחבילות יתחילה להיזרק.

הרצאה 4:

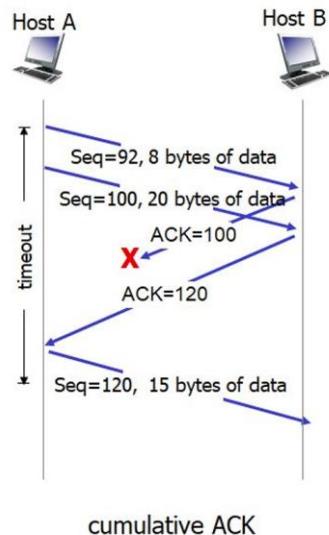
בהרצאה הקודמת סיימנו עם הסבר על hand shake. בעת נסביר על התהליך הקשור בעת סיום חיבור החיבור מסטיים ע"י שלושה/ ארבעה שלבים:

1. FYN – תחילת שלוחים חיבור עם header TCP כאשר אין מידע בשכבה האפליקציה, ורק דגל הNPN דלוק. כך מקבל מבחן שהשולח רוצה לסיים את החיבור.
2. ACK / FYNACK – מחריז לשולח בחזרה header TCP שאגם הוא ריק בשכבה האפליקציה עם שני דגלים دولקים בלבד: FYN – מבקש מהשולח לסיים את החיבור אותו, ACK – מאשר את קבלת הבקשה לFYN. נשים לב שגם נשלח בהתחלה רק ACK על סיום התקשרות, לאחר מכן מקבל שולח גם הוא FYN כדי שהשולח יסגור עמו את החיבור (ולכן יכול להיות ארבעה שלבים במקום שלושה).
3. ACK – מאשר את קבלת הבקשה לFYN מהמקבל.

עת נסביר איך TCP יודע להתמודד עם אובדן חבילות. תחילת, כל חיבור שmagie TCP יאחסן received buffer, במקומות המתאים לה לפי sequence number. במידה והחביבה הבאה שהגיעה מתאימה למקום הבא received buffer שלוחים עליה ACK וממשיכים. במידה ולא – מאחסנים את החביבה במקום המתאים לה received buffer, ושלוחים שוב ACK על החביבה האחורונה שהגעה לפי הסדר (נקרא גם ACK כפול שכן כבר אישרנו את קבלת החביבה הקודמת). במידה והגעה החביבה "סותמת" לנו חור בbuffer, שלוחים ACK מצטבר – לא על החביבה בלבד אלא על החביבה האחורונה ברצף שנוצר כתוצאה מסתימת החור.



האיור ממחיש מה קורה במידה ו-ACK מסויים אבד בדרך לפני הזמן של החבילה, מכיוון שהACK על החבילה השנייה כבר הגיע, מחשב A לא ישלח שוב את החבילה הראשונה מכיוון שקיבל ACK מצטבר עד 120 (שכלול בתוכו גם את 100).

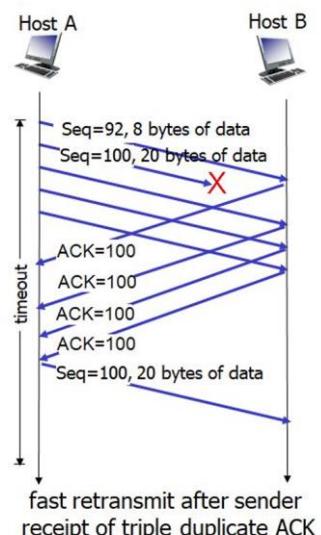


השאלה המתבקשת היא – למה לשЛОח כל כך הרבה ACKים? בדוגמה לעיל אפשר לראות שהוא מספיק לשЛОח רק את ACK 120 מכיוון שהרעיון חל אחריו. וכך ניהול ה-ACKים של TCP מוגדר ככזה:

1. אם קיבלת חבילה עם sequence number שצפית לה – עכב את שליחת ה-ACK עד 500ms. אם לא התקבלה חבילה הבאה – שלח את ה-ACK.
2. אם קיבלת חבילה עם sequence number שצפית לה, ויש ACK מעוכב (בגלל הסעיף הקודם) – שלח מיד את ה-ACK המציג (עד לחבילה השנייה שהתקבלה).
3. אם קיבלת חבילה עם sequence number שהוא לא לפי הסדר, שלח מיד ACK כפול (כלומר ACK הآخر שכבר שŁוחת).
4. אם קיבלת חבילה עם sequence number שסוטם חור בbuffer – שלח מיד ACK מצטבר עד להיכון שהחור נסתם.

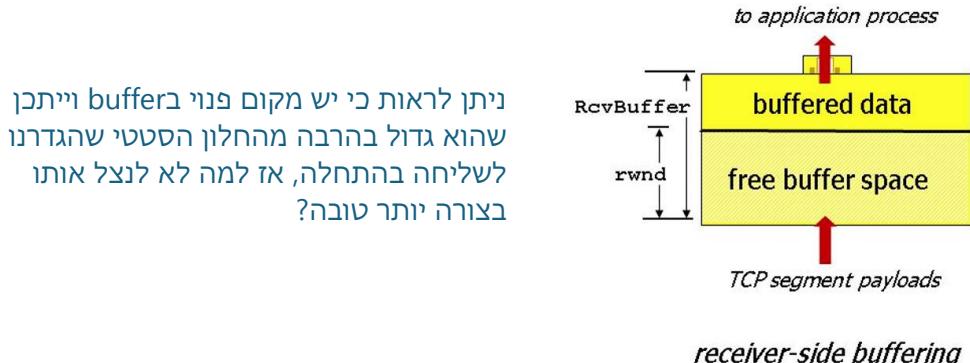
נשים לב לעד נקודה – ACK כפול יכול להיות כתוצאה משני אירועים – או שחבילה הלאה לאיבוד ולכך חבילות אחרות המשיכו להגיע, או שהחבילות הבאות שנשלחו הצליחו עקפו את החבילה הראשונה שנשלחה. ככל שנקל יוטר AC'ים כפולים, הסבירות שמדובר בחבילה שמתעכבות ולא בחבילה שהלאה לאיבוד קטן, וניתן לעלות על זה עוד לפני timeout שלה. ובהתאם על נקודה זו פועל מגנון ה **:retransmit**

מחשב A שלח 4 חבילות למחשב B אך החבילה השנייה אבדה. מחשב B שלח כבר שלושה ACK'ים כפולים (4 ACK'ים סה"כ) על קבלת 100. מכיוון שעדין לא תם הזמן לחבילה השנייה, אך הסבירות שהיא לא הלאה לאיבוד קטן מאוד, מחשב A ישלח אותה שוב. כאשר מחשב B קיבל אותה, מכיוון שהיא סותמת לו את החור שנוצר הוא ישלח ACK מצטבר עד **ללא sequence number** של החבילה הרובעתית.



נניח כי אנחנו שולחים חבילות, ובעת הנו עוברים את תהליך העיבוד במחשב המקלט (מפורקות לheaders ולבסוף עוברות לbuffer של שכבת האפליקציה על מנת שתתחליל למצות ממנו את המידע). בעת נתיחס

לחalon בקרת הזרימה. בהרצאה הקודמת תתייחסנו לחלונות סטטיים, ככלומר חלונות שנשאים בגודל קבוע לאורך כל החיבור. מטרתו של חalon בקרת הזרימה הוא לדאוג לכך שמידע ימשיך להישלח כל עוד received bufferן פניו לקבל עד מידע. וכך בTCP header היכן שרשום window נציג בכל פעם את גודל bufferן הזמן לקבלת מידע חדש, וכך השולח ידע בהתאם את מספר החבילות לגודל bufferן הפניו.



על מנת לחשב כל פעם כמה חבילות ניתן לשלוח נגדיר את הפרמטרים הבאים:

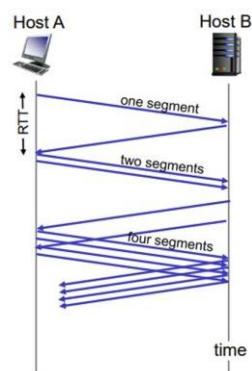
1. – המיקום הפניו buffer (המידע נמצא בTCP header נמצוא בTCP header האחרון שקיבלונו receive window).
2. .2 – החבילה האחרונה עליה קיבלנו ACK. LBA (last byte ACKed)
3. .3 – הבית האחרון אותו שלחנו. LBS (last byte send)

כל עד > 0 (LBS-LBA) – send window נתון לחalon לגלוש לכיוון החבילות הבאות.

מנגנון בקרת העומס – דואג לבקרת העומס על הרשות, במקרה שיש יותר מחשב אחד שרוצה להתחבר לממחשב א. המנגנון צריך להיות:

1. למקסם את כמות החבילות שניתן לשלוח כל עוד הרשות מאפשרת זאת.
2. הוגן – שכל משתמש מקבל משאבים שווים ככל הניתן.
3. דינامي – אם חל שינוי בכמות המשתמשים ברשות, ידע לחלק מחדש את המשאבים בצורה הוגנת.
4. במקרה של עומס על הרשות – שידע להתואוש ממנו.

Slow start – תחילת נשלוח segment אחד ונכחקה לקבל עליו ACK, אם קיבלנו ACK נשלח בפעם הבאה שניים, ולאחר מכן ארבעה וכן הלאה – כך שקצב השילוח שלנו גדל אקספוננציאלית. איך slow starts עובד בפועל? נגיד חalon בקרת עומס – Congestion window – בהתחלה הוא יוגדר להיות 1 ולאחר קבלת כל ACK הוא יגדל ב-1, האפקט שנקבל הוא גידלה אקספוננציאלית בכמות החבילות שניתן לשולח בכל RTT (מכיוון שchlон בקרת הזרימה זו גם הוא, לדוגמה אחרי שקיבלונו שניות להגדיל ב1 בהתחלה, מכיוון שכבר קיבלנו ACK על החבילה הראשונה מזיא אתchlון לחבילה הבאה + גידיל אותו באחד ולכן ניתן לשלוח שתי חבילות חדשות וכן הלאה).



נשים לב שתמיד נלך לפני החלון הקטן יותר מבין חלון בקרת הזרימה וחלון בקרת העומס.

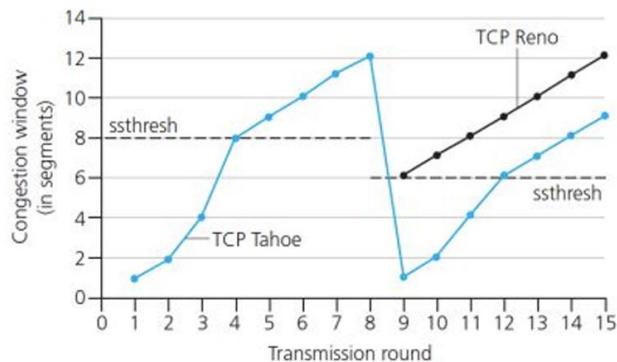
כשמדובר בAIMD (Additive Increase Multiplicative Decrease) – כאשר נרצה להימנע מעומס על הרשת כשמדובר בthreshold (כשמתחלים לשילוח חבילות, הערך שלו מוגדר להיות גדול מאוד, עד לאובדן החבילה הראשונה – ובהמשך נראה איך מתעדכן הערך שלו בהתאם לעומס) נעבד לפי מנגנון AIMD, במקום גידלה אקספוננציאלית בכל RTT נבצע גידלה לינארית – ככלomer לא נגדיל את החלון לאחר קבלת כל ACK, אלא החלון יגדל רק בסיום RTT (של כל החבילות שנשלחו). ישנן שתי שיטות למימוש המנגנון:

1. אם גודל החלון הוא n , נמitten לקבלת כל ACKים על n החבילות שנשלחו, ולאחר מכן להגדיר שגודלו חלון בקרת העומס בRTT הבא הוא $n+1$.

2. אם גודל החלון הוא n , אחרי קבלת כל ACK נגדיל אותו ב $\frac{1}{n}$, כך שבסוף RTT נקבל שגודלו החלון הוא $n + \frac{1}{n} = n + 1$.

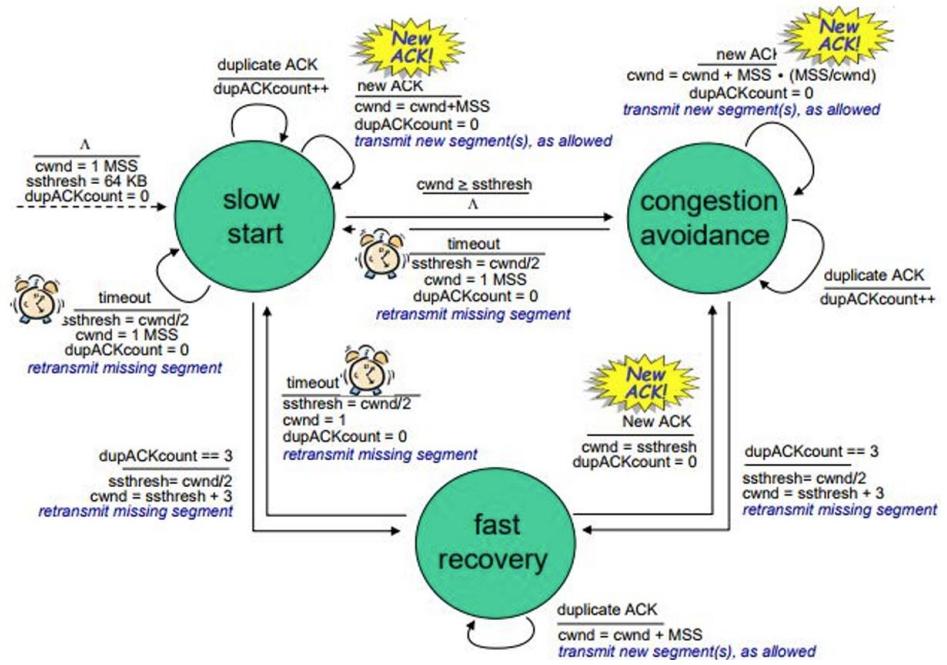
במידה ומזהה אובדן חבילה מכיוון שהרשות עמוסה מדי תחילת נגידר את ה-threshold להיות חצי מגודל החלון האחרון, ולאחר מכן נקטין את חלון בקרת העומס:

- במקרה של $out = 1$ – נקtinן את החלון בחזרה ל-1, ובצע慢 start slow עד שנגיעה ל-threshold, וממנו נעבור לגידלה לינארית באמצעות AIMD. אם אנחנו עובדים בTCP גרסה Tahoe תמיד נקtinן את גודל החלון ל-1 וنمישר כמפורט מעלה (ונגידר את ה-threshold להיות חצי מגודל החלון בו חל $timeout$).
- אם קיבל 3 ACKים כפולים, זהה שהרשות הייתה בעומס רגעי (כלומר החבילות שנשלחו אחריה חבילה שאבדה הצלicho להגיע ולבן כהה"ן מדובר בעומס רגעי) נקtinן את חלון בקרת העומס בחצי, ונמשיך בגידלה לינארית לפי AIMD. אם אנחנו עובדים בTCP Reno נקtinן כך את החלון במקרה של ACKים כפולים (ובמקרה של $timeout$ נפעל כמו ב1).



Fast recovery – בסeno, אם קיבלנו 3 ACKים כפולים, לא נקtinן את חלון בקרת העומס בחצי ישן, אלא "ונפח" אותו בצורה מלאכותית כך שיגדל ב-1 על כל ACK כפול, כאשר נקבל ACK מצטבר על החבילות שנשלחו (כתוצאה מסתימת החור של חבילה שאבדה), נקtinן את החלון לחצי מגודל החלון שהוא (לפנינו לו 1). ההיגיון האחורי זה הוא שאם קיבלנו ACKים כפולים סביר מאד להניח שגם שאר החבילות שהושפנו לו לא הגיעו yet, ואז החלון יקטן בחצי, אך כבר יהיה חבילות חדשות שנשלחו בחלון של חבילה שאבדה הגיעו לעדן, ואז החלון יקטן בחצי, אך כבר יהיה חבילות חדשות בחוץ מהחלון זה (מסורבל – ממליצה לראות את הדוגמה של חממי מסוף הרצאה 4, עשויה סדר).

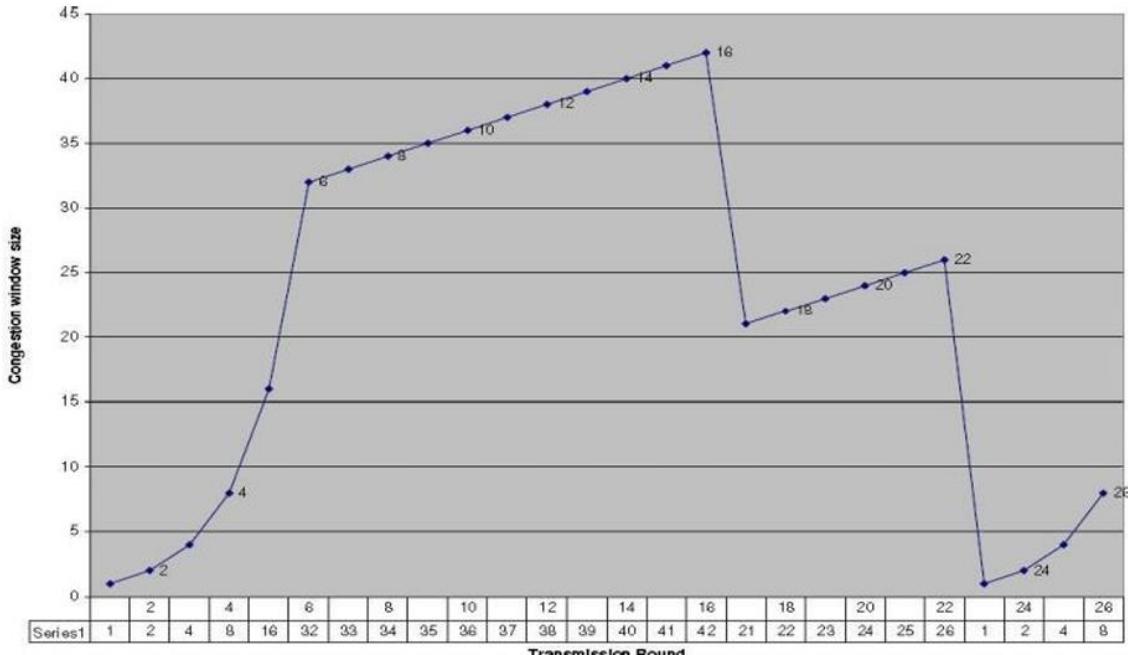
להלן תרשימים שמספרת בדיקות איך פועלים בהתאם לכל מצב בו נמצאים:



תרגול 4:

בדוגמה הבאה השורות למיטה מפרטות את RTT ואת גודל החלון (ציר הX וציר הY בהתאם).

הערה: בדוגמאות הבאות התעלמו מ fast recovery



.[23,26],[1,6] – Slow start

[17,22],[6,16] – Congestion Avoidance

כמו כן נשים לב שנייתן לראות שאנו חנכו עובדים על TCP מסוג Reno שכן החלון קטן בחצי ב-17 RTT – ככלומר קיבלו שלושה ACKים כפולים. בנוסף ניתן לראות שב-22 RTT היה timeout ולכן גודל החלון קטן יותר ל-1 והתחלנו lagiיל שוב אקספוננציאלית ב-slow start.

כמו כן, נשים לב שערך threshold הוא 32. לאחר מכן הוא מוגדר להיות 21 (חצי מגודל החלון בו אבדה החבילה הראשונה). בת-24 RTT גודל threshold הוא 13 (חצי מ-26, גודל החלון בו חלה timeout).

החבילה ה-70 נשלחה בסיבוב 7 ($1 + 2 + 4 + 8 + 13 + 32 = 63$).(1).

נניח ומזה אובדן חבילה לאחר הסיבוב ה-26 ע"י 3 ACKים כפולים, מה יהיה גודל החלון והthreshold לאחר מכן? גודל החלון וגודל threshold יהיו זרים – 4.

שאלה: אליס שולחת לבוב הודהה בגודל 4,400B, כאשר MSS=900B.sequence numbern שאליס בחרה הוא 1,100.

1. לכמה סגמנטים המידע יחולק?
2. ציינו את sequence numbern הראשון והאחרון של כל סגמנט.
3. ציינו את ערכיו ACK שישלח לבוב בחזרה לאלייס.
4. הניחו שחבילות ACK ששולח לבוב אינן מכילות מידע אלא רק תחיליות. לבוב בוחר את sequence numbern שלו להיות 659. מה יהיו ערכיו ACK שאליס תחזיר לבוב כחלק מהמידע שהוא שולחת לו?

תשובה:

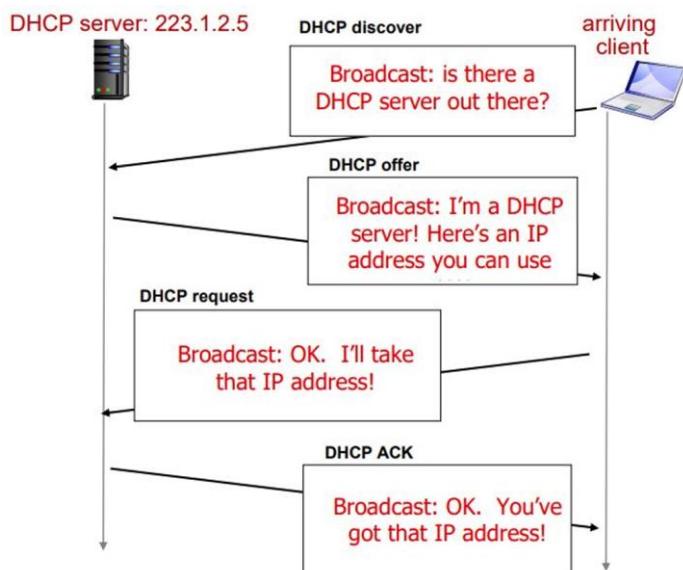
$$1. \text{ המידע יחולק ל } 5 \text{ סגמנטים שכן } \frac{4400}{900} = 4\frac{8}{9}$$

- .(1100, 1999), (2000,2899), (2900, 3799), (3800, 4699), (4700, 5499) .2
.2000, 2900, 3800, 4700, 5500 .3
4. כולם יהיו בעלי ערך 659 (שכן בוב לא שלח כל מידע).

הרצאה 5

DHCP – Dynamic Host Configuration Protocol – פרוטוקול להקצת כתובת IP למחשב שהתחבר לרשת, מאפשר גם לקבל מידע על המחשבים שנמצאים איתך ברשת, לקבל מידע על היעד default gateway וכו'. בשמה של מתחבר לרשת.

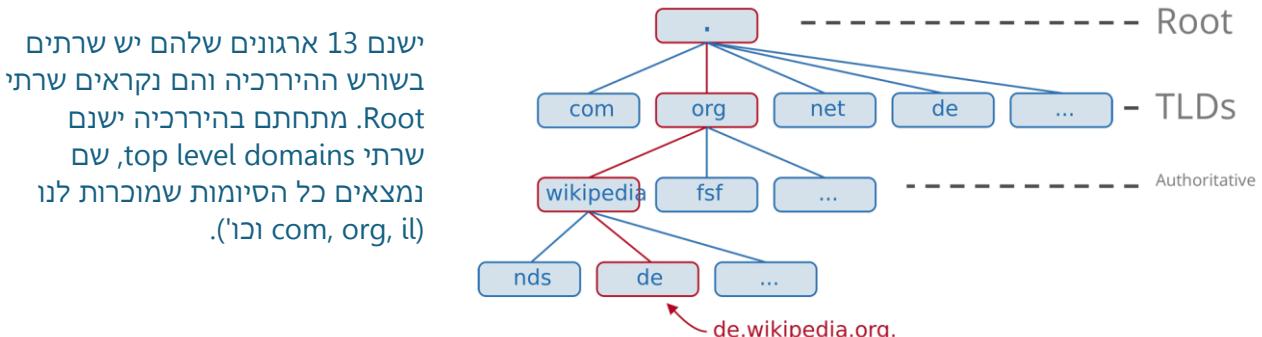
1. הוא מחפש משהו שיוכל לסייע לו ולכ שולח הודעה הודיעת – מהו שירות DHCP של הרשת? הודעה זו נקראת DHCP discover.
2. השירות (או השירותים) מגיבים בה הודעה DHCP offer ומציעים כתובת IP למחשב כדי שייתחבר לרשת (במקרה של כמה שירותי ניתנים לבחור מביניהם).
3. לאחר שהמחשב קיבל את בקשה Offer הוא שולח עדכון לשירות DHCP שהוא מעוניין בכתובת הנ"ל – DHCP request. כמו כן, הבקשה יכולה להכיל רשימה של פרמטרים שאנו רוצים לקבל מהשירות, כמו שירות DNS, כתובת הדומיין, default gateway וכו'.
4. אם הכתובת עדין זמינה לשימוש אצל השירות, הוא מאשר ACK.



DHCP עובד על פרוטוקול UDP שכן אנחנו רוצחים פשוטות ומהירות, אין טעם להקים חיבור כל עוד אין אFILEO כתובת IP.

טיפ לזכור את תהליך ההתחברות – DORA :discover, offer, request, ACK

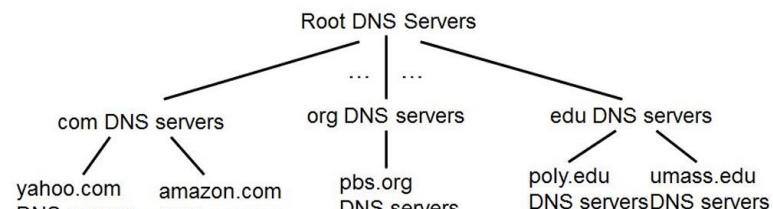
DNS (domain name system) – כמו ספר טלפונים שמקשר בין domain לכתובת IP. גם הוא עובד על UDP מכיוון שהוא צריך להיות מהיר ופשוט. כל שרת יש מיפוי לכתובות שישובות תחתיו או לשירותים שישובים תחתיו / מעליון.



אם נפנה לשרת שנמצא בהיררכיה מעל הכתובת שאנו מחפשים הוא יגאל אחריות עד לשרת הרלוונטי. במהלך התהליך זהה מתבצע תהליך למידה באמצעות cache, כך שבבקשה הבאה שתגיע נדע לנצל את המידע שכבר אגרנו על השירותים.

Resolver – תוכנה שפותרת "חידות" DNS, תבודק קודם אם הכתובת המבוקשת נמצא/cache ואם לא תמשיך את התהליך שפורט מעלה. על מנת שכל התהליך לא יהיה ארוך מדי, יוכביד על שירות DNS, כאשר מבצעים DHCP request בדרכו המחשב יבקש גם את IP של DNS resolver, כך ידע להפנות אליו שירות בקשות DNS.

לדוגמה, פנו resolver בבקשת לקבלת את כתובת IP של resolver.amazon. מכיוון שהזיהוי לא ידע ישר את התשובה (לא הייתה לו/cache) הוא עובר לשרת Root וմבקש ממנו את התשובה, וכן הזריק resolver מוגלала עד לשרת הרלוונטי.

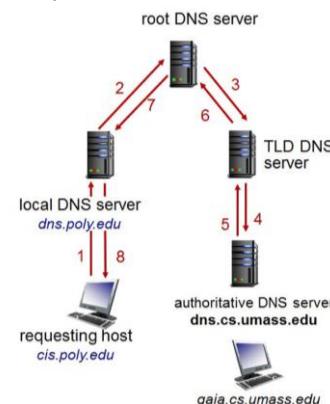


client wants IP for www.amazon.com; 1st approx:

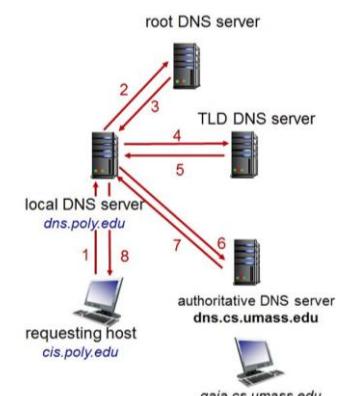
- ❖ client queries root server to find com DNS server
- ❖ client queries .com DNS server to get amazon.com DNS server
- ❖ client queries amazon.com DNS server to get IP address for www.amazon.com

את הפניה של DNS ניתן לבצע באופן רקורסיבי או באופן איטרטיבי:

1. רקורסיבי – פונים לresolver, אם הוא לא יודע את התשובה מעביר לשרת Root שמעביר אותו לשרת TLD וכן הלאה עד למציגים לשרת הרלוונטי, וזה כל הדרך חוזרת מעבירים את IP. Resolver לא מבצע כך תהליך מיידי וכמובן זהה מאוד לא עיל.



2. איטרטיבי – פונים לresolver עם בקשת החזקה domain, והוא מבצע כל פעם את השאלת DNS בהתאם לתשובה שחזקה מהשרת (או מחזיר לבסוף את כתובת IP ללוקוח).



כאשר נרצה לרשום כתובות domain חדשה, נפנה לשירות DNS registrar נשלם לו כמה שצורך, ובתמורה הוא ירשום את המיפוי בין כתובות החו"ן domain שלו והIP לשירות DNS הרלוונטי. לדוגמה אם אנחנו רוצים לרשום את האתר שלנו – networkutopia.com, שירות ה-TLD של com ירשום בטבלת המיפויים שתירשומות:

- .1 NS (name server) לשירות name server networkutopia.com, dns1.networkutopia.com, dns1.networkutopia.com, NS לשירות DNS של networkutopia.com הוא dns1.networkutopia.com כתובות ה-IP בגרסה 4 של השירות name server dns1.networkutopia.com, 212.212.212.1, A לשירות name server dns1.networkutopia.com, 212.212.212.1, A של networkutopia.com הוא dns1.networkutopia.com, 212.212.212.1, A.
- .2 dns1.networkutopia.com, 212.212.212.1, A לשירות name server dns1.networkutopia.com, 212.212.212.1, A של networkutopia.com הוא dns1.networkutopia.com, 212.212.212.1, A.

DNS מבצע הליך של caching, היתרון בכך הוא חיסכון בכמות פניות DNS. החיסרון הוא שאם נשמר כתובות ליותר מדי זמן יתכן שכותבות ה-IP הספיקה להשתנות. לכן בטבלה שמתוחזק DNS לטובה מצוינים הפרטים הבאים בצורה של key-value caching :key-value

- .1 A: כתובות ה-IP בגרסה 4 (או 6).
- .2 NS: שירות השמות של כל כתובות החו"ן domain שמסתימות באותו סימולט (לדוגמה, כל הכתובות שנגמורות בcom).networkutopia.comMX: השירות mail exchange של השירות המייל הקשור לחו"ן domain הרלוונטי.
- .3 CNAME: שמות נרדפים לכתובות החו"ן domain שלו.
- .4

(hypertext transfer protocol) HTTP: באמצעותו מבקשים את המידע שמכיל דף האינטרנט עם כתובות החו"ן resolve שכך עשויה לה. TCP, ש肯 נדרש אמינות. ה-PORT שוגדר לחיבור לשירות HTTP הוא 80.

ישנו מספר סוגי חיבור HTTP:

- .1 Non persistent: על כל חיבור, ניתן לבצע רק בקשה HTTP אחת, לקבל עלייה את התשובה ולסגור את החיבור. בירית המחדל של HTTP גרסה 1.0.
- .2 Persistent: ניתן לבצע מספר בקשות HTTP על כל חיבור שמדובר, אולם חובה לקבל תשובה לכל בקשה לפניו שליחת הבקשה החדשה. בירית המחדל של HTTP גרסה 1.1.
- .3 Pipelining: ניתן לבצע מספר בקשות עד לפניו שקיבלו תשובה לכל אחת. אנחנו לא נניח את זה בקורס, ובדרך"כ גם הדפדףים שאנו עובדים עליהם לא מאפשרים את זה כי זה גורם לביעות.

תרגילים 6+5:

כעת נסביר על TCP sockets. שרת:

```

import socket
IP בגרסה 4 TCP Socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
קובעים באיזה PORT השרת מازין, וכן שהוא מאזין לכל
server.bind(('12345', 12345) כרטיסי הרשות של המחשב (המחורת הרקאה)
השרת יכול להאזין ל5 סוקטים
server.listen(5)

השרות מצין כי הוא מוכן לקבל את
while True: IP PORT של השולח
    הלקוח הבא, יתקע כאן עד שיגיע
    socket client_socket, client_address = server.accept() הקזאה של
    הלוקה הבא. (client_socket, client_address = server.accept())
    print('Connection from: ', client_address)
    שומרים במשנה data את המידע
    data = client_socket.recv(100) שנשלח מהלקוח, בגודל עד 100 בתים
    print('Received: ', data) אין צורך במו"ך recv שכן הsocket של
    הלוקה הוא ייחודי
    client_socket.send(data.upper())

סגורים את הsocket מול הלוקה, כשההsocket הראשי
print('Client disconnected') (server) עדין ממשיך לחיות.

```

ללקוח:

```

import socket
IP בגרסה 4 TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

מקים חיבור מול השרת ('127.0.0.1', 12345) (server)

s.send(b'hello') שולחים לשרת הודעה ('hello')
מקבלים מהשרת תשובה בגודל של כל
data = s.recv(100) היותר 100 בתים
print("Server sent: ", data)

סגורים את החיבור
s.close()

```

תרחישים שונים להרצאת התוכנית:

1. מרכיבים לקוח אחד לנמל השרת, הלקוח ישלח הודעה ויקבל הודעה, יסגור את socket והשרות יפסיק להמתין `.accept()`.
2. אם נריץ מספר לקוחות מול אותו שרת, נראה שלכל לקוח יש PORT אחר (סביר שהם יקבלו PORTים עם מספרים עוקבים/כמעט עוקבים).

.3. אם ננסה להריץ שרת נוסף על גבי המחשב שרצ' בשרת, ולא נשנה את הPORT שלו, נקבלavigation
שגיאה bind (אלא אם עבר ה-timeout של PORT), שכן מערכת הפעלה שומרת את הנתונים של הPORT לזמן מה כדי לוודא שגם לאחר שהתוכנית נסגרה לא זולגים נתונים מהתוכנית.

.4. אם נוסיף לשרת (sleep(10) – ה-kills לא משפיע על שכבת התעבורה! חיבור TCP ממשר' לעובד, ה-kills פועל רק על שכבת האפליקציה. ניתן להמחייב זאת ע"י פתיחת שני ליקוחות ומעבר על התקשרות Wireshark, שם נראה שבזמן ה-kills של הליקוח הראשון כבר מוקם חיבור של הליקוח השני.

```

GNU nano 4.8          tcp_server.py          Modified
import socket, time

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('',12345))
server.listen(5)

while True:
    print('Good night')
    time.sleep(10)
    print('Good morning')

    client_socket, client_address = server.accept()
    print('Connection from: ', client_address)

    data = client_socket.recv(100)
    print('Received: ', data)

    client_socket.send(data.upper())

    client_socket.close()
    print('Client disconnected')

```

.5. נשנה בשרת את השורה של ה-client_socket.send(data_upper()) (כלומר נשלח את ההודעה שהתקבלה באותוות גודלות 10000 פעמים). מצד הליקוח נכפל את ה-s.send 5 פעמים.

```

GNU nano 4.8          tcp_server.py          user@pc1:-
import socket, time

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('',12345))
server.listen(5)

while True:
    client_socket, client_address = server.accept()
    print('Connection from: ', client_address)

    data = client_socket.recv(100)
    print('Received: ', data)

    client_socket.send(data.upper() * 10000)

    data = client_socket.recv(100)
    print('Received: ', data)

    client_socket.close()
    print('Client disconnected')

```

```

GNU nano 4.8          tcp_client.py          user@pc1:-
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('10.0.2.6', 12345))

for i in range(5):
    s.send(b'hello')
    s.send(b'hello')
    s.send(b'hello')
    s.send(b'hello')
    s.send(b'hello')

    data = ''
    while len(data) < 50000:
        temp = s.recv(6)
        print('Server sent: ', temp)
        data += temp

s.close()

```

נשים לב שעל אף של הליקוח שלח חמיש פעמים את ההודעה, השרת ידפיס כבר לאחר הrecv הראשון את חמיש ההודעות – שוב הסיבה לכך היא שיש להפריד בין שכבת האפליקציה לשכבת התעבורה, אמנם הליקוח העביר 5 הודעות עם data של hello, אך בפועל הן הועברו על גבי אותו segment והמתינו בו buffer של השרת (יתכן שהן יועברו גם על גבי segments שונים, תלוי TCP), ולכן כשהוא ביצע recv הוא קרא את כל 25 הבטים שנשלחו מהליקוח. לכן בעת השרת ישיב לליקוח ב-25*10000 בתים, והליקוח יקרא בכל פעם מהbuffer 6 בתים ובסה"כ קיבל את

ההypoזות האלו:

```
user@pc1:~$ python3 tcp_server.py
Connection from: ('10.0.2.7', 45272)
Received: b'hellohellohellohellohello'
Traceback (most recent call last):
  File "tcp_server.py", line 18, in <module>
    data = client_socket.recv(100)
ConnectionResetError: [Errno 104] Connection reset by peer
user@pc1:~$ python3 tcp_server.py
[1] 12345 python3 tcp_server.py
Received: b'hello'
Received: b'hellohellohellohello'
Client disconnected
[1]+ 12345 Terminated                 python3 tcp_server.py
```

המשך שכבת התעבורה מתרגול 4:

프로그램נטציה – כאשר חבילה מגיעה לרכיב תקשורת והיא אגדולה כדי עברו הערוץ שאנחנו צריכים להעביר אותה אליו, מתבצעת פרגמנטציה. ככלומר מחלקים את החבילה לתתי חבילות בצדד שיווכלו לעברן הцентр יותר. פרגמנטציה יכולה להתבצע הרבה פעמים לאורור המסלול, אך הרכבת החבילות מחדש תבוצע רק אצל המקבל. TCP מנסה להימנע מפרגמנטציה ע"י שליחת סגמנטטים נפרדים.

דוגמא: נניח שהעט MTU הוא 1500 בתים, ואנו רוצים להעביר data בגודל של 3972 בתים (אליו יתווסף 8 בתים של header וכן 20 בתים של IP header, כולל בסה"כ 4000 בתים). שכבת הערץ לא יכולה להכיל את הנקודות הנ"ל, ולכן מה שיתבצע זה פרגמנטציה – נחלק את המידע (כולל IP header) ל-3 חבילות וניקח בחישובו שכל חבילה צריכה להכיל גם 20 בתים של IP header. בדוגמה שלנו, מכיוון שישנם 3980 בתים נדרשים להישלח (data + header), כל פרגמנט יכול להכיל 1480 בתים (שכנן הורדנו 1500 בתים 20 בתים של IP header), וכך החבילה הראשונה והשנייה יכילו 1480 בתים של מידע, והחbillה השלישית תכיל 1020 בתים של מידע. ב-IP יהיה כתוב אותו IP שכן מדובר בסופו של דבר באותה חבילה שפורקה למספר חבילות, ובשדה fragflag יהיה כתוב 0 או 1 בהתאם לכך שישנן חבילות נוספות מאותו פרגמנט או לא. בשדה offset יהיה כתוב כמה בתים כבר נשלחו לפני חלקיקי 8 (כולומר מאייה בית הfragmant הנ"ל מתחילה), הסיבה לכך היא שהשדרה הנ"ל הוא 13 בתים ולא 16 בתים.

example:

- ❖ 4000 byte datagram
 - ❖ MTU = 1500 bytes

	length =4000	ID =x	fragflag =0	offset =0	
--	-----------------	----------	----------------	--------------	--

one large datagram becomes several smaller datagrams

1480 bytes in
data field

offset =
1480/8

	length	ID	fragflag	offset
	=1500	=x	=1	=0

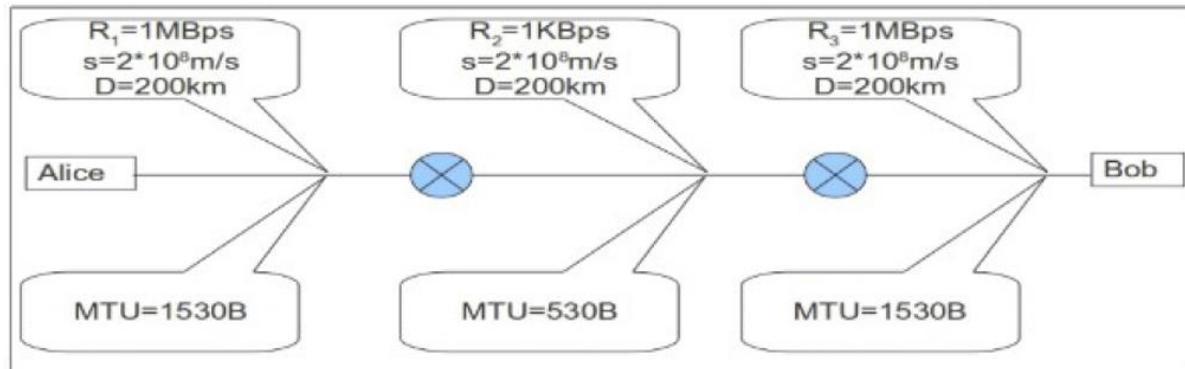
	length	ID	fragflag	offset
	=1500	=x	=1	=185

length =1040	ID =x	fragflag =0	offset =370
-----------------	----------	----------------	----------------

נשים לב שבפרגמנט הולך לאיבוד יש לשЛОח מחדש מחדש את כל הfragments מהם חלק מההודהה שלו – מאוד לא יעיל.

Path MTU discovery: כדי ליעיל את תהליך שליחת החבילות ולהימנע מפירוק שלהן, TCP מפעיל מנגנון לגילוי ה- MTU הקטן ביותר בדרך. תחיליה שלוחים חבילה עם גודל ה- MTU הראשון שידוע לנו כאשר דגל *don't fragment* מופיע. כאשר החבילה תעבור לעורוץ בו ה- MTU קטן מהחביבה, תשלח בחזרה הודעה ICMP שאותה פוטוקול השוארת שהחביבה גדולה מדי ומגדירה מה גודל ה- MTU העדכני. השולח יעדכן אצלו את ערכיו ה- MTU וה- MSS בהתאם, ובמידה ובהמשך הדרכו ישנה עחיצים שגדל זה עדין גודל מידי עבורם נבצע שוב אותו תהליך.

שאלה:



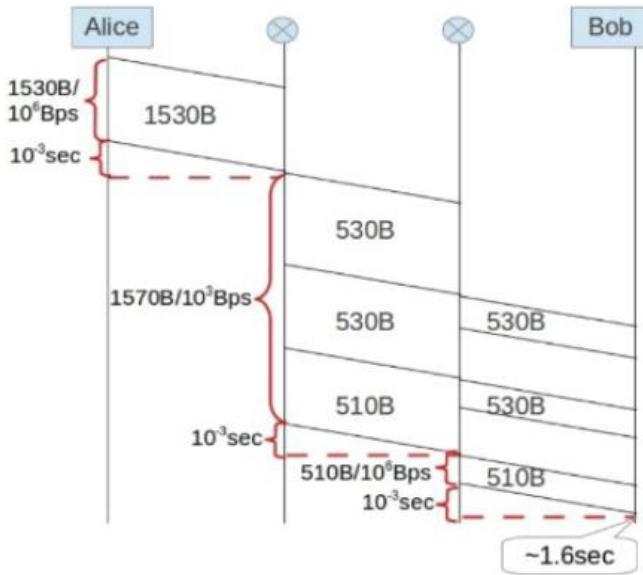
באирו לעיל ייושם במחשב של אליס מסדר חבילה שמכילה 15 דגימות לבוב מעל UDP. גודל כל דגימה הוא 100B הניחו גודל תחילתית UDP של 10B וגודל תחילתית IP של 20B. הניחו כי אליס לא מדילקה את הדגל *don't fragment* ולא יודעת מה ה- MTU המינימלי במלול ממנה לבוב. שאר המידע מופיע באирו. לצורך התשובה נניח כי ניתן לשדר פרגמנטים שהם כמות הנתונים ולא בкопיות של 8 בתים.

1. צירו תרשימים העברת החבילות חמנים, ובפרט חשבו את הזמן עד שבוב קיבל את המידע כלומר את כל 15 הדגימות בהנחה שאין אובדן חבילות. ציינו במדוק את אורכה של כל חבילה.
2. בהנחה שהנתב השני זורק כל חבילה שSEGUE בהסתברות ק:
 - א. מה ההסתברות שהdagima הראשונה תעבור בהצלחה ליישום של לבוב?
 - ב. מה ההסתברות שכל הדגימות יגיעו בהצלחה ליישום של לבוב?

תשובה:

1. העורך האיטי הוא בין שני הנתבים ולכן עורך זה הוא זה שמקטייב את הקצב בו החבילות יתתקבלו אצל מקבל. ולכן נחשב את השהייה התפשטות והשידור של החביבה הראשונה מאليس לנtab, נוסף את השהייה השידור של כל החבילות בין שני הנתבים ואת השהייה התפשטות של הביט האחרון בינהם, ולבסוף נוסף את השהייה השידור וההתפשטות של הביט האחרון בין הנתב השני לבוב.

$$\begin{aligned}
 \text{נחשב: } d_p Alice \rightarrow \text{router1} &= \frac{2 \cdot 10^5 m}{2 \cdot 10^8 mps} = 0.001 sec, d_t Alice \rightarrow \text{router1} = \frac{1530B}{10^6 B} = 0.00153 sec \\
 d_t \text{ route1} \rightarrow \text{route2} &= 2 \cdot \frac{530B}{10^3 Bps} + \frac{510B}{10^3 Bps} = 1.06 + 0.51 = 1.57 sec \\
 d_t \text{ route2} \rightarrow Bob &= \frac{510B}{10^6 Bps} = 0.00051 sec, d_p \text{ router1} \rightarrow \text{router2} = 0.001 sec \\
 d_p \text{ router2} \rightarrow Bob &= 0.001 sec \\
 \text{ובסה"כ: } 0.00153 + 0.001 + 1.57 + 0.00051 + 0.001 &= 1.57504
 \end{aligned}$$



.2

- א. ההסתברות שהדגם הריאיון הגיע בהצלחה ליישום של בוב היא $(k - 1)$ שכן אם אחד מהפרגמנטים לא הגיע ליישום של בוב, בוב לא יוכל להרכיב בחזרה את הפרגמנטים ולכן כל הפרגמנטים ירתקו.
- ב. כנ"ל.

$$\text{היחס בין גודל החולון וקצב השידור: } RTT = \frac{W \cdot L}{R}$$

1. אם $RTT < \frac{W \cdot L}{R}$ אז סיימנו לשדר לפני שהגיע ACK ולאחרנו מוחכים ולא משדרים עד שיגיע ACK.
2. אם $RTT > \frac{W \cdot L}{R}$ אז ACK על החבילה הראשונה מגעתו'כ שידור החבילות בחולון, ולכן כאשר אנחנו מסיימים לשדר את החולון הראשון נתחיל לשדר מיד את החולון השני ללא בזבוז זמן מת. בטעות, מכיוון שגודל החולוןTCP מתחילה בגודל חבילת 1, בחולון הראשון תמיד RTT יהיה גדול יותר מזמן השידור של החולון. נרצה לדעתה החולון הראשון RTT כבר לא גדול יותר מאשר זמן השידור של החולון. כאשר אנחנו נמצאים בstate slow, בחולון זה, אנחנו משדרים d_t^{i-1} בתים, ולכן בהינתן שאלה מסוימת, נפתרו את אי-השוויון הבא: $RTT \geq d_t^{i-1} \cdot 2$. וכך עד החולון הזה אנחנו משדרים בחולון שקטן ממה RTT והחל מחולון זה החולון כבר גדול ממה RTT .

שאלה: כמה זמן ייקח לשולח קובץ בגודל 122,000B כאשר נתנו:

1. גודל חבילת 1,500B
2. גודל התחלילות הוא 40B
3. קצב השידור הוא 12Mbps
4. השהיית ההתפשטות היא 12ms

- א. הציגו את החישוב כאשר משתמשים בstop and wait.
- ב. הציגו את השימוש כאשר משתמשים בpipelining עם 8 חבילות בחולון.

תשובה:

- א. נחשב כמה חבילות נדרש לשולח $\frac{122 \cdot 10^3 B}{(1500 - 40)B} = 83 \frac{41}{73}$. קלומר צורך לשולח 84 חבילות, אשר החבילת האחרונה בגודל 820B.

$$\begin{aligned}
 \text{משך הזמן שלוקח לשולוח חבילת אחת: } & d_t + d_p = \frac{8 \cdot 1500b}{12 \cdot 10^6 bps} + 12 \cdot 10^{-3} = 0.013 \\
 \text{משך הזמן שלוקח לשולוח את החבילת האخונה: } & d_t + d_p = \frac{8 \cdot 860}{12 \cdot 10^6 bps} + 12 \cdot 10^{-3} = 0.012573 \\
 \text{משך הזמן שלוקח לשדר ACK: } & d_t + d_p = \frac{8 \cdot 40b}{12 \cdot 10^6 bps} + 12 \cdot 10^{-3} = 0.012027 \\
 \text{ובסה"כ נקבע: } & .83 \cdot 0.013 + 0.012573 + 84 \cdot 0.012027 = 2.101841 \text{ sec} \\
 \text{ב. נבדוק את היחס } & \frac{\frac{W \cdot L}{R}}{RTT} ? \\
 \text{מכיוון שהRTT של חבילת ה-ACK הוא } & \frac{W \cdot L}{R} = \frac{8 \cdot 8 \cdot 1500b}{12 \cdot 10^6 bps} = 0.008 \text{ sec} \\
 \text{כלומר RTT } & \frac{W \cdot L}{R} < RTT. \text{ ככלומר, ה-ACK מתקבל לאחר שנשלחו 8 החבילות בחלון ולכן נקבע שישנים 11} \\
 \text{RTTים, אליהם צריך להוסיף את השהייה השידור של כל החבילות שנשלחו בחלון האחרון, מלבד} \\
 \text{החבילה הראשונה (שהיא נספירה כבר ב-RTT ה-11), ולכן נקבע:} \\
 & \frac{1460B + 40B}{12 \cdot 10^6 bps} + \frac{820B + 40B}{12 \cdot 10^6 bps} = 0.2779 \text{ sec}
 \end{aligned}$$

Silly window syndrome – האפליקציה עלולה לגרום לנזק בקצב איטי ביחס לקצב בו המידע מגיע למחשב (בקרט הזרימה תודא שלא ישלחו יותר מידע הזיכרון בכדי שלא יירק). הבעיה – עלולה להיווצר תופעה בה כמות המידע בשלוחים בכל חבילה הולכת וקטנה, מה שגורם לכך שאנו נושא שלוחים הרבה תחיליות. בעיה דומה יכולה להתறחש גם במקרה הפוך – כאשר האפליקציה בצד השולח שלוחות הרבה מאוד מידע אבל בחתיות קטנות (אם TCP ייכחה מעט הוא יוכל לחבר חתיות קטנות לחבילת אחת).

הדריכים למניעת Silly window syndrome:

1. המקלט לא יפרסם חלון קטן מדי, لكن אם גודל החalon קטן MSS 1 המקלט יפרסם חלון בגודל 0 (כלומר יחסום את השולח מלשלוח לו עוד חבילות), כאשר יתרונה מקום בצד המקלט הוא ישלח ACK עם גודל חלון עדכני / לחלוין השולח ימתין מעט וינסה לשולוח שוב מעת חבילות בגודל חלון סביר.
2. מניעה בצד השולח מתבצעת באמצעות אלגוריתם Nagle: אם אין מידע שימושיים ל-ACK עליון, שלח את המידע שהאפליקציה רוצה לשולח. אם יש מידע שימושיים ל-ACK עליון – נעכט בקשורת שליחת האפליקציה אם הן קטנות מדי עד שנתקבל את ACK או עד שיצטבר מספיק מידע, ככלומר לפחות MSS.

שאלה: אפליקציה מסויימת מתקשרת בעזרת ערוץ לוויי על גבי מרחק של 60,000km כאשר באמצעותם מהירות התפשטות היא 300,000kmps, ככלומר השהייה התפשטות היא 0.2sec. נניח שלמקלט יש buffer בגודל 200,000B.

1. נניח שקצב השידור הוא 10Mbps, והאפליקציה קוראת בקצב קבוע של 8Mbps כיצד ניתן לשפר את קצב העברת המידע?
2. מהו החסם העיקרי על גודל buffer שיכל להיות מפורסם?

תשובה:

1. נשים לב שעל כל 10MB buffer, 8MB נקראים ממנו ע"י האפליקציה. מכיוון שיש לנו buffer בגודל 200,000B, כנסנלה תחילת את ה-200,000B נקלט buffer ישנו 80,000B שעד לא נקראו, מכיוון שהזמן שייקח להחזיר ACK ע"י הלוין הוא עצום, בזמן זהה buffer כבר יתרוקן – ככלומר הניצולות כאן לא טובה בכלל. לכן הלוין יפרסם גודל חלון אגרסיבי בהתאם, שמחושב ע"י הנוסחה $\frac{W}{1 - \frac{R}{S}}$, כאשר R קצב השידור, S קצב הקראיה של האפליקציה מהbuffer, W גודל buffer בפועל. לכן, על מנת לשפר את קצב העברת המידע, הלוין יפרסם חלון גדול יותר מהbuffer שלו.
2. נחשב: $\frac{W}{1 - \frac{R}{S}} = \frac{200000B}{1 - \frac{8Mbps}{10Mbps}} = 10^6 B$

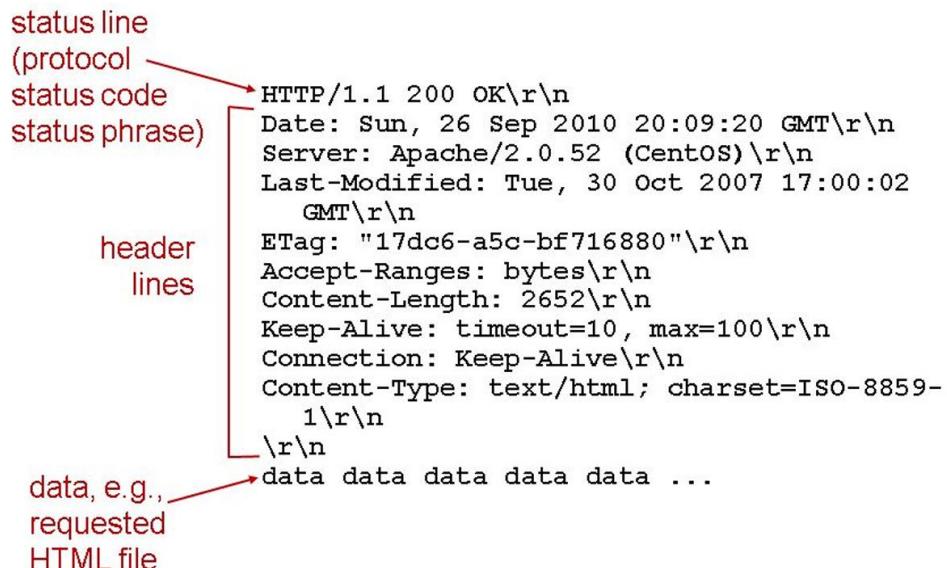
הרצאה 6

דוגמאות לבקשת HTTP



נשים לב שלאחר **header**ים תמיד ישנה שורה ריקה.

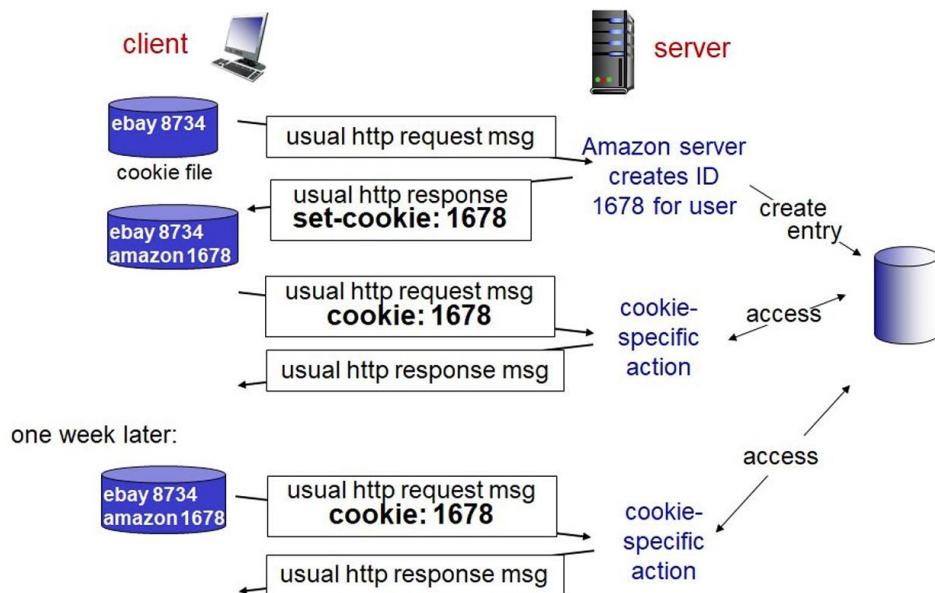
דוגמאות לתשובה HTTP



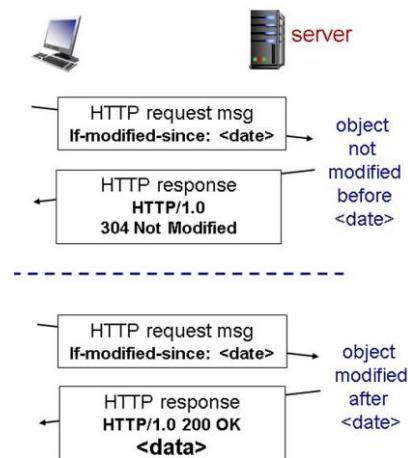
רשימת הסטטוסים בHTTP

- .1 200 – הבקשה הצלילה, פירוט התשובה בהמשך.
- .2 301 Moved Permanently – האובייקט המבוקש כבר לא נמצא היכן שביקשת, המיקום הנוכחי הוא: ...
- .3 400 Bad Request – הבקשה לא הובנה ע"י השרת.
- .4 404 Not Found – לא נמצא תיעוד לאובייקט המבוקש בשרת זה.
- .5 505 HTTP Version Not Supported

Cookies (קוקיזות) – השרת לא זוכר בעצמו כל לקוחות שייצר אותו קשור, ובשביל זה קיימות קוקיזות. כאשר פונים לשרת, יתכן והוא ייחזיר לנו עם התשובה cookie שחייבת כלשהי, ויגדר לו למשך אליו פרק זמן לשמרו אותה. בנוסף, בדרך כלל הclient תאגיר אצל הלוקו מידע שהשרת מעוניין בו, וכך בפעם הבאה שלлокו יתחבר לשרת (ובתנאי שהclient שומרה עד שמורה אצל הלוקו), הלוקו יפנה לשרת בצריך הclient ו_az השרת יכיר את הלוקו. יכול לשמש בעיקר לטובות טיבן עגלות קניות, המלצות על סמרק נתוני הגלישה, כניסה מהירה לחשבון וכו'.

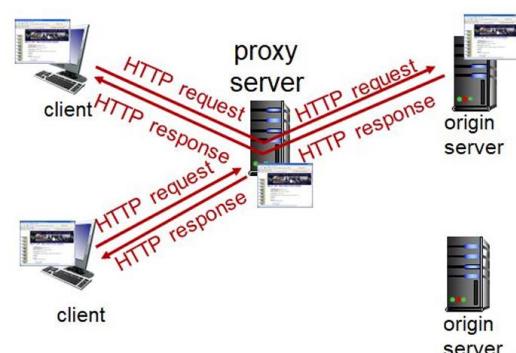


Caching – מכיון ששרת HTTP מודעת ללקוחות מטבעם, אם יבוצע caching אצל הלוקו זה יחשב הזמן ופניות לביקשות חדשות אצל השרת. וכך הלקוח ישמר אצל cache last modified של המידע, בעת פניה לשרת HTTP הלקוח יזכיר מה המידע שיש אצל cache ולמתי הוא עדכני, ואם השרת יראה כי אין אצלו עדכון לקובץ הוא יגיב ב304, כך שהלקוח יוכל להשתמש בגרסה שנמצאת אצל cache.



שרת Proxy – תפקידו הוא לעשות פעולות בשם הלקוחות שלו. למשל להציג לאילו אתרים מותר לגלוש, אילו פעולות חוקיות וכו'. בנוסף, התקשרות ככה נתיה יותר מהירה כי סביר שהשרת ישמור/cache פעולות שבוצעוות תכופות ע"י הלקוחות שלו.

בדוגמה ניתן לראות שהלקוח הראשון בבקשת HTTP לשרת proxy. לשרת אין אותו/cache פונה ולכן הוא פונה לשרת המקורי (חיבור TCP אחר!), מקבל את תשובתו, שאותה/cache ולאחר מכן מgive ללקוח שלו עם HTTP הרטלוני. לאחר מכן, לקוח שני פונה עם אותה בקשה HTTP, ומכיון שהוא כבר שומרה/cache של proxy, הוא עונה אותה ישר מבלי לפנות לשרת המקורי.



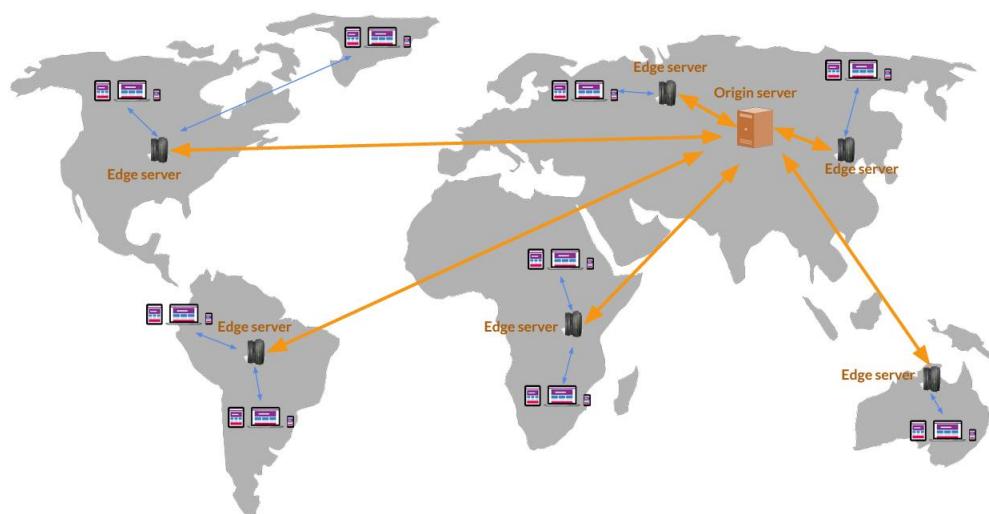
סוגי שרתים proxy:

1. Proxy רגיל – מטרתו לעזור ללקוחות, מבצע עבורות caching, מנהל לוגיקה של שמירה על ביטוחן הלקוחות שלו, פרטיותם וכו'. הוא דרש מהלקוח לבצע התאמה בדףו שלו.
2. Proxy שקוֹף – על אף שלדף לא הוגדר לעבוד מול שרת proxy, בפועל מה שקרה הוא שהוא גולש מול שרת proxy.
3. Proxy הפוך – נועד לשרת את השירותים ולא את הלקוחות, כלומר proxy הוא של השירות ולא של הלקוח.

כיצד לחלק לקוחות בין שירותים זחים? המטרה שלנו היא להפחית עומס מכל שירות ולכן נרצה חלוקה יחסית הוגנת של לקוחות בכל שירות. כמה אפשרויות לחלוקת זו:

1. אלגוריתם סטנדרטי, לפיו כל לקוח חדש מוקצה לשרת הבא בתור, כאשר מגיעים לאחרון חחרים להקצות מהראשון.
2. אלגוריתם שמנפה בין כתובת IP של לשרת הרלוונטי, על סמך פונקציה שנגדר. היתרון של שימוש באלגוריתם זה היא שבכל פעם שהלקוח יפנה לקבלת השירות, תמיד הוא יפנה לאותו השירות (מכיוון שיש לו את אותה כתובת IP).
3. אלגוריתם שמנפה ללקוחות על סמך העומס בשירותים, הלקוח הבא יפנה לשרת הכى פחות עומס.
4. שימוש DNS לטובת קבלת IP של השירות בו אנחנו מעוניינים, DNS יחשב עבורנו איזה שירות מתאים לנו על סמך עומס, קרבה וכו'.

– ישנן כל מיני חברות שנותנות שירותים זה, הן מפוזרות ברחבי העולם (content distribution network CDN) – שרת קצה (edge server) כך שהיה קרובים ללקוחות, לרובם ממוקמים ממש היכן שנמצאים השירותים של ספקית האינטרנט שלנו. מדובר בשירותים חזקים מאוד הן תוכניתית והן חומרתית. וכאשר לקוח ורוצה לפנות לשרת של חברה מסוימת, הוא יפנה אל שרת CDN הקרוב ביותר אליו, אם התשובה נמצאת אצליו הוא יקבל תשובה הרבה יותר מהירה מאשר אם היה צריך לפנות בעצמו לשרת המקורי, שיתכן ונמצא מעבר לאוקיינוס.



הרצאה 7:

SMTP (simple mail transfer protocol) – מדובר בפרוטוקול שמאפשר העברת מיילים בין משתמשים, מזכיר תהליך שליחת מכתבים פיזיים בדואר. מספר מונחים רלוונטיים ל프וטוקול:

1. User agent – התוכנה שבעזרתה קוראים וכותבים את המיילים (לדוגמה gmail).

2. Mail server – השירות שמולו מתנהל ה-user agent. הוא מחזיק את תיבת הדואר הנכנס של לקוחותיו וכן את תוכן הדואר היוצא שלהם. אם הלקוח אליו שלחתי מייל לא משתמש באותו שירות מייל, על שירות המail שלי להעביר את המייל לשרת הרלוונטי.

כדי לשלוח מיילים "ולדחוף" אותם לשרת המייל, נשתמש בSMTP. בכדי להוריד מיילים מהשרת, נשתמש באחד משני פרוטוקולים שיפורטו בהמשך.

כעת, איך שירות המail שלי יודיע למשהו שכתובת המייל שלו נגמרה בzo.biu.ac.il? הוא מבצע שאלחת DNS על מנת להבין של מי החומר, ככלומר מבצע שאילתת MX לשרת DNS.

דוגמא: ננich ואלייס ובוב אינם משתמשים באותו שירות מייל.

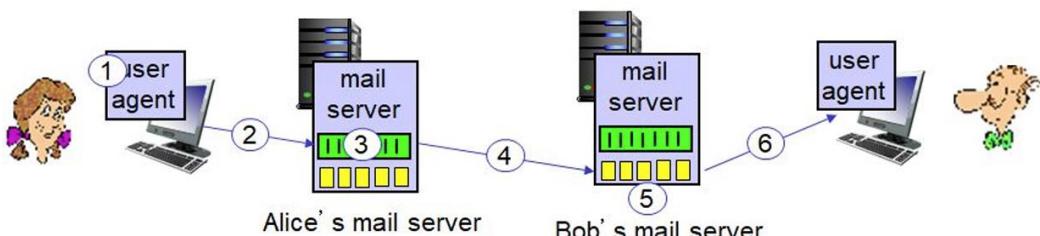
1. אליס משתמש בתוכנת user agent ל佗ת שליחת הודעה [�bob@someschool.edu](mailto:bob@someschool.edu).

2. הודעה של אליס מועברת לתור ההודעות היוצאות של שירות המייל שלו (אם היו משתמשים באותו שירות ההודעה שלה הייתה נמחפת לטיבת הדואר הנכנס של בוב שנמצאת בשרת, ולא מועברת לדואר היוצא של השירות שלו).

3. השירות של אליס מקיים חיבור TCP מול שירות המייל של בוב, ושולח לו את ההודעה של אליס.

4. השירות של בוב שומר את ההודעה של אליס בדואר הנכנס של בוב.

5. בוב מושך את הדואר שקיבל מטיבת הדואר הנכנס.



דוגמא לתקשורת SMTP (נראה רק את שכבות האפליקציה, מהורי הקלעים מתבצע hand shake וכו'):

S: 220 hamburger.edu - הצלחה, השירות מציג את עצמו

C: HELO crepes.fr - הצלחה, השירות ממתין

S: 250 Hello crepes.fr, pleased to meet you - בקשת הלקוח

C: MAIL FROM: <alice@crepes.fr> - הלקוח רוצה לשלוח מייל מאליס

השירות מאשר שהוא הבין שיש מייל מאליס

C: RCPT TO: <bob@hamburger.edu> - הלקוח מציין שהנושא הוא בוב

S: 250 bob@hamburger.edu ... Recipient ok - השירות מאשר שהוא הבין

C: DATA - הלקוח מציין שמעתה והלאה הוא יעביר לו את תוכן המייל

S: 354 Enter mail, end with "." on a line by itself - השירות מאשר, ומציין כי בסוף

C: Do you like ketchup? - המail צריכה להיות נקיודה

C: How about pickles? - בשורה ריקה

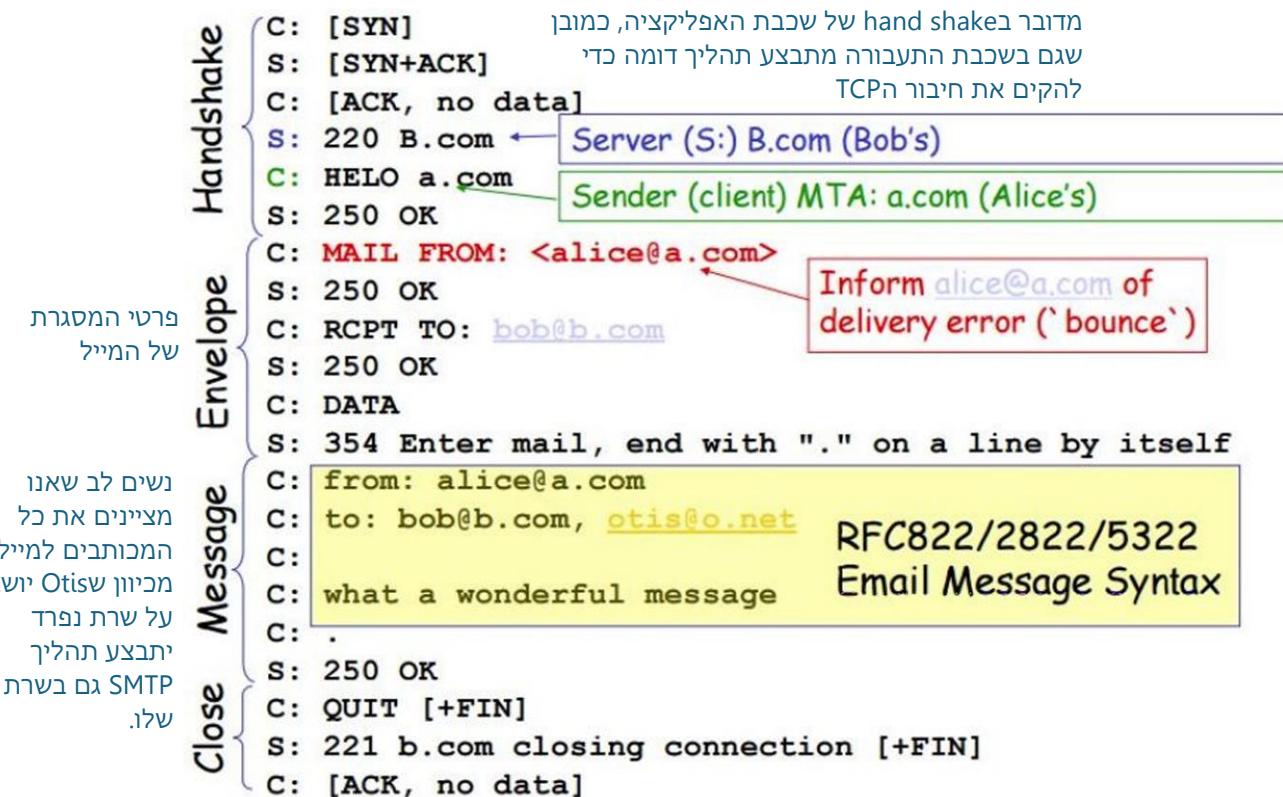
C: . - סוף התוכן .

S: 250 Message accepted for delivery - השירות מאשר שהמייל התקבל אצלנו

C: QUIT - אם ללקוח היה מייל נוסף לשילוח במקום quit הוא יכול להתחיל בתהליך מחדש

S: 221 hamburger.edu closing connection - השירות סגור את החיבור (ובשכבות ה-TCP)

דוגמיה נוספת:



דוגמיה נוספת:

תווית: מכיוון שכאן שאר המcotובים נמצאים על אותו שרת מייל נציגו את הנמענים אחד אחרי השני

```

S: 220 sponge.com
C: HELO A.com
S: 250 OK
C: MAIL FROM: <alice@a.com>
S: 250 OK
C: RCPT TO: manager@sponge.com
S: 250 OK
C: RCPT TO: bob@sponge.com
S: 250 OK
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: from: alice@a.com
C: to: bob@sponge.com
C:
C: I _still_ did not receive the goods you promised
C: .
S: 250 OK
C: MAIL FROM: <alice@a.com>
S: 250 OK ..... .

```

תווית: Notice also multiple RCPT TO (one per recipient, all in same domain)

תווית: אין quit כי הלקוח מעוניין לשלחו מייל נוסף דרך הרשת

כיצד מושכים מיילים מהטיב הדואר הנכנס בשרת? ישם שני פרוטוקולים:

.1 – מזכיר תא קולי: Pop3

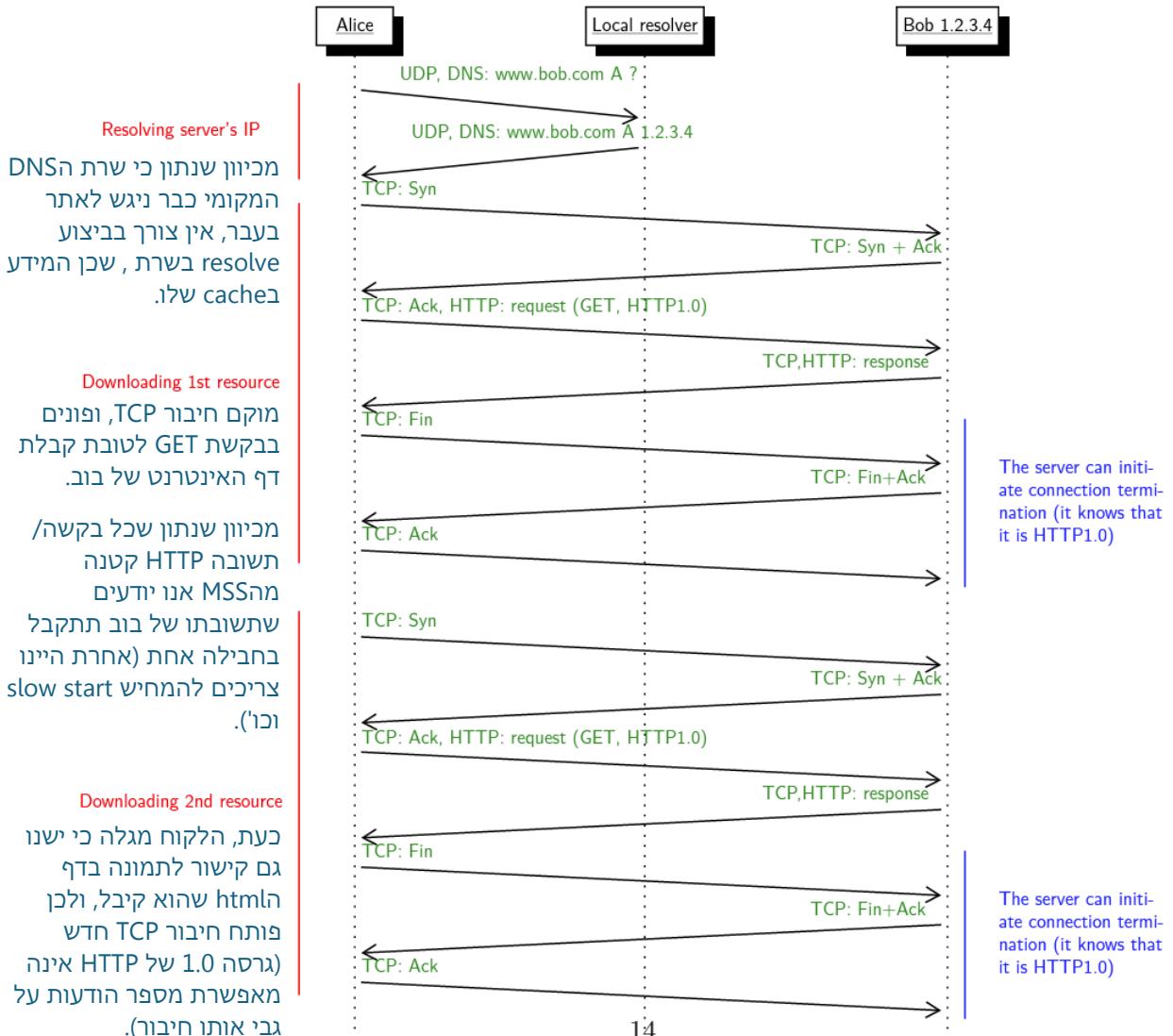
שלב אימות זהות	S: +OK POP3 server ready C: user bob S: +OK C: pass hungry S: +OK user successfully logged on
רשימה של המailים שהתקבלו וגדלים	C: list S: 1 498 S: 2 912 S: .
הלקוח מבקש לקבל את הودעה מס' 1	C: retr 1 S: <message 1 contents> S: .
הלקוח מבקש למחוק את הודעה מס' 1	C: dele 1 C: retr 2 S: <message 1 contents> S: . C: dele 2 C: quit S: +OK POP3 server signing off

.2 – שומר את כל ה הודעות בשרת, מאפשר חלוקת ה הודעות לתיקיות: IMAP

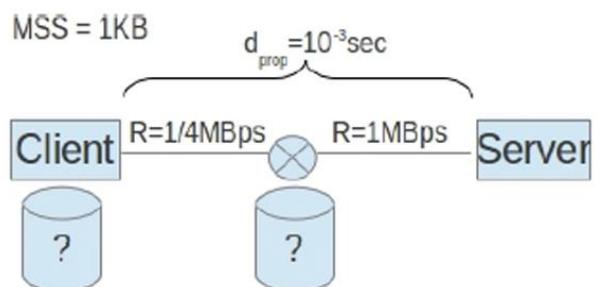
תרגול 7:

שאלה: דפדפן במחשב של אליס רוצה לגלוш לאינטרנט של בוב. עמוד האינטרנט המודבר מכיל גם תמונה אחת. נניח שהדפדפן של אליס מعلوم לא ניגש לאתר של בוב, אבל שרת DNS המקומי של אליס דוקא כן. כמו כן, נניח שהדפדפן של אליס עובד ב프וטוקול HTTP 1.0. הניחו שככל בקשה או תשובה HTTP קטנה מה-MSS. הראו תרשימים הודיעות מתאים.

תשובה: מכיוון שאليس עובדת בגרסת HTTP 1.0 מדובר בחיבור non-persistent.



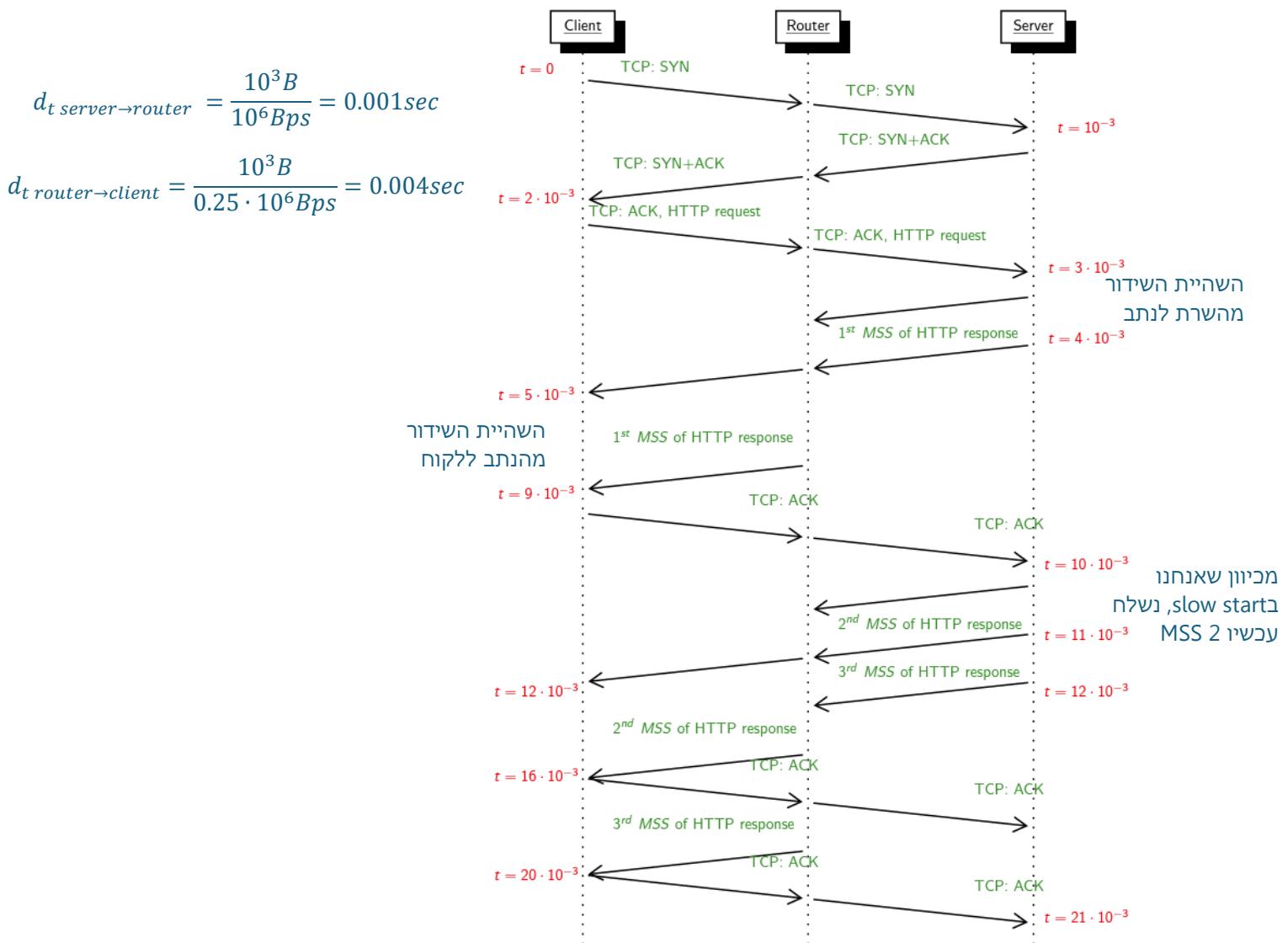
שאלה:



ברשת באירן לעיל לקוח מחובר (מחובר פיזית, לא הטענו דרך hand shake (חיבור TCP 1.0), ובשרת קובץ בגודל 3KB. אין delayed ACK's. ניתן להתעלם מתחילה. כמו כן, התעלמו מהשהייה השידור של בקשת HTTP(.).

ציירו תרשימים העברת הדעות לקובץ עד שהשרת יודע כי הקובץ התקבל בלקוח (בהתבהה שהחוצץ בנתב לא מוגבל) וחשבו את זמן העברת הקובץ.

תשובה: מכיוון שנתנו שניות להתעלם מתחילה – ניתן להתעלם מהשהייה השידור שלהם, אולם לא ניתן להתעלם מהשהייה התפשטות שלהם. בהמשך לכך, ניתן להתעלם מהשהייה השידור של בקשת HTTP (.ולא מהשהייה התפשטות!). כמו כן, מכיוון שהשאלה מבקשת לתאר תרשימים עד שהשרת יודע שהקובץ התקבל, אין צורך להראות בתרשימים גם את סגירת החיבור.



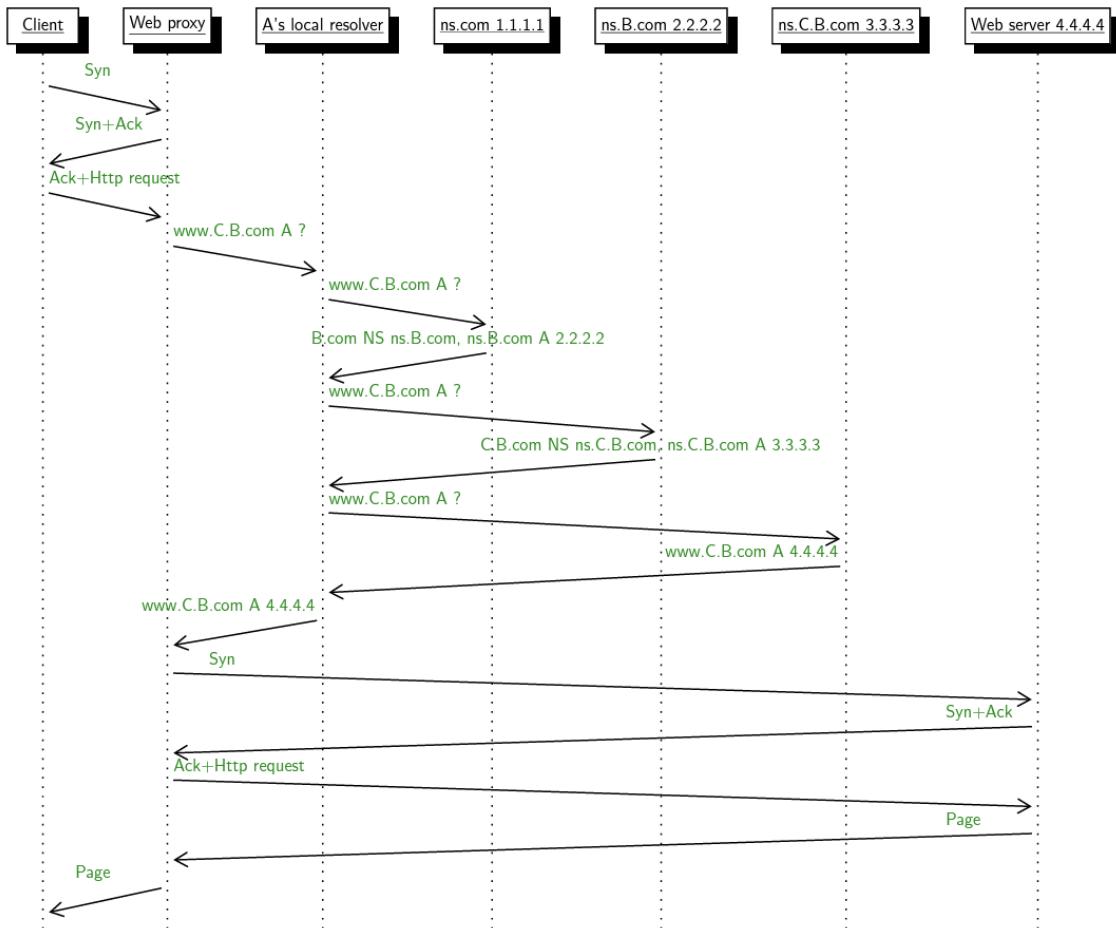
שאלה: לקוח נמצא ברשת A ורואה להוירד את עמוד הבית של www.C.B.com. ברשת A ישנו שרת proxy, אך הנינו שהדף אינו נמצא בשרת proxy. כמו כן, הנינו שהcache בשרת DNS resolver המוקומי של A ריק, למעט המיפוי בין כתובת IP לבן שרת השמות שאחראי על החזאים תחת .com.

1. הציגו דיאגרמת זמנים מתאימה.

2. בהמשך לשאלת הקודמת, כמה זמן עבר עד לתשובה בהנחה שהdelay בין שתי תחנות קצר באינטראנט הוא קבוע בגודל 500ms, העד בינהו תחנות קצרה באוטה רשת הוא קבוע בגודל 50ms, וכל הودעה מספיקה חבילה 1.

תשובה:

.1



תחליה מקימים חיבור עם שרת proxy, בעת שרת proxy בודק אם הדף נמצא בcache ומגלה (לפי נתוני השאלה) שלא. שרת proxy פונה לlocal resolver של A וմבוקש את כתובות IP של www.C.B.com, מכיוון שבזיהוי domain על החומרה com, הולך local resolverן פונה לח'ן של com ומבקש את הכתובת www.C.B.com.Ns של com מפנה את הכתובת local resolverן לשרת local resolver של B.com (נשים לב שאנו פונים לאופן איטרטיבי ולא רקורסיבי). הפונה בעת ל-B.com ומבקש את הכתובת www.C.B.com לשרת B.com. www.C.B.com מפנה את הכתובת IP של www.C.B.com לשרת השמות של com. ולבסוף מקבלים מ-www.C.B.com את הכתובת IP של www.C.B.com. ישן 10 חבילות שנשלחות באינטראנט ועוד 6 חבילות באוטה הרשת וכן נקבע: $.6 \cdot 50ms + 10 \cdot 500ms = 5300ms = 5.3sec$

שאלה: משה לוחץ על קישור כלשהו בדף. נתון שכתובות IP של החוות domain שמופיע בקישור אינה שמורה בcache של משה, ולכן משתמשים DNS. הניחו שתהlixir הדרוש DNS פניה לח'ן DNS וכל פניה שזכה ל- RTT_i זמן. כזכור שאלה + תשובה משרת DNS ה- i לקחה RTT_i שניות. הקישור שמשה לווח עליו מהירות עמד html קטן (נכנס בחבילה אחת) על גבי HTTP ללא הפניות למשאים נספחים. זמן RTT בין המחשב של משה לשרת מוגדר בתור T_0 . חבילות בקרה כמו SYN ו-ACK הינן חבילות לכל דבר ועניין ואין להתעלם מהן. שימושם לבשין השוואות נוספות נספחו מעבר למתחואר בשאלת, חמני

השידור וההתפשטות מובטאים באמצעות ערכי RTT שהוגדרו בשאלה.
כמה זמן לוקח מהרגע שימושה לחץ על הקישור ועד שמשה קיבל את העמוד?

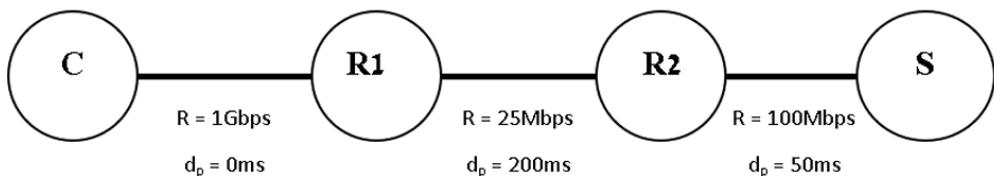
תשובה: יש ח שרתי DNS שהינה צריכה לפןות אליהם, זהה לכך $RTT_n + RTT_2 + \dots + RTT_1 + RTT_0$. בעת בתwichis לחיבור TCP, שליחת SYN וקבלת ACK עליו לוקח RTT_0 ושליחת ACK עם הבקשה וקבלת התשובה לוקח עד $RTT_0 + \sum_{i=1}^n RTT_i + 2 \cdot RTT_0$. ובזה"כ: $RTT_0 + \sum_{i=1}^n RTT_i + 2 \cdot RTT_0$. חמיה ציין שהשאלה הופיע יותר מפעם אחת ב מבחנים, צריך לשים לב שהמטרה של כל המיל הוא לנסות להפיח, כשבפועל התשובה מאוד פשוטה.

שאלה: לךוח ושרות מחוברים דרך שני נתבים. קצבי השידור וההתפשטות כתובים באיזור מטה. גודל חבילות שנשלחות על גבי ערוצים מהשרות ללקוח הוא 20Kb. הלוקוט מבקש מהשרות להוריד קוובץ (נתעלם מזמן השידור וההתפשטות של הבקשה של הלוקוט).

1. כמה זמן ייקח להעביר חבילה בודדת מהשרות ללקוח על גבי UDP? הניחו שאין תעבורה אחרת ברשות.

2. מהו הקצב המירבי בו השרות יכול להעביר מידע אל הלוקוט?

3. בעת, נניח שR1 מתנהג כשרת cache עבור הלוקוט. ככלומר, כאשר הלוקוט שולח בקשה דרכו אזי R1 קודם בודק אם התשובה נמצאת כבר(cache שלו). אם כן, הוא מוחזר אותה ללקוח ללא צורך לשילוח הودעה לשרת. אחרת, מעביר את הבקשה לשרת ומתנהג כמו שמדובר בסעיפים הקודמים. נתון שבהתשובות 60% מידע שהлокוט נמצא cache של R1. בהנחה שלatkות יש הרבה בקשנות, מהו הקצב הממוצע של מידע שהлокוט מקבל?



תשובה:

$$d_{t\ server \rightarrow R2} = \frac{20 \cdot 10^3 b}{100 \cdot 10^6 bps} = 0.0002sec$$

$$d_{t\ R2 \rightarrow R1} = \frac{20 \cdot 10^3 b}{25 \cdot 10^6 bps} = 0.0008sec$$

$$. d_{t\ R1 \rightarrow client} = \frac{20 \cdot 10^3 b}{10^9 bps} = 0.00002sec$$

$$. d_{E2E} = 0.0002 + 0.05 + 0.0008 + 0.2 + 0.00002 = 0.25102sec$$

2. כגדל הערזן המינימלי – **25Mbps**.

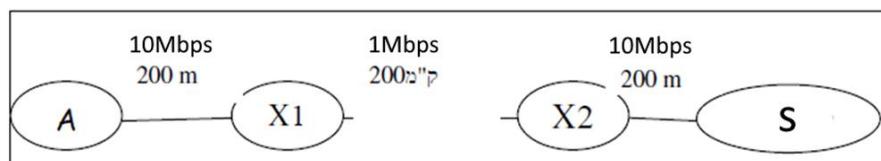
נשים לב שב% 60 מהפניות התשובה ללקוח כבר הייתה/cache של R1 ולכנן מיד חזרנו עם תשובה, ואילו ב% 40 מהפניות נאלכנו להגעה עד השרת, וכבר אמרנו שהקצב המירבי של השרות הוא 25Mbps ולכנן זה"כ: $0.6 \cdot 25 \cdot 10^6 bps + 0.4 \cdot 25 \cdot 10^9 bps = 610Mbps$

שאלה: מחשב A מחובר לשרת אינטרנט דרך שני נתבים כמפורט בתרשימים מטה. A שולח בקשה לדף שבו יש אובייקט 1. גודל בקשה 900B וגודל האובייקט 1400B. קצב התפשטות בכל ערוץ ומשתמשים ב 1.1. HTTP. MTU=1500B. הניחו TCP 1.0. MTU=200,000kmps וזמן שידור של תחילותות.

1. כמה זמן ייקח עד שהדף והתמונה יגיעו לא?

2. מה ייריה השינוי המשמעותי אם MTU בין הנתבים יהיה 1000? הניחו TCP יידע מראש את MTU המינימלי.

3. מה יהיה השינוי אם משתמש ב 1.0 HTTP?



תשובה:

$$1. \text{ נחשב שהיית התפשטות כוללת: } d_p E3E = \frac{200m}{2 \cdot 10^8 mps} + \frac{200 \cdot 10^3 m}{2 \cdot 10^8 mps} + \frac{200m}{2 \cdot 10^8 mps} = 1.002ms \text{ מכיוון שניית להעלם מזמן השידור של תחיליות, נחשב את זמן } hand shake \text{ של } hand shake = 2 \cdot d_p = 2.004ms$$

$$\text{כעת נחשב את זמן בקשה HTTP: } .HTTP_{request} = d_t E2E + d_p E2E = \frac{100B}{10Mbps} + \frac{100B}{1Mbps} + \frac{100B}{10Mbps} + 1.002ms = 1.962ms \text{ כעת נחשב את זמן תגובה HTTP לדג':}$$

$$.HTTP_{response\ file} = \frac{900B}{10Mbps} + \frac{900B}{1Mbps} + \frac{900B}{10Mbps} + 1.002ms = 9.642ms \text{ כעת נחשב את זמן תגובה HTTP לאובייקט:}$$

$$.HTTP_{response\ object} = \frac{1400B}{10Mbps} + \frac{1400B}{1Mbps} + \frac{1400B}{10Mbps} + 1.002ms = 14.442ms \text{ ובסה"כ נקבל: } total = hand\ shake + request + response + response = 2.004 + 1.962 + 9.642 + 14.442 = 30.01ms$$

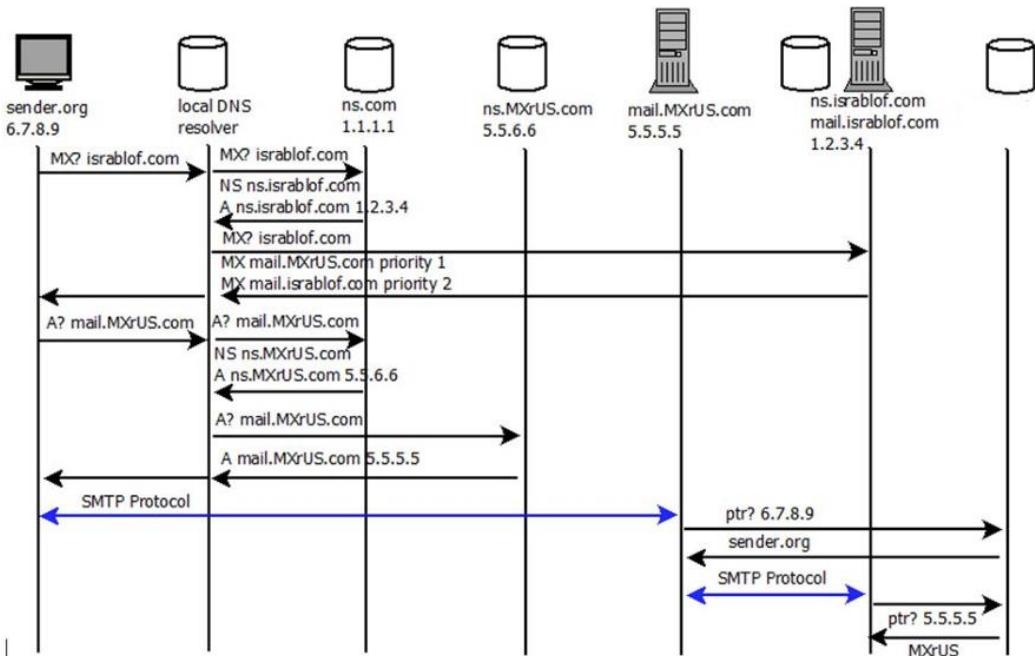
2. במקורה שזכה, האובייקט היה עבר סגמנטציה, ונסלח בשתי חבילות. אבל מכיוון שהחיבור הוא עקבי, ניתן היה לשולח אותן ברצף כי חלון בקרת העומס היה מתיירן (slow start), התחלנו בחבילה אחת של הדף, וכך מוטר כבר להגדיל את החלון לשתי חבילות – שזה מספיק לשילוח האובייקט שלו בסגמנטים).

3. החיבור יהיה persistent וכך וכך נדרש לסגור את החיבור ולפתח חיבור אחר בשביל להorie את האובייקט. אם היה מדובר גם במקורה של MTU=1000, האובייקט היה נשלח בשתי חבילות, אך לא אחת אחרי השניה, אלא קודם היינו מקבלים ACK על החבילה הראשונה ורק לאחר מכן על השניה.

שאלה: חברת MXrUS.ORG מציעה שירות טיפול בדואר נכנס עבור חברות וארגונים שונים, ע"י שירות SMTP שלה שממוקמים באזוריים שונים בעולם. השירותים מקבלים את הדואר ביעילות, מסננים ספאם והתקפות, ואת הדוא"ל התקין מעבירים לשרת הדוא"ל הנכנס של הארגון. חברת israblof.com מחזיקה רק מחשב אחד בכתובת 1.2.3.4 שמרץ, בין היתר, את שרת הדוא"ל ואת שרת השמות. להודת העומס עליו היא נעזרת בשירותי MXrUS. עדין הדרישה היא שההודעות יגיעו לאוטו שרת 1.2.3.4 והוא גם יישאר שרת השמות, אבל ניתן גם להיעזר בשרת השמות היחיד של MXrUS, בכתובת 5.5.6.6.

1. הציגו תרשימים העברת הודעות שמאירה תהליך שליחת דוא"ל משרת שלוח sender.org אל israblof.org, תוך הצגת כל שאלות DNS הרלוונטיות. הינו ש商量 שרת DNS יש את רשומות NS של com, וכתובות IP המתאימות, אבל אין רשומות נוספת. אין צורך לפרט את הודעות SMTP, מספיק להראות רק את תחילתו וסופה.
2. בתחילת הצגתם, יש לגרום לכך שהדוא"ל יועבר לשרת israblof ע"י השרת הקרוב ביותר של MXrUS, להזה של השולח.

תשובה:



מכיוון שהשלוח רוצה לשלוח מייל ל-mail.israblof.com, תחילה עליו לבצע שאלת MX של mail (MX) לשרת DNS המקומי שלו. בcut, מכיוון שבcache של שרota DNS המקומי של השולח יש רק את הפרטים של com, ולא את הפרטים של NS של ns.com, הוא פונה(ns.com) וambil שאלת PTR של ns.israblof.israblof. שרota com מшиб לשרת DNS המקומי את החו"ם domain ואט IP של ns.israblof. מכון הרשות המקומי מ�行 שאלת MX לIP של ns.israblof לוטובת קבלת שרota המייל שלהם. בתגובה, הרשות של ns.israblof מגיב בשתי אפשרויות לשורת המייל שלו – הראשונה (שהיא גם בעדיפות ראשונה) היא שרota המייל שמתזקקים עבורי MXrUS, השנייה הוא שרota עצמו (ns.israblof). בcut, שרota DNS המקומי קיבל את התשובה לשאלת שבקשו ממנו, וכך הוא מחזיר לשולח מהו domain של שרota המייל שלו ns.israblof. בcut, הלקוח מעוניין לקבל גם את כתובות A (IPv4) של שרota המייל הנ"ל, וכן מבצע שוב שאלת DNS לשרת המקומי. מכיוון לשרת DNS אין את IP cache, הוא פונה לcom לשאלת A. com מגיב לו עם NS של MXrUS, וכן עם IP של NS. בcut השרota DNS המקומי פונה לNS של MXrUS ובמבקש את רשומה A של com, mail.MXrUS.com מקבל את כתובות IP של שרota המייל. בcut השרota המקומי מחזיר לשולח את IP הרלוונטי. לאחר מכן ממבצעת תקשורת SMTP (שנדרנו בשאלת לא לפרט) בין השולח לשרת המייל.

9.2 – reverse DNS ptr? – זו שאלת NS, כלומר על סמך IP מבקשים לקבל את החו"ם domain. אם השרota קיבל בתגובה את החו"ם domain שפנה אליו, הוא בעצם אימת את זהותו ומוכן להמשיך הלאה (זהו אקט אבטחתי). בcut, ממבצעת תקשורת בין MXrUS לבין ns.israblof – כאשר שוב ns.israblof מאמתים קודם את זהותם של MXrUS, ורק לאחר מכן מבצעים תקשורת SMTP ביניהם.

הרצאה 8:

האלמנט המרכזי בשכבה הרשת הוא **router**s. תפקידם הוא לנצלabilות בין רשתות כדי שיגיעו מנקזה לנקזה.

Forwarding – הפעולה שמתרבצת כאשר הנתב מקבל חבילה באחד הממשקים שלו ומחליטה להעביר אותה לאחד הממשקים האחרים שלו כדי שתוכל להמשיך בדרך.

Routing – קובע את המסלול כולו מהמקור ליעד, ע"י אלגוריתמי ניתוב.

מישור המידע – קובע כיצד מידע המגיע לממשק הכנסה בנתב מועבר לממשק היציאה של הנתב.

מישור הבדיקה – כיצד הנתבים מחליטים ביניהם באיזה מסלול החבילות יעברו.

שכבת הרשת עושה את המינימום האפשרי, לא מובטח שהמידע יגיע בהצלחה ליעדו או שסדר החבילות שנשלחו ישנה.

כתובת IP מזיהה הן את המשתמש והן את הרשת בה אתה נמצא, הרעיון הוא שהדיאט most significant bits יזהה את הרשת שלך, והדיאט least significant bits יזהה את המשתמש שלך. השאלה היא, איך שמים את החוץ בין MSB לLSB? ישנן מספר שיטות:

.1 – חלוקה מבוססת קבוצות. Class based

class	
A	אם הבית השמאלי בeyond הוא 0, אנחנו בclass A, אם שני הBITSים השמאליים הם 10 אנחנו בclass B וכן להלאה.
B	10 network host
C	110 network host
D	1110 multicast address
המשתמש	
32 bits	
שאר הביטים (בקפיצות של שמונה) מייצגים את המשתמש	

מה החיסרון של שיטה זו? נניח ואנו כורא לחישית קטנה, ולכן לא נדרש לנו הרבה כתובות – על מנת שנשלם כמה שפচות נקנה כתובות **class**. אולם אם החיבור צמחה ממשמעותית, נדרש לקנות כבר כתובות **class**, זה גדול ממשמעותית מכמות הכתובות שנוצר בפועל, וכותזאה מכך נשלם הרבה יותר.

.2 – subnet mask – כל כתובת IP יוגדרsubnet mask – מספר שנוסיף לכל כתובת IP והוא יידיר לכל כתובת IP כמה ביטים משמאליים הם מזיהה הרשת, והשאר הם מזיהה המשתמש. לדוגמה: אם כתובת IP היא: 200.23.18.0/23 אז subnet mask הוא 23, כלומר 23 הביטים הימניים הם מזיהה הרשת, ו9 הביטים הנותרים הם מזיהה המשתמש.



בכל טווח של כתובות IP אנחנו נשמר בדר"כ שתי כתובות – הכתובת הגדולה ביותר לטובת שידור הודעות broadcast, והכתובת הראשונה לטובת זיהוי הרשת. כמו כן, ישנו טווחים ששמורים לשימוש ברשת פרטית כמו 10.0.0.0 – 10.255.255.255, וישנן כתובות IP מיוחדות כמו 127.0.0.1.

כמה תת-רשתות ניתן ליצור לכתובת * 200.39.19.27? מכיוון שה subnet mask הוא 27, וcutet ישנים 24 ביטים קבועים, נקבל שישנים 3 = 27 – 24 ביטים בהם ניתן להעתיקם כדי להגדיר תת-רשתות, כלומר ישן מה"כ 2^3 תת-רשתות (ולכן ישנים 5 ביטים לטובת מזיהה המשתמש). $(32 - 27) = 5$

טבלת Destination-based forwarding – זהה טבלה שמנדרה לנtb טווחי רשות ודרך איזה ממשך לטפל בהן.

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010000 00000100 through	3
11001000 00010111 00010000 00000111	
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
otherwise	3

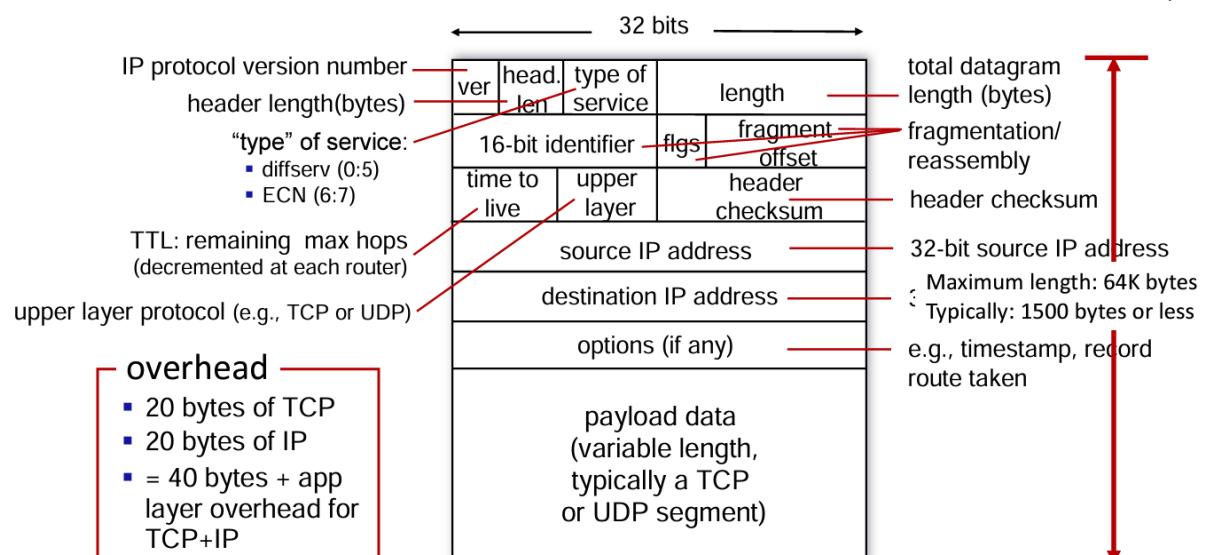
כאשר אנו מחפשים בטבלת היעד, נחפש את הכתובת עם ה`prefix` הארוכה ביותר שמתאימה לכתובת היעד. דוגמה:

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

11001000 00010111 00010110 10100001 ממשק 0

11001000 00010111 00011000 10101010 ממשק 1

להלן פירוט IP header: (מכיוון שאין לנו יותר מדי מה לחפש כי דומה ל`header`ים שכבר ראיינו רץ על זה)



נניח שהחברה שלנו – fly by night ISP בעלת קבוצת IPים הבאים - 200.23.16.0/20. החברה שלנו מוכנה למכור כתם חלק מכתובות ה-IP, לשמונה ארגונים נוספים, כלומר מקצה עוד 3 ביטים מה subnet

mask שלה לכל ארגון. זה הבלוק של החברה שלנו:
 $.11001000\ 00010111\ 00010000\ 00000000\ 200.23.16.0/20$

ואם נחלק אותו לשמונה ארגונים נקבל:

organization 0 - $.11001000\ 00010111\ 00010000\ 00000000\ 200.23.16.0/23$

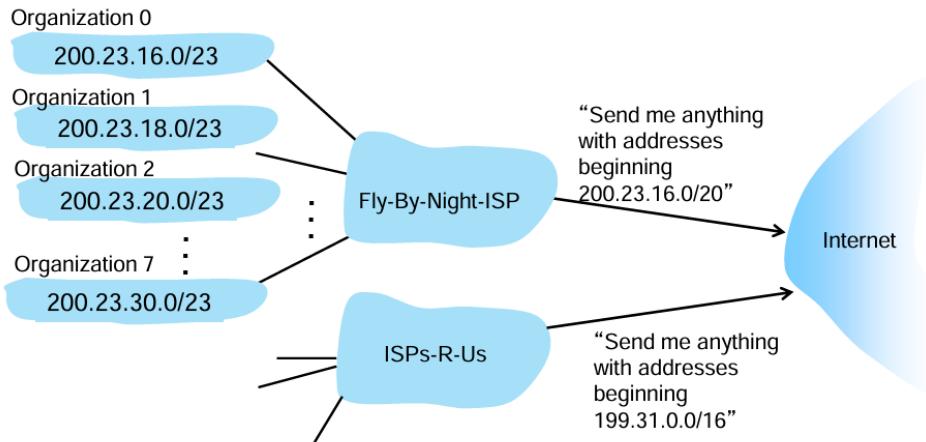
organization 1 - $.11001000\ 00010111\ 00010010\ 00000000\ 200.23.18.0/23$

organization 2 - $.11001000\ 00010111\ 00010100\ 00000000\ 200.23.20.0/23$

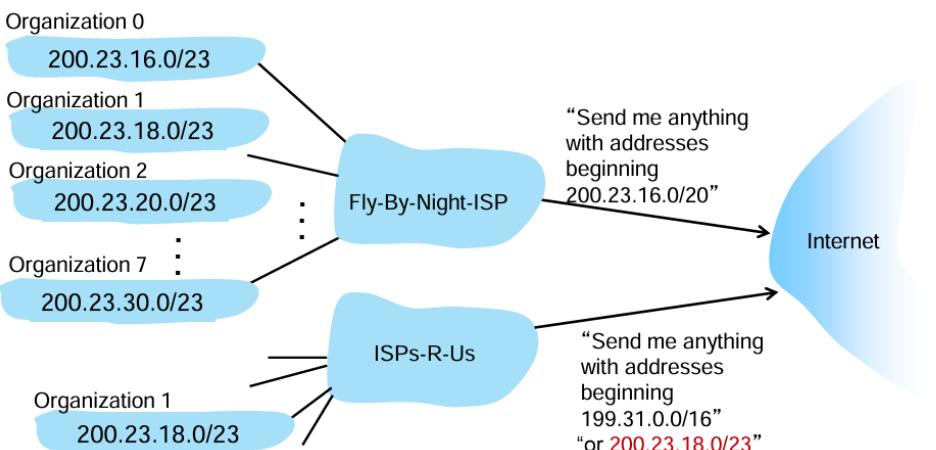
...

organization 7 - $.11001000\ 00010111\ 00011110\ 00000000\ 200.23.30.0/23$

ב_TBL subnet mask של הנטב אכן מתקיים שכל הכתובות של הארגונים מתאימות גם לmask של 20, והוא תחת החברה שלנו ולכן הכל נכון.



כעת, מה קורה אם אחד הארגונים עבר לספקית אחרת Us R Us, ומנייד גם את טווח הכתובות שהוא קיבל מאייתנו אליו? האם علينا לשנות את טווח הכתובות שהנטב צריך להעביר אלינו?



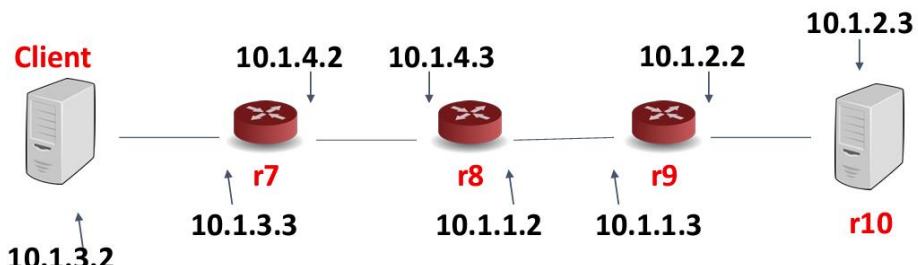
התשובה היא שלא צריך לשנות כלום, כיון שהנטב עובד לפי האזוף הארץ ביחס לכתובת שמתאימה, ולכן מכיוון שהאקסיפ בחברה החדשה מורכבת מ23 ביטים ולא מ20 – המידע יועבר לארגון.

לדוגמה, אם נריץ בטרמינל את הפקודה ping 8.8.8.8 – מה שקרה בפועל זה שתשליח הודעה בעזרת פרוטוקול ICMP לIP 8.8.8.8(header 8 typ 01 coden 01) בשדהTYPE&CODE שמי שלח לו את ההודעה רוצה לבדוק האם ההודעה הגיעה אליו, ולכן יחזיר לו הודעה באמצעות

פרוטוקול ICMP של echo, שברור header יהיה כתוב 0 בשדה type ו-0 בשדה code.

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

באמצעות ICMP ניתן לבצע גם traceroute – פקודה שתפקידה לארור את הנתבים במסלול שהחביבה שלנו תשלח. איך זה מתבצע? שלוחים חבילת UDP כשבורד header שלהttl מגדירים את הערך להיות 1. כאשר הנתב הראשון במסלול מקבל את החביבה הוא יוריד את ttl=1, ואז יגלה שהttl עבר ולכן יזרוק אותה ויזיר ICMP עם type 11 code 0 (TTL expired). כך נגלה את פרטיו של הנתב הראשון במסלול. בעת חזרה על התהיליך עברו הנתב השני, כאשר מגדירים ttl=2, ואז יתבצע אותו תהליך רק שהפעם הנתב השני יזרוק אותה, וכך נגלה את פרטיו של הנתב השני במסלול. משיכים כך עד לתנאי העזירה – נגיע לעד, אולם הידע לא ימצא אפליקציה רלוונטית לPORT שלחנו בהודעה, וכן יחזיר ICMP עם type=3 code=3 (bad IP header). וכך למדנו את פרטייהם של כל הנתבים במעלה הדרן.

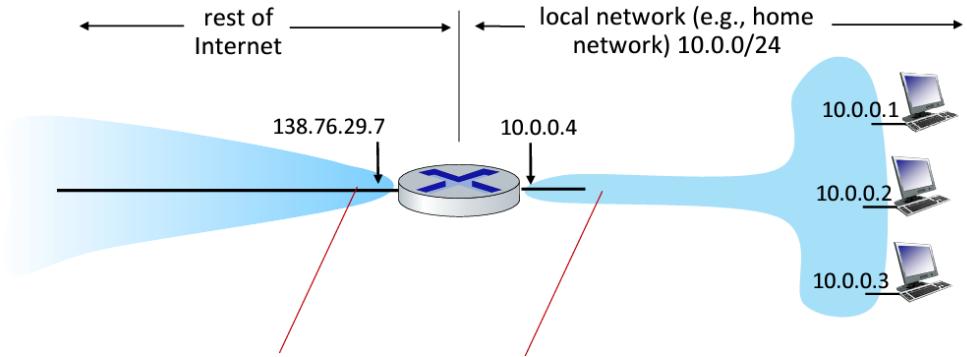


```

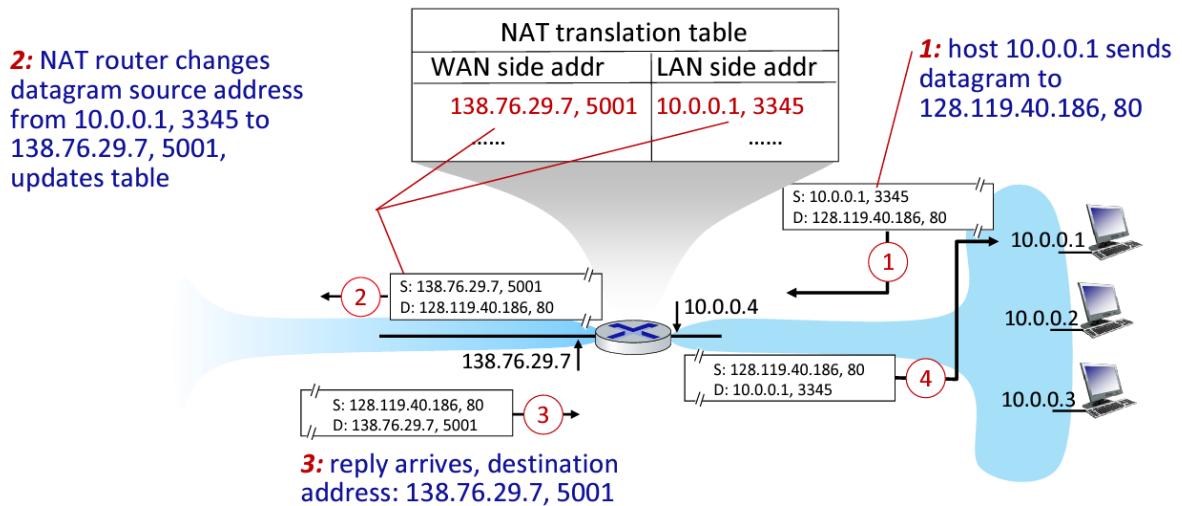
@client:~$ traceroute 10.1.2.3
traceroute to 10.1.2.3 (10.1.2.3), 30 hops max, 60 byte packets
1 r7-link0 (10.1.3.3) 0.211 ms 0.194 ms 0.179 ms
2 r8-link1 (10.1.4.3) 0.342 ms 0.336 ms 0.324 ms
3 r9-link2 (10.1.1.3) 0.692 ms 0.698 ms 0.690 ms
4 r10-link3 (10.1.2.3) 1.043 ms 1.050 ms 1.039 ms
  
```

ישנה בעיה בIPv4 – ישן רק ² כתובות אפשריות, והגענו למיצוי שלhn כבר זמן. גם אם נרצה לעבור לשימוש כוללני בIPv6 זה לא צזה פשוט – צריך לקבוע flag בו כל העולם יעבור לשימוש בIPv6, זה מרכיב מאד! מה הפתרון לכך? NAT.

NAT (network address translation) – כל המכנים ברשת מקומית כלשי חולקים אותה כתובות IPv4, כך שЄחרוצים לפנות לאינטרנט הנתב שלנו מתרגמים את כתובתה המקומית שלו לכתובת IPv4 של הרשת המקומית שלנו. כשאנחנו גולשים בתוך הרשת הפנימית לכל אחד מהכנים יש את הIP שלו, אולם כאשר פונים לרשת החיצונית, יכולים מזוהים ע"י אותה כתובות IP.



כיצד הנטב יודע להחזיר אלינו תשובה שקיבלו מרשת החיצונית? הוא משתמש במספר ה-port שלחנו בשאיילטה, מתאים לו מספר port לשיקולו וכשהתשובה חזרת הוא בודק לפי מספר ה-port מי ביקש מהתשובה הנ"ל ומעביר אותה למכשיר הרלוונטי.



תרגול 8:

שאלה: לאיזה רשת שיכת כתובות IP הבהא: 138.64.22.5 ?

תשובה: מכיוון שלא ניתן לנתח כאן subnet mask, אזי מדובר על כתובות בשיטה המבוססת מחלקות. נמיר את 138 ליבנארי – 10001010, כלומר IP שיר ל B class. ולכן הרשת היא 138.64.

שאלה: לאיזה רשת שיכת כתובות IP הבהא: 255.255.254.0 subnet mask 138.64.22.5 עם ?

תשובה: הכוונה ב subnet mask 255.255.254.0 היא 23, שכן 255 – 8 ביטים Dolkim, ואז שוב 8 ביטים Dolkim, 254 – 7 ביטים Dolkim, ולכן סה"כ נקבל 23 = 8 + 8 + 7 = 8. בעת, כדי לדעת מה מזהה הרשת, נבצע פעולה_and בין כתובות IP לבין subnet mask:

$$\text{and } \frac{10001010.01000000.00010110.000000101}{11111111.11111111.11111110.00000000} = 10001010.01000000.00010110.00000000 = .138.64.22.0$$

שאלה: נתונה כתובות IP הבהא: 101.101.101.64/26 .

1. תננו דוגמה לכתובות IP ששייכת לרשת.
2. האם כתובות IP 101.101.101.75 שייכת לרשת הקודמת?
3. האם כתובות IP 101.101.101.128 שייכת לרשת הקודמת?

תשובה:

1. ראשית, מהה כמה כתובות יש ברשת הזאת לכל היותר: מכיוון שה subnet mask הוא 26, יש нам 6 ביטים שמייצגים מחשב בתוך הרשת. ולכן יש לנו $2^6 = 64$ כתובות אפשריות. מכיוון שאחנחנו תמיד שומרם את הכתובת הנמוכה ביותר והגבוהה ביותר לטובת צרכיהם פנימיים, ולכן במקרה שלנו כל כתובות בטווח 101.101.101.65-101.101.101.128 מתאימה. נבחר לדוגמה את 101.101.101.65.
2. כן, נמצאת בטווח מהסעיף הקודם. ניתן גם לחשב ישירות באמצעות **and** עם ה subnet mask ולוודא שמקבלים את מזהה הרשת.
3. לא.

שאלה: ספק אינטרנט הוא הבעלים של בלוק הכתובות הבא: = 101.101.101.64/26 01100101.01100101.01000000 4 לקוחותיו. הציגו חלוקה שכזו.

תשובה: ברשת זו יש 64 כתובות אפשריות. נשים לב שהספק מעוניין לדעת לכל כתובות לאיזה לקוח היא שייכת. מכיוון שיש 4 לקוחות, הספק מקריב את 2 הביטים הכי שמאליים מהחלוקת של הכתובת (ולא מהחלוקת של מזהה הרשת), בכדי להזות את הלוקוח. הסיבה לשני הביטים היא שבعزيزת 2 ביטים ניתן לייצג 4 לקוחות ($2^2 = 4$). ולכן, אלו יהיו בלוקי הכתובות של הלוקוחות:

- | | |
|--------------------|----|
| 101.101.101.64/28 | .1 |
| 101.101.101.80/28 | .2 |
| 101.101.101.96/28 | .3 |
| 101.101.101.112/28 | .4 |

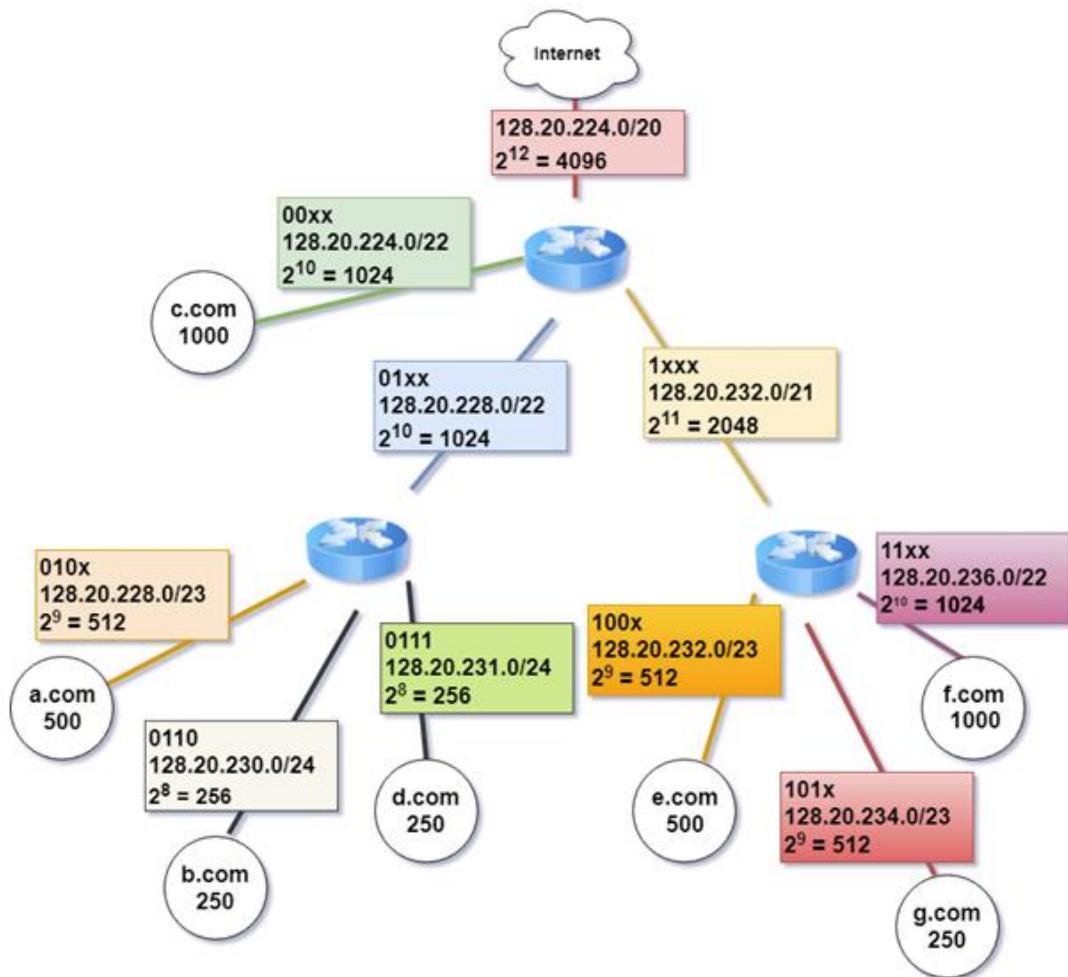
Route aggregation – כאשר הרשת יפרעם לעולם את הכתובות שהוא מעוניין שישלחו אליו, הוא לא יפרעם 4 פרסומים במקורה של השאלה הקודמת, אלא יפרעם 101.101.101.64/26, כאשר חבילה המיועדת לכתובת IP בטווח זהה תעיגע לנtab, הוא יסתכל על שני הביטים הנוספים ויקבע לפיהם לאיזה לקוח עליו להעביר את החבילה. ההליך זהה נקרא route aggregation.

שאלה: לספק אינטרנט יש את בלוק הכתובות 128.220.224.0/20 והוא מעוניין להקצות את הכתובות ל-7 הלקחות שלו, כך שהוא משתמש במינימום נתבים ככל האפשר. נתון שלו כל נתב יש לכל היותר 4 משקים. הלקחות זוקרים לכמויות הכתובות הבאה:

a.com	→ 500	.1
b.com	→ 250	.2
c.com	→ 1000	.3
d.com	→ 250	.4
e.com	→ 500	.5
f.com	→ 1000	.6
g.com	→ 250	.7

הציגו הקצאות כתובות אפשריות.

תשובה:



תחילה צריך לשים לב שאנו מוגבלים בקפיצות בין ארונות, גם אם לקוח רוצה רק 1000 כתובות הוא יהיה חייב לקבל את הקפיצה הבינארית הקרובה ביותר אליו מלמעלה, כלומר 1024. עצם אם מישחו רוצה לשולח מידע לIP שכתובתו 128.220.224.0/20 הוא יגיע לנטב הראשון, משם הוא יחפש את הaxfex prefaciיה בירוכה נוספת שמתאימה לו, ויעבור לנטב הרלוונטי.

שאלה: בשאלת הקודמת נשארו 512-250=262 כתובות ללא שימוש. כמה תתי רשתות נוכל לחלק אותן, אם כל תנת רשת היא בעלת 50 מחשבים?

תשובה: מכיוון שכל כתובות מבוססת בינהרי, אנחנו צריכים לבדוק כמה כתובות אנחנו צריכים לפחות 50 כתובות, ונקבל שאנו צריכים $6 \text{ ביתים} = 2^6$. וכך ניתן לחלק את הכתובות ל-4 תת-רשתות.

הרצאה 9:

RFC (request for comment) – מסמך שמנדר את הפרטוקול מהתחלת ועד הסוף, מה מותר מה אסור, אילו שדות יש בתפקיד וכו'.

סוגי NATים:

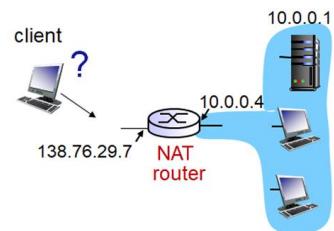
1. Full cone NAT – אם ביצענו מיפוי של IP וPORT פנימיים לIP וPORT חיצוניים, המיפוי הנ"ל יישמר לפחות זמן אורך מאד. יתרה מכך, כל מכשיר ברשת האינטרנט יכול לשלוח הודעות למחשב שלי באמצעות המיפוי הנ"ל (כלומר לIP ולPORT שהנתב הקצה לו), והן יגיעו אליו.
2. Restricted cone NAT – הגבלה נוספת על הcone, רק כתובות IP שאני פניתי אליהם בעבר יכולו לשלוח לי הודעות. יש רעיון אבטחתי מאחוריו שיטה זו, ברשות מקומית כל המכניםים אמורים להיות לקוחות ולא שרתים, ומה שנרצה לקבל פניה מבוחר אם אני לא שרת? יכול למנוע התקפות על המכשיר שלי.
3. Port restricted cone NAT – הגבלה נוספת על cone, רק כתובות IP וPORT שפניתי אליהם יכולו לשלוח לי הודעות.
4. symmetric NAT – הגבלה נוספת על cone, אם אני שולח חבילה לעד אחר – מתבצע מיפוי שונה מהמיופיע שכבר קיים אצל הנתב.

ארQUITטורות:

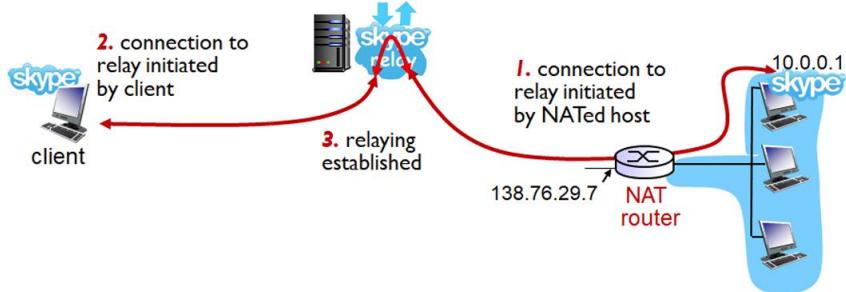
1. שרת-לקוח: השרת עובד תמיד, עם אותה כתובת IP ואילו הלקוח יכול גם לא לעבוד תמיד ולשנות כתובת IP. כמו כן הלקוח לא מתקשר לשירותים עם לקוחות אחרים.
2. Peer 2 peer: כל לקוח בשרת יכול לשחק גם תפקיד של שרת. יתרונות – לא ממורכז, ניתן להוריד קבצים מכל אחד מהלקוחות ששמרו אצלן עותק, כי הם מתפקדים גם בשרתים. חסרונות – לקוחות לא מחוברים לרשת 24/7 וכן לא תמיד זמינים לתפקיד כשרתים.
3. Hybrid: הלקוחות נתונים לשרת מידע על IPם שלהם ועל תוכן שהם מכילים ומבקשים מהשרת מידע על לקוחות אחרים בהתאם לתוכן שהם רוצים לצור. לאחר מכן, מתחברים לשירות לקוחות שאליו הם רוצים לפנות.

איך יכול לשלוח הודעה לממשק אחר תחת NAT, אם המכשיר לאשלח לו קודם הודעה? אין מיפוי מתאים בטבלה של הנתב. ישנו מספר פתרונות:

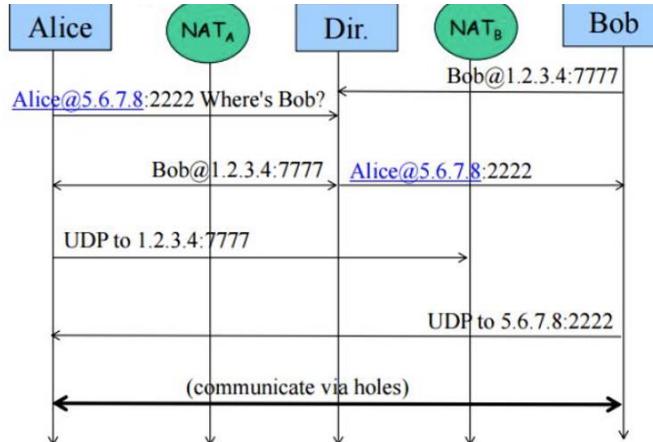
1. הגדרת רשומות סטטיות שישמשו את הנתב כאשר מגיעה בבקשת התחברות חדשה.
 - ❖ client wants to connect to server with address 10.0.0.1
 - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
 - only one externally visible NATed address: 138.76.29.7
 - ❖ **solution 1:** statically configure NAT to forward incoming connection requests at given port to server
 - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000
2. אפשרות אחרת היא להשתמש בשרת מתווך – מכיוון ששרתים מעוניינים שייפנו אליהם לקוחות, גם במקרה שאינם מוכרים את IPם שליהם, נستخدم בשרת על מנת לתקשר עם המכשיר באינטרנט: המחשב שלנו יועבר דרך נתב NAT לשרת, ושם השרת יעביר את המידע למכשיר הרצוי ולהפוך. חסרונות בשיטה זו – מה קורה אם השרת נופל? מה אם השרת לא מאובטח כמו



שצריך או שמי שמתוחק את השירות רוצה להאזין למידע שעובר בו?



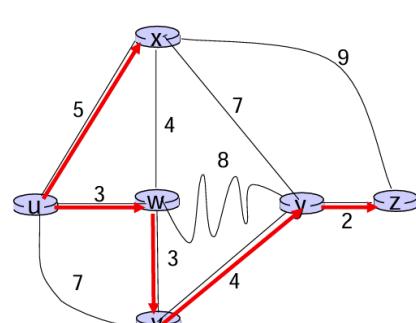
.3 – הלקוחות ישתמשו בשרת רק על מנת לבצע היכרות בין הNATים שלהם, ולאחר מכן יבצעו תקשורת באמצעות הNATים שלהם.



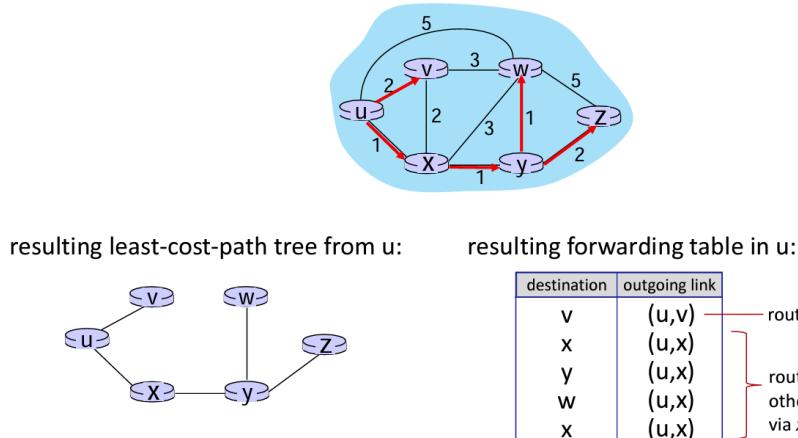
תחליה בוב מתחבר לשרת (Dir) שמהווה מעין מאגר ללקוחות. בפועלות ההתחברות בוב בעצם "מחורר חור" בNAT שלו שיאפשר לשרת לשלוח לו הודעות. בעת אליס רוצה לשלוח הודעה לבוב, ולכן היא שואלת את השירות היקן בוב נמצא, בפעולה זו גם היא "מחוררת חור" בNAT שלה שיאפשר לשרת לשלוח לה הודעה. בעת השירות שלוח הודעה לשנייהם: לאלייס – בוב נמצא בIP 1.2.3.4 PORT 7777, ובבוב הוא מוסר שאלייס מחפש אתו, והוא נמצא נמצאת בIP 5.6.7.8 PORT 2222. בעת אליס שלוחת הודעה UDP לIP ולPORT של בוב דרך NAT שלו, ולכן היא מחוררת חור עבור הודעה של בוב – אולם NAT של בוב זורק את ההודעות של אליס שכן שבוב עדין לא חורר חור עבור הודעה של אליס. לאחר מכן, בוב שלוח הודעה לאלייס, שתצליח להגיע ליעדה כיון שהוא יכבר מוכן לקבל הודעה מבוב, אז – הם יכולים לבצע תקשורת ישירה ביניהם דרך NATים שלהם, ולא באמצעות השירות.

כיצד קובעים את המסלול בעל העלות הנמוכה ביותר, כלומר כיצד מנתבים את החבילות? נציג את הבעיה כgraf ממושקל: כאשר הקודקודים הם הנתבים, הקשותות בין זוג נתבים שיש ביניהם משקל, והמשקל על כל קשת הוא עלות ההעברה בין נתב לנtab.זיכרון מאלגוריתמים, דיקטורה פותר את הבעיה. דוגמה להמחשה:

Step	N'	v	w	x	y	z
0	u					
1	uw	7,u	3,u	5,u	11,w	∞
2	uwx		6,w		11,w	14,x
3	uwvx		6,w		10,v	14,x
4	uwxv					12,y
5	uwxvz					



באמצעות הרצת האלגוריתם של דיקסטרה, נוכל לקבוע לכל נתב את טבלת forwarding שלו, כלומר דרך איזה מסלך עדיף לו לצאת בהתאם לעדילו הוא רוצה להגיע:



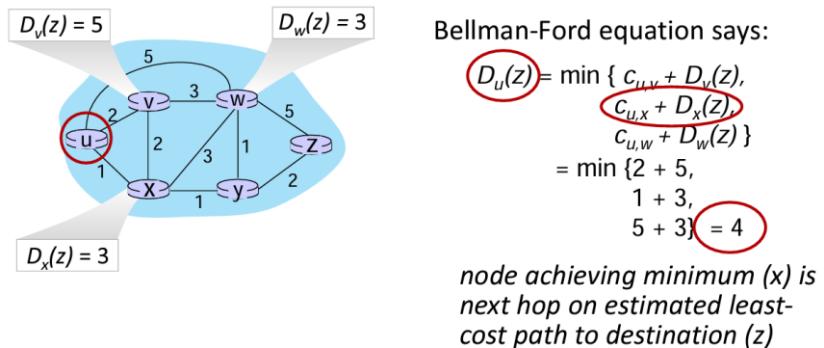
על מנת שלכל נתב תהיה את טבלת forwarding נציג להריץ מכל נתב דיקסטרה. כלומר לכל נתב צריך להיות כוח חישובי די חזק... מה קורה במקרה שבנתב מתקל במסלול? נרץ שוב דיקסטרה מכל הנתבים? מאד לא עיל.

תנודות אפשרויות בדיקסטרה: נדים מדו"ע השימוש בדיקסטרה מעורר אי יציבות במסלולי הניתוב. נגיד את משקלי הקשתות להיות מספר הפעמים שהעברנו מידע בין נתב לנתב (כלומר אין קשותות דו-כיוונית), אלא קשת מנtab A לנtab B וממנtab B לנtab A, כי ככל כיוון יש משקל אחר). נמבחן באמצעות דוגמה:

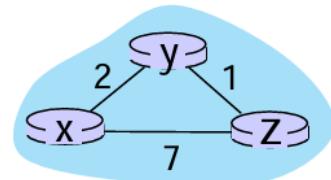


לודג'ס open short path first (OSPF) – נדרש לדעת את המיפוי המלא של הרשת, ונרץ דיקסטרה כפי שראינו למעלה.

Distance vector algorithm – מבוסס על אלגוריתם בלמן-פורד. אין צורך בהיכרות עם מבנה הרשת כולה, אלא רק לדעת מי השכנים שלו. משווהת בלמן פורד $\{y\}_v = \min_x \{c_{x,v} + D_x(y)\}$, כלומר המסלול הקצר הקצר ביותר מא לע הוא המינימום מבין הקשת (v, x), כאשר v הוא אחד משכנים x , ועד המסלול הקצר ביותר מט לע. לדוגמה, המסלול הקצר ביותר מט לע בדוגמה הבאה הוא 4.



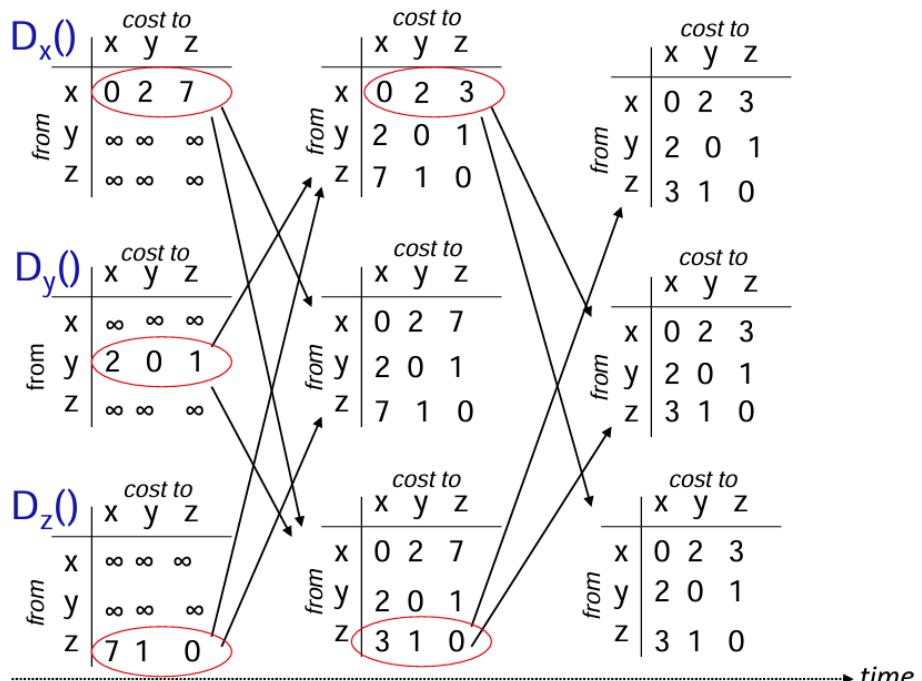
דוגמה הרצה לאלגוריתם: להלן הגרף עליו נריץ את האלגוריתם-



תחילה, כל אחד מהנתבים מעדכן אצלו את המרחק ממנה לשכניו (לרובות לעצמו). בשלב הבא, כל נתב ישלח לשכניו את וקטור המרחקים שהוא יצר, והשכנים יעדכו אצלם בהתאם למידע שהם קיבלו.

ניתן לראות שבטוטו
השני x קיבל מידע מעז
שהוא יכול להציג לפחות
בעלות 1. מכיוון שאז
יודע שהוא יכול להציג
לע בעלות 2, הוא יעדכן
את עלות המסלול לפחות
להיות 3.

לבסוף, הנתבים יגלו
שאין שיפורים אפשריים
ולכן יסימנו את
האלגוריתם.



תופעות הקשורות ב-algorithm:
distance vector

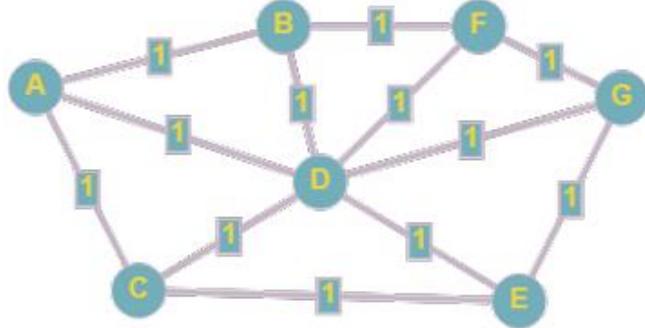
1. חדשות טבות מתרשות מהר – אם מחיר קשת החול, שכניו יגלו זאת ויעדכו מהר מאוד את המסלולים דרכו.
2. חדשות רעות מתרשות לאט – אם מחיר קשת עלה, ייקח לשכניו המון זמן לעלות על המחיר העדכני הנוכחי (ייתכן לו לאו אינסופית). ניתן לנסות פטור זאת באמצעות poision reverse – אולי גם זה לא פתרון שיעבוד תמיד.

RIP (routing information protocol) – מחיר התחלתי של קשותות הוא 1. אם אחרי 180 שניות לא קיבלו ערך משכנן, הקשת ביןו לא קיימת יותר. שאר האלגוריתם מבוסס על distance vector.

תרגול 9:

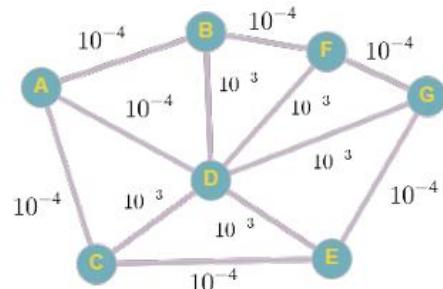
שאלה: קצב השידור בכל הערכות הוא $s = 2 \cdot 10^8 \text{ bps}$. מהירות ההתפשטות היא $R = 10^6 \text{ m/s}$. משקל קשת הוא הזמן שŁוקט להעביר חבילה בקשת, כולל הזמן שהיית התפשטות. המרחק מפ לשאר הקודקודים הוא $m = 10^4 \cdot 2$ לא כולל a . המרחק של שאר הערכות הוא $m = 10^4 \cdot 2$.

1. A רוצה להוריד קובץ בגודל 5KB מ-Google HTTP 1.1. מה הנתיב שייעברו הabiliaות בעזרת RIP?
2. A רוצה להוריד שני קבצים מ-Google בגודל 3KB כל אחד בעזרת OSPF 1.1 HTTP.
3. אותו הדבר אבל הפעם d הוא proxy.



תשובה:

1. מכיוון RIP הינו פרוטוקול מסוג distance vector, כל נתב לומד את המרחקים לכל היעדים דרך שכניו. RIP אוחנו מדיפנים את המסלול הקצר ביותר מבחינת מספר הקפיצות. לאחר מספר סיבובים תחנה A תלמד מתחנה D שהיא יכולה להגיע ל-G דרך B במסלול הקצר ביותר.
2. נחשב את משקל הקשתות שנן השהייה התפשטות לפי נתוני השאלה ונקבל:



המסלול שיבחר הוא $a \rightarrow f \rightarrow b$.

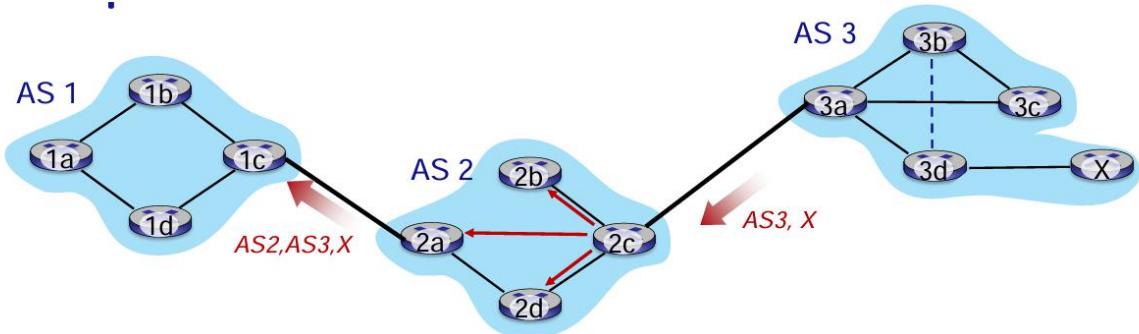
3. מכיוון שכעת יש proxy, אז A יפתח חיבור TCP מול proxy ויבקש את המשאב ממנו. D יפתח חיבור נפרד מול G ויבקש את המשאב, יוריד את המשאב אליו ואז יעביר אותו לא.

הרצאה 10:

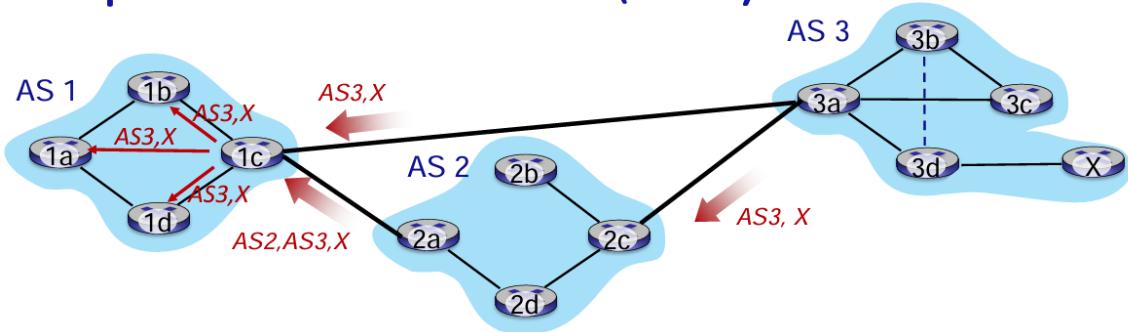
– נחלק את האינטרנט לאזורים אוטונומיים, כך שכל אזור יחליט באיזה אלגוריתם ניתוב הוא משתמש (RIP או OSPF), וכן יוכל לבחור בהערכת האלגוריתמים לא תיקח זמן רב. בנוסף, אם בוצע שינוי כלשהו באזורי אוטונומי אחר, או שהוא משתמש באלגוריתם ניתוב אחר, זה לא משנה לאיזור האוטונומי שלנו. אם אני רוצה לתקשר עם מילוי AS'ו, פונים כרגע ליעיר על בסיס פרוטוקולי הניתוב את הבקשה וממשיכים כרגע. אך אם אני רוצה לפנות למשהו שנמצא מחוץ ל-AS שלי? עליי להגיע לנット מיוחד שנקרא border gateway. הוא יעביר את זה בין נתבי, עד שנגיע לנット הגבול של ה-AS שאליו אני רוצה לשולח בקשה, ושם ה-*border gateway* שלהם ינתב אותו לעד.

path (border gateway protocol) – פרוטוקול לטובות ניתוב חבילות בין AS. הוא מבוסס על vector, כלומר על נתב הגבול לקבל החלטה מראש מסלול והוא רוצה לעבור על מנת שהחבילת תגיע לידי.

בדוגמה למטה, AS 3 מפרස את המסלול X, AS, ככלומר הוא מבטיח ל-2 AS'ים שהוא יעביר נתוניים דרכו הוא יעביר אותם לא-AS 2 מקבל את הצעתו של AS 3, וכך מפרסם לכל הנתבים שת חתיות | את המסלול 3 AS הציע. בהתאם, נתב הגבול 2a ב-AS 2 במאזעות נתב הגבול 2a מפרסם את המסלול X, AS 2, AS 3, ומוציא אותו ל-1 AS.



כעת נניח ש-1 AS ו-3 AS חתמו ביניהם על הסכם ולכן ישנו גם מסלול ישיר ביניהם:

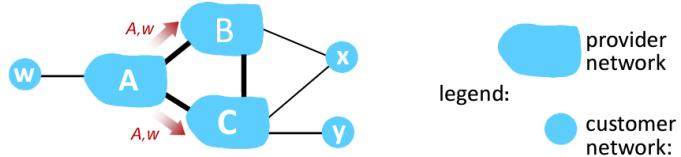


כעת, נתב הגבול של 1 AS, 1c, לומד שני דרכי להגעה לא-AS:

$$\begin{aligned} \text{AS } 2 \rightarrow \text{AS } 3 \rightarrow X &.1 \\ \text{AS } 3 \rightarrow X &.2 \end{aligned}$$

כעת, בהתאם למידניות של 1 AS נתב הגבול בוחר באיזה מסלול להגעה לא-AS (לאו דווקא המסלול הקצר ביותר!).

באיור ניתן לראות שישנו ספקיות – ה-ASes
הגדולים, ולכל ספקית יש לקוחות, שהם ה-ASes
הקטנים יותר. הספקיות מנהלות ביניהן קשרי
עמיתים, ולא גובות תשלום אחת מהשנייה,
אלא מהלוקחות שלהן.



מכיוון שלספקיות אין אינטראס להעביר ביניהן חבילות שלא לצורך – כי הן לא מתוגמלות על כך, B לא תפרנס את המסלול $w \rightarrow A \rightarrow B \rightarrow C$. X הוא dual homed – מחובר לשתי ספקיות.

להלן הכללים הבסיסיים של BGP:

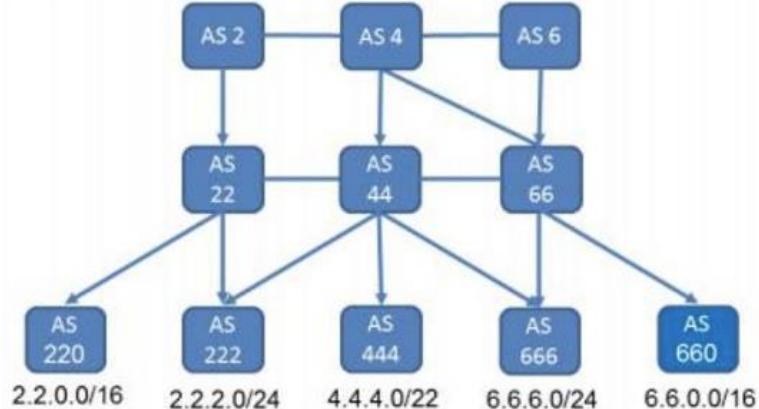
1. אם יש במסלול תחנה שאני לא מוכן לעבור בה (ייתכן ומדובר בתחנה שנמצאת במדינה עיינית או סתם כי לא מתאים לי לעبور דרכה) – אל תפרנס את המסלול.
2. מבין המסלולים שאני מוכן לעבור בהם, נבחר את המסלול שהכי משתלם לי כלכלית – נעדיף מסלולים שעוברים דרך לקוחות שלי, ואם אין כלו נבחר במסלול שעובר דרך ספקיות עמיתות. בplibית ברירה, אם אין מסלול שעובר דרך לקוחות או ספקיות עמיתות, נבחר במסלול שעובר דרך ספק שלי.
3. אם ישנו מסלולים שבהם מבחינת הקרייטריוניים מעלה, נבחר את הקצר ביותר מבחינת מרחק. אחרת, נבחר שובר שוין כלשהו (לדוגמה סדר אלפביתי).

למי מפרנסים את המסלול?

1. אם מדובר במסלול שימושתי עליו דרך לקוח ללקוח – נפרנס אותו לכלום: לעמיתים, לספקים ול לקוחות, על מנת למקם את התגמול שנתקבל במעבר דרכו.
2. אם מדובר במסלול שימושתי עליו מעמית או מספק – נפרנס אותו רק לקוחות שלו.
3. אם מדובר בלקוח – הוא יפרנס את הגישה אליו לספק.

תרגול 10:

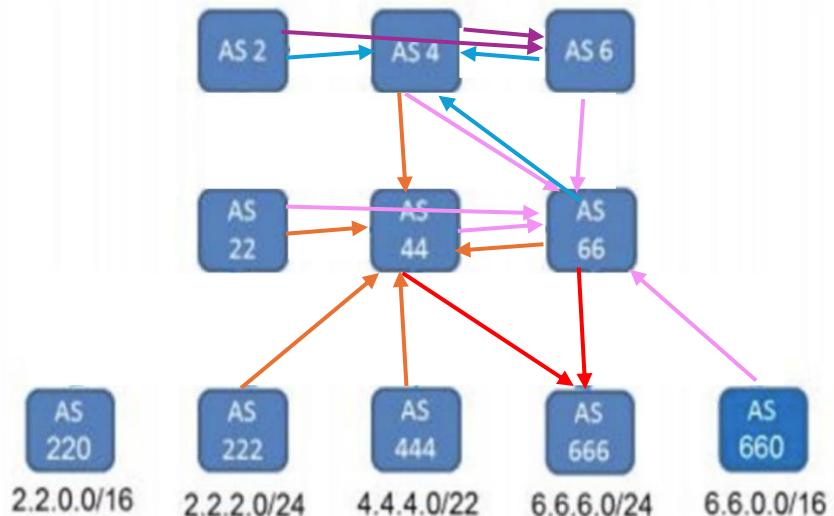
שאלה: התייחסו לרשות האזוריים האוטונומיים בציור להלן, כאשר יחש לקו-ספק מובעים לפי הגובה, החיצים והמספרים. הניחי כי כל האזוריים נוקטים מדיניות BGP רגילה, שמשתמשת ומופרסת נתיבים משתלים כלכליות, וכך שבחירה בין שני נתיבים משתלים אותה במידה מעדיפים את הנתיב הקצר יותר. אם הם באותו אורך, בוחרים את זה שהתקבל מס' ספורי נמוך יותר.



1. רשמי את כל ה-AS-path **שיקבל** AS4 בפרסומי BGP, ומאהה שכן, עברו תחילית 0.6.6.6.0.
2. עברו תחילית 0.2.2.2.0.
3. רשמי את כל ה-AS-path **шибירסם** AS4 ולמי משכניו, עברו תחילית 0.6.6.6.0.
4. עברו תחילית 0.2.2.2.0.

תשובה:

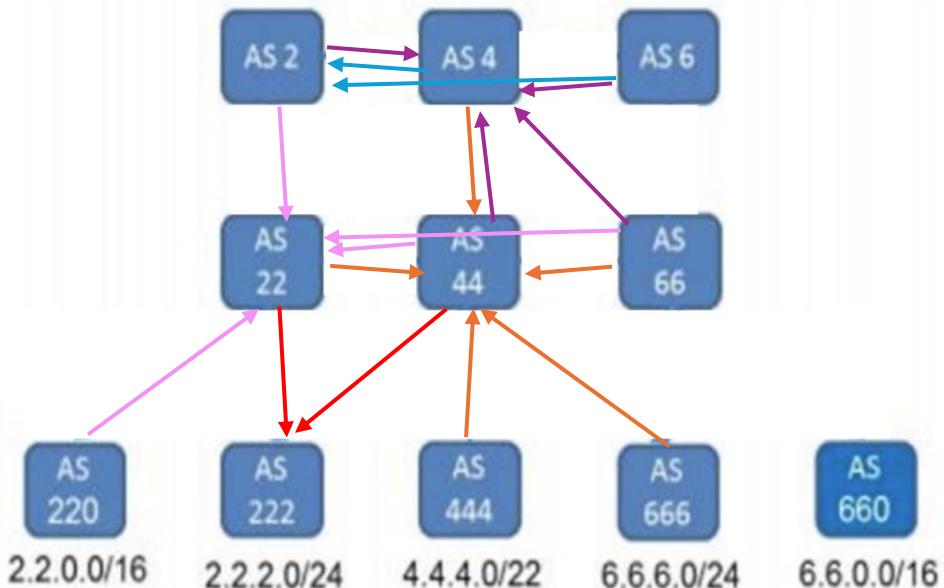
1. באדום – AS666 מסחר על המסלול אליו לספקיות שלו.
בורד – הספקית AS66 מעבירה לכל שכינה את המידע על מסלול שעובר דרכה (כי AS666 לקוח שלה, ולכן משתלים לה לפרסם אותו).
בכחות הספקית AS44 מעבירה לכל שכינה את המידע על מסלול שעובר דרכה (כי AS666 לקוח שלה, ולכן משתלים לה לפרסם אותו).
- בתכלית – הספקית AS4 מעבירה לכל שכינה את המידע על מסלול שעובר דרכה (כי AS44,AS66 liquego משתלים לה לפרסם אותם).
- בסגול – הספקית AS6 מעבירה לכל שכינה את המידע על מסלול שעובר דרכה (כי AS666 liquego משתלים לה לפרסם אותם).
(לא הוסתי חיצי לולאה, אבל גם הם קורים)



ולכן אלו המסלולים שיבורסמו ל-AS4:

Next AS	AS path
AS 6	AS6→AS66→AS666
AS66	AS66→AS666
AS44	AS44→AS666

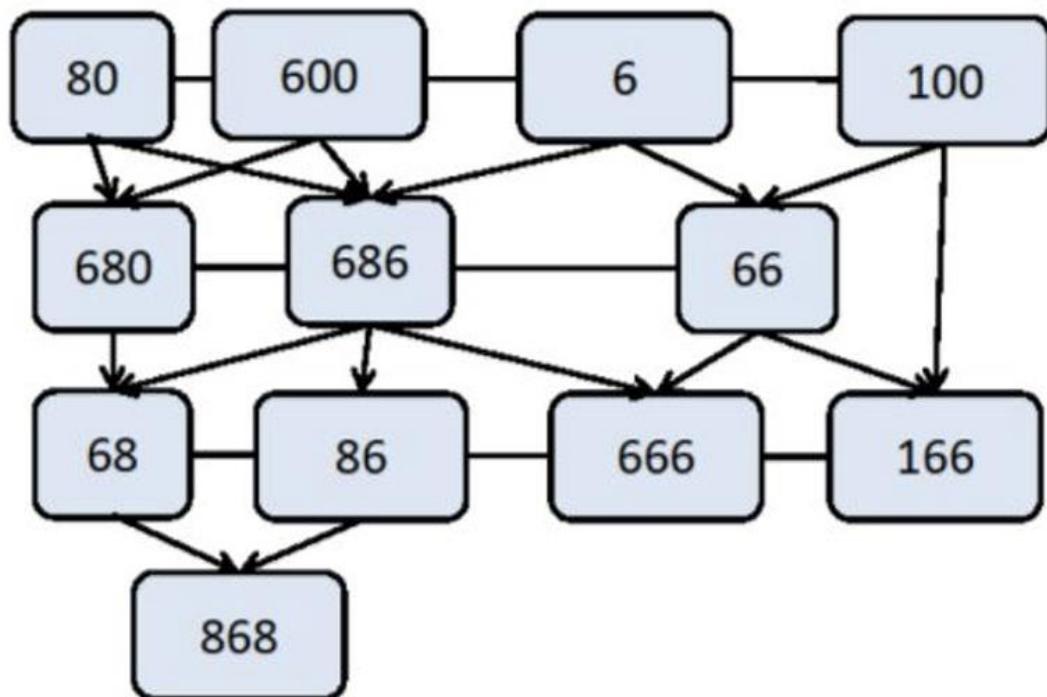
- .2. באדום – AS222 מסוף על המסלול אליו לספקיות שלו.
 בוורוד – הספקית AS22 מעבירה לכל שכניה את המידע על מסלול שעובר דרכה (כי AS222 ל Koh)
 שלאה, ולכן מעתולם לה לפרסם אותו).
 בכתום הספקית AS44 מעבירה לכל שכניה את המידע על מסלול שעובר דרכה (כי AS222 ל Koh)
 שלאה, ולכן מעתולם לה לפרסם אותו).
 בתכלת – הספקית AS2 מעבירה לכל שכניה את המידע על מסלול שעובר דרכה (כי AS222 ל Koh)
 שלאה, ולכן מעתולם לה לפרסם אותו).
 בסגול – הספקית AS4 מעבירה לכל שכניה את המידע על מסלול שעובר דרכה (כי AS44 ל Koh)
 שלאה, ולכן מעתולם לה לפרסם אותו).



Next AS	AS path
AS44	AS44→AS222
AS2	AS2→AS22→AS222

- .3. בהסתמך על הציור מסעיף 1, AS4 יפרסם לאט המסלול AS6,AS66,AS2,AS44, AS4 → AS44 → AS666
 .4. בהסתמך על הציור מסעיף 2, AS4 יפרסם לאט המסלול AS6,AS66,AS2,AS44, AS4 → AS44 → AS222

שאלה:



1. בין איזה אזוריים אוטונומיים לא תתכן תקשורת, לפי הקשרים שבציגו?
2. הנח כי רשת 666 אינה מריצה BGP וגיל אלא מעוניינת לקבל תובורה שנשלחת אל רשת 8.8.8.0/24 מراتות אחריות, אפילו ע"י חריגה מהפרוטוקול ו/או מדיניות "לא כלכלית". תקשורת מאיזה רשתות 666 תוכל לכלוך וכייד? הנח שרשת 24 מחוברת לאזור 868.

תשובה:

1. לא תתכן תקשורת בין 100 ל600, 80, 680. כמו כן לא תתכן תקשורת בין 80 ל6, 100, 66, 68, 166.
 2. AS666 יכול ללכוד תקשורת מכל שאר העמיטים או הספקים שלו (חו"ן 60 AS86, AS68, AS680, AS6).
- ע"י כך שהוא יפרנס נתיב קצר יותר, ככלומר הוא יカリ אליו הרשת מחוברת ישירות אליו.

חזרה על פרגמנטציה:

שאלה: מחשב A שולח הודעה למחשב B. ה-SUT בין A לR1 וכן בין R1 לR2 הוא 1500B. ה-SUT בין R2 לבין B הוא 660B. מחשב A שולח למחשב B כמות של 1980B של שכבת אפליקציה. כיצד ישלח את המידע בTCP או UDP?

תשובה: $A \xrightarrow{1500B} R1 \xrightarrow{1500B} R2 \xrightarrow{660B} B$
מחשב A: header TCP = 20B, header UDP = 10B, header IP = 20B
תחילה עברו UDP TCP

:A→R1 .1

מספר פרוגמנט	כמות בתים	header IP + header UDP + header TCP	length	IP id	MF	DF	offset
1	1470B של שכבת אפליקציה + 20B + 10B	1500B	1500	x	1	0	0
2	510B של שכבת אפליקציה + 20B + 10B	530	530	x	0	0	($\frac{1480}{8} = 185$) 185

.2. R1→R2: אותו דבר.

מספר פרוגמנט	כמות בתים		IP id	length	MF	DF	offset
1א	630B של שכבת אפליקציה .20B + 10B +		x	660	1	0	0
1ב	640B של שכבת אפליקציה .20B +		x	660	1	0	80
1ג	200B של שכבת אפליקציה .20B +		x	220	1	0	160
2	510B של שכבת אפליקציה .20B +		x	530	0	0	($\frac{1480}{8} = 185$) 185

cut-through TCP: מכיוון TCP קודם כל מגלה מי הוא MTU המינימלי, החבילות יחולקו מלכתחילה ל-60B לכל היותר. כמו כן, מכיוון TCP מבצע סגמנטציה ולא פרגמנטציה – כל חבילה תשלוח גם עם sequence number המתאים לה.

שאלה: לאיזה interface ישלו החבילות עם כתובות ה-IP:

- .1 195.65.127.3
.2 196.94.100.13

	Interface
196.80.0.0/12	Eth0
196.94.16.0/20	Eth1
196.96.0.0/12	C
196.104.0.0/14	D
128.0.0.1/1	E
64.0.0.0/2	F
0.0.0.0/2	G

תשובה:

- .1 את 4 הראשונות אפשר לפסול מידית מכיוון שניתן לראות שיותר מ-8 הביטים שמאליהם שלהם מקובעים, ולכן אין מהчинיות כolumn ב-196. כתעת נשים לב שעבור 195 קיבל השביט השמאלי ביותר חייב להיות דלוק מכיוון שהוא תורם 128, ולכן רק E מספק את התנאי זהה.
.2 .Eth0