רשתות תקשורת

רובנו גלשנו ל-Facebook, הרשת החברתית העצומה שמונה מעל למיליארד משתמשים. אך האם עצרנו לשאול את עצמנו - מה בעצם קורה מאחורי הקלעים כשגולשים? איך יתכן שאנו נמצאים בבית, מקישים בדפדפן (Browser) את הכתובת "www.facebook.com", לוחצים על מקש ה-Enter, ומקבלים תמונת מצב של כל החברים שלנו?
על מנת לענות על שאלה זו, עלינו להבין ראשית מה האתר Facebook צריך כדי לתפקד.

כל אתר וכך גם אתר Facebook זקוק ל**אחסון (Storage)** - הכוונה למקום בו יימצאו דפי האתר ואליו יפנו המשתמשים. אתר Facebook ישמור גם את המידע על כל המשתמשים כגון: מי חבר של מי, התמונות שהועלו לאתר, סטטוסים וכו'. בנוסף, אתר Facebook זקוק ל**עיצוב**. יש לעצב לוגו, להחליט היכן תוצג רשימת החברים, היכן יוצגו העדכונים שלהם, איפה יוצגו הפרסומות ועוד.
האתר זקוק גם ל**אימות (Authentication)** - עליו לזהות את המשתמש שפונה אליו. לשם כך, Facebook צריך לזכור את כל המשתמשים והסיסמאות שלהם, ולאפשר לכל משתמש להתחבר ולהזדהות. כלומר, Facebook נזקק ל**תקשורת**. צריכה להיות דרך שתאפשר לאדם לתקשר עם האתר של Facebook בין אם מהמחשב שלו בישראל, ובין אם מהסמארטפון שלו כשהוא נמצא בטיול באיטליה.

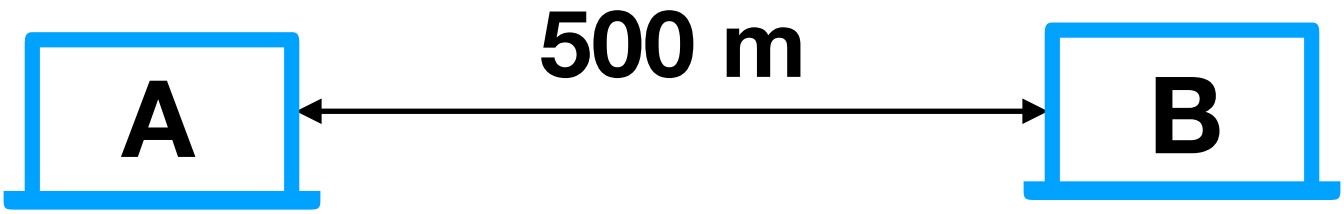
WWW – World Wide Web

כשאנו אומרים "אינטרנט", אנו מתכוונים בדרך כלל ל- WWW (World Wide Web). זהו אוסף עמודי האינטרנט אליהם אנו גולשים בדפדפן. משמעות המילה Web היא רשת. עמודי האינטרנט מקושרים אחד לשני כמו רשת של קורי עכביש המקושרים זה לזה. עמודי האינטרנט מפוזרים בכל רחבי העולם - World Wide - בעוד אתר אחד נמצא בחיפה, שני יכול להיות בטוקיו ושלישי בניו-יורק. העמודים השונים מקושרים ביניהם באמצעות לינקים.

על מנת שהדפדפן יוכל להציג את העמוד הראשי של Facebook, עליו לדעת כיצד העמוד נראה. הכוונה היא לדעת איזה טקסט קיים בעמוד, היכן כל חלק מהעמוד ממוקם, באיזה גופן הטקסט כתוב ובאיזה גודל, האם צריך להציג תמונות על המסך, אילו תמונות ועוד. את כל המידע הזה, הדפדפן משיג מאתר Facebook עצמו. בכדי ש- Facebook ישלח לדפדפן את המידע, על הדפדפן להודיע שהוא מעוניין בו. כלומר, הדפדפן צריך לשלוח הודעת **בקשה (Request)** אל Facebook, ובה להגיד: "שלח לי את המידע הנדרש כדי להציג את העמוד הראשי של האתר www.facebook.com". כאשר Facebook מקבל את הבקשה, הוא שולח **תגובה (Response)** שמכילה את המידע הדרוש.

האתר של Facebook מקבל בקשה ושולח תגובה. כלומר, הוא מספק שירות לדפדפן. אי לכך, נאמר כי האתר של Facebook הינו **שרת (Server)**, והדפדפן הינו **לקוח (Client)**. באופן כללי, כאשר גולשים ב-WWW, ישנם **לקוחות** (דפדפנים) ששולחים בקשות אל **השרתים** (אתרי האינטרנט), והשרתים מחזירים תגובות ללקוחות. אותו מידע שמגיע בתגובות, משמש את הדפדפנים בכדי להציג על המסך את אתרי האינטרנט למשתמשים.

כמה זמן לוקח ל-A להעביר חבילת נתונים שלמה ל-B?



מבוא

מהירות שידור / קצב שידור / רוחב פס-
כמות הנתונים אשר ניתן להעביר בקו תקשורת בזמן נתון.
מסומן ב- R ונמדד ביחידות bit per sec (bit/s, bps)

קו התפשטות-
סוג של ערוץ תקשורת, המשמש להעברה של אותות תקשורת (כבל)

מהירות התפשטות / קצב התפשטות-
מהירות התקדמות המידע בקו התקשורת, כתלות בסוג הקו.
מסומן ב- S ונמדד ביחידות m/sec

תווך רשת (network media)-
מנגנון הנושא פיזית נתונים ממחשב למחשב.
מחשבים מחוברים באמצעות תווך רשת, וברוב הרשתות אלה הם כבלי נחושת המחברים בין המערכות.
תווך הרשת יכול להיות גם סיבים אופטיים או טכנולוגיה אלחוטית, בכל המקרים, החיבורים הפיזיים נקראים תווך.

מרחק פיזי בין מקור ליעד-
מסומן ב- L ונמדד ביחידות m (מטרים)

גודל הקובץ-
מסומן ב- F ונמדד ב bit

זמן שידור (transmission time)-
הזמן שלוקח למידע להעלות מהמכשיר המשדר לקו התקשורת (הזמן שלוקח לקחת את החבילה ולשים על הקו).
נסמן ב- t_x ונמדד ביחידות bit per sec (bit/s, bps)

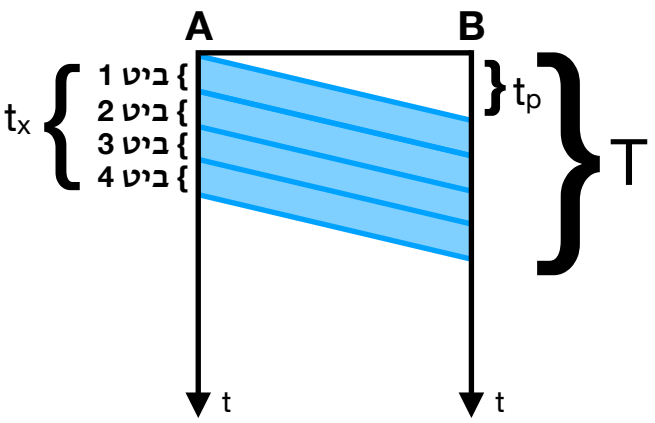
$$tx = \frac{F}{R} = \frac{\text{גודל הקובץ}}{\text{קצב השידור}}$$

זמן התפשטות / השהיה (propagation time)-
הזמן שלוקח לביט הראשון לעבור בקו התקשורת עד שהוא מגיע ליעד.
מסומן ב- t_p ונמדד ביחידות m/sec

$$tp = \frac{L}{S} = \frac{\text{מרחק מהיעד}}{\text{קצב התפשטות}}$$

זמן שידור כללי-
הזמן שלוקח מרגע התחלת השידור עד שההודעה התקבלה ביעד (זמן הגעה מהמקור ליעד).
נסמן ב- T

$$T = tx + tp$$



תקורה-
ביטים אשר לא מכילים מידע אך מועברים כחלק מהחבילה שמשדרת כגון כתובת יעד, כתובת מקור, ביטים לגילוי ולתיקון שגיאות ועוד.

זמן השהיה כולל בין תחנות A ו- B-
הזמן מרגע שתחנת המקור שידרה עד שתחנת היעד קיבלה את הביט הראשון.
כאשר אין צמתים בדרך —> שווה לזמן התפשטות (ללא זמן השידור של ההודעה

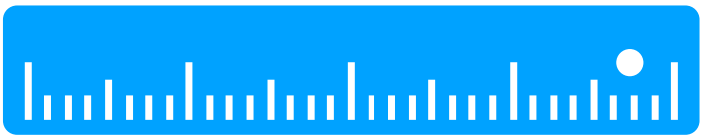
גדלי קבצים:

1 Byte = 8 bit
1 Kbyte = 10^3 Byte = $10^3 \cdot 8$ bit
1 Mbyte = 10^3 Kbyte = 10^6 Byte = $10^6 \cdot 8$ bit
1 GByte = 10^3 Mbyte = 10^6 Kbyte = 10^9 Byte = $10^9 \cdot 8$ bit

המרת זמנים ומרחקים:

- מ- cm ל- m נחלק ב- 10^2
- מ- cm ל- m נכפיל ב- 10^2

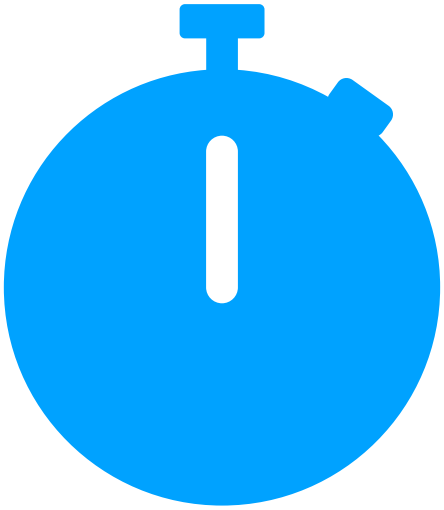
- מ- mm ל- m נחלק ב- 10^3
- מ- m ל- mm נכפיל ב- 10^2



•ממילי שנייה (ms) לשנייה (sec) נחלק ב- 10^3
•משנייה (sec) למילי שנייה (ms) נכפיל ב- 10^3

•ממיקרו שנייה (μs) לשנייה (sec) נחלק ב- 10^6
•משנייה (sec) למיקרו שנייה (μs) נכפיל ב- 10^6

•מננו שנייה (ns) לשנייה (sec) נחלק ב- 10^9
•משנייה (sec) לננו שנייה (ns) נכפיל ב- 10^9



יחידות למדידת קיבולת ערוץ / רוחב הפס והתפוקה

Bit per sec - bps
Kilobits per sec - Kbps
Megabits per sec - Mbps
Gigabits per sec - Gbps
Terabits per sec - Tbps

מעבר מבסיס הקסה דצימלי לבסיס דצימלי (עשרוני)

יש להמיר את הספרה ההקסה דצימלית לערכה הדצימלי.

הקסה	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
עשרוני	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

לאחר מכן נכפיל את הספרה המומרת בחזקה המתאימה של 16 לפי מיקום הספרה במספר

$$n_1 \cdot 16^0 + n_2 \cdot 16^1 + n_3 \cdot 16^2 + \ldots + n_m \cdot 16^{m-1}$$

למשל- 00**A**0 יומר ל- 00**10**0 וכעת:

$160 = 0 \cdot 16^3 + 0 \cdot 16^2 + 10 \cdot 16^1 + 0 \cdot 16^0$ בדצימלי

מעבר מבסיס עשרוני לבסיס 2 (בינארי)

x: מספר עשרוני.

1. חלק את x ב- 2 ורשום את השארית.
2. חלק את המנה שהתקבלה שוב ב- 2 ורשום את השארית החדשה.
3. חזור על שלב 2 עד לקבלת מנה 0.
4. המספר בבסיס 2 של x הוא סדרת השארית בסדר כתיבה הפוך.

המספר בבינארי: 10001001

חלוקה	מנה	שארית
137/2=	68	1
68/2=	34	0
34/2=	17	0
17/2=	8	1
8/2=	4	0
4/2=	2	0
2/2=	1	0
	0	1

ובחזרה למספר בבסיס עשרוני:

$1 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$

כלומר, המספר בבסיס עשרוני הינו 137.

כלל אצבע לחיבור בינארי

0+0=0
1+0=1
0+1=1
1+1=10
1+1+1=11

חיבור מספרים בבסיס הקסה דצימלי

כאשר נבצע את החיבור עצמו, נמיר את הספרה ההקסה דצימלית לערכה הדצימלי. במידה ותוצאת החיבור הינה מספר עד 15 (כולל) נרשום אותו בספרה ההקסה דצימלית המתאימה.
במידה והסכום יצא מעל 15, נחסר 16 לתוצאה ולמספר הבא נוסיף 1.

למשל-

```
  11
 C395
+A9BF
-----
16D54
```

The Internet

הגדרות:

-hosts

כל מכשיר/מחשב/פלטפורמה שמחובר לרשת ומריץ יישומי רשת תקשורת.
כלומר, כל מכשיר שמחובר לרשת, לא משנה אם הוא שרת או לקוח, כל עוד הוא מחובר לאינטרנט הוא נחשב כ- Host.

יישומי רשת (Network Apps)-

יישומים (אפליקציות) הנגישות דרך רשת מחשבים (האינטרנט).
הם מורכבים מדפי אינטרנט דינמיים המסתמכים על תוכנה ש"מורצת" בצד הלקוח ובצד השרת.
לדוגמה- אימייל, דפדפן.

קווי תקשורת (Communication Links)-

קווי התקשורת הם למעשה הסיבים, נחושת, רדיו, טלוויזיה בלוויין וכד' המקשרים אותנו לרשת התקשורת.
קצב השידור הינו כרוחב הפס (תזכורת- R)

נתב/ראוטר (router)-

רכיב תקשורת שתפקידו להעביר הודעות בין ערוצי התקשורת שמחבורים אליו.

-port

תהליך ספציפי שדרכו יכולות תוכנות להעביר נתונים באופן ישיר.
ניתן לדמות זאת לשליחת מכתב דואר בין שתי משפחות הגרות בשכונה של בתים רבי קומות.
אם כתובת ה- IP הינה מזהה הבניין של המשפחה- למשל "הרצל, תל אביב, בית מספר 1"
אז הפורט במקרה זה- הוא מזהה הדירה הספציפית בבניין, למשל "דירה 23".

צומת (Node)-

מכשיר אלקטרוני פעיל המחובר לרשת אשר מסוגל להעביר מידע על ערוץ תקשורת.

פרוטוקול (protocols)-

אוסף של כללים המגדירים את אופן בקשת וקבלת הנתונים במערכת תקשורת.
זוהי דרך לתקשר בין הגורמים ברשת.
שימוש בפרוטוקולים הכרחי על מנת שהצדדים יבינו זה את זה ויוכלו לספק שירותים זה לזה.

-Internet Standards

1. מסמכי *RFC: Request for Comments (מסמכי בקשה להערות).*
מסמכים טכניים ומפורטים שבהם מוגדרים הפרוטוקולים.
מסמכי RFC נכתבים בדרך כלל על ידי מהנדסים וחוקרים בתחום מדעי המחשב לשם ביקורת עמיתים או לשם הפצת תפיסות חדשות ומידע.

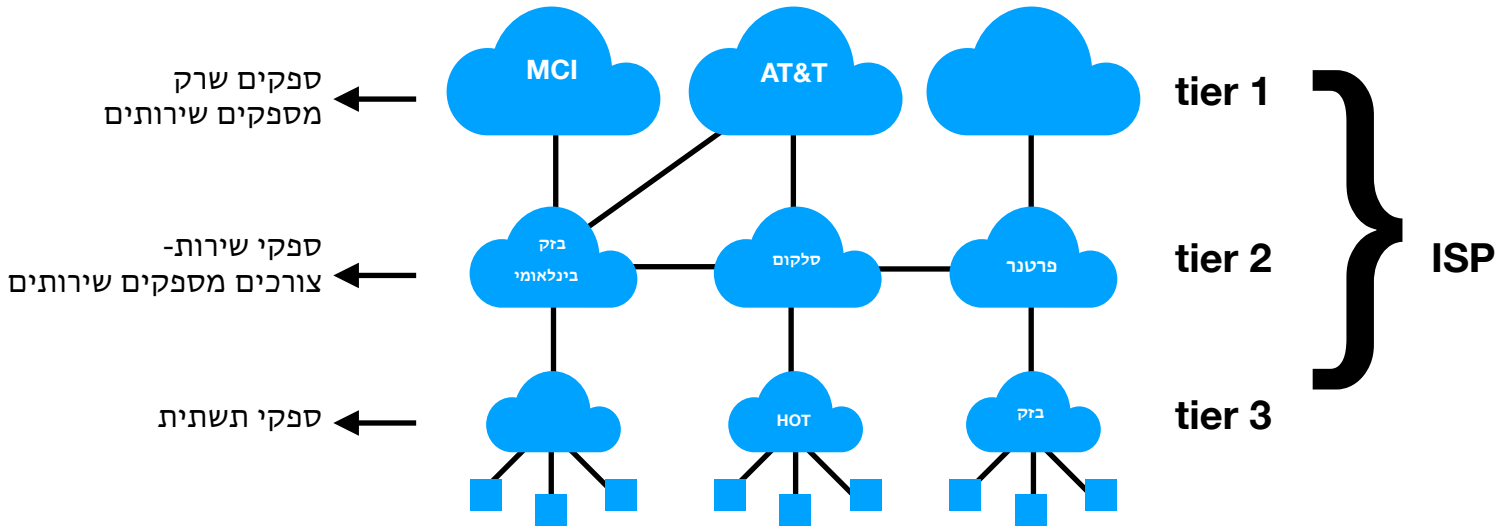
2. *IETF: Internet Engineering Task Force.*
הגוף הממליץ על תקנים (סטנדרטים) שמשמשים להפעלת האינטרנט. תקנים אלו מתפרסמים במסמכי RFC. בזכות תקנים אלו יכול מחשב מתוצרת חברה אחת לתקשר עם מחשב מתוצרת חברה אחרת, ולהעביר מידע שיהיה מובן למגוון תוכנות.

Network Structure

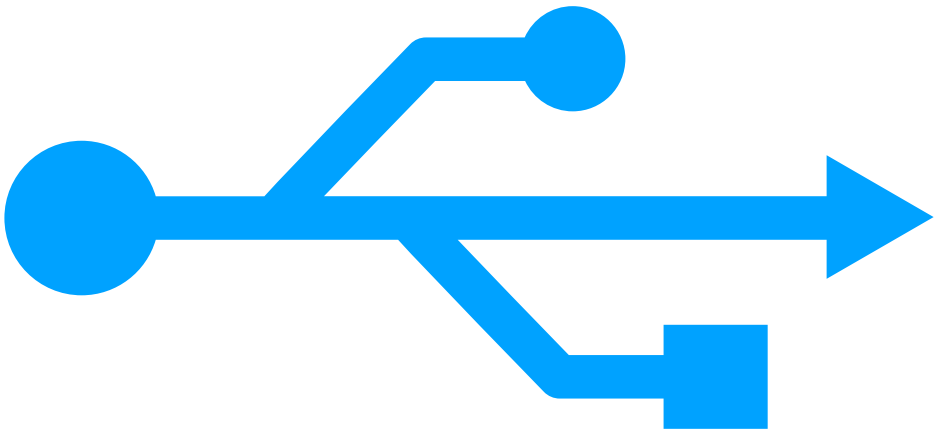
-network edge
קצוות הרשת, כלומר ה- Hosts (מערכות קצה)

-Access Network
הרשת שפיזית מחברת את מערכות הקצה לאינטרנט.
כמו בזק- יוצרים תשתית עד הבית וכך יוצרים את האפשרות להתחבר.

-Network Core
ספק אינטרנט ברמה ארצית/עולמית.
נקראים ISP- Internet service provider (ספקי שירות אינטרנט)



- ISP מהשכבה ה- 2 לקוחות של ISP מהשכבה ה- 1 וגם אחד של השני.
- ISP מהשכבה ה- 3 לקוחות של ISP מהשכבה ה- 2 (הכי קרוב למערכות הקצה)



The Network Edge

Client-Server (שרת-לקוח)-

צורת התקשורת הנפוצה ביותר כיום באינטרנט נקראת שרת-לקוח (Client-Server). בצורת תקשורת זו קיים רכיב כלשהו המשמש כשרת (Server), דהיינו מספק שירות כלשהו, ורכיב כלשהו הנקרא לקוח (Client) אשר משתמש בשירות המסופק.
מחשבי הלקוח משתמשים במשאבי הרשת ומחשבי השרת מספקים משאבים אלו.
לדוגמה, כאשר אתם פונים אל האתר של YouTube בכדי לצפות בסרטון, הלקוח הוא המחשב שלכם (או הדפדפן שבמחשב שלכם), והשרת הוא אותו שרת של YouTube. במקרה זה, הדפדפן שלכם שולח בקשה לשרת של YouTube, והשרת מחזיר לכם תשובה.

שרת-

- בעל כתובת IP קבוע- נגיש.
- תמיד מספק נתונים, עונה על בקשות של Hosts אחרים- זמין.
- רשתות מבוססות שרת יכולת לגדול ממספר משתמשים בודדים לאלפים ושימוש בחוות שרתים.
- שרת פסיבי- מחכה לבקשות מהלקוח.
- תמיד רץ ב- Host (מחשב קצה)

לקוח-

- יכולים להיות בעלי כתובת IP דינמית או קבועה.
- יכולים להיות מחוברים לסירוגין.
- לקוח אקטיבי- יוזם תקשורת, שולח בקשות לשרת.
- לא מתקשרים ישירות זה עם זה

חיסרון- אם השרת נופל כל המערכת עלולה להיות בלתי שמישה.

Peer To Peer :P2P (עמית לעמית)-

רשת תקשורת בה כל אחד מהקצוות מתפקד הן כלקוח והן כשרת, וכל אחד מהקצוות מסוגל ליזום או לסיים התקשרות וכן לספק או לדרוש שירותים.
בהשוואה לרשת שרת-לקוח בה יש הבחנה ברורה בין הצדדים השונים ברשת, ברשת עמית לעמית אין הבחנה כזו, והרשת מתפקדת באופן מבוזר, ללא צורך בתיווך או תלות בגורם מרכזי כלשהו.
רשתות עמית לעמית משמשות בעיקר ברשתות שיתוף קבצים, בהן כל משתמש מאפשר לשאר המשתמשים להוריד קבצים מהמחשב שלו, ומקבל בתמורה גישה לקבצים ממחשבי המשתמשים האחרים.

- אין שרת שזמין תמיד.
- רשת תקשורת בה כל אחד מהקצוות מתפקד גם כשרת וגם כלקוח.
- מערכות הקצה מתקשרות ישירות.
- העמיתים מחוברים לסירוגין ומשנים כתובות IP.

בעיות-

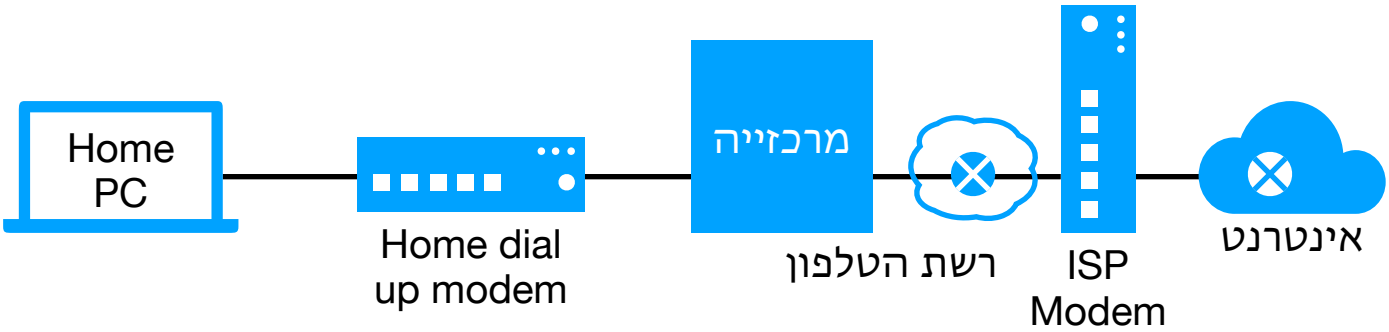
1. מציאת הרשת- אם אני לא מחובר לאף מחשב, אינני יודע היכן לחפש אותם.
2. מציאת המידע- אם כולם יכולים לשתף נתונים, איך עוקבים היכן הוא נמצא?

כלומר, הרשת קשה לניהול, גודלה משתנה בהתאם ללקוחות וכך גם כמות המידע.

Access Net

-Dial Up modem

חיבור Dial-up היא צורת תקשורת נתונים בה המשתמשים נעזרים במודם על מנת לחבר בין המחשב לקו הטלפון החוטי. באמצעות קו זה המשתמשים מחייגים ומתחברים לספק שירותי האינטרנט (ISP) על מנת ליצור חיבור מודם למודם אשר מנותב לרשת האינטרנט.
היתרון בשיטה זו היא שהיא מסתמכת על תשתית הטלפון הקיימת בכל בית וכך חוסכת את ההוצאות שבהקמת תשתית חדשה.
החיסרון בשיטה זו הוא שכאשר לקוחות רבים באותו אזור (ולפיכך גם באותה תשתית טלפונית) משתמשים בה, מהירות האינטרנט נפגעת קשות נוסף על כך שרוחב הפס צר מלכתחילה.
כאשר גולשים לא ניתן להשתמש בטלפון וליהפך.



DSL: Digital Subscriber Line (בתרגום חופשי: קו מנוי דיגיטלי)-

משפחה של טכנולוגיות לתקשורת נתונים דיגיטלית באמצעות תשתיות הטלפון המקומיות.
מהצד של הלקוח, הציוד הדרוש הוא מודם DSL. המודם ממיר את האותות האנלוגיים של מתח חשמלי בטווח תדרים מתאים לקו הטלפון לאותות דיגיטליים של מחשבים.
מהצד של ספקית השירות, נמצא התקן DSLAM (Digital Subscriber Line Access Multiplexer - מֻרַבֵּב כניסת קו מנוי דיגיטלי) אשר מפריד בין קול הדיבור של שיחת טלפון, לאותות הדיגיטליים שמועברים בטכנולוגית ה-DSL.
העיקרון- יש קו אחד שמחובר מהבית למרכזייה, ה- DSLAM יודע להעביר את המידע ל- ISP שלי שמחבר אותי לאינטרנט, כך מנצלים את תחום התדרים היותר רחב שניתן להעביר בקו טלפון, כלומר תוך כדי השימוש בטלפון נשתמש בחלק מהתדרים להעברת מידע.

ADSL: Asymmetric Digital Subscriber Line-

היא טכנולוגית DSL שמשמשת לתקשורת נתונים ברוחב פס גבוה בקווי טלפון רגילים. המילה "אסימטרית" (Asymmetric) בשם הטכנולוגיה רומזת לעובדה שהטכנולוגיה מנצלת את רוב רוחב הפס להעברת נתונים **אל** המשתמש, ומותירה רק חלק קטן מרוחב הפס להעברת נתונים **מ**המשתמש.

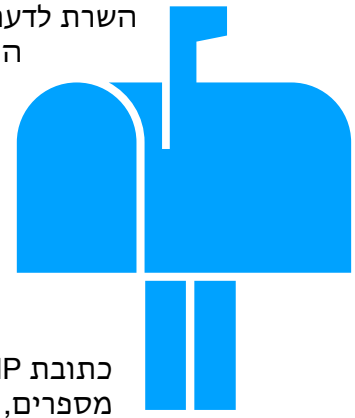
Cable Network (אינטרנט בכבלים)-

אינטרנט בכבלים הוא יצירת תקשורת בין ספק שירותי אינטרנט (ISP) ובין צרכן של שירותים אלה (משתמש ביתי או עסקי), באמצעות תשתית הכבלים שהונחה לשם אספקת שירותי טלוויזיה בכבלים.
העברת תקשורת בכבלים היא טכנולוגיה נפוצה לתקשורת נתונים ברוחב פס גבוה. בישראל, טכנולוגיה זו, אשר בשימוש על ידי חברת הכבלים HOT מהווה מתחרה עיקרית לטכנולוגיית ה־ADSL אשר בשימוש על ידי חברת התקשורת הגדולה בישראל, בזק.

-Ethernet

טכנולוגיה לתקשורת נתונים ברשתות מחשבים מקומיות (LAN).
כלומר, מערכות הקצה מחוברות למתג Ethernet.

Internet Addresses



על מנת שנוכל לשלוח ולקבל הודעות באינטרנט, עלינו לדעת לאן לשלוח את הבקשות ועל השרת לדעת להיכן לשלוח את התגובות. כאשר אנו שולחים מכתב בדואר, אנו מציינים על המעטפה את כתובת היעד (נמען) ואת כתובת המקור (מוען). באופן דומה, גם כאשר נשלח מידע ברחבי האינטרנט, יש צורך בכתובות מתאימות שיזהו את השולח ואת היעד של ההודעה.
יש המכנים את כתובת ה- IP של מחשב כ- "תעודת הזהות של המחשב", אם כי תיאור זה הולם יותר כתובת MAC, מאחר שכתובת IP לא קבועה, וייתכנו 2 מחשבים עם אותה כתובת IP במידה והם לא מחוברים לאותה הרשת.
כאמור, IP היא כתובת לוגית בלבד, בניגוד לכתובת פיזית. כלומר, זו לא כתובת ש"צרובה" על כרטיס הרשת, אלא עשויה להשתנות עם הזמן ולהתחלף.

כתובת IP מיוצגת באמצעות ארבעה בתים (bytes), ולכן תיראה כצירוף של ארבעה מספרים, שכל אחד נע בין הערכים 0 ו-255. להלן דוגמה לכתובת IP: 192.168.2.5

-Interface

ממשק חיבור בין נתב/Host לקו תקשורת.
לנתב יש כמה ממשקים ולכל ממשק יש כתובת IP משלה.
למחשב יש בדרך כלל ממשק אחד אך ישנם כאלה עם יותר.
לממשק רשת יש בדרך כלל יש צורה כלשהי של כתובת רשת, כתובת זו עשויה להכיל מזהה צומת (Node) ומספר יציאה (Port) או להיות מזהה צומת ייחודית בפני עצמה.
ממשקי רשת מספקים פונקציות סטנדרטיות כגון העברת הודעות, חיבור וניתוק וכו.

ובכן, חלק מכתובת ה- IP הוא קוד זיהוי הרשת (Network ID) אליה הוא שייך, ושאר חלקי הכתובת מציינים את קוד זיהוי המארח (Host ID), המציין את המחשב של המשתמש.

Subnet (תת-רשת)-

חלוקה לוגית של כתובת IP.
למחשבים המשתייכים ל- Subnet יש חלק זהה בכתובות ה- IP שלהם. התוצאה היא חלוקה לוגית של כתובת ה- IP לשני שדות, אחד הוא מספר הרשת או "קידומת הניתוב", והשני הוא שדה מזהה למארח מסוים או לממשק רשת.

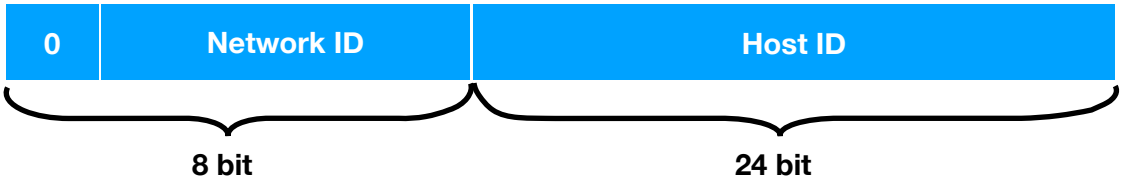
Subnet Mask(מסכת רשת משנה)-

הגדרה של מספר הסיביות בכתובת ה־ IP המשמשות לקביעת כתובת הרשת.
(אחדות- כתובת תת הרשת, אפסים- כתובת Host)
כל נתב מכיר רק את ה- Subnet באינטרנט ולא כל Host בפועל.
לדוגמה- הביטו בכתובת הבאה: 192.168.0.1. נאמר שמסיכת הרשת שלה מוגדרת כ-16 ביטים (או שני בתים). מכאן ש-192.168 הינו מזהה הרשת, ו-0.1 הינו מזהה הישות.
מקרה זה ניתן להציג בדרכים שונות: 192.168.0.1/16 - הוספת "/16" בסוף הכתובת מציינת שה-Subnet Mask מכיל 16 ביטים, כלומר שזהו מזהה הרשת הרלוונטי.
דרך נוספת היא לציין שהכתובת היא 192.168.0.1, וה-Subnet Mask הינו 255.255.0.0 (16 הביטים הראשונים דולקים, ולכן הם מזהה הרשת. 16 הביטים הבאים כבויים, ולכן הם מזהה הישות).

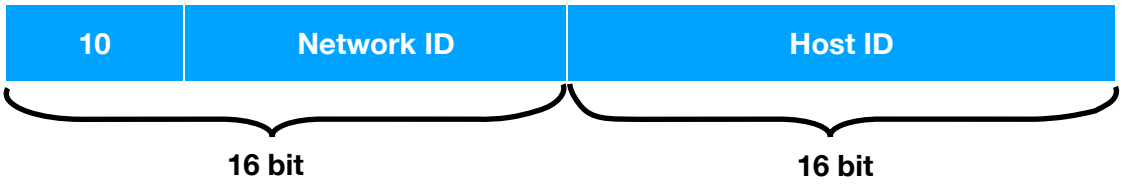
Classful Addressing (מחלקות כתובות)-

כיוון שהניחו שלארגונים שונים יהיו דרישות שונות, חילקו מתכנני הפרוטוקול את מרחב הכתובות לשלוש מחלקות, לפי גודל הרשתות שנועדו לשרת:

- **מחלקה A** - כתובות ממחלקה זו יועדו לרשתות גדולות מאוד, שיכילו עד 2^{24} מחשבים.
כתובת ממחלקה A מתחילה תמיד ב- 0. מספר הרשת מיוצג ע"י 7 ביטים, ושאר 24 הביטים מייצגים את מספר המחשב ברשת.
המספר המקסימלי של רשתות שונות מסוג A הוא 126 $(2^7 - 2)$. הסיבה שהמספר אינו 2^7 היא שהכתובת 0.0.0.0 (32 אפסים) שמורה עבור ייצוג של נתיב ברירת-מחדל(link), והכתובת 127.0.0.0 היא כתובת שמורה, המיועדת לפונקציית ה-(link) loopback.



- **מחלקה B** - כתובות אלה יועדו לרשתות בינוניות. כל כתובת מתחילה בביטים "10", שאחריהם מספר רשת בן 14 ביטים, ומספר מחשב בן 16 ביטים.
המספר המקסימלי של מחשבים ברשת מסוג B הוא 65534 $(2^{16} - 2)$, ויכולות להיות עד 16,384 (2^{14}) רשתות כאלה.



- **מחלקה C** - כתובות אלה יועדו לרשתות קטנות. הכתובת מתחילה בביטים "110", שאחריהם כתובת רשת בת 21 ביטים, וכתובת מחשב בת 8 ביטים. המספר המקסימלי של רשתות כאלה הוא 2,097,152 (2^{21}), ובכל רשת יכולים להיות עד 254 $(2^8 - 2)$ מחשבים שונים.

שתי מחלקות נוספות שהוגדרו בפרוטוקול הן:
- מחלקה D - כתובות שמתחילות בביטים "1110" ומיועדות לשימוש ב- IP Multicasting
- מחלקה E - כתובות שמתחילות בביטים "1111", ומיועדות לשימושים ניסויים.

חיסרון- כתובות מבוססות מחלקות —> מבוזבז.

CIDR: Classless Inter-Domain Routing (תרגום חופשי: ניתוב פנים תחומי ללא שימוש בקלאסים)-

מאפשר גמישות ביצירת כתובות בכך שמבצעים חלוקה של הביטים בכתובת ה- IP.

- חלק התת-רשת (Subnet) של כתובת האורח (Host) הוא שרירותי.
- פורמט הכתובת: a.b.c.d/x, כאשר x הוא הביטים המשותפים שמוקצים ל- Subnet.
- נשים לב כי לא משתמשים בכתובת הראשונה והאחרונה של ה- Subnet.
- כתובות פנימיות (באותו ה- Subnet) יכולות להיות זהות.
- מספר הכתובת = $2^m - 2$, כאשר m הוא מספר bit שמוקצה ל- Host.
- הנתב בוחר את ה- Subnet הכי ספציפיים- זה עם כמות ה- bit של ה- Subnet הגדול ביותר, כיוון שאז טווח הכתובות יותר מצומצם, כלומר ספציפי.
- ניתן לשלב 2 ממשקים לאחד:

200.23.16.0/23 = 11001000 00010111 00010000 | 00000000

200.23.17.0/24 = 11001000 00010111 00010001 | 00000000

200.23.16.0/23

כתובת ראשונה בטווח- 200.23.16.0 — 11001000 00010111 00010001 00000000

כתובת אחרונה בטווח- 200.23.17.255 — 11001000 00010111 00010001 00000000

כיצד ניתן לגלות בדרך יותר קלה מהי הכתובת האחרונה בטווח-

כתובת ראשונה בטווח- 200.23.16.0

כתובת אחרונה של ה- Host- + 0.0.1.255

200.23.17.255 ← כתובת אחרונה בטווח!

איך Host מקבל כתובת IP-

בצורה סטטית- מבזבז כתובות IP כאשר לא בשימוש.

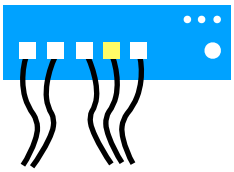
DHCP: Dynamic Host Configuration Protocol- (בתרגום חופשי: פרוטוקול הגדרת מארחים דינמי) פרוטוקול תקשורת המשמש להקצאה של כתובות IP ייחודיות למחשבים ברשת מקומית (LAN). בנוסף לכתובת ה-IP, שרת DHCP בדרך כלל יספק למחשב גם נתונים כמו ה-Subnet mask, כתובת שרת ה-DNS וכתובת שער הגישה (Gateway), כך שהמחשב יוכל להתחיל לתפקד ברשת ללא צורך בנתונים נוספים.

- קבלה של כתובת IP בצורה דינמית משרת DHCP.
- מאפשר ניהול בקלות יותר, טוב למחשבים שמשנים את מיקומם.
- חוסך כתובות IP- משתמשים רק בצורך, ניתן להשתמש מחדש בכתובת.
- בעיה במציאת רשת (כתובת משתנה ולא ידעו כיצד להשיג אותו)

כתובות IP מיוחדות-

- Localhost (מחשב זה)/כתובת Loopback- כתובת המחשב שאני עובד עליו.
  כלומר, כתובת ה-hostname הניתנת לממשק הרשת הפונה אל תוך הציוד עצמו.
  הוא משמש כדי לגשת לשירותי הרשת הפועלים על ה- Host דרך ממשק הרשת Loopback.
  שימוש בממשק Loopback עוקף את כל חומרת ממשק הרשת המקומית.
- כאשר כל ה- bit של ה- Host אפסים- מתקבל כתובת הרשת.
- כאשר כל ה- bit של ה- Host אחדים- Broadcast Address
- כאשר הכתובת מהתבנית x.x.x.1- לרוב זוהי כתובתו של הנתב.
- 0.0.0.0/0- כל הכתובת.

אופן העבודה של נתב:



זמן השהיה (עיבוד) בנתב-
הזמן שבו הנתב בודק את תקינות הביטים שהתקבלו, ומחליט באיזה מהיציאות שלו להוציא את החבילה.

זמן המתנה בתור-
הזמן בו החבילה נמצאת בתור של הנתב וממתינה לטיפול.

Store & Forward (שמור והעבר)-
נתב הפועל באופן של "שמור והעבר" הוא נתב אשר יקרא את כל ההודעה, יוודא את תקינותה ורק אם היא תקינה יעביר אותה הלאה ליעדה.
שיטה זו דורשת שמירה זמנית של החבילות בזיכרון הנתב.

- בכל כניסה יש תור, כל החבילות שמגיעות מאותה כניסה מצטברות באותו התור.
- בכל יציאה יש תור, ביציאה מחוברים הקווים שיוצאים החוצה ודרכם ההודעות משודרות.
- כל כניסה + יציאה = קו תקשורת.
- תפקיד ה- CPU (מעבד) לבחון את החבילה ואת יעדה.
- בטבלאות הניתוב רשום:

Subnet	Interface

- השהיה בזמן שליחת החבילה נובע מהמתנה של החבילה בתור, וכן מהמתנה של קבלת החבילה בשלמותה.
- נתב לא יודע מה המסלול של החבילה ליעד, אותו "מעניין" לאן החבילה צריכה להגיע, מה התחנה הבאה.
- אם תור יתמלא והחבילה שמגיעה לנתב לא יכולה להיכנס לתור, או שהחבילה שהגיעה תלך לאיבוד או שאחת החבילות בתור תלך לאיבוד וזו שהגיעה תתפוס את מקומה- **Packet loss**.
- חבילה שאבדה יכולה להיות משודרת פעם נוספת ע"י הקודקוד הקודם, מהמערכת בנק' הקצה, או לא להיות משודרת שנית כלל.

Network Switching

Circuit switching (מיתוג מעגלים)-

שיטת ניתוב הפועלת כך שלפני העברת חבילות מידע בין תחנת המקור לתחנת היעד יש להקים "מעגל" (או ערוץ תקשורת) ייעודי להתקשרות (session) זו. כל עוד ההתקשרות ממשיכה, מידע אחר לא יכול לעבור באותו ערוץ תקשורת. רשתות טלפונים ישנות הן דוגמה מובהקת למיתוג מעגלים - היה צריך לבקש מהמרכזיה לחבר את המתקשר אל היעד בין אם ישירות או דרך מרכזיה אחרת, ובכל אופן בסופו של דבר היה צורך ליצור באופן פיזי מעגל חשמלי בין שני מכשירי הטלפון.

יתרונות- מסלול קבוע, כלומר אמינות, איכות מובטחת, משאבים ייעודים (אין שיתוף, טוב לתקשורת טלפונית).

חסרונות- נדרשת הכנה של שיחה, בזבוז משאבים (אם המעגל לא נמצא בשימוש עדיין שמור ל"שיחה").

Packet switching (מיתוג מנות/חבילות)-

שיטת ניתוב שלפיה מחלקים את החבילה שאותה יש לשלוח לחבילות קטנות ולאחר מכן הנתב שולח חבילה אחר חבילה אל היעד. לחבילות יש את כתובת ה- IP של מחשב היעד ובנוסף הן משתמשות בטלת הניתוב של הנתב.

בשיטה זו אין מסלול קבוע מראש, כל חבילה נשלחת במסלול המתאים לאותו רגע, ומנותב בין צמתים בקווי התקשורת.

כל צומת מיתוג מקבל את החבילה בשלמותה, בודק את תקינותה ומעביר הלאה.

בשיטה זו החבילות יכולות לעבור בנתיבים שונים ולהגיע ליעד לא לפי סדר יציאתן.

יתרונות- את הקצאת משאבים, ניצול משאבי הרשת, שיתוף משאבים ע"י כל הלקוחות, מימוש פשוט.

חסרונות- דרישת משאבים משותפת יכולה לחרוג מהכמות הזמינה בפועל, סתימות בתור- גרימה להמתנה לקבלת החבילה בשלמותה ואף לאובדן חבילות.

עיכוב בהעברה-

ככל שקצב השידור יותר מהיר, ההעברה יותר מהירה.
(זמן שידור)

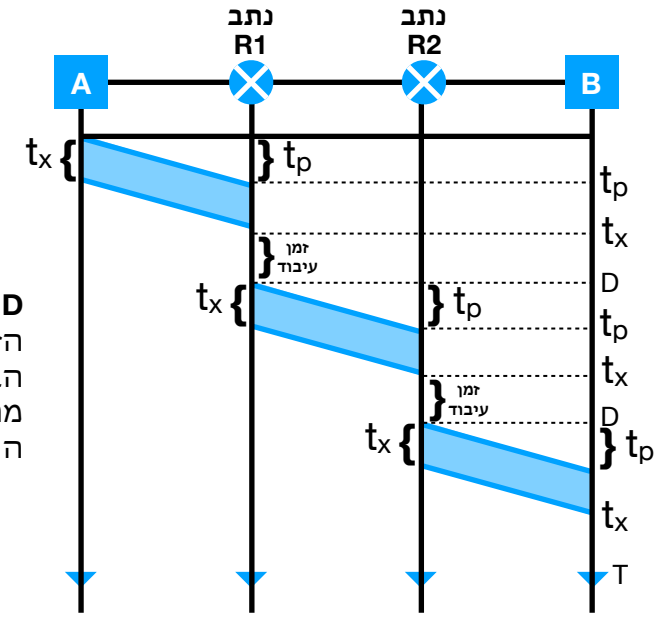
$$tx = \frac{F}{R} = \frac{\text{גודל הקובץ}}{\text{קצב השידור}}$$

עיכוב הפצה-

ככל שהמרחק יותר קטן, כך העיכוב ברשת קטן.
(זמן התפשטות/השהיה)

$$tp = \frac{L}{S} = \frac{\text{מרחק מהיעד}}{\text{קצב התפשטות}}$$

תרשים מעבר רגיל:

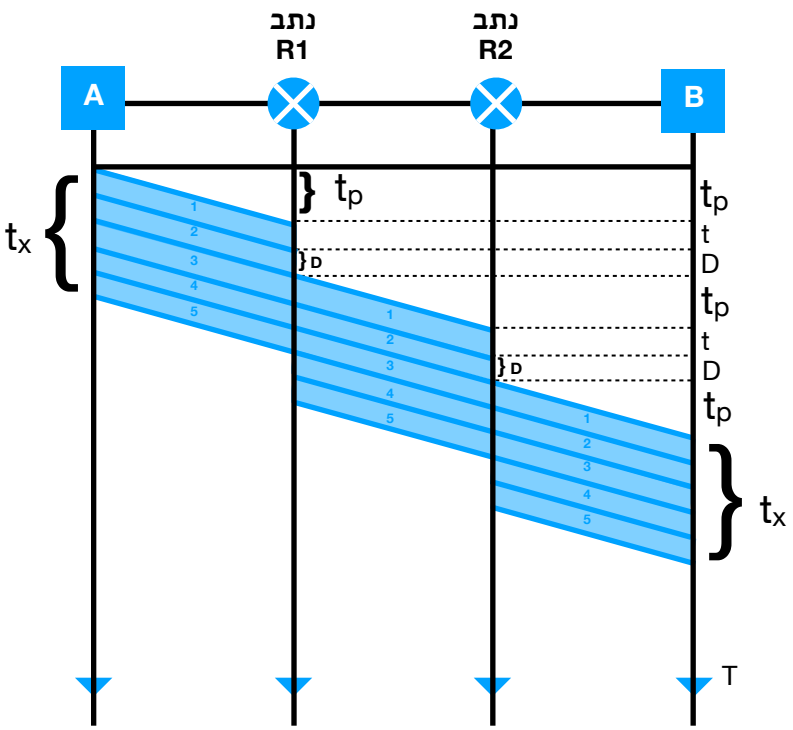


$$T = 3t_p + 3t_x + 2D$$

D = זמן עיבוד-

הזמן שהנתב בודק את תקינות הביטים שהתקבלו, ומחליט באיזה מהיציאות שלו להוציא את החבילה.

תרשים מיתוג מנות/חבילות:



$$T = 3t_P + 2D + \frac{7f}{R}$$

$$f = \frac{F}{5} \rightarrow t = \frac{f}{R} = \frac{F}{5R} \rightarrow t_x = 5t = \frac{F}{R}$$

המקביליות היא בכך שאני משדר מהמחשב הראשון בזמן שהמחשב השני מקבל וגם הוא מעביר.
לא צריך להמתין עד שהתחנה השנייה תקבל את ההודעה הראשונה כדי להעביר את ההודעה השנייה,
אלא, אני משדר את כולם ברצף וכך הם עוברים בצורה מקבילה.

הנתב עובד ב"שמור והעבר" (Store&Forward), אם אני מקטין את החבילה כך אני מצמצם את הזמן שהוא מתחיל לעבוד על כל חבילה. מקודם הנתב חיכה עד שקיבל את כל הקובץ כדי להתחיל לעבד וכאן זה נחסך.

מקודם- $T = 3t_P + 2D + \frac{15f}{R}$

עכשיו- $T = 3t_P + 2D + \frac{7f}{R}$

חסכתי מ- A לנתב הראשון $\frac{4f}{R}$

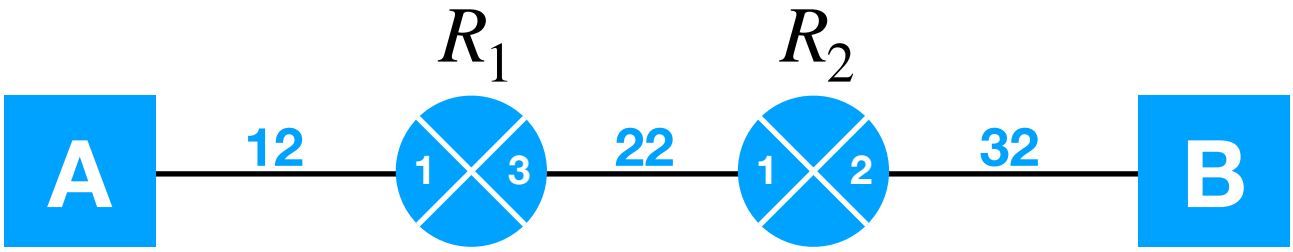
ומהנתב הראשון לשני חסכתי עוד $\frac{4f}{R}$

שזה בדיוק ההבדל בניהם.

Virtual Switching (מעגלים וירטואלים)-

שיטה בה איחדו את 2 השיטות הקודמות, חלוקה לחבילות + יצירת מסלולים.
החיבור הווירטואלי מורכב מ:

1. קביעת המעגל מקצה לקצה- נתיב בין המקור ליעד (הנתבים ידברו זה עם זה ויבטיחו משאבים).
2. א. כל חבילה נושאת זיהוי של חיבור וירטואלי (VC) ולא של מחשב היעד. (כי היעד ידוע לפי הנתיב, וכל הנתבים בדרך ידועים)
   ב. כל נתב במסלול של מקור-יעד, שומר את החיבור עבור כל חבילה שעוברת בטבלת הניתוב.
   ג. מספור של המעגל הווירטואלי, מספר אחד לכל ערוץ לאורך הנתיב כדי לדעת לאן לשלוח את החבילה (הערוץ נשאר בטבלת הניתוב)
3. סגירת מעגל, ה-VC נמחק.



טבלת ניתוב/כניסות R_1

In Interface	In VC	Out Interface	Out VC
1	12	3	22

טבלת ניתוב/כניסות R_2

In Interface	In VC	Out Interface	Out VC
1	22	2	32

המקור ששולח את ההודעה יכול לבקש מהרשת להקים VC בינו ובין היעד.
הרשת בוחרת את המסלול שבו תעבור ההודעה ונותנת לכל ערוץ מספר VC
ברגע שחבילה מגיעה עם מספר VC לאחד המתגים, היא צריכה להחליף מספר VC שיתאים לערוץ הבא.
תפקיד המתג הוא להחליף בעזרת טבלת הכניסות את מספר ה- VC של החבילה המגיעה מהערוץ הנכנס במספר VC חדש שיתאים לערוץ שבו החבילה יוצאת.
הטבלה מתעדכנת עם כל יצירת VC וכש-VC נמחק, הכניסות המתאימות נמחקות.

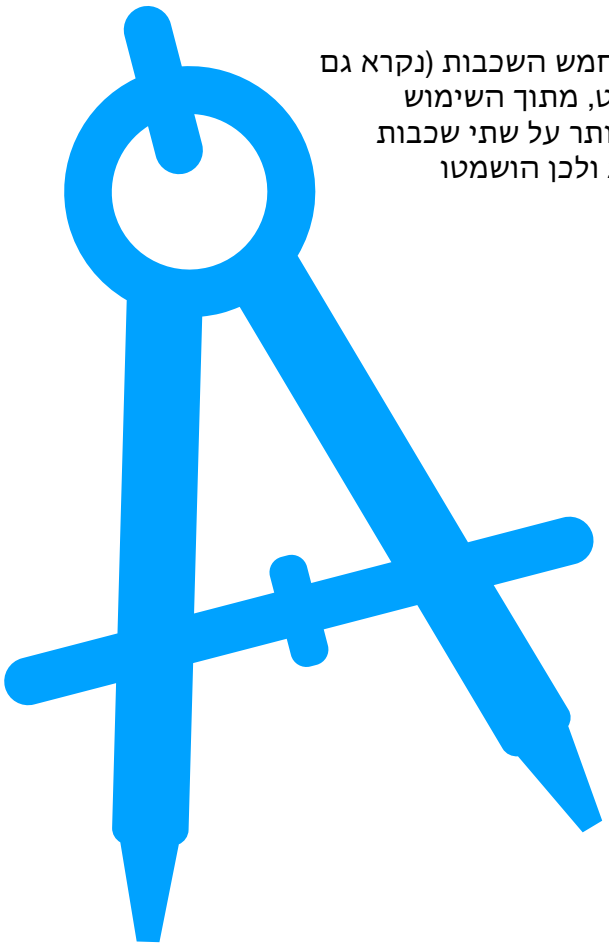
OSI Protocol Stack

האינטרנט גדול ומורכב מאוד, הוא מכיל אינספור ישויות (Entities). ישות ברשת היא כל דבר המחובר לרשת - בין אם זה סמארטפון, מחשב נייד, שרת של Google, רכיב רשת שנמצא בדרך בין ישויות אחרות, או רכיב בקרה של תחנת כוח המחובר גם הוא לרשת לצורך שליטה מרחוק. העברת המידע בין כל הישויות הללו זו משימה כלל לא פשוטה
נשאלת השאלה: איך אפשר לארגן את כל המידע הזה, כך שיאפשר למערכת המורכבת הזו לעבוד – ולעבוד בצורה טובה? הרי ברור שאם כל אחד ייתן את פתרונו נגיע למצב שבו כל ישות יודעת לדבר ב"שפה שלה" ואין אף "שפה משותפת" לכל רכיבי הרשת בעולם.
ובאמת כך קרה בפועל, נוצר מצב בו הרבה יצרניות חומרה שיווקו מכשירים אשר תומכים אך ורק בתקן אחד ספציפי (תקן אשר החברה עצמה ייצרה), מה שחייב את הלקוחות להמשיך ולרכוש מוצרים נוספים מאותה היצרנית אם הם היו רוצים לתקשר בין שני מכשירים שונים. ברור כי המצב הזה אינו רצוי, והוא אינו מאפשר למכשירים שונים באמת לדבר ב"שפה אחידה" המשמשת את כל רשת האינטרנט.

כדי לפתור את בעיה זו וליצור סטנדרטיזציה של המידע העובר על גבי רשת האינטרנט, יצר ארגון התקינה הבינלאומי ISO (International Organization for Standardization) (שאחראי על פיתוח ופרסום של תקנים בינלאומיים) את מודל שבע השכבות.
מטרתו של מודל זה, שנקרא OSI (Open Systems Intercommunications) הינה לתת קווים מנחים כיצד צריכה להיראות תקשורת בין כל מערכת מחשב אחת לשנייה, ללא תלות ביצרן של אותה מערכת.

שימו לב: מודל השכבות הינו מודל, ולא תקן. הוא לא מחייב את מערכות התקשורת לדבר בצורה הזו אחת עם השנייה, אלא מנחה את יצרניות החומרה כיצד לממש את מערכות התקשורת כך שתהיה אחידות בין כולם.

אנו נעשה שימוש במודל חמש השכבות, מודל TCP/IP.
ISO יצרו את מודל שבע השכבות באופן תיאורטי. מודל חמש השכבות (נקרא גם Protocol Stack) נוצר לאחר העבודה עם רשת האינטרנט, מתוך השימוש היישומי, והוא דומה למודל שבע השכבות אולם הוא מוותר על שתי שכבות (השכבה החמישית והשישית) שבפועל התגלו כמיותרות ולכן הושמטו מהמודל.



שכבת האפליקציה (Application)-

אפליקציות הם תוכנות שהמשתמש מפעיל ישירות כדי לתקשר באמצעות האינטרנט.
שכבת האפליקציה ממונה על אספקת שירותי הרשת לתוכנות בהן משתמש משתמש הקצה.
היא זו הקובעת את סוג התקשורת בין מחשבים. למשל, היא קובעת האם מדובר בתקשורת "שרת-לקוח" (client-server) או שמדובר דווקא בתקשורת "קצה לקצה" (peer to peer).
תפקיד השכבה- תקשורת עם משתמש, עיבוד נתונים בהתאם להחלטות שמבצע המשתמש וניהול תהליכים בין תחנת המוצע ליעד.
פרוטוקולים- FTP, HTTP, SMTP, POP3.

שכבת התעבורה (Transport)-

שכבה זו מספקת העברה של מידע מנקודת המוצא לנקודת היעד ברשת.
אחראית על ניהול התקשורת, אמינות החיבור, ואמינות הנתונים. כמו כן, היא משמשת את שכבת היישום (אפליקציה) של המודל, כל אפליקציה בוחרת את פרוטוקול התעבורה שמתאים לה. כמו כן משתמשת בשכבת הרשת לצורך העברת נתונים ברשת.
פרוטוקולים- TCP/UDP

שכבת הרשת (Network)-

אחראית על מיפוי לוגי של הרשת, וניתוב חבילות נתונים על פי מיפוי זה.
שכבת הרשת היא השכבה בה מתבצעות כל ההחלטות הנוגעות לדרך בה יועברו הנתונים על גבי הרשת, היא זו שתקבע האם קיים קשר בין המקור ליעד, היא תבחר באיזו דרך יעברו הנתונים (בהנחה שקיימת יותר מדרך אחת בין המקור ליעד) על פי שיקולים שונים, ביניהם - מהירות, נגישות, יעילות, עומס ועלויות.
שכבת הרשת משמשת את שכבת התעבורה ומשתמשת בשכבה הפיזית של המודל לצורך העברת נתונים על־גבי הרשת.
פרוטוקולים- IP

שכבת הערוץ/קו/קשר (Link)-

שכבה זו אחראית לתקשורת בין שתי תחנות קרובות שבאותה תת רשת לוקאלית (לא בהכרח קרובות פיזית, אלא נמצאות על קו אחד)
שכבת הקו אחראית על מעבר הסיביות בין שתי תחנות של הרשת. שכבת הקו אחראית על העברת המידע למרות רעשים או תקלות או התנגשויות שיש בקו. חבילות נתונים בשכבת הקו מכונות "מסגרות" (frames) ומכילות, בנוסף למידע ולתוספות הפרוטוקולים של השכבות הגבוהות יותר, את כתובות ה-MAC של תחנות המוצא והיעד.
משמשת את שכבת הרשת, ומשתמשת בשירותיה של השכבה הפיזית.
פרוטוקולים- Ethernet

שכבה פיזית (Physical)-

התווך הפיזי עליו עוברת התקשורת, למשל- חוט נחושת, כבל, סיב אופטי.
השכבה הפיזית אחראית על המיפוי הפיסי של הרשת, בין אם מדובר ברשת מקומית (LAN) או רשת אזורית (WAN). מיפוי זה נעשה באמצעות כתובות MAC, כתובות ייחודיות אלו משויכות לכל רכיב המסוגל לתקשר ברשת מחשבים, ומשמשות לזיהוי תחנת המוצא והיעד של חבילות נתונים.
השכבה הפיזית קובעת גם תקנים שונים ביחס לכבלים בהם יש להשתמש, אורכם המקסימלי, תעבורה מקסימלית, מחברים, ועוד. כל אלו חיוניים על מנת למנוע טעיות בזיהוי שעשויות להגרם בשל הפרעות שונות באותות הבינאריים.

שכבת האפליקציה

אפליקציות (יישומים, בעברית)- כינוי לתוכנות שבהן אנחנו עושים שימוש במחשב, וזהו מושג שנעשה הרבה יותר נפוץ ומוכר מאז שהתחיל השימוש הנרחב בסמארטפונים ובטאבלטים. גם יישומים שרצים על המחשב שלנו (דוגמה נפוצה - דפדפנים), וגם אפליקציות ב-iPhone או ב-Android שלנו (כמו Whatsapp או האפליקציה של Facebook), עושים שימוש בתקשורת דרך האינטרנט כדי לשלוח ולקבל הודעות (Whatsapp), להעלות תמונות (Facebook / Instagram), לקבל מיילים (Gmail) ועוד.

שכבת האפליקציה היא אוסף הפרוטוקולים בהם עושות האפליקציות שימוש באופן ישיר, והיא מספקת הפשטה מעל תקשורת הנתונים ברשת האינטרנט.

יישום רשת-

יישום רשת הוא יישום הבנוי במודל שרת-לקוח ונגיש דרך רשת מחשבים, כגון רשת האינטרנט או אינטראנט. לרוב, יישומי רשת משתמשים בדפדפן כתוכנת לקוח, והם מורכבים מדפי אינטרנט דינמיים המסתמכים על תוכנה שמורצת הן בצד השרת והן בצד הלקוח.
לדוגמה- דואר אלקטרוני, גוגל מפות, FTP.

פרוטוקול שכבת היישום-

פרוטוקול התקשורת עבור יישומי רשת.
הפרוטוקול משתמש בשכבת התעבורה כדי להעביר את תקשורת זו בשימוש ב-UDP/TCP.
הפרוטוקול מגדיר את ההודעות ואת תהליך עיבודן.

פרוטוקול אפליקציה צריך להגדיר:

1. חוקים לגבי מתי ואיך מתקשרים.
2. סוג התקשורת, לדומה- בקשה ותגובה, הודעות.
3. פורמט ההודעות, כלומר אילו שדות ואיך שדות מופרדים.
4. הגדרות סמנטיות של השדות, לדוגמה- מה המשמעות של מידע בשדה מסוים.

תהליך (Process)-

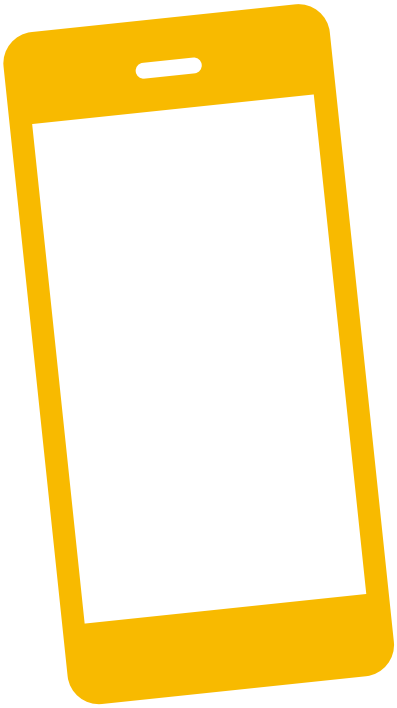
מופע של יישום שמופעל על ידי מערכת הפעלה שיש לה היכולת להפעיל מספר תהליכים בו זמנית. תכנית מחשב היא בעצמה רק אוסף פקודות, בעוד שתהליך הוא ההפעלה של אותן פקודות.

-תהליך לקוח- תהליך שמתחיל את כל ההתקשרות.
-תהליך שרת- תהליך שממתין ליצירת קשר.

- ביישומי P2P בעלי תהליך ותהליך שרת- כל קצה שולח ומבקש.
- בשביל שתהליך יקבל הודעות, חייב להיות לו מזהה:

כתובת IP
לזיהוי ה-Host

פורט
המקושר לתהליך
על ה- Host



שכבת האפליקציה פועלת עם שכבת התעבורה,
לכן בחירת שירות התעבורה הרלוונטי אליה יבחר על פי-

אמינות המידע-
אבדן חבילות, סדר הגעה.
ישנם יישומים שאינם מאפשרים אבדן מידע, לדוגמה העברת קבצים, דואר אלקטרוני.
ארי יש ישומים שמאפשרים אבדן עד רמה מסוימת, לדוגמה אודיו, וידאו.

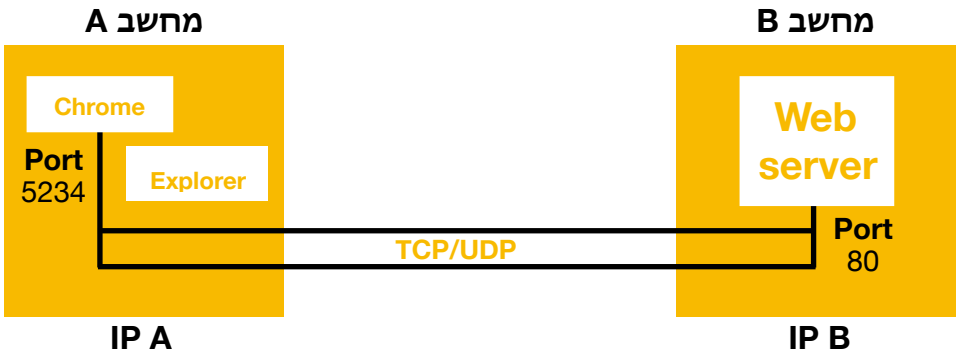
תקשורת Setup-
ישנם ישומים דלילים שתקשורת ההקמה שלהם משמעותית אך הם שולחים הודעה קטנה מדי פעם. אולם מרבית היישומים תקורת הקמתם זניחה- "מאריכי חיים", מקימים את התקשורת פעם אחת והיא רציפה לזמן ממושך.

רוחב פס ועיבוד-
ישנם יישומים גמישים המשתמשים ברוחב הפס שהם מקבלים
ואחרים לעומת זאת דורשים איכות מינימלית של שירות רוחב הפס.
לדוגמה דוא"ל, web- גמישים
וידאו, אודיו- לא גמישים

גודל וסיבוכיות-
גודל הקוד והסיבוכיות שלו יכולים להיות משמעותיים ויש לבחור פרוטוקול מתאים לכך.

מתי נבחר להשתמש ב-UDP ומתי ב-TCP עבור האפליקציה שלנו?
(על UDP/TCP בהרחבה בשכבת התעבורה)

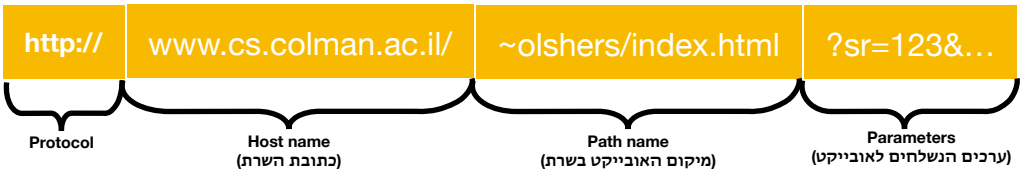
TCP	UDP
● כאשר האפליקציה שולחת קבצים גדולים- רגיש לאמינות המידע וגמיש מבחינת קצב העברת הנתונים.	● רוחב הפס רב יותר מ-TCP
● אפליקציה רגישה לשלמות המידע.	● אפליקציה שולחת מעט הודעות קצרות
● אפליקציה לא רגישה לזמן שידור.	● אפליקציה רגישה לזמן שידור.
● שליטה בעומס- מתאים את הקצב ליכולת של הרשת.	● אין שליטה בעומס או בזרימה
● שליטה בזרימה- בקרה בנוגע לקצב העברת הנתונים בין 2 תחנות בכדי שהודעות לא ילכו לאיבוד.	
● תעבורה אמינה- כל הודעה שנשלחת תגיע לצד השני בסדר שהיא נשלחה.	



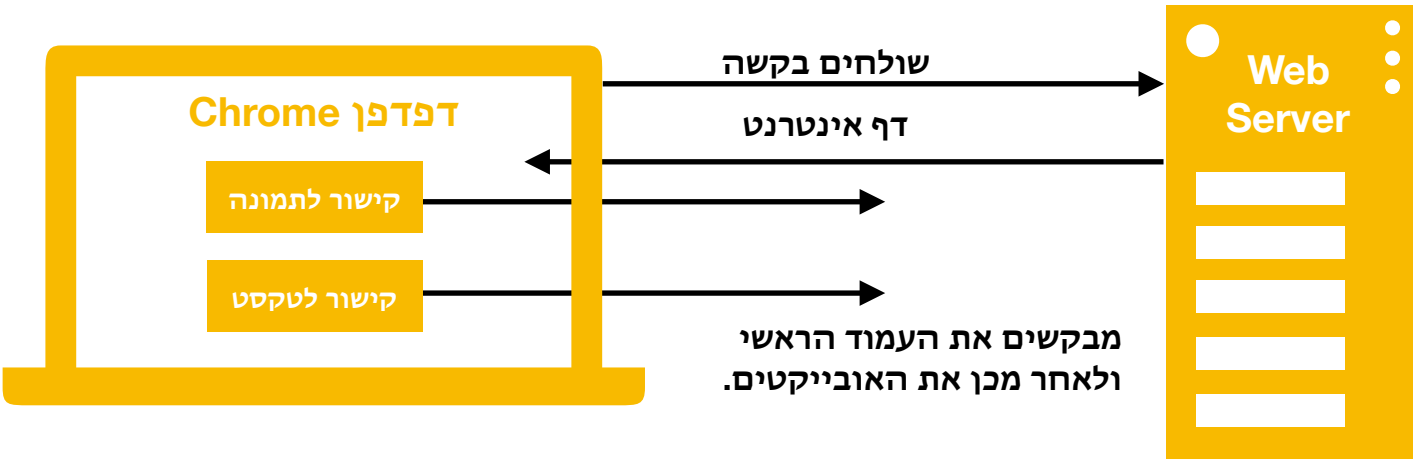
- כאשר אנו פותחים ערוץ תקשורת בין 2 תהליכים, הכתובות שמצינות את הקצוות של הערוץ הם ה-IP וה-Port. כלומר, הצירוף של ה-IP וה-Port מגדיר לי כתובת של תהליך.
- ניתן שיהיה כפילות בין מחשבים במספר הפורט (בניגוד לכתובת IP)
- כל אפליקציה מגדירה פורט משלה.
  מספר הפורט חייב להיות קבוע בשרת ולכן עבור פרוטוקולים סטנדרטים ישנה קבוצת פורטים קבועה.

HTTP

- HTML- Hyper Text Markup Language (שפת סימני עריכה לתמללי-על), הינה שפת תגיות לתצוגה ועיצוב דפי אינטרנט והתוכן המוצג בדפדפן. זו שפת התגיות המרכזית בעולם האינטרנט, המהווה שלד למרבית עמודי התוכן באינטרנט. השפה מאפשרת עיצוב תוכן בצורה מהירה, קלה ללימוד באופן יחסי וקלה לכתיבה.
- דף web (דף אינטרנט) מורכב אובייקטים המכילים קובץ HTML, תמונות JPEG, קובצי וידאו וכו'.
  לרוב, דף web מורכב מקובץ HTML. בסיסי (דף בסיסי) אשר בדרך כלל מכיל הפניות למספר אובייקטים, כלומר קישורים לאובייקטים המרכיבים את דף האינטרנט.
  הפנייה לאובייקט נעשית באמצעות URI/URL.



- HTTP (Hypertext Transfer Protocol)- הוא פרוטוקול תקשורת שנועד להעברת דפי HTML ואובייקטים שהם מכילים (כמו תמונות, קובצי קול, סרטוני פלאש וכו') ברשת האינטרנט וברשתות אינטראנט. שרתי HTTP הם שרתי התוכן המרכזיים ברשת האינטרנט ודפדפנים הם תוכנות הלקוח הנפוצות ביותר לפרוטוקול HTTP, הדפדפן הוא זה שיודע לשלוח את הבקשה, לקבל בחזרה את הדף ולהציג במסך.



סקירה על HTTP:

הלקוח-

הלקוח יוזם תקשורת אמינה (מבוססת TCP) ולכן מחויב בשיטה פתוחת קשר.
הלקוח שולח את הודעת החיבור אל כתובת ה-IP ולפורט שבו השרת נמצא, בדרך כלל פורט 80.
לאחר מכן נשלחת הבקשה, הכוללת את הכתובת של האובייקט המבוקש (למשל, דף HTML) ופרטים נוספים על הבקשה ועל הלקוח.

השרת-

השרת מקבל חיבור TCP מהלקוח, קורא את הבקשה, מפענח אותה, שולח ללקוח תשובה בהתאם ולרוב מנתק את החיבור ללקוח כשהשליחה הסתיימה.

- פרוטוקול HTTP הינו stateless ("חסר מצב"), הפרוטוקול מתייחס לכל בקשה כבקשה עצמאית שאינה קשורה לבקשות הקודמות.
  כלומר, השרת אינו שומר מידע על בקשות קודמות של הלקוח.

תהליך ההתקשרות-

1. לקוח יוזם חיבור TCP (יוצר Socket) לשרת ה-Web דרך פורט 80.
2. שרת ה-Web מקבל חיבור TCP מלקוח (דפדפן)
3. מוחלפות הודעות HTTP (request/responses)
4. שירות TCP נסגר.

בקשות ב- HTTP-

1. **Get-** בקשה שמיועדת לקבלת מידע מהשרת.
   (קבלת אובייקט שנמצא בשרת בכתובת שצוינה בהודעת הבקשה).
   בהודעת ה-Get אין משמעות לגודל ההודעה, כאשר נרצה להוסיף מידע מעבר לשם הדף עלינו להוסיף זאת ב- URL באמצעות תוספת של "&", כך ניתן לשלוח נתונים אל השרת אך בהגבלה של כמות הנתונים והפורמט.
   בשל תוספת מידע ב-URL המידע הנוסף אינו חשוף.

2. **Post-** נועד להעביר מידע לשרת מהלקוח.
   המידע שנרצה להעביר יהיה בגוף ההודעה.
   ניתן לשים איזה מידע שנרצה, בכל פורמט וגודל רצוי.
   המידע שנשלח מוסתר משום שהוא לאחר ה-Header.

תגובות ב- HTTP-

1. **200 (OK)-** קוד מצב הנפוץ ביותר, מציין שהבקשה עובדה בהצלחה על ידי השרת, האובייקט נמצא בהמשך הודעה זו.

2. **301 (Moved Permanently)-** האובייקט המבוקש עבר מיקום והמיקום החדש מפורט בהמשך לתשובת השרת תחת Location.

3. **400 (Bad Request)-** השרת לא הצליח לפענח את הבקשה, ככל הנראה משום שהיא אינה כתובה בהתאם לפרוטוקול.

4. **404 (Not Found)-** הכתובת המצוינת בבקשה לא תואמת אף אובייקט שנמצא על השרת. קוד זה משמש גם במקרים שבהם השרת לא ממלא אחר הבקשה של הלקוח, אבל לא מעוניין לחשוף את הסיבות לכך.

5. **505 (Version Not Supported)-**
   הבקשה עשתה שימוש בגרסת HTTP שאינה נתמכת ע"י השרת.

מעט היסטוריה..

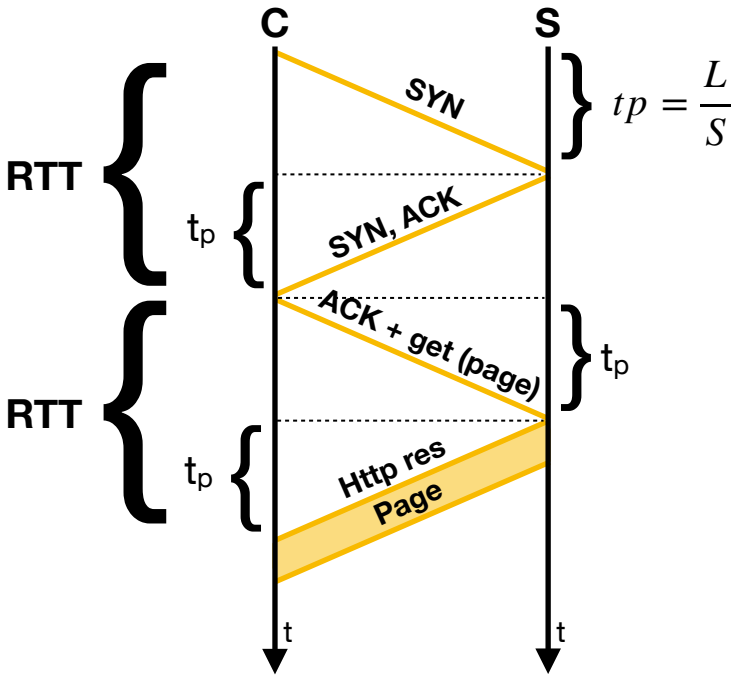
ב-1991 פורסמה הגרסה הראשונה של הפרוטוקול שהייתה בשימוש (0.9). גרסה זו הייתה פשוטה ביותר ולא תמכה ברוב האפשרויות הקיימות כיום. שיטת הבקשה היחידה בגרסה זו הייתה GET, לא הוגדרו שדות כותרת והבקשות לא כללו את גרסת ה-HTTP כפי שהן כיום.

במאי 1996 התפרסמה גרסה 1.0 של הפרוטוקול, שנמצאת עדיין בשימוש נרחב בעיקר על ידי שרתי פרוקסי. בגרסה זו נוספו שיטות הבקשה HEAD, POST, PUT, DELETE, LINK ו-UNLINK ובקשות נדרשו לציין את גרסת הפרוטוקול.

הגרסה הנוכחית, HTTP/1.1, פורסמה ביוני 1999 והתבססה ברובה על גרסה 1.0. ההבדלים העיקריים בין גרסה זו לקודמתה הם שליטה טובה יותר במטמון, הוספת שיטות הבקשה OPTIONS ו-TRACE, תמיכה ב-virtual hosts, ותוספות מתקדמות שמטרתן לייעל את אופן הפעולה של הפרוטוקול.
מאפיין חשוב שנתמך בגרסה זו הוא האפשרות להשתמש בחיבור יחיד עבור מספר בקשות, במקום לפתוח חיבור חדש עבור כל אובייקט שנמצא בדף שהתקבל בתשובה הראשונה.
נעשו ניסיונות להרחיב את הפרוטוקול וליצור גרסה 1.2, אך הם לא יצאו לפועל.

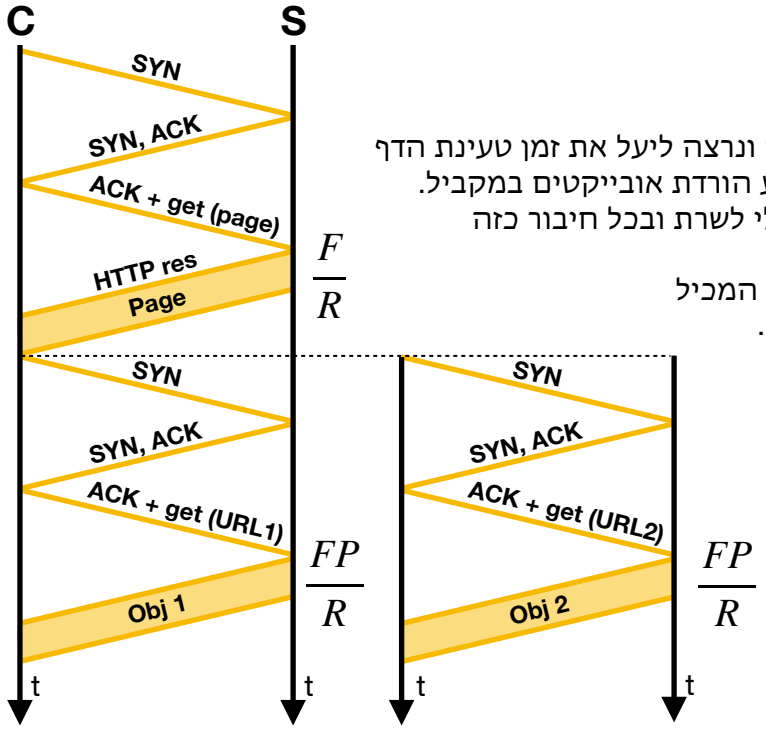
בפברואר 2015 אושרה גרסה HTTP/2 אשר מבוסס על פרוטוקול הרשת SPDY שפותח במקור על ידי גוגל, אותו פרוטוקול מכוון להפחתה של זמן השהייה בתעבורת המידע וכן להגברת אבטחת המידע הזורם ברשת.

HTTP 1.0 לא עקבי-
יוצרים חיבור TCP, מעבירים אובייקט אחד וסוגרים את החיבור.



$$tp = \frac{L}{S}$$

$$T = 2RTT + \frac{F}{R}$$

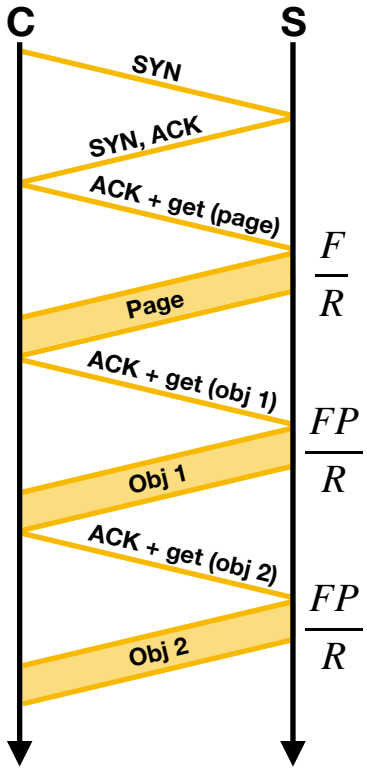


HTTP לא עקבי, מקבילי-

כיוון שדף אינטרנט מכיל מספר אובייקטים ונרצה ליעל את זמן טעינת הדף למחשב מבלי לשנות את הפרוטוקול, נבצע הורדת אובייקטים במקביל. הדפדפן ייצור מספר חיבורים באופן מקבילי לשרת ובכל חיבור כזה יבקש אובייקט אחד.
כמובן שתחילה יש צורך בקבלת דף HTML המכיל את מספר האובייקטים והקישורים אליהם.
האובייקטים יורדים במקביל אך הם חולקים את קו התקשורת עם השרת, כלומר את אותו רוחב הפס.

$$T = \underbrace{2RTT + \frac{F}{R}}_{\text{הדף הראשי}} + \underbrace{2RTT}_{\text{נלקח 1 עבור כל מה שמבוצע במקביל}} + \left.\frac{\frac{FP}{R}}{2}\right\} \text{קצב השידור קטן ב-2}$$

$$T = 4RTT + \frac{F}{R} + \frac{2FP}{R}$$

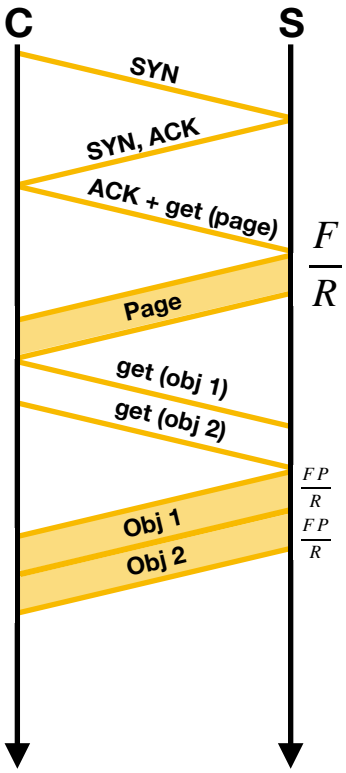


HTTP עקבי לא מקבילי-

בעזרת שיפור זה, משאירים את חיבור ה-TCP פתוח מול השרת במקום לסגור אותו אחרי כל בקשה, ובכך משפרים את זמן טעינת הדף למחשב (שינוי בפרוטוקול).

$$T = 2RTT + \frac{F}{R} + 2 \cdot (RTT + \frac{FP}{R})$$

$$T = 4RTT + \frac{F}{R} + \frac{2FP}{R}$$



HTTP 1.1 עקבי ומקבילי (Pipline)-

בשיפור זה ניתן לשלוח בקשות מבלי להמתין לקבלת האובייקט.

$$T = \underbrace{2RTT + \frac{F}{R}}_{\text{הדף הראשי}} + \underbrace{RTT}_{\text{RTT אחד עבור כל האובייקטים}} + \frac{2FP}{R}$$

Web Session

session היא החלפת מידע אינטראקטיבית חצי-קבועה, הידועה גם כ"שיחה", "התקשרות", "דיאלוג" או "פגישה", בין שני מכשירים מתקשרים או יותר, או בין מחשב למשתמש קצה. סשן מוקם או מבוסס (established) בנקודה מסוימת בזמן, ונהרס בנקודה מאוחרת יותר.
בדרך כלל, אבל לא תמיד, סשן הוא stateful, כלומר, לפחות אחד מהצדדים המתקשרים צריך לשמור מידע אודות ההיסטוריה של השיחה על מנת שיוכל לתקשר,
וזאת בניגוד לתקשורת stateless, המורכבת מבקשות ותגובות נפרדות.

Web session הוא האפשרות שהשרת יעקוב אחר בקשות המשתמש, וידע מי שלח את הבקשה הספציפית ומה היו הבקשות הקודמות של אותו משתמש.
לדוגמה, לצורך ניהול עגלת קניות או חשבון בנק.

כיצד אפשר לנהל Session ב- HTTP-

1. **לפי חיבור ה-TCP, חיבור עקבי-** לא טוב משום שהדבר מחייב את השרת לשמור חיבורים פתוחים עבור כל המשתמשים וכן מחייב את כל המשתמשים לשמור את החיבור פתוח. הדבר מסבך את שרת ה-web.

2. **לפי כתובת ה-IP של המשתמש-** לא טוב משום שהמשתמש מקבל כל פעם כתובת חדשה.

3. **שימוש בפרמטרים הנשלחים ב-URL (הוספת id)-** השרת יוסיף בתגובתו לבקשת הלקוח מספר מזהה (id) כך שיוכל לשייך את מספר ה- id למשתמש מסוים ולפי זה לעקוב אחר הפעולות של המשתמש. אך גם שיטה זו אינה טובה, משום שבמידה והמשתמש יפתח דף או לחלופין יסגור את הדפדפן הוא יקבל מספר id חדש ואיו לשרת דרך לעקוב אחריו.

אז מה בעצם צריך בכדי לזהות את המשתמש בשרת ולנהל Web Session-
יש צורך במזהה ייחודי ללקוח אשר קבוע ללקוח ולא תלוי בכתובת IP או בחיבור TCP או בדף שאליו הלקוח נכנס באתר מסוים.
על המזהה להישמר במחשב הלקוח ולהישלח בכל פעם ששולחים בקשה לשרת.

Cookies

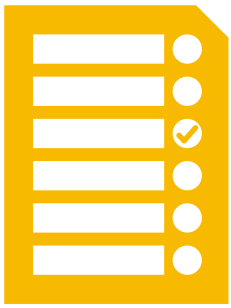
Cookie (עוגייה) היא מחרוזת אותיות או מספרים, המשמשת לאימות, למעקב ולאגירת מידע על אודות גולש באתר אינטרנט, כגון שמירת העדפות המשתמש. העוגייה מייצגת מצב מסוים של גלישה באתר או שימוש באפליקציה. לעתים, גם אתרים שאינם האתר שיצר את העוגייה יכולים להשתמש בה.
העוגייה נוצרת על ידי השרת כאשר מתקבלת הודעה ראשונה מלקוח, השרת שמעביר אותה לדפדפן ששומר אותה בזיכרון המחשב.

המחרוזת מוחזרת חזרה לשרת בכל פעם שהדפדפן יוצר קשר עם השרת. כך למעשה יכול השרת לזהות את המשתמש ולאחזר מידע שנשמר בין שיחות שונות. כאשר הדפדפן ישלח הודעת get נוספת,
הוא יפנה אל הקובץ השמור אצלו ויחפש את כתובת ה- URL של האתר. במידה וקיים בקובץ Cookie המתאים לאותה כתובת URL, הוא ישלוף אותה מהקובץ ויצרף ל-Header של HTTP תחת שדה ה-Cookie.

Web Cash (Proxy Server)

דרך אחת "להאיץ" את חוויית השימוש באינטרנט היא באמצעות ההבנה שיש משאבים שאותם מבקשים שוב ושוב, וכך למרות שאין בהם כל שינוי, המידע שלהם עובר מספר רב של פעמים על גבי רשת האינטרנט.

Cash (מטמון)-



הוא אוסף נתונים על בסיס ערכים מקוריים אשר מאוחסנים (מוטמנים) במיקום אחר, או שהופקו קודם לכן באמצעות חישוב כלשהו. השימוש במטמון מאפשר שליפה מחודשת של המידע במהירות במקום לחזור אל המאגר המקורי שהוא יחסית איטי או מרוחק. השימוש מאפשר גם לאחזר את הנתונים הללו במהירות ובהשקעה מינימלית.

לצורך הדוגמה, חישבו על כך שבכל פעם שאתם גולשים לאתר Ynet כדי להתעדכן בחדשות, תמונת הלוגו של Ynet תעבור מהשרת אל הדפדפן שלכם. הדבר נכון גם לגבי שאר הכותרות ואמצעי העיצוב שקבועים בדף - כל זה גורם לתעבורה "מיותרת", שהרי המידע היה כבר בדפדפן שלכם בעבר, ומאז הוא לא השתנה. התעבורה המיותרת גורמת גם לבזבוז של עלויות
(רוחב הפס וכמות המידע שעוברת על גביו), וגם של זמן (בהמתנה למשאב שיגיע).
הרעיון של מנגנון ה-Cache הוא לשמור משאבים כאלה על המחשב של הלקוח, וכל עוד הם לא משתנים, לטעון אותם מהדיסק המקומי ולא על גבי הרשת.

- המטמון פועל גם בלקוח וגם בשרת.
- מטמון מותקן בדרך כלל על ידי ספק שירות האינטרנט.
- מטמון יכול לבדוק האם האובייקט עדכני באמצעות פרוטוקול ה-HTTP בכך שיעביר לשרת (יחד עם בקשת ה-GET) את הזמן בו שמר את המשאב. השרת יחזיר את המשאב רק אם יש לו גרסה חדשה יותר (כלומר, שנוצרה לאחר הזמן שמצוין בבקשה).

יתרונות- הקטנת זמן תגובה לבקשת לקוח, צמצום תעבורה על קו תקשורת של הארגון, הורדת עומס על ספק התוכן.
חסרונות- דורש גישה לכל אובייקט, נכשל בדפים דינמיים.

Proxy Server (בתרגום חופשי: נציג, שליח, בא-כח)-

הוא שרת שתפקידו העיקרי לספק גישה מהירה למשאבים חיצוניים ברשת מחשבים.
תפקידו העיקרי של שרת הפרוקסי מתאפשר באמצעות תוכנה השומרת בזיכרון המטמון שלו עותקים של דפי אינטרנט ומאפשרת גישה מהירה אליהם מהמחשבים המחוברים לפרוקסי.
בחלק מהמקרים הפרוקסי משמש נקודת הפרדה וביקורת בין רשת המחשבים הפנימית של הארגון אליו הוא שייך לבין העולם החיצוני. הוא מאפשר מעקב, חסימה או שינוי לדפי HTML על פי מדיניות הארגון בענייני אבטחת מידע ותכני מידע קבילים (אפשר, למשל, לחסום אתרים פורנוגרפיים).

Security and HTTP

HTTP Authentication

על מנת להתמודד עם איומים כמו חדירה לרשת או קבלת מידע שגוי מגורמים מתחזים,
יש לוודא את זהותם של הגורמים עימם מתקשרים.
שלב האותנטיקציה (אימות) חשוב משום שבכל מערכת שמנהלת מידע של משתמשים,
בדרך כלל תשמור גם מידע על אותם משתמשים ועל המערכת לשמור על המידע
הפרטי שלא יזלוג למשתמשים אחרים או לגורמים אחרים לא מורשים.
הזיהוי בשיטה זו נעשה באמצעות שם משתמש וסיסמה.

תהליך ההתקשרות-

1. הלקוח שולח בקשת Get ל- URL מסוים שנמצא בשרת. ה-URL על השרת מסומן כ- URL עם Access Control כלומר, מי שניגש לדף זה חייב לעבור תהליך אימות.
2. השרת שולח לדפדפן הודעת Response עם בקשב לשם משתמש וסיסמה.
3. המשתמש מכניס את שם המשתמש והסיסמה, הדפדפן מעביר אותם קידוד בסיסי ומכניס אותם ל-Header לתוך שדה בשם Authentication ושולח בחזרה לשרת.
4. השרת מקבל את שם המשתמש והסיסמה ובודק מול ה- DB שלו האם הפרטים תקינים. במידה וכן, השרת שולח את הדף המבוקש.

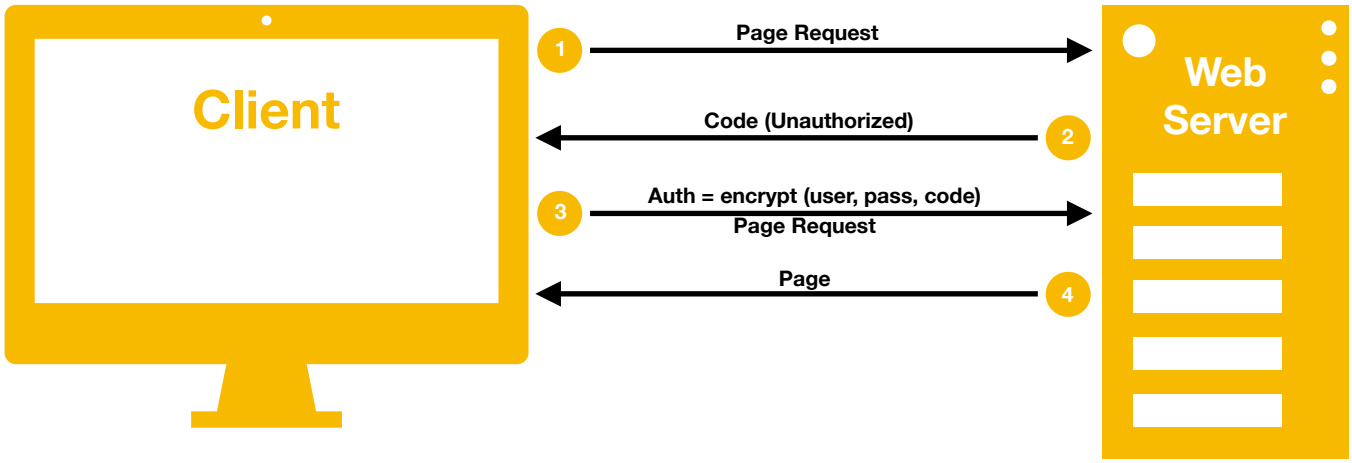
יתרונות- פשוט (בסך הכל שדה אחד נוסף ב-Header), מספק אמצעי לאימות משתמש.
חסרונות- שם המשתמש והסיסמה אינם מוצפנים, אין אימות של שרת ה-Web, פגיע להתקפות.

HTTP Digest Authentication

אחת השיטות המוסכמות של שרת אינטרנט המאפשר הצפנת פרטי משתמש.
הפרוטוקול פועל בעזרת פונקציה מסוימת (Hash) אשר מצפינה את הסיסמה לפני שליחתה לשרת.
במהלך שיטה זו מתקבל מהשרת Code אשר תלוי בזמן ומשתנה תמיד.
הלקוח מצפין רת שם המשתמש, הסיסמה וה-Code שאותו קיבל בעזרת פונקציית ה- Hash.
בשיטה זו ההצפנה עובדת בכיוון אחד.

לפונקציית ה-Hash יש 3 שימושים-

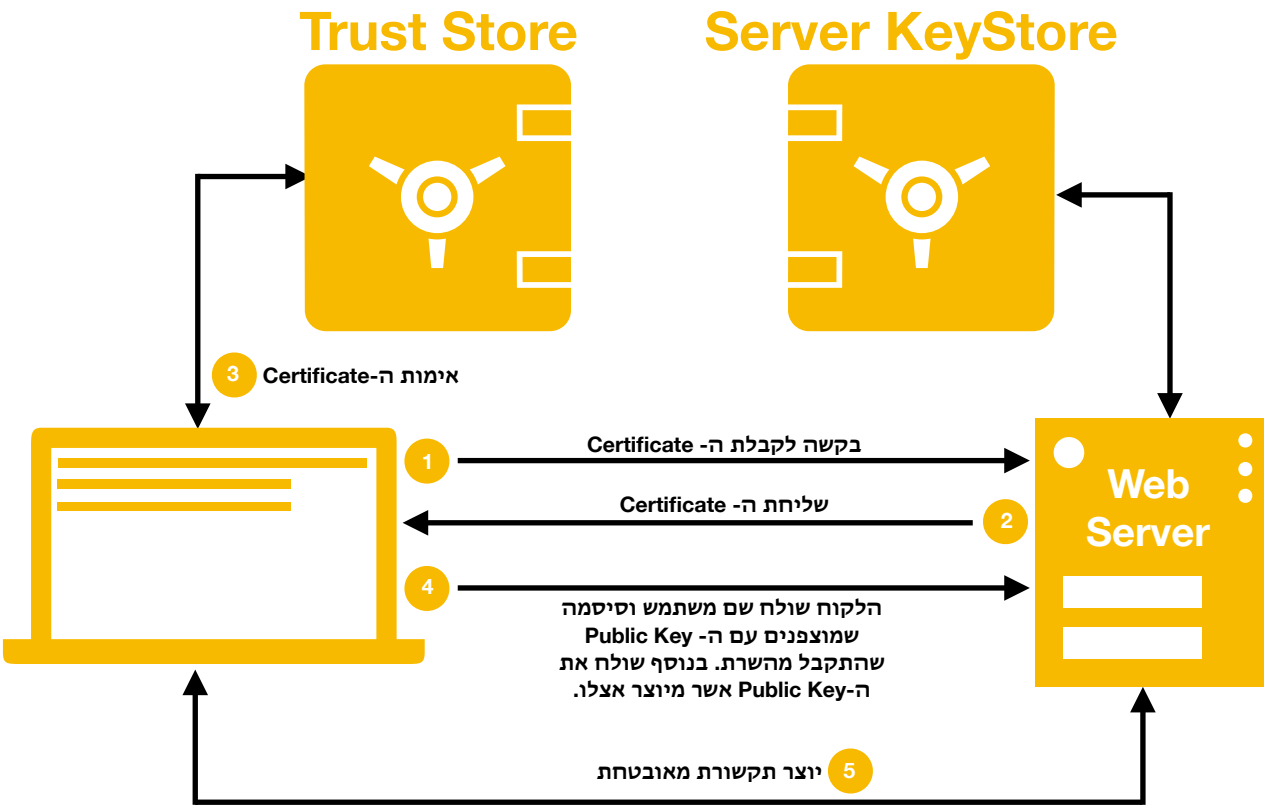
1. חיפוש.
2. הצפנה.
3. אימות.



HTTPS (S = Secure)

פרוטוקול תקשורת נפוץ במיוחד לאבטחת מידע ברשתות מחשבים, ומיושם במיוחד באינטרנט. באופן רשמי, הוא אינו פרוטוקול תקשורת כשלעצמו, אלא שימוש בפרוטוקול HTTP על שכבת SSL/TLS (תעבורה) ובכך מקנה יכולות אבטחה לתקשורת HTTP רגילה.

בשיטה זו מניחים כי ישנו גורם שלישי אשר הוא הישות האמינה ובלעדיו המנגנון לא יכול להתקיים.
הגורם השלישי הינו ה- Trust-Store אשר כלל הישויות מכירות בו והוא אחראי על האימות.
בתום הרישום ב- Trust Store נוצר מסמך אימות בשם certificate אשר מאמת את ה- URL של אותו אתר ואת ה-Public key הדרוש להצפנה.
כלומר, בשיטה זו אנו מאמתים את זהותו של האתר אליו אנו גולשים ומוודאים עי החיבור לשרת הינו אמין.
מעבר לכך, שיטה זו מאפשרת הצפנה דו כיוונית של תעבורת מידע בין הלקוח לשרת אשר מגינה מפני האזנות סתר או שינויים של תוכן המידע העובר בין הצדדים.



Public Key - קובץ → אלגוריתם הצפנה → קובץ מוצפן

Private Key - קובץ מוצפן → אלגוריתם פענוח → קובץ מפוענח

יתרונות- תקשורת שרת - לקוח מאובטחת.
חסרונות- מסובך, יש לסמוך על רשת אישורים מרכזית (ה- Trust Store)

DNS - Domain Name System

שמות של דומיינים (google.com) נבחרו להיכתב מילולית כמילים וביטויים על מנת ליצור נוחות ופשטות עבור בני האדם. אך "מאחורי הקלעים" אנו עובדים עם כתובות IP ולפיכך קיים הצורך לתרגם שמות דומיין לכתובות IP. הדבר דומה לדרך בה אנו שומרים מספרי טלפון: כאשר אנו מתקשרים לאדם, בדרך כלל נעדיף שהטלפון יזכור עבורנו את המספר שלו, וכך נעדיף לחייג אל משה כהן מאשר לזכור את המספר 054-1234567. עם זאת, כשם שהטלפון צריך בסופו של דבר להשתמש במספר בכדי להתקשר למשה, כך גם המחשב צריך לדעת את כתובת ה-IP של מחשב היעד בכדי לפנות אליו.

על מנגנות ה-DNS לעמוד בדרישות הבאות:

1. תרגום שמות דומיין לכתובות IP.
2. שירות מספר עצום של שמות וכתובות.
3. שיורת מספר עצום של מחשבים.
4. מאפשר שינוי ועדכון כתובות IP כיוון ששרתים נופלים וקמים ולכל דומיין יכולים להיות מספר כתובות פעילות.
5. מאפשר הוספת דומיינים חדשים.

בשל כך, ברור כי שרת אחד אינו יעמוד בעומס הדרישות ולכן פיתחו מערכת היררכית של שרתי DNS. (עובד מעל UDP)

היררכיית שמות

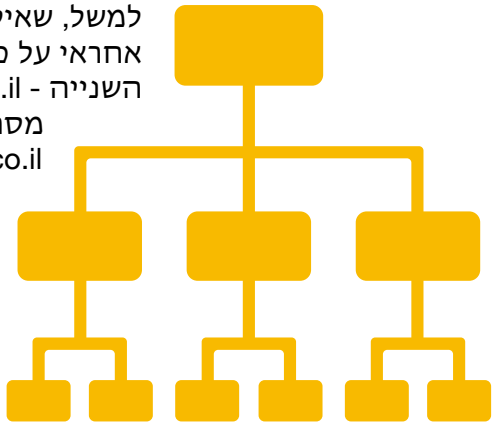
פרוטוקול ה-DNS משתמש במבנה היררכי של אזורים (Zones).
התו המפריד שיוצר את ההיררכיה הוא התו נקודה (“.”). כך למשל, הדומיין www.facebook.com מתאר שרת בשם "www" בתוך האזור "facebook" שבתוך האזור “com”.
הדומיין "he.wikipedia.org" מתייחס לשרת בשם "he" בתוך האזור "wikipedia", שבתוך האזור "org".

חלוקת ה-DNS לאזורים מאפשרת חלוקת אחריות ומשאבים. במצב זה, אף שרת DNS לא צריך לטפל בכל הדומיינים באינטרנט. לכל אזור יכול להיות שרת DNS שידאג אך ורק לאזורים והדומיינים שנמצאים תחתיו. כך למשל, השרת שאחראי על כל הדומיינים ותת-הדומיינים (subdomains) של google.com כגון: mail.google.com ו-www.google.com, לא צריך להכיר את www.facebook.com, ובוודאי שלא את he.wikipedia.org. שרת זה מכיר רק את הדומיינים ותת הדומיינים שתחת google.com.
האזור הראשי ברשת הינו האזור Root המיוצג בידי התו נקודה (“.”). למעשה, האזור com, כמו גם האזור org, מוכלים בתוך האזור Root. כך ששם הדומיין המלא עבור "www.google.com" הינו למעשה - "www.google.com.” (שימו לב לנקודה שמופיעה בסוף הכתובת)

ישנם רק 13 שרתי שמות בסיס (Root) בכל העולם.
שרתי הבסיס מקבלים פניות משרתי שמות מקומיים שלא יכולים לפענח שם כלשהו, אם שרת הבסיס מכיר את השם הזה אז הוא יחזיר את המיפוי המתאים אליו.
במידה והוא אינו מכיר בשם זה, הוא יפנה לשכבה הבאה בהיררכיה- שרת TLD ויבקש ממנה.
עבור כל סוג של סיומת יש שם של שרת אחראי עליה.

כתובת לשרת TLD מתאים	סיומת
ab.ns.il **cd.ns.net**	**.com** **.il** **.org** **.net**

למשל, שאילתה לשרת מרמת השורש לגבי www.google.co.**il** תופנה לשרת אשר אחראי על כלל שמות התחום הישראליים. שרת זה יוכל להפנות לשרת ברמה השנייה - www.google.**co**.il, כלומר השרת שאחראי על כתובות של אתרים מסחריים בישראל. שרת זה יטפל בחלק השלישי של הכתובת - www.**google**.co.il ויפנה את הבקשה לשרת ה"מארח" (host) של google, שיבדוק וימצא שהמשתמש מעוניין בכתובת הIP של שרת האינטרנט (**www**.google.co.il) ויספק את הכתובת המבוקשת.



שרת Authoritative (מוסמך)

איתור שם מתחם ב-DNS מתחיל, עקרונית, בשרת השורש, ומשם לאחר שורה של הפניות, מאותר שרת השמות שבו נמצא המידע המלא על השם אותו מחפשים.
שרת זה נקרא השרת המוסמך (Authoritative Server) עבור אותו שם מתחם.
לדוגמה- כאשר ניגשים לאתר האינטרנט www.example.org.il, יפנה שרת השורש את המחפש לשרת המוסמך של il (במקרה זה, השרת של מרשם השמות הישראלי) , משם תהיה הפנייה לשרת של org.il, ומשם לשרת המוסמך של example.org.il. שם, יימצא המידע המלא על מיקומו של האתר המבוקש.

קביעתו של שרת שמות כמוסמך עבור שם מתחם מסוים נעשית ע"י שרת השמות ברמה שמעליו– בדוגמה דלעיל, קביעתו של שרת המוסמך ל- example.org.il נעשית במרשם השמות האחראי, בין היתר, על השרת של org.il.
כך למשל, כאשר מחזיקו של שם-מתחם מבקש להחליף ספק שירות אירוח (שירות אחסון אתרי אינטרנט), הוא פונה למרשם השמות- במקרה שבדוגמה, איגוד האינטרנט הישראלי (ישירות, או באמצעות הספק החדש) , ומעדכן במרשם שמות המתחם את פרטי השרת המוסמך החדש.
במקביל, מתבצעות שתי פעולות חשובות נוספות-

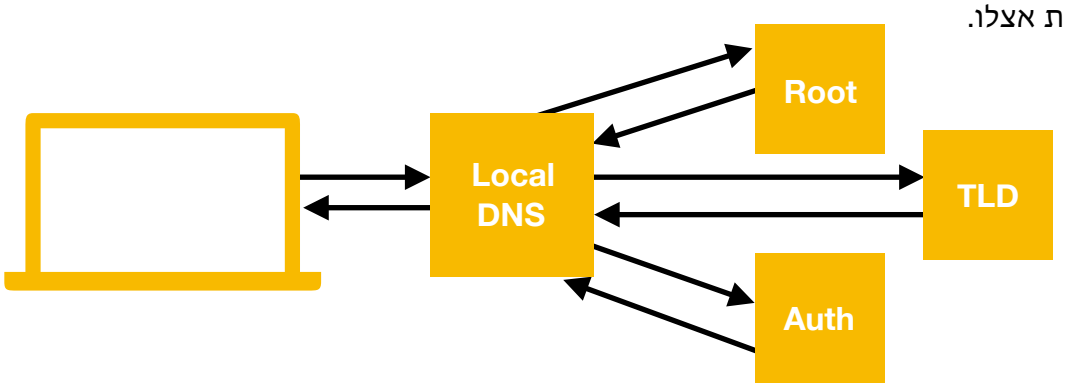
1. ספק השירות החדש (כלומר, זה שאליו עוברים) , מעדכן את שרת השמות שלו להציג מידע מוסמך עבור שם המתחם המועבר.
2. ספק השירות הקודם מעדכן את שרת השמות שלו ע"י הסרת הגדרת ההסמכה.

חברות וארגונים גדולים יכולים להחזיק שרתים משלהם לעומת אנשים פרטיים או חברות קטנות שישתמשו בשירותים של חברות אחרות.

Local DNS-

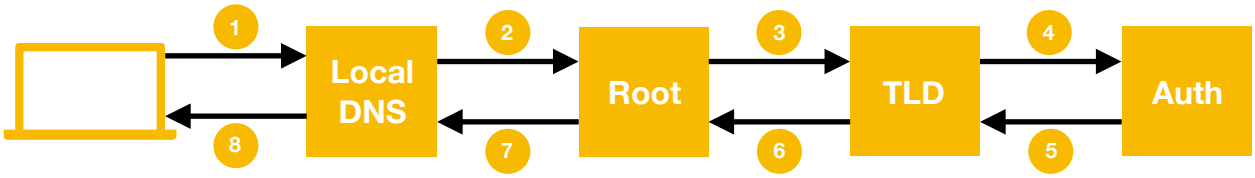
אינו חלק מההיררכיה של ה-DNS.
כל ISP מתחזק שרת מקומי (Local DNS) במשמש כמתווך (כמו Proxy).
כאשר אנו מקבלים כתובת IP אוטומטית נקבל גם את כתובת ה-Local DNS.

תחילה, ה-ISP מפנה אותנו ל-Local DNS. במידה והשרת המקומי מכיר את כתובת ה-URL הרצויה, יחזיר תשובה מיידית.
במידה ולא, יעשה את כל ההפניות ל-Root, TLD, Authoritative, יחזיר לי את התשובה וישמור זאת אצלו.



שיטות לביצוע שאילתת DNS-

1. **איטרטיבית-** בשיטה זו, ה-Local DNS שולח בקשה לאחד משרתי ה-Root. לאחר קבלת התשובה (קבלת כתובת של שרת TLD) ה-Local DNS פונה לשרת ה- TLD ולאחר קבלת כתובת שרת ה-Auth הוא פונה אליו בעצמו לקבלת כתובת IP של האתר.
2. **רקורסיבית-** בשיטה זו ה-Local DNS שולח בקשה אחת לשרת ה-Root, שרת ה-Root אינו מחזיר עדיין תשובה, אלא שולח בעצמו את הבקשה לשרת ה-TLD שגם הוא אינו מחזיר עדיין תשובה, אלא שולח את הבקשה לשרת ה-Auth שמחזיר תשובה לשרת ה-TLD שמחזיר תשובה לשרת ה-Root שמחזיר תשובה ל-Local DNS.



מה צריך בשביל לקנות אתר-

1. קונים Domain (URL), דבר אשר מבטיח לי שאף אחד אחר לא יעשה שימוש בשם זה. את ה- Domain קונים מסוכן Domain של הרשות המרכזית של השמות. למשל GoDaddy.
2. קונים שירות אחסון מאחד מספקי שירותי האחסון.

כיצד לקשר בין כתובת ה-URL לכתובת ה-IP-

(כיצד אנשים יוכלו להגיע לשרת שלי)

1. אני מעדכן את סוכן ה-Domain לגבי Authoritative DNS של ספק השירות שלי.
2. סוכן ה-Domain מעדכן את שרת ה-TLD (מוסיף שורה חדשה עם ה-Domain של האתר שלי, וקישור ל-Auth DNS המתאים).
3. אני מעדכן את ספק שירותי האחסון שלי לגבי שם ה-Domain שלי.
4. ספק שירותי האחסון מעדכן את ה- Authoritative DNS שלו ומוסיף את שם ה-Domain שלי אם כתובת ה-IP שהוא הקצה לי.

מה קורה כאשר ספק שירותי האחסון משנה את כתובת ה-IP של האתר-

ספק שירותי האחסון הוא הראשון שמודע לשינוי ולכן הוא ניגש ל-Authoritative DNS שלו ומחליף את כתובת ה-IP.

מה קורה כאשר מחליפים ספק שירותי אחסון-

(לדוגמה- מ -GoDaddy ל- iPage)

1. אני מעדכן את סוכן ה-Domain לגבי ה- Auth DNS של iPage, והוא מעדכן את ה-DNS TLD בהתאם.
2. נאמר ל- iPage את שם ה-Domain שלי והוא יעדכן את ה- Auth DNS שלי בהתאם.
3. נעדכן את GoDaddy על מנת שימחק את הרשומה המתאימה ב- Auth DNS שלו.

מדוע כתובות IP מאוחסנות בשרתי ה- Authors ולא בשרתי TLD-

כתובות IP משתנות ולפיכך נשמרות קרוב ככל האפשר למקום השינוי.

FTP - File Transfer Protocol

מטרת הפרוטוקול להעביר קובץ ממחשב אחד אל השני.
הפרוטוקול משתמש במודל שרת-לקוח.
לקוח- הצד שיוזם את ההעברה של הקובץ למחשב המרוחק השני.
שרת- המחשב המרוחק.

- הפרוטוקול משתמש בפרוטוקול שכבת התעבורה. TCP ולכן עבור כל שליחת קובץ יפתח חיבור יעודי.
- לקוח FTP מתקשר לשרת ה-FTP בפורט 21 שהוא משמש ל-Control כלומר, בערוץ זה ניתן להעביר פקודות בקרה (login, העברת סיסמה, בקשה להורדת קובץ).
  כאשר הלקוח פונה לשרת, ניתן להזדהות (או שלא) ולפי ההזדהות השרת יודע לאיזה קבצים מותר לי לגשת ונותן את הרשאות הגישה לקבצים.
- כאשר נשלחים פקודות בקרה, בין היתר ניתן לשלוח פקודות העברת קבצים. לאחר קבלת פקודה זו השרת יפתח חיבור TCP נוסף עבור העברת קבצים.
  הערוץ הינו Data Transfer הפועל בפורט 20.
- ערוץ הבקרה נשאר פתוח, אך ערוץ העברת הקבצים נסגר לאחר העברת קובץ אחד.
- ב- FTP בכדי לזהות אובייקטים (קבצים) משתמשים בשם הקובץ, ודרך שם הקובץ ניתן לזהות את הקבצים שאנו מעוניינים להעביר או להוריד. (אין URL)
- ניתן להוריד קובץ יחיד באמצעות פקודת Get ובנוסף ניתן להעביר קובץ יחיד באמצעות פקודת Put.
- ניתן להוריד אוסף של קבצים באמצעות פקודת Mget ובנוסף ניתן להעביר אוסף של קבצים באמצעות פקודת Mput.
- שרת FTP שומר על "מצב"- שומר את הספרייה שנוכחית עבור משתמש מסוים.

MAIL

מערכת דואר אלקטרוני מורכבת מ- 3 מרכיבים מרכזיים-

1. **סוכני משתמש-** לקוח דואר,אפליקציית המייל, שמיועדת ללקוח על מנת שיוכל לקרוא ולכתוב הודעות.

2. **שרת הדוא"ל-** Mail Server שדרכו נשלות ההודעות. השרת מורכב מהרבה תיבות דואר של משתמשים שונים. הוא מחזיק תור של הודעות יוצאות שצריכות להישלח לשרתים אחרים (צריך להיות זמין ובמקום ידוע- Port/IP)

3. **פרוטוקול הדוא"ל-** הפרוטוקול העיקרי בו משתמשים לשליחת דואר אלקטרוני הינו SMTP. הפרוטוקולים העיקריים לאיסוף דואר אלקטרוני הנם pop3 ו- IMAP.

פרוטוקול שליחת דואר אלקטורני

-SMTP - Simple Mail Transfer Protocol

פרוטוקול פשוט להעברת דואר, הוא הפרוטוקול הסטנדרטי לשליחת דואר אלקטרוני דרך האינטרנט.

- משמש למשלוח דואר אלקטרוני בין שרתים שונים, עד שיגיע לשרת היעד, אך אינו מאפשר למשתמש לשלוף את הודעות הדואר המיועדות אליו מן השרת. קיימים פרוטוקולים אחרים המיועדים לשליפה של הודעות דואר, כגון POP3 ו-IMAP..
- שרת הדואר של השולח מוודא שהשולח הוא משתמש ידוע כדי לא לשלוח מיילים מזויפים. בנוסף, שרתי הדואר מוודאים כי ההודעה הנשלחת מגיעה לשרת דואר מורשה.
- עושה שימוש ב-TCP כדי להעביר בצורה אמינה דוא"ל ומשתמש בדרך כלל בפורט 25.
- ניתן לזהות את שרת המקור לפי כתובת IP.
- פרוטוקול זה שם דגש על חיבוריות ושיתוף משאבים.

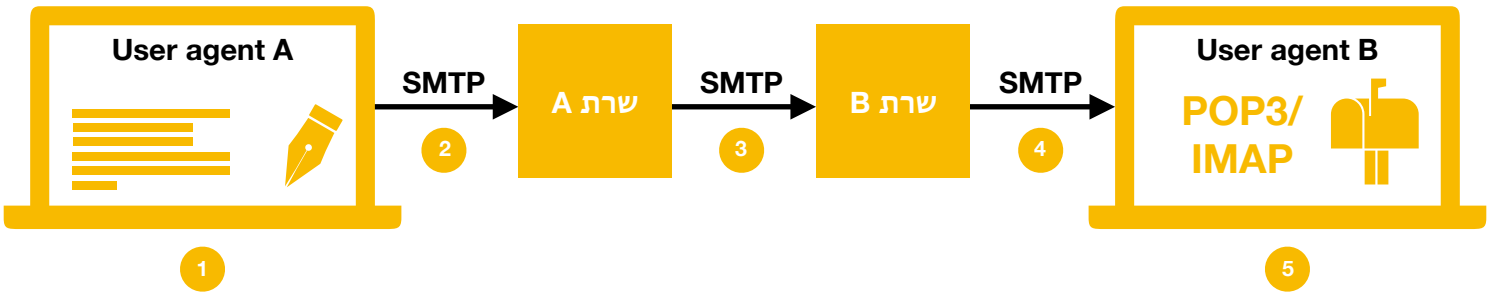
פרוטוקול איסוף דואר אלקטרוני

-POP3 - Post Office Protocol

פרוטוקול שרת-לקוח לשליפה של הודעות דואר אלקטרוני משרת דואר מרוחק. לפרוטוקול קדמו גרסאות POP ו-POP2, והמונח POP הפך שם-נרדף ל-POP3. הפורט הסטנדרטי של פרוטוקול POP3 הוא 110.
תוכנן במטרה לאפשר למשתמשים שאין להם חיבור-קבוע לשרת הדואר, דוגמת משתמשי מודם חיוג או ADSL, גישה לדואר האלקטרוני שלהם. הלקוח מתחבר אל השרת, מבצע מולו אימות, מוריד ממנו את ההודעות במלואן בסדר שבו התקבלו, ושומר אותן על מחשבו. לאחר מכן ההודעות נמחקות מהשרת. הפרוטוקול תומך באפשרות להשאיר עותק מן ההודעה המקורית על השרת. בצורה זו יכול המשתמש לקרוא את הדואר שלו מכמה מקומות וגם מכמה תוכנות לקוח שונות.

-IMAP - Internet Message Access Protocol

פרוטוקול IMAP מהווה אלטרנטיבה מתקדמת יותר לפרוטוקול POP3. בעוד פרוטוקול POP3 מאפשר למשתמש לעבוד עם תיבת דואר אחת בלבד, היתרון בפרוטוקול IMAP הוא באפשרות עבודה עם מספר תיבות דואר ושמירת הדואר על השרת. בנוסף לכך, IMAP מאפשר עבודה יעילה יותר על ידי קריאת חלקים בודדים מתקן ה-MIME (תקן להרחבת יכולות הדוא"ל כמו- טקסט שאינו בקידוד ASCII, צירוף קבצים ועוד).



1. משתמש A כותב הודעה בתכנת הדואר שלו (user agent A) וממען את ההודעה לכתובת הדואר של B.
2. תכנת הדואר של A שולחת את ההודעה לשרת דואר A וההודעה נכנסת לסוף התור של ההודעות היוצאות.
3. שרת הדואר A פותח חיבור TCP עם השרת של B, ושולח את ההודעה של A על גבי חיבור ה-TCP תוך שימוש ב- SMTP אל שרת הדואר של B.
4. שרת הדואר של B שם את ההודעה בתיבת הדואר של B.
5. B מפעיל את תכנת הדוא"ל שלו ומקבל את ההודעה מהשרת תוך שימוש ב-IMAP/POP3.

Webmail

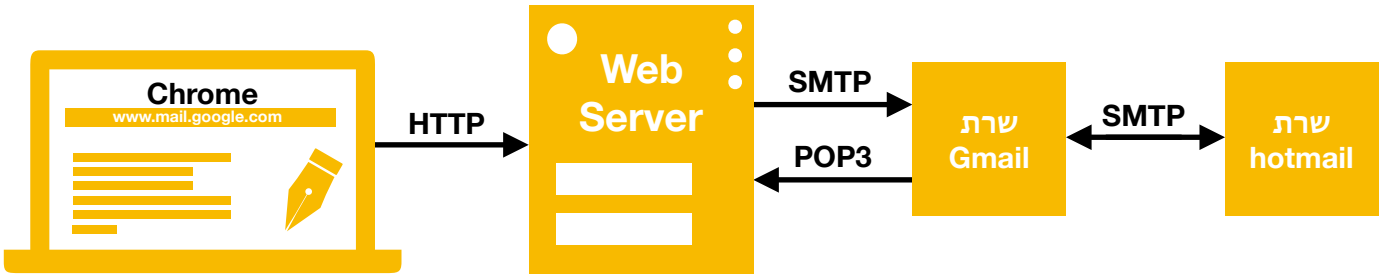
דואר מבוסס רשת הוא שירות דואר אלקטרוני המאפשר גישה אל כל שירותי הדואר דרך דפדפן, באמצעות תוכנת דואר אלקטרוני המותקנת על שרת Web של ספק השירות, ומשמשת כממשק המציג את הודעות הדואר האלקטרוני בדף אינטרנט בדפדפן.
מטרת השירות היא לספק יכולת גישה נוחה ופשוטה לחשבון הדואר האלקטרוני מכל מחשב המקושר לרשת האינטרנט, ללא תלות בתכנת דואר אלקטרוני המותקנת על המחשב המקומי (כדוגמת Microsoft Outlook), וללא צורך להתקין ולהגדיר תוכנה כזו.
לדוגמה- Gmail, הוטמייל, Yahoo!, וואלה!.

יתרונות-

- שימוש בתכנת דפדפן, אשר מותקנת ופועלת ברוב רובם של המחשבים שמחוברים לרשת האינטרנט,
- השימוש פשוט יותר, ודורש רק ידיעה של כתובת האתר, שם וסיסמה.
- הודעות הדואר האלקטרוני מאוחסנות בשרת ונגישות למשתמש מכל מחשב, ולא רק ממחשב אישי אחד.
- פשטות השימוש הובילה לתחרות בין ספקים שונים, להגדלת ההיצע, ולהגדלת סל השירותים שמסופק עם תיבות הדוא"ל. למשל, באמצע שנות התשעים תיבת דוא"ל סטנדרטית הייתה בגודל של מגהבייט בודד. עם התבססות שירותי הרשת גודל זה עלה עד ל-10 מגה בשירותים חינמיים, ו-100 מגה בשירותים בתשלום. עם התגברות התחרות, Gmail הציעה תיבה בגודל אחד גיגהבייט, ונפחי האחסון המקובלים כיום יכולים להגיע ל-1 TB.
- חשבון הדואר האלקטרוני וכתובת הדואר האלקטרוני אינם קשורים לספק שירותי הרשת, מה שמאפשר החלפת ספק שירותי הרשת ללא צורך להחליף כתובת דואר אלקטרוני.

חסרונות-

- יש צורך להשתמש במחשב המחובר לאינטרנט ואין יכולת מובנית לעבודה עם דוא"ל כאשר לא מחוברים לרשת.
- הצורך להשתמש בדפדפן, שהיא תוכנה כבדה יחסית עבור יישום שהוא טקסטואלי ברובו.



Peer To Peer v.s. Client-Server

סימונים-

u_s- קצב העלאת נתונים (רוחב פס) של השרת (מקור ב-P2P).

u_i- קצב העלאת הנתונים של הלקוח ה-i.

d_s- קצב הורדת הנתונים של השרת (מקור ב-P2P).

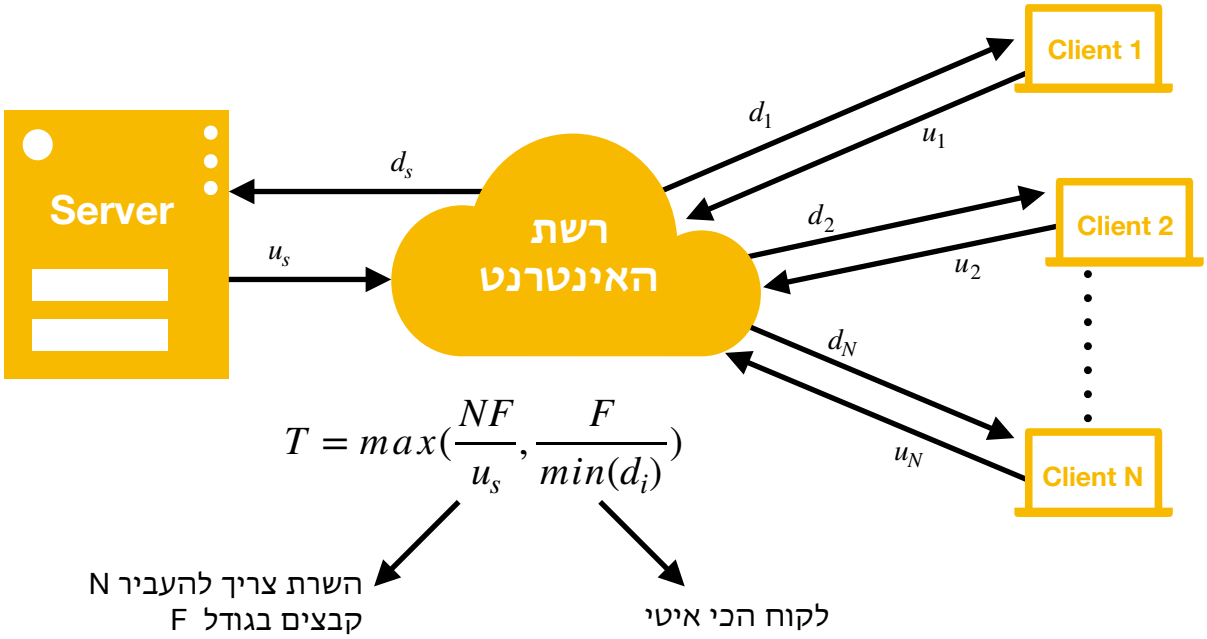
d_i-קצב הורדת הנתונים של הלקוח ה-i.

F- קובץ.

N- מספר הלקוחות.

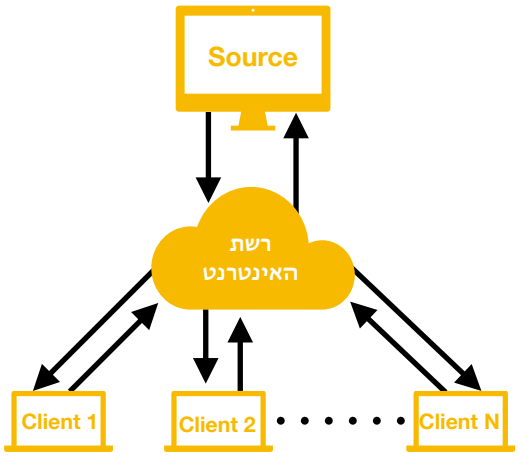
Client - Server

מודל שרת-לקוח היא ארכיטקטורת תוכנה לחישוב מבוזר, אשר מגדירה את היחס בין תוכנות המשתפות פעולה. המודל מחלק את המשימות, או עומס העבודה בין ספק השירות, או המשאבים- כלומר השרת, לבין מבקש השירות- כלומר הלקוח. שרת-לקוח היא אחת מתצורות ההתקשרות הנפוצות ברשתות מחשבים. השרת הוא תוכנה פסיבית, המאזינה לרשת ומחכה לקבל בקשות. הלקוח לעומתו בדרך כלל מהווה את ממשק המשתמש - הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו.
בדרך כלל, תוכנות השרת והלקוח רצות על גבי מחשבים שונים והתקשורת ביניהן מתבצעת על גבי רשת מחשבים. עם זאת, תוכנות השרת והלקוח יכולות לפעול גם על גבי אותו מחשב.
מכונת שרת הוא מחשב המריץ תוכנת שרת אחת או יותר, אשר חולקת את המשאבים שלה עם הלקוחות. הלקוח אינו חולק את המשאבים שלו, אלא מבקש תכנים ושירותים (ביצוע פונקציות) מהשרת. לפיכך, הלקוח הוא זה שיוזם את ההתקשרות עם השרת, אשר ממתין לבקשות נכנסות.
מודל שרת-לקוח הפך לאחד מהרעיונות המרכזיים ברשתות מחשבים. הרבה מהיישומים העסקיים הנכתבים כיום משתמשים במודל זה, כמו גם פרוטוקולי התקשורת העיקריים של האינטרנט, כדוגמת: Telnet, SSH, SMTP, DNS, HTTP, ועוד.



נקבע את הזמן כ- max בין-

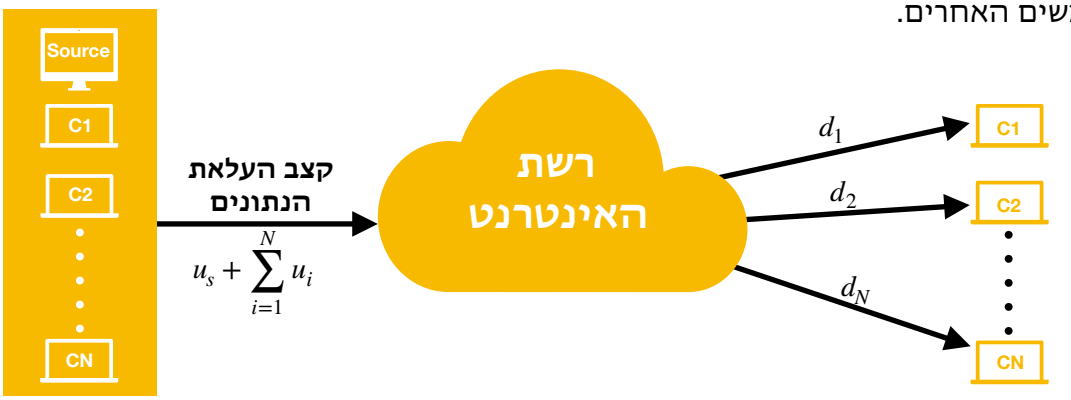
1. זמן העלאת הקבצים של השרת לכל N הלקוחות (קובץ עבור כל לקוח).
2. זמן הורדת הקובץ של הלקוח בעל קצב הורדה הנמוך ביותר.



Peer To Peer

כל אחד מהקצוות מתפקד הן כלקוח והן כשרת, וכל אחד מהקצוות מסוגל ליזום או לסיים התקשרות וכן לספק או לדרוש שירותים.
בהשוואה לרשת שרת-לקוח בה יש הבחנה ברורה בין הצדדים השונים ברשת, ברשת עמית לעמית אין הבחנה כזו, והרשת מתפקדת באופן מבוזר, ללא צורך בתיווך או תלות בגורם מרכזי כלשהו.
רשתות עמית לעמית משמשות בעיקר ברשתות שיתוף קבצים, בהן כל משתמש מאפשר לשאר המשתמשים להוריד קבצים מהמחשב שלו, ומקבל בתמורה גישה לקבצים ממחשבי המשתמשים האחרים.

כולם ביחד מקור אחד



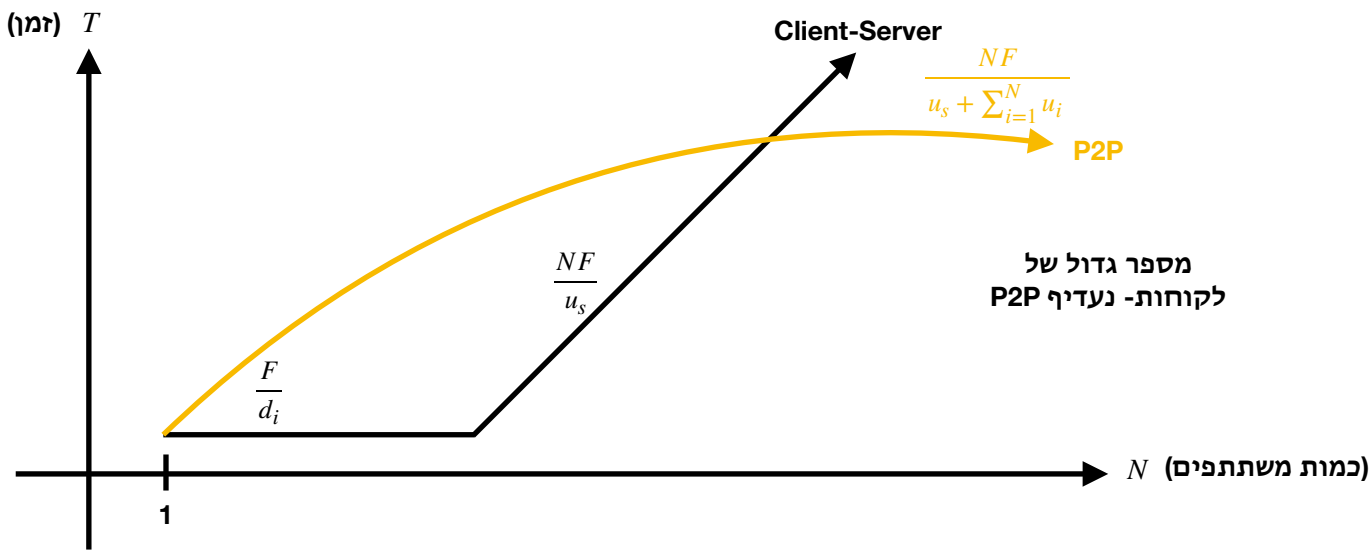
כאשר מסתכלים על כל הצרכנים יחד כמקור אחד, יש צורך בשליחת קובץ עבור כל אחד מהלקוחות.
אצל כל צרכן רק חלק מהקובץ ויחדיו כולם מספקים את הקובץ במלואו.

$$T = max(\frac{F}{u_s}, \frac{F}{min(d_i)}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i})$$

כדי שהקובץ יכנס לרשת במלואו, על המקור להעלות את הקובץ לפחות פעם אחת

הלקוח הכי "חלש"

כאשר כולם יחדיו משמשים כמקור ועליהם להפיץ $N \cdot F$ אינפורמציה לכל הלקוחות



Napster

הרעיון- P2P עם ספרייה מרכזית, שילוב בין P2P ל- Client - Server.

נאפסטר היא אחת מחלוצות התוכנה בתחום שיתוף קובצי מוזיקה בפורמט MP3, ברשת האינטרנט. התוכנה זכתה לפרסום רב בתקשורת עקב השאלות המשפטיות שהיא העלתה לגבי שיתוף קבצים ואחרי מאבק משפטי וציבורי התוכנה איבדה מכוחה ויכולתה.

קיים שרת מרכזי ששומר את כל כתובות הלקוחות (Peer)
ואילו קבצים יש לכל לקוח.
העברת הקבצים מתבצעת בארכיטקטורת P2P.
רשת זו הייתה מבוססת על שרת מרכזי שיצר אינדקס של כל השירים אשר כל המשתמשים ברשת שיתפו באותו זמן. לאחר התקנת נאפסטר על המחשב, נהפך המחשב לשרת קטן אשר ביכולתו להפוך את קובצי המוזיקה אשר מאוחסנים על המחשב הספציפי הזה, לזמינים לכל משתמשי נאפסטר. בשלב זה המחשב מחובר לכל השרתים המרכזיים של נאפסטר ומעביר להם מידע על כל קובצי המוזיקה במחשב, וכך לשרתי נאפסטר היו רשימות מלאות של כל השירים שנמצאו ושותפו על הדיסקים קשיחים שחוברו לתוכנה. כל קובץ מחולק לחתיכות קטנות ואצל משתמשים שונים חלקי קובץ אחרים, לכן כאשר משתמש היה רוצה להוריד שיר מסוים הוא הקליד את שם השיר ובחר בו מתוך רשימה של כל השירים הקיימים בתוכנה, ואז החלה ההורדה ישירות מהמחשבים האחרים.
בשלב מסוים מאגר השירים הגיע לממדים כאלה שלשום שרת מרכזי לא היה דיסק קשיח מספיק גדול שיוכל להכיל את כל השירים, או רוחב פס כדי לטפל בכל הבקשות.

איך מתמודדים עם הבעיות של P2P-

1. **מציאת הרשת-** דרך ספרייה מרכזית בעלת IP קבוע.
2. **מציאת המידע-** תשאול הספרייה המרכזית.

חסרונות-

1. נקודת כשל אחת.
2. הפרת זכויות יוצרים.
3. צוואר בקבוק בביצועים (הספרייה המרכזית)

eMule

הרעיון- התחברות לשרת מתוך רשימת שרתים.

תוכנת שיתוף קבצים המפותחת כתוכנה חופשית.
השרת אינו יחיד ואינו נחשב כשרת מרכזי (כל אדם יכול לפתוח שרת), לאחר מכן הלקוח מברר על לקוחות נוספים שמחוברים לאותו השרת, כלומר השרת מחזיר רשימה של כתובות IP ופורטים של לקוחות. כאשר הלקוח מבקש להוריד קובץ, הוא מקבל את רשימת הלקוחות ומדבר ישירות עם לקוח אחר, זהו ה- P2P. כאשר קובץ שלם מורד הוא משותף גם על ידי הלקוח שהוריד.
התוכנה מנהלת תור אשר אליו נכנסות כל הבקשות להורדה. על מנת לתגמל הדדית משתמשים אשר מעלים חלקי קובץ אחד לשני, התוכנה מנהלת מעקב אחרי הכמות שכל משתמש העלה, ובהתאם מתגמלת אותו בהתקדמות יותר מהירה בתור ההורדות. משתמש יכול להתקדם בתור עד פי 10 יותר מהר עקב העלאה מרובה. שיטת תגמול זו נקראת מערכת הקרדיט של אימיול.

איך מתמודדים עם הבעיות של P2P-

1.**מציאת הרשת-** חיפוש רשימת שרתים באינטרנט במנועי החיפוש.
2.מציאת המידע- חיפוש ברשת

Gnutella

הרעיון- רשת P2P טהורה, אין שום שימוש בישות של שרת.
בעל פרוטוקול ציבורי אשר לקוחות יכולים לממש.

הרשת בנויה בצורה של Overlay Network, רשת אשר מחברת בין הלקוחות בצורה של גרף.
קשת נוצרת בין לקוח x ללקוח y כתוצאה מיצירת קשר TCP בין אחש לשני.
קשת היא אינה חיבור פיזי, אלא חיבור באמצעות הרשת.
לקוחות יכולים להיות מחוברים ליותר מלקוח אחד בו זמנית.

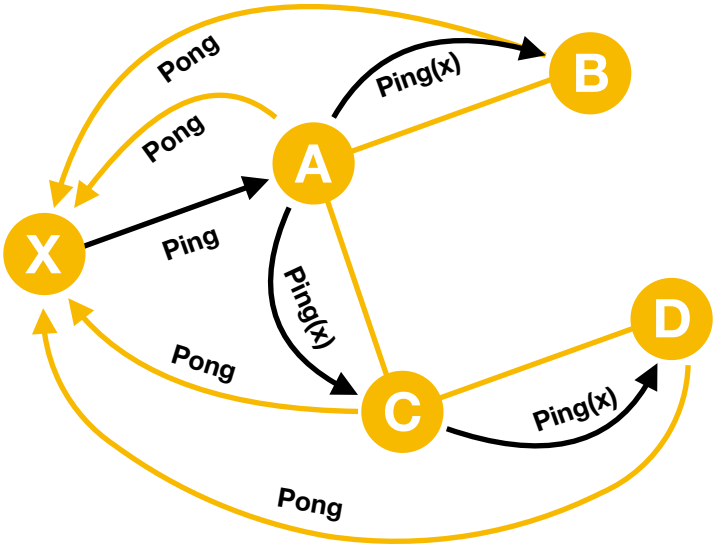
איך מתמודדים עם הבעיות של P2P-

1. **מציאת הרשת-** חיפוש רשימת לקוחות במנועי החיפוש ולאחר מכן מתבצע Ping-Pong.
2. **מציאת המידע-** הצפת שאילתה (Ping-Pong).

בעיה שנוצרה במציאת הרשת- היא שכולם יתחברו לאותו לקוח חזק.

הפתרון-

שימוש ב-Ping-Pong, כאשר אנו פונים למחשבים שקיבלנו מהרשימה שהורדה מהאינטרנט, אנו שולחים לכל Peer הודעת Ping.
המחשבים לא שולחים ישר הודעת Pong אלא מפיצים את הודעת הPing לכלל שכניהם.
כאשר לקוח x שולח Ping ובסופו של דבר מתקבלים אצלו הרבה Pong והלקוח בוחר בצורה רנדומלית לקוח אחש ואליו הוא יתחבר.
להודעות Ping יש TTL (Time To Live, סוג של Counter) שמטרתו להגביל את זמן המחייה של ההודעה בכדי להקטין את כמות התקשורת (הודעות Pong) על מנת לא להעמיס את הרשת.



הצפת שאילתה-

כאשר מחפשים קובץ מסוים שולחים query(y) האומר שאנו מחפשים את קובץ y.
נשלח את ה- query(y) אל הצמתים המחוברים אלי.
כאשר צומת מקבל את השאילתה, תחילה הוא בודק האם הקובץ קיים אצלו, במידה וכן הצומת יחזיר תשובה ישירות למי ששלח את ה-query. במידה ולא, הצומת יעביר את השאילתה לשאר הצמתים השכנים. כלומר, אנו מעמיסים את הרשת בשאילתות ולכן גם כאן ישנו TTL המגביל את המרחק שיוכל להתקדם עם השאילתה.
מחשב המחפש קובץ מסוים יקבל אוסף של תשובות ויכול לבחור מול איזה Peer או מספר Peers לעבוד.
ישנו חיסרון בשימוש ב-TTL משום שלא נוכל להגיע את כל הלקוחות ויכול להיות שקובץ מסוים שחיפשנו נמצא ברשת אך לא נמצא ב"רדיוס" שלנו ולכן לא נמצא אותו.

KaZaA

הרעיון- שיפור המנגנון של Gnutella

כל Peer מתחלק לאחת מן הקבוצות הבאות:
1. חבר בקבוצה.
2. Group Leader.

כלומר, יוצרים היררכיה בין ה- Peers השונים.

Group Leader- עוקב אחר המידע שיש בקבוצתו.
הם ה-Peers שמפורסמים ברשת והם יציבים בעלי IP קבוע.
חיבורי TCP נוצרים בין חברי הקבוצה ל- Group Leader שלה.
בנוסף ישנם חיבורי TCP בין ה- Group Leader השונים.
כלל ה- Peers המחוברים ל- Group Leader מסוים משתפים אותו במידע הקיים אצלם, כלומר לכל Group Leader מסוים ישנו המידע הקיים אצל ה- Peers שבקבוצתו.

לכל קובץ יש מתאר (Descriptor) ו- Hash של אותו מתאר.
כאשר לקוח מסוים מחפש קובץ, הבדיקה מתחלקת ל-2-
1. בדיקה ראשונית הינה ברמת ה- Group Leader של הלקוח המחפש. ה- Group Leader מבצע סינון ראשוני ע"י בדיקה בטבלה שנמצאת אצלו לפי ה- Hash. כלומר מתבצע Hash לשאילתת החיפוש וה- Group Leader מחפש התאמה ב- Hash של חברי קבוצתו.
2. במידה ונמצאה התאמה בקבוצת ה- Group Leader, עבור כל הלקוחות שעבורם נמצאה התאמה, מתבצעת בדיקה של המתאר (ה- String עצמו) באופן מלא אצל אותם Peers.

במידה ולא נמצאה התאמה כלשהי אצל ה- Group Leader, השאילתה תופץ ל- Group Leader אחרים.
אם כן נמצאה התאמה, יוחזר כתובת IP, Hash ו- "מידע אודות המידע" (MetaData).
בהתאם לכך, הלקוח בוחר מול איזה Peer לעבוד ומכאן תהיה התחברות ישירה בין ה- Peers.
משום שההפצה היא רק ל- Group Leader, לא מתבצעת הפצה לכל הרשת ובכך נמנע query flooding.

איך מתמודדים עם הבעיות של P2P-
1. **מציאת הרשת-** חיפוש רשימת Group Leader במנועי החיפוש. לאחר מכן כמו ב- Gnutella מתבצע Ping-Pong.
2. **מציאת המידע-** תשאול ה- Group Leader בצורה חכמה דרך Hash עד שלבסוף השאילתות מגיעות ספציפית ליעד שלהן.

BitTorrent

הרעיון- קודם מחפשים את המידע ולאחר מכן מתחברים לרשת.

התוכנה עובדת עם קבצים בעלי הסיומת torrent. קבצים אלו הם למעשה מצביעים שמטרתם להפנות את התוכנה למידע הדרוש על מנת להוריד את הקובץ המבוקש. קובץ ה-torrent מפנה את התוכנה ל-Tracker, שהוא למעשה שרת המכוון את כל התנועה בין המשתמשים המורידים את אותו קובץ.

כאמור, כל קובץ מתואר ע"י Torrent ונחפש אותו במנועי חיפוש ייעודים.
בתוך ה- torrent יש:
1. **מידע על הקובץ-** שם קובץ, חתימה וכו'.
2. **כתובת ה- Tracker-** קיים Tracker עבור כל קובץ והוא למעשה שרת נוסף שתפקידו לשתף את כל המשתמשים הפונים אליו.

ביטורנט שולחת ומקבלת מספר חלקים מקובץ ההורדה בו-זמנית, על פי הוראות ה-Tracker. היכולת לשלוח ולקבל חלקים שונים של הקובץ בו זמנית, ממשתמשים שונים, מאפשרת לביטורנט לשמור על קצב העברה גבוה מאוד.

-Pulling Chunks

זהו מנגנון שמטרתו לדאוג שקובץ ישמר בשלמותו ברשת בזמן נתון.
משום שמחשבים מתחברים ומתנתקים לסירוגין, ייתכן שברגע נתון לא יהיה קובץ מסוים בשלמותו ברשת. לכן, כאשר לקוח מתחבר לרשת הוא מתשאל את הלקוחות שקיבל מה-Tracker ומתחיל להוריד את החלקים הכי נדירים עד הורדה של הקובץ בשלמותו.
גם אם נוריד חלקים במהירות נמוכה יותר, עדיין נעדיף להוריד את החלקים הנדירים ביותר.
כך נגיע לפחות מקרים בהם חסרים לנו חלקים בשביל להוריד קובץ בשלמותו.

-Tit For Tat

מטרת המנגנון למנוע מצב שיהיו משתמשים שרק מורידים קבצים ולא משתפים אף אחד אחר.
משתמש מסוים יבחר לעבוד עם משתמשים שמשתפים אותו באינפורמציה ורק איתם ישתף מידע.
במידה ומשתמש משתף במידע שקיים אצלו ומשתמש שמקבל ממנו אינו מעביר אליו בחזרה מידע דרוש, המשתמש יסגור את הקשר. לכן תמיד עובדים עם משתמשים מהירים.
בשביל לאפשר למשתמשם חדשים להיות חלק מהמנגנון, כל 30 שניות, בצורה רנדומלית, יילקח משתמש וישלח לו קובץ למרות שאותו משתמש לא משתף כלל מידע.

איך מתמודדים עם הבעיות של P2P-

1. **מציאת הרשת-** דרך ה- Tracker
2. **מציאת המידע**- חיפוש במנועי חיפוש ייעודיים את קובץ ה- torrent. שלב זה כאמור מתבצע לפי מציאת הרשת.

שכבת התעבורה

עד כה התעסקנו באפליקציות. הצלחנו לשלוח מידע מתוכנה שנמצאה במחשב אחד לתוכנה במחשב אחר. אבל איך כל זה קרה? איך עובד תהליך העברת מידע בין מחשב אחד למחשב אחר?
כעת נכיר ונסביר לעומק איך הדברים עובדים.

שכבת התעבורה אם כן אחראית להעביר מידע מתכנית (תהליך) לתכנית (תהליך) מרוחקת. כחלק מכך, יש לה שתי מטרות עיקריות:

1. **ריבוב מספר אפליקציות עבור אותה הישות-** כלומר היכולת לתקשר עם ישות רשת אחת (אל מול אותה כתובת IP בודדת) ולהשתמש בכמה שירותים שונים של הישות, כך שהישות תדע להבדיל איזה זרם מידע שייך לאיזה שירות שהיא מספקת. מטרה זו קיימת תמיד בשכבת התעבורה.

2. **העברה אמינה של מידע-** זוהי מטרה אופציונלית, ומכאן שהיא לא קיימת בכל המימושים של שכבת התעבורה (כלומר, לא בכל הפרוטוקולים של שכבת התעבורה).

ריבוב אפליקציות - פורטים

נאמר ויש לנו חיבור בין שרת ללקוח. כעת, הלקוח שולח בקשת אימייל לשרת מעל חיבור זה:

הדבר הגיוני בהנחה והשרת מריץ שירות של אימייל. עם זאת, יתכן שהלקוח ישלח יותר מבקשה אחת לשרת. למשל, יתכן והשרת מריץ גם שירות של שרת אימייל וגם שירות של שרת Web (למשל - HTTP).
מה יקרה אם הלקוח ישלח אל השרת גם בקשה של אימייל, וגם בקשת HTTP?

כעת, על השרת להבין לאיזה שירות שלו נשלחה הבקשה. במקרה זה, תהיה אצל השרת תוכנה שתטפל בבקשות מלקוחות הקשורות באימייל, ותוכנה שתטפל בבקשות HTTP. על השרת להצליח להפריד ביניהן, כדי להפנות את הבקשה לתוכנה המתאימה.

לשם כך, יש לנו צורך במזהה התוכנה. לא מספיק שהלקוח יודע לפנות אל השרת (כתובת IP), הוא צריך גם לספק מזהה של התוכנה הספציפית אליה הוא פונה. זהו הפורט (Port) אותו הכרנו במבוא.

בפועל, פורט הינו מספר בין 0 ל-65,535. על מנת שתוכנה אחת תוכל להתחבר לתוכנה מרוחקת, עליה לדעת את הפורט שבו התוכנה המרוחקת מאזינה.
לשם כך, ישנם פורטים מוכרים (Well known ports). אלו הם הפורטים מ-0 ועד 1023, והם הוקצו בידי IANA (Internet Assigned Number Authority). כך למשל ידוע שהפורט 80 משויך לפרוטוקול HTTP.
במקרה אחר, מפתחי אפליקציות פשוט צריכים להסכים על הפורט בו הם משתמשים.

העברה אמינה של מידע

הרשת בה משתמשת שכבת התעבורה בכדי להעביר מידע עשויה להיות לא אמינה. כלומר, חבילות מידע יכולות "ללכת לאיבוד" בדרך ולא להגיע ליעדן, או אולי להגיע בסדר הלא נכון (חבילה מספר 2 תגיע לפני חבילה מספר 1). שכבת האפליקציה לא רוצה להתעסק בכך. היא רוצה לבקש משכבת התעבורה להעביר מידע מתוכנה אחת לתוכנה שניה, ולא לדאוג למקרה שהחבילה לא תגיע. לשם כך, שכבת התעבורה צריכה לספק העברה אמינה של מידע מצד לצד.
עם זאת, לא תמיד נרצה ששכבת התעבורה תספק העברה אמינה של המידע. לכן, מטרה זה היא אופציונלית בלבד - ובחלק מהמימושים של פרוטוקולי שכבת התעבורה אין הבטחה שהמידע יגיע ושיגיע בסדר הנכון.

פרוטוקולים מבוססי קישור ולא מבוססי קישור

בשכבת התעבורה, פרוטוקולים יכולים להיות מבוססי קישור (Connection Oriented) או לא מבוססי קישור (Connection Less).

פרוטוקולים מבוססי קישור-

ניתן להמשיל פרוטוקולים מבוססי קישור למערכת הטלפוניה. לא ניתן פשוט לדבר אל מכשיר הטלפון, ולצפות שאדם בצד השני יקבל את המסר שלנו, אם כלל לא חייגנו אליו. באופן דומה, על מנת לתקשר עם מישהו באמצעות פרוטוקול מבוסס קישור, יש ראשית "להקים" את הקישור, לאחר מכן להשתמש בקישור שהוקם ולבסוף לנתק את הקישור. מבחינת המשתמש, הוא מתייחס לקישור כמו לשפופרת הטלפון: הוא מזין מידע (במקרה שלנו - רצף של בתים) לקצה אחד, והמשתמש השני יקבל את המידע בצד השני.
דוגמה לפרוטוקול מבוסס קישור היא פרוטוקול TCP.

פרוטוקולים מבוססי קישור מבטיחים אמינות בשליחת המידע. כלומר, הם מבטיחים שכל המידע שנשלח יגיע אל המקבל, וכן שהוא יגיע בסדר שבו הוא נשלח. עם זאת, יש להם תקורה (Overhead) גבוהה יחסית. כלומר, ישנו מידע רב שנשלח ברשת בנוסף על המידע שרצינו להעביר.

פרוטוקולים שאינם מבוססי קישור-

ניתן להמשיל פרוטוקולים שאינם מבוססי קישור לרשת הדואר. כל מכתב שאנו שולחים באמצעות הדואר כולל את כתובת היעד שלו, וכל מכתב עומד בזכות עצמו: הוא עובר ברשת הדואר מבלי קשר למכתבים אחרים שנשלחים. ברוב המקרים, אם נשלח שני מכתבים מכתובת אחת לכתובת שנייה, המכתב הראשון שנשלח יהיה זה שיגיע
ראשון. עם זאת, אין לכך הבטחה, ולעתים המכתב השני שנשלח יגיע קודם לכן.
דוגמה לפרוטוקול שאינו מבוסס קישור היא פרוטוקול UDP.

מתי נעדיף פרוטוקול מבוסס קישור ומתי פרוטוקול שלא מבוסס קישור-

לפרוטוקולים מבוססי קישור, כמו TCP, יתרונות רבים. הם מבטיחים הגעה של המידע בצורה אמינה ובסדר הנכון. אי לכך, נבחר להשתמש בהם במקרים רבים.
עם זאת, לא תמיד נרצה להשתמש בפרוטוקול מבוסס קישור כגון TCP.
ל-TCP יש תקורה גבוהה יחסית: יש צורך בהקמה וסגירה של קישור, יש צורך לוודא שהמידע הגיע ליעד והגיע בסדר הנכון.. למעשה, שימוש ב-TCP גורר יותר זמן ומשאבים מאשר שימוש בפרוטוקול שאינו מבוסס קישור כגון UDP.
לעתים, העברה מהירה של המידע תהיה חשובה לנו הרבה יותר מאשר העברה אמינה של המידע.

מדוע צריך שכבת תעבורה לא אמינה מעל שכבת רשת לא אמינה?

אם שכבת הרשת שלנו אינה אמינה ולא מבטיחה העברה של מידע מצד לצד, מדוע להשתמש בכלל בשכבת תעבורה לא אמינה? מדוע להשתמש בפרוטוקול UDP ולא לשלוח חבילות ישר מעל שכבת הרשת?

לשימוש בפרוטוקול לא אמין של שכבת התעבורה מעל שכבת רשת לא אמינה, יש שתי סיבות עיקריות. ראשית, השימוש בפורטים. כפי שהוסבר קודם, השימוש בפורטים הוא הכרח בכדי לדעת לאיזו תוכנה אנו פונים בשרת המרוחק. בנוסף על כן, שימוש של שכבת האפליקציה בשכבת הרשת "ישבור" את מודל השכבות: איננו רוצים שמתכנת של שכבת האפליקציה יכיר את שכבת הרשת. דבר זה יגרום למפתח של שכבת האפליקציה להעמיק בסוגיות הקשורות לשכבת הרשת, ולכתוב מימוש שונה עבור כל פרוטוקול בשכבה זו.

הגדרות

-Chacksum

תפקידו לאתר תקינות. גם חבילות מידע (הודעות) וגם הודעות ACK.

-ACK

תפקידו לאשר קבלת חבילות.

ההודעה תישלח מהצד המקבל לצד השולח לאחר קבלת הודעה תקינה.

-Cumulative ACK

הודעה המאשרת את כלל החבילות שהתקבלו עד כה.

-Seq num

תפקידו לידע את התחנה המקבלת בסדר ההודעות שנשלחו. בנוסף לכך מסייע במניעת כפילויות בצד המקבל. במידה ויגיעו הודעות משוכפלות, לפי ה- Seq num הצד המקבל ידע לא להעבירה לשכבת האפליקציה. מעבר לכך מאפשר זיהוי של הודעות משובשות.

-NACK Negative Acknowledgement

אישור על הודעה שהתקבלה משובשת.

כיצד ניתן לוותר על NACK-

כאשר שולחים הודעת ACK הכוללת Seq num, ניתן לזהות הודעה שהתקבלה משובשת משום שה- ACK יכול את ה- Seq num של ההודעה אותה הוא אישר. במידה וה- Seq num של ה- ACK אינו תואם את ההודעה שנשלחה, ניתן לזהות שגיאה.

-TimeOut

מסייע בהתגברות על אבדן חבילות משום שאחרי T.o נבצע שידורים חוזרים.

-Window

גודל החלון הינו גודל ה- Buffer.

קצב שידור אפקטיבי של פרוטוקול-

$$\tilde{R} = \frac{\text{כמות מידע}}{\text{זמן}} \quad \text{(Bit per sec)}$$

נצילות של פרוטוקול-

$$u = \frac{\tilde{R}}{R} = \frac{\text{קצב שידור אפקטיבי של פרוטוקול}}{\text{פרק זמן כולל}}$$

$$u = \frac{\text{פרק זמן שליחת המידע}}{\text{פרק זמן כולל}}$$

-Stop & Wait

כל עוד לא התקבל אישור על הגעת החבילה מהצד המקבל לא תשלח חבילה נוספת.

UDP - User Datagram Protocol

פרוטוקול תעבורה שאינו מבוסס קישור שמתאפיין בכך שאינו אמין ואינו משמר סדר. כלומר, חבילות מידע עלולות להגיע בסדר שונה מזה שנשלחו, להגיע מספר פעמים או ללכת לאיבוד ולא להגיע כלל.

-UDP Connectionless

- כלומר לא מתבצע 3HS (לחיצת יד משולשת)
- כל חבילת UDP מטופלת בנפרד.
- ישנו חיסכון ביצירת חיבור.

פרוטוקולים העושים שימוש ב- UDP-

DNS, SNMP, DHCP, RIP.

מדוע יש צורך בפרוטוקול UDP-

- אין הקמת חיבור (דבר המוסיף עיבוד)
- פשוט
- Header בגודל 8 bytes

UDP הינו "Best effort" משום שאין בקרה על העברת הנתונים, משתמש במקסימום היכולת של השרת (העמסת הרשת ואפשרות לניצולת מקסימלית).

UDP Segment Header

Source Port	Dest Port
Length	Checksum
Message (Application Data)	

1. **Source Port-** מספר הפורט של התהליך ששלח את ההודעה (מקור).
   מוצג בבסיס עשרוני וגודלו 16 bit (2 byte).
2. **Dest Port-** מספר הפורט של תהליך היעד.
   מוצג בבסיס עשרוני וגודלו 16 bit (2 byte).
3. **Length-** מייצג את גודל ההודעה כולל את גודל ה- Header (8 byte).
   כלומר, אורך המידע + אורך ה- Header ב- bytse.
   גודלו 16 bit (2 byte).
4. **Checksum-** מטרתו למצוא שגיאות בהודעה (בדיקת תקינות).
   גודלו 16 bit (2 byte).

Checksum

סיכום ביקורת מספר האימות של הפתיח והנתונים. שדה אופציונלי, שבמידה ולא ימומש יכיל רשומה מלאה ב-0 בינארי.

עד כה ציינו שבעיות ברשת יכולות לגרום לחבילה לא להגיע כלל, או לרצף של חבילות להגיע ברצף הלא נכון. אך בעיות ברשת יכולות גם לגרום לשגיאות בחבילה עצמה - כלומר שהחבילה תגיע עם תוכן שונה מהתוכן שנשלח במקור.

לדוגמה, אם נרצה לשלוח הודעה ובה מספר טלפון סלולרי, נשלחות 10 ספרות של מספר טלפון אחד.
הבעיה היא, שיתכן והחבילה השתנתה בדרך בגלל תקלה כלשהי. כך למשל, יתכן והשרת יקבל את החבילה ומספיק שהשתנתה ספרה אחת.
במקרה זה, נרצה שהשרת ידע שאירעה שגיאה, ולא יתייחס לחבילה התקולה. דרך אחת לעשות זאת, היא להשתמש ב-Checksum.

כיצד ה-Checksum מחושב ב- Header-

הרעיון הוא כזה: נבצע פעולה כלשהי על המידע שאנו רוצים לשלוח, ונשמור את התוצאה. בצד השני החישוב יתבצע שוב, ויושווה לתוצאה שנשלחה. אם התוצאה שונה, הרי שיש בעיה.

פונקציית ה-Checksum מבצעת את החישוב הבא: חילוק ההודעה לקטעים של 16 סיביות, סכימת כל הקטעים והוספת השארית. לאחר מכן התוצאה נשללת לפי ייצוג One's Complement - כל ביט 0 מוחלף ב-1 ולהפך.

הצד השולח- תחילה הצד השולח מחלק את ההודעה למקטעים של 16 bit.
את שדה ה- Checksum ממלא באפסים. לאחר מכן מחבר את כל השדות של המקטעים.
במידה והתקבל מספר שגודלו יותר מ- 16 bit, מחברים את הספרה השמאלית ביותר לצד הימיני ביותר (ההתחלה). לאחר מכן נחשב את ההופכי של המספר שקיבלנו (ההופכי ייווצר השלמה ל-1 או ל-F בהקסה)

Src Port	00	A0	00000000	10100000	
Dest Port	00	15	00000000	00010101	00B5 → 0000000010110101
Length	00	28	00000000	00101000	00DD → 0000000011011101
Checksum	00	00	00000000	00000000	00DD → 0000000011011101
1-16 bit message	72	D4	01110010	11010100	73B1 → 0111001110110001
17-32 bit message	1E	BE	00011110	10111110	**926F → 1001001001101111**
.....	..	..	.....	.....	

לתוצאה זו נעשה הופכי

0110110110010000 **6DA4** **וזהו ה- Checksum**

הצד המקבל- מחלק את הסגמנט למקטעים של 16 bit, סוכם את כל ההודעה כולל ה-Checksum.
במידה והסכום כולו אחדים (בבינארי) או כולו F-ים (בהקסה), ההודעה הגיעה בצורה תקינה.
אחרת, התגלתה טעות בהודעה.

- גודל השדה Length קובע מגבלה תאורטית של 65,535 בתים להודעת UDP בודדת (8 בתים Header + 65,527 בתים של נתונים). עם זאת, המגבלה בפועל עבור אורך הנתונים, המוטלת על ידי פרוטוקול IPv4 הבסיסי, היא 65,507 בתים (65,535 - 8 UDP Header - 20 IP Header).

Principles Of Reliable Data Transfer

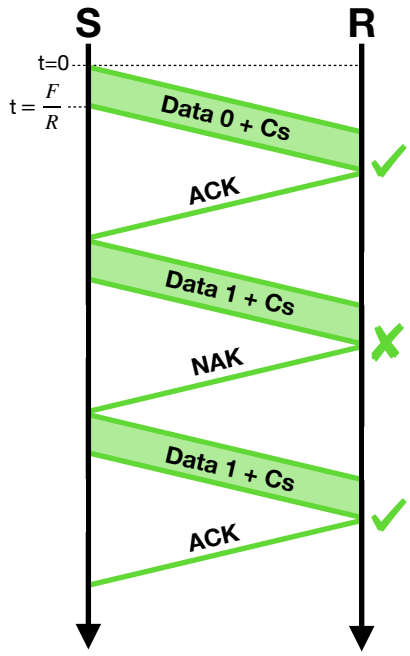
כדי להבטיח שהמידע אכן מגיע ליעדו, בפרוטוקולים רבים נהוג לאשר הגעה של חבילות מידע בדרכים שונות. אישור כזה נקרא לרוב Ack (Acknowledgements). יש פרוטוקולים בהם חבילות מידע שלא מתקבל עליהן אישור בפרק זמן שהוגדר (timeout) נשלחות שוב. בצורה זו מבטיחים שכל המידע מגיע לבסוף ליעדו.
ככל שהתווך יותר משובש כך על פרוטוקול התעבורה האמין,RDT (reliable data transfer protocol) להתמודד עם יותר מצבים. נבצע סקירה של פרוטוקולים, כל אחד מהם מורכב יותר מקודמו, עד שנגיע לפרוטוקול האמין ביותר.

פרוטוקול פשוט שמניח שהערוץ שמשתמשים בו אמין-

לשולח ולמקבל יש רק מצב אחד- המתנה לקבלת הודעה. הצד השולח מחכה לקבלת הודעה מהרמה העליונה. כאשר מתרחש מאורע של קבלת הודעה מתבצעת פעולת שליחת ההודעה, תוך העברתה לרמה התחתונה.. הצד המקבל מחכה לקבלת הודעה מהרמה התחתונה. כאשר מתרחש מאורע של קבלת הודעה- המקבל מוציא מהחבילה את המידע.

ערוץ תקשורת לא אמין-
הנחות:
- אין אבדן חבילות. (ייתכן שיגיעו משובשות)
- החבילות מגיעות בסדר הנכון.
- כל הטעויות מתגלות על ידי מנגנון ה- checksum.

איך מתגברים על טעויות (Bit Error)-
- ACK (acknowledgmen)- אישור על קבלת חבילה תקינה.
- NAK (negative acknowledgment)- אישור על קבלת הודעה משובשת
- חבילה שהגיעה משובשת תשלח מחדש (retransmission).



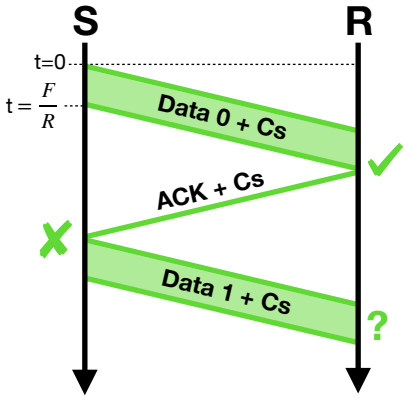
יש שני מצבים בצד **השולח**:
מצב 1- מחכה לקבלת מידע לשליחה מהרמה העליונה. כאשר יתרחש המאורע של קבלת המידע השולח ייצור חבילה שמכילה את המידע ואת שדה ה checksum. השולח ישלח את החבילה דרך הערוץ הלא אמין.
מצב 2: השולח מחכה לקבלת ACK או NAK מהצד המקבל. אם מתקבל ACK השולח יודע שהחבילה האחרונה שנשלחה הגיעה תקינה וחוזר למצב 1. אם התקבל NAK השולח ישלח מחדש את החבילה האחרונה ששלח ויחכה לקבלת ACK או NAK, נשאר באותו המצב.

פרוטוקול זה הוא מסוג **stop&wait** כל עוד לא התקבל אישור על הגעת החבילה מהצד המקבל לא תשלח חבילה נוספת.

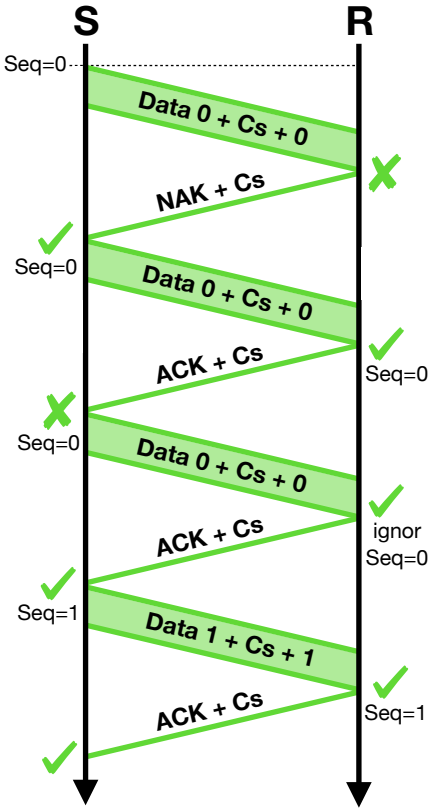
בצד **המקבל** יש מצב אחד:
המקבל ממתין להגעת חבילה, כשמגיעה חבילה הוא יבדוק את שדה ה- checksum, במידה והחבילה משובשת תשלח הודעת NAK. אם החבילה תקינה ישלח ACK.

על פניו נראה כי הפרוטוקול מבטיח אמינות אבל יש בו פגם חמור. ייתכן שיבוש של הודעת ACK או NAK. יכול להיווצר מצב של שכפול הודעות או אי קבלת הודעות. אם יהיה שיבוש הודעת NAK אז חבילה שהגיעה משובשת לא תשלח שוב כנדרש ויהיה איבוד הודעה. אם יהיה שיבוש הודעת ACK אז תשלח אותה ההודעה פעמיים.

פתרון שיבוש הודעות NAK -ו ACK-
כאשר הצד המקבל שולח הודעת ACK או NAK הוא מוסיף להודעה שדה checksum.
כעת, הצד השולח שמקבל אישור שליחה על המידע ששלח מוודא את תקינות ההודעה.
אם ההודעה שקיבל היא ACK היא תקינה, ממשיך לשדר את ההודעות הבאות במידה ויש.
אם אינה תקינה, או לחלופין קיבל הודעת NAK תקינה, שלוח שוב את ההודעה.



אך גם כאן קיימת בעיה מהותית, מה יקרה וכאשר הצד המקבל קיבל את ההודעה בשלמותה, בדק אותה ושלח הודעת ACK והצד השולח מקבל את הודעת ה- ACK אבל שגויה.
על פי הפרוטוקול ההודעה תישלח שוב.
התחנה המקבלת לא תדע להבדיל בין ההודעות, אין לה אפשרות לדעת אם זו הודעה חדשה או הודעה חוזרת.

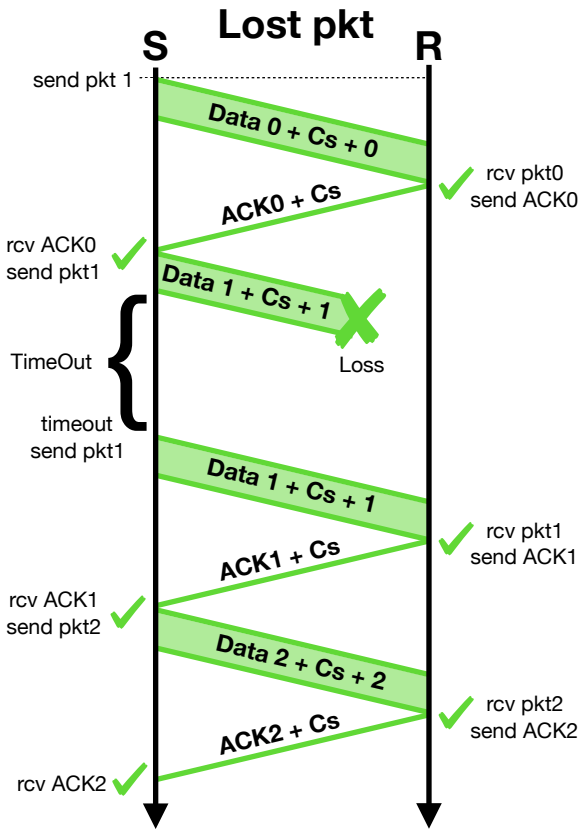


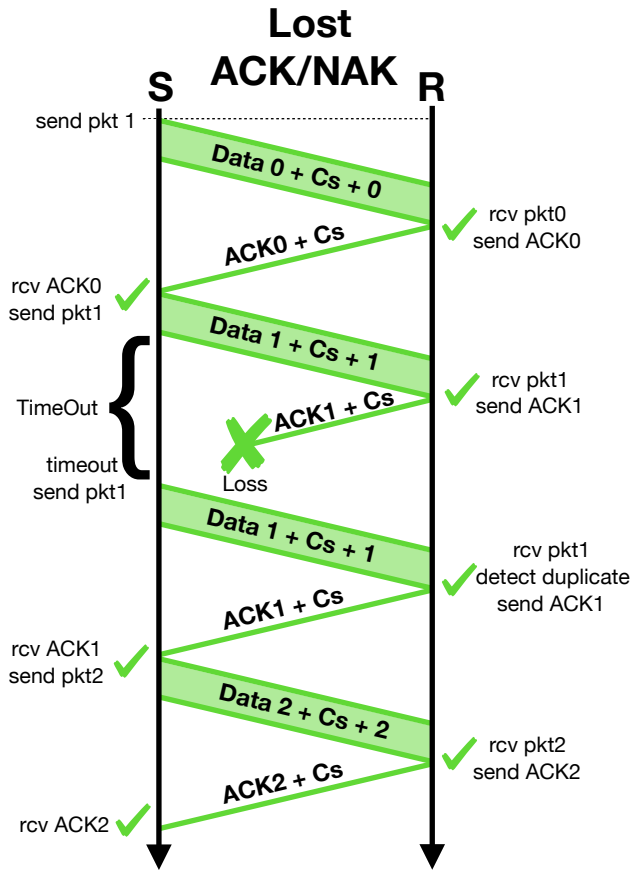
טיפול בשכפולי הודעות-
בכדי שהצד המקבל ידע לזהות את סדר החבילות שהוא מקבל ובכך להתעלם מהודעות משוכפלות שכבר נשלחו אליו, יש צורך במספור החבילות הנשלחות. זהו ה-Seq num.
אם הצד המקבל קיבל את החבילה אך ה-checksum אינו תקין הוא ישלח הודעת NAK.

במידה והצד השולח קיבל הודעה בעלת Bit Error, כלומר, ה-checksum נכשל, הוא ישלח שוב את אותה ההודעה.
מכיוון שכעת יש להודעות מספר מזהה, הצד המקבל יוכל לדעת אם זו הודעה שהוא כבר קיבל ואישר וכך יוכל לדעת להתעלם ממנה (הכוונה להתעלמות מבחינתו, מבחינת הצד השולח, הוא עדיין צריך לקבל על הודעה זו אישור ACK).

עד כאן טיפלנו בבעיות של Bit Error.
אך מה יקרה אם יש Loss, הכוונה שההודעה כלל לא הגיע לצד השני.
כך לצד השולח אין אפשרות לדעת אם החבילה שלו אכן הגיע ליעדה ובנוסף הוא תקוע במצב האזנה (stop&wait).

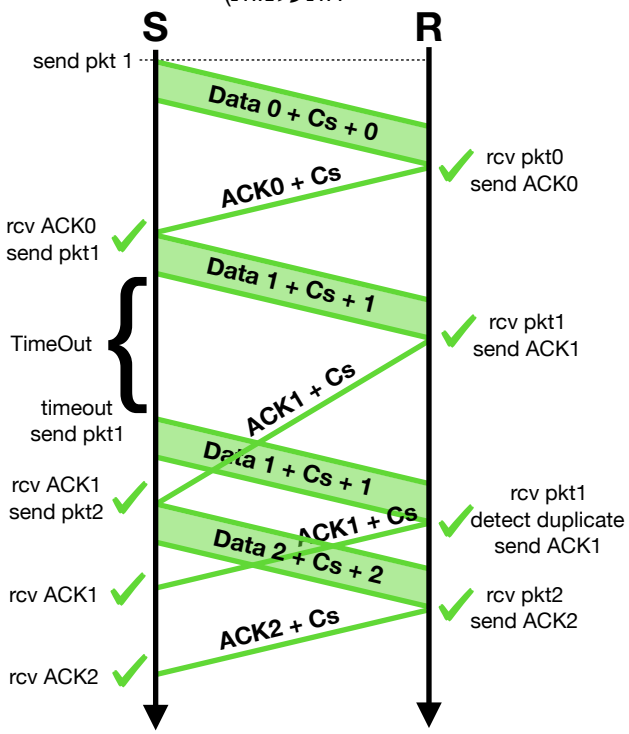
טיפול באבדן חבילות-
במצב של Loss, חבילה שנשלחה כלל לא הגיעה ליעדה, השולח מחכה ל- ACK פרק זמן "סביר" אשר קטן מהעיכוב המקסימלי (RTT). במידה ולא התקבל ACK בזמן הזה תתבצע שליחה מחודשת. במידה והיה רק עיכוב ולא אבדן, יתבצע שכפול שבו הצד המקבל יודע לטפל (בעזרת מס' סידורי, Seq num).
בכדי להתגבר על מצב ובו ה-TimeOut היה קצר מהזמן שבו לקח ל-ACK להגיע, נוסיף את ה-Seq num גם ל- ACK/NAK. כך התחנה המשדרת תדע לזהות כל ACK/NAK למי הוא שייך.
ביצועי הפרוטוקול יכולים להיות גרועים בגלל שה- RTT יכול להיות גדול עקב תורים בנתבים.





premature timeout

(timeout שהתרחש לפני שהגיע ה-
ACK וגרם להכפלה ממנה הייתה
התעלמות)



דוגמת חישוב

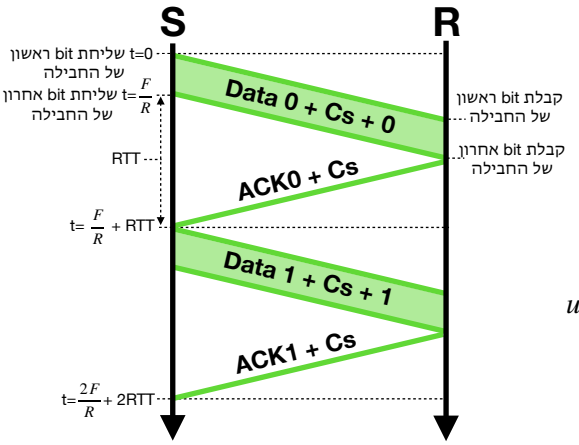
גודל החבילה = F
קצב השידור של הרשת = R
קצב שידור אפקטיבי של הפרוטוקול= $\tilde{R}$
נצילות של הפרוטוקול = u
(כמה מהקצב של הרשת באמת השתמשנו)

$$\tilde{R} = \frac{2F}{\frac{2F}{R} + 2RTT} \Rightarrow \tilde{R} = \frac{F}{\frac{F}{R} + RTT} \Rightarrow u = \frac{\tilde{R}}{R}$$

$$u = \left(\frac{F}{\frac{F}{R} + RTT}\right) \cdot \frac{1}{R}$$

$$u = \frac{\frac{F}{R}}{\underbrace{\frac{F}{R} + RTT}_{\text{זמן ההמתנה שבו קצב הרשת לא מנוצל}}}$$

$\frac{F}{R}$ } הזמן בו הפרוטוקול שולח את החבילה



החישוב מדגיש שמערוץ רחב קיבלנו ניצולת מאוד נמוכה.

-Pipeline

בפרוטוקולים מסוג stop and wait נוצרה בעיה של ניצולת נמוכה, ניתן לשלוח הודעה אחת בכל פעם בלבד. כמו כן ה- RTT יכול להיות גדול בגלל תור שנוצר בנתבים. פרוטוקולים מסוג pipeline מאפשרים שליחת הרבה הודעות במקביל מבלי שקבלנו עליהן ACK.
יש צורך בחוצצים בצד השולח כדי לשמור הודעות שנשלחו ועדיין לא התקבל עליהן ACK למקרה שנצטרך לבצע שליחה מחודשת של ההודעה. לפעמים צריך גם חוצצים במקביל.
מתמודד גם עם שנויי סדר בחבילות ישנות. שימוש ב- pipelining גורם להגדלה של הניצולת, אבל תמיד תהיה מגבלה מסוימת שנובעת מגודל החוצצים שיש לנו.

-Sliding window

פרוטוקולים של שכבת התעבורה מפקחים על זרימת הנתונים. פיקוח זה מתבטא בשליטה על מהירות ההעברה (רוחב פס), המתבצע על ידי שימוש ב"חלונות" (Sliding window), כאשר גודל החלון קובע את כמות נתונים שמחשב המקור (SOURCE) יעביר למחשב היעד (DESTINATION), לפני שהם יתאמו מחדש פרטים שונים הנוגעים לאופן העברת הנתונים (ביניהם גם גודל החלון). ככל שהחלון גדול יותר, כך מהירות העברת הנתונים עולה. בדרך כלל, כאשר שני המחשבים פנויים, וקו התקשורת ביניהם אמין, החלון יהיה גדול למדי. לעומת זאת, אם אחד המחשבים עסוק, הוא יבקש להקטין את החלון ובכך להאט את קצב העברת הנתונים.

Go Back N

פרוטוקול אמין להעברת חבילות מבוסס על חלון שליחה אשר לא מתחשב במצב הרשת ובעומסים.
ניתן לשלוח חלון הכולל בתוכו N הודעות, כלומר גודל החלון הינו N.
ההודעות ישלחו אחת אחרי השנייה.

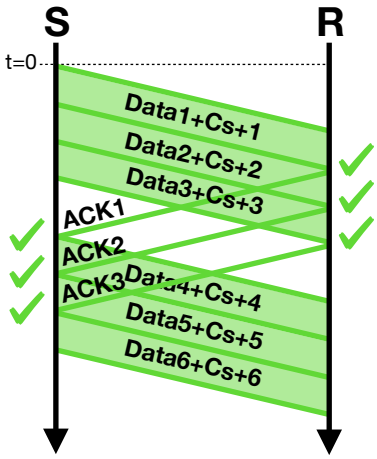
תחנה משדרת (שולחת)-

לשולח יש Timer עבור ההודעה הישנה ביותר שטרם התקבל עבורה ACK.
כאשר מתקבל ACK שמאשר הודעה ועדיין יש הודעות בחלון, ה-Timer יאופס.
כאשר ישנו אירוע Time Out כל החלון נשלח מחדש וה-Timer מתאפס.
כאשר מתקבל ACK שמאשר הודעות בחלון, החלון "מחליק" קדימה ושולח את ההודעות החדשות שנוספו לחלון.

תחנה מקבלת-

התחנה המקבלת מצפה לקבל הודעות לפי הסדר (ולכן לא שומרת הודעות שלא לפי הסדר).
במידה והתקבלה הודעה לא תקינה (משובשת או לא לפי הסדר), ישלח ACK (חוזר) על ההודעה האחרונה שהתקבלה תקינה וברצף.
בשל כך, המקבל שולח Cumutative ACK, כלומר, מאשר אוטומטית הודעות שקדמו לאותה הודעה כולל ההודעה עצמה.

מצב תקין

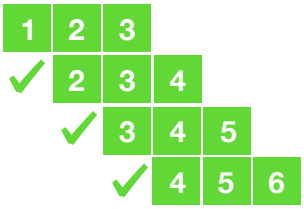


תחילה, w=3

התקבל ACK1- נשלחה הודעה 4.

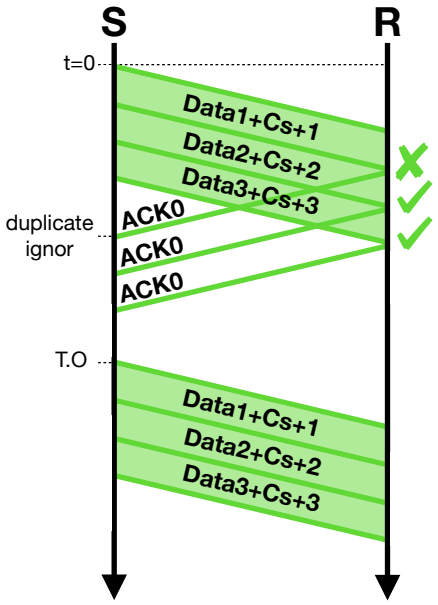
התקבל ACK1- נשלחה הודעה 5.

התקבל ACK3- נשלחה הודעה 6.



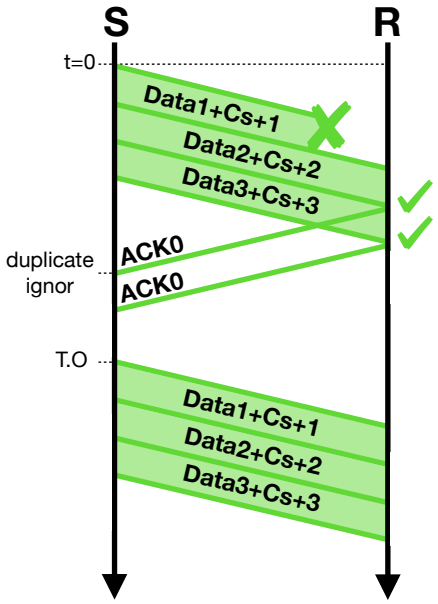
הודעה לא תקינה

התחנה המקבלת לא שומרת הודעות שלא לפי הסדר, לכן החלון לא מקודם ולא נשלחות הודעות חדשות. כאשר יהיה T.O, כל החלון ישלח מחדש (שידור חוזר)



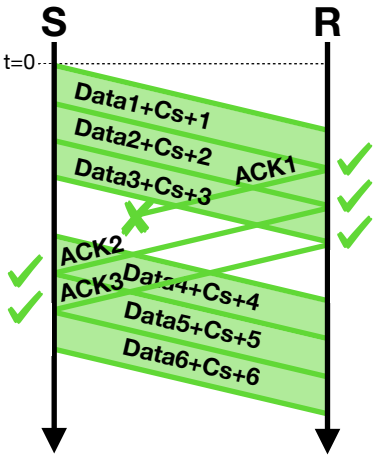
הודעה לא מגיעה

משום שההודעה ה-2 התקבלה לא ברצף (מבחינת הצד המקבל הוא אינו יודע שהודעה 1 הלכה לאיבוד), ישלח ACK להודעה האחרונה שהתקבלה תקינה וברצף. וכך הצד המקבל ממתין לקבלת הודעה מספר 1.



ACK הולך לאיבוד / ACK לא הגיע תקין

כאשר ACK לא הגיע תקין ואחריו מתקבלים ACK-ים נוספים, הם מאשרים את ה- ACK שלא הגיע תקין.



Selective Repeat

פרוטוקול אמין להעברת חבילות הדומה לפרוטוקול Go Back N אם כי מטפל בתקלות בצורה שונה.
הפרוטוקול מונע שליחה מחדש מיותרת שכן הצד השולח, שולח מחדש רק את החבילות שאבדו או הגיעו למקבל משובשות. לשם כך המקבל צריך לשלוח ACK לכל חבילה שהגיעה תקינה בנפרד.

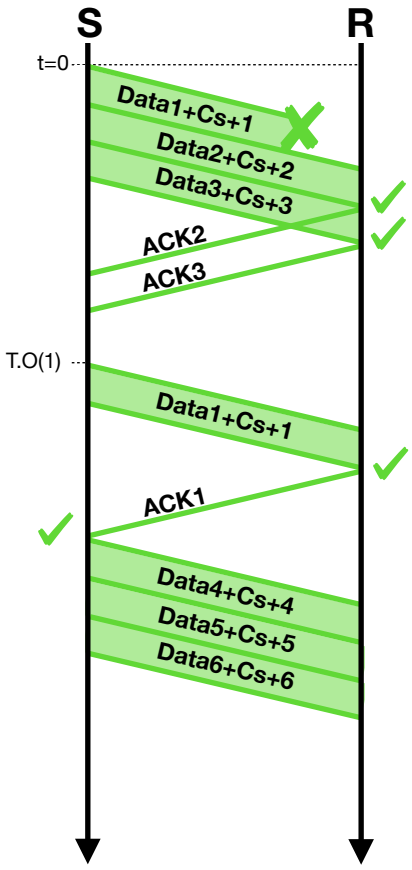
שלא כמו בפרוטוקול GBN, השולח כבר קיבל ACK למספר הודעות שנמצאות בתוך החלון. הצד המקבל שולח ACK לחבילה תקינה שהתקבלה גם אם לא התקבלה בסדר הנכון, ולכן חבילות שמגיעות בסדר לא נכון נשמרות בחוצץ אצל המקבל עד שהחבילות החסרות מתקבלות, ואז החבילות בחוצץ יועברו בסדר הנכון לרמה העליונה.

תחנה משדרת (שולחת)-

אם כן, ניתן לשלוח חלון הכולל בתוכו N הודעות והן ישלחו אחת אחרי השנייה.
לשולח ישנו Timer עבור כל הודעה. כאשר מתקבל ACK(i) הוא מאשר את ההודעה ה-i ואותה בלבד.
במידה ואותה הודעה i הינה ההודעה בעלת המספר הנמוך ביותר בחלון שטרם אושר, הודעה זו מפונה מהחלון ומתבצע Sliding Window, כלומר, החלקה של החלון קדימה ושליחת ההודעה החדשה שנוספה לחלון.

תחנה מקבלת-

התחנה המקבלת יכולה לקבל הודעות לא לפי הסדר.
כלומר, במידה והתקבלה הודעה i לא בסדר הנכון, הצד המקבל ישמור את ההודעה עד אשר יהיה ניתן להעביר את ההודעה לפי הסדר הנכון לשכבה מעל (כלומר, כשהתקבלו ההודעות החסרות).
במידה וההודעה i מתקבלת כן לפי הסדר, התחנה המקבלת תעביר את ההודעה לשכבה מעל (אפליקציה) ותבצע החלקה של החלון קדימה.
התחנה המקבלת שולחת ACK(i) עבור הודעה i שהגיע תקינה גם אם לא ברצף.



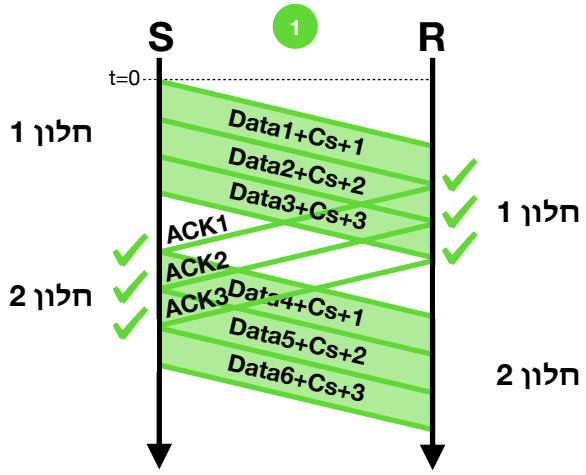
הודעה לא תקינה / הודעה לא הגיעה

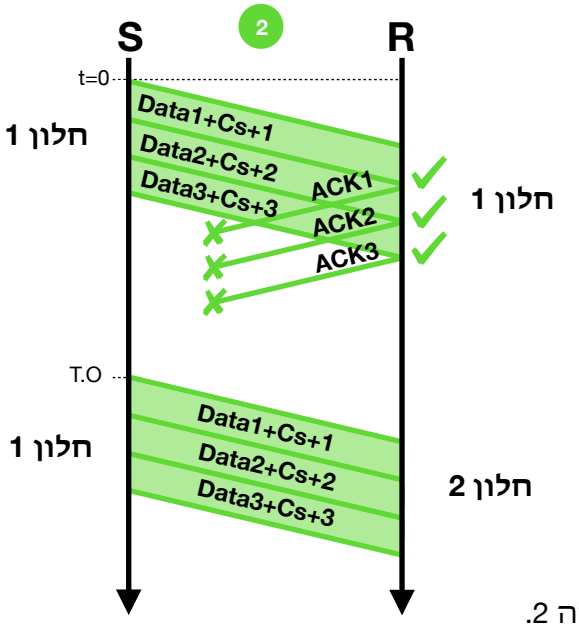
התחנה השולחת- כאשר התקבלו ACK2 ו-ACK3 הופסקו הטיימרים שלהם אך לא ניתן לקדם את החלון משום שלא אושרו על ההודעות ברצף.
כאשר בוצע T.O(1) נשלחה הודעה 1 בלבד.
כשמתקבל ACK(1) ניתן לקדם את החלון ולשלוח את הודעות 4-6

התחנה המקבלת- כאשר התקבלו הודעות 2 ו-3 הן נשמרות עד לקבלת הודעה 1.
כאשר התקבלה הודעה 1, נשלח ACK בהתאם וההודעות עלו לאפליקציה.

בעיות מספור-

נשאלת השאלה כמה מספרים אנו צריכים בשביל למספר את החלון,
לדוגמה- אם הקצנו bit 2 אז יש לנו 4 אפשרויות מספור.





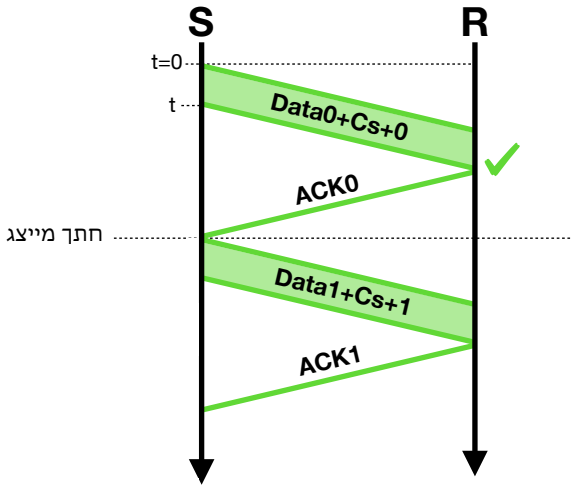
לתחנה המקבלת אין דרך להבדיל בין מקרה 1 למקרה 2.
כלומר, מבחינתה במקרה 2, החלון ה-2 המתקבל הינו חדש
ואין לה דרך לזהות שזהו שידור חוזר.

לכן, המספור המינימלי הינו- גודל החלון X 2

נצילות הפרוטוקולים Go Back N ו- Selective Repeat

$$t = \frac{F}{R}$$ זמן שידור ההודעה

$$\tilde{R} = \frac{N \cdot F}{t + RTT} = \frac{\text{מידע}}{\text{זמן}}$$



כלומר, בעבור החתך המייצג אנו שולחים
N=1 (הודעה 1) בזמן t+RTT

$$u = \frac{\tilde{R}}{R} = \left(\frac{N \cdot F}{\frac{F}{R} + RTT}\right) \cdot \frac{1}{R} = \frac{N \cdot \frac{F}{R}}{\frac{F}{R} + RTT} = \frac{\text{פרק זמן שליחת המידע}}{\text{פרק זמן כולל}}$$ (נצילות)

- N- גודל החלון (גודל ה- buffer).
- ככל שנגדיל את גודל החלון, כך נגדיל את קצב השידור האפקטיבי.
- גודל החלון קובע את נצילות הפרוטוקול.
- כמובן שהנצילות לא יכולה לעבור את ה-100% ולכן גודל החלון מוגבל לפי זמן המחזור (RTT)

TCP
Transmission Control Protocol

פרוטוקול תעבורה מבוסס קישור אשר מבטיח העברה אמינה של נתונים בין שתי תחנות ברשת מחשבים באמצעות יצירת חיבור מקושר (Connection Oriented). מעביר את הנתונים שהועברו באמצעות IP, מוודא את נכונותם, ומאשר את קבלת הנתונים במלואם או מבקש שליחה מחדש של נתונים שלא הגיעו בצורה תקינה.

קשר נקודה לנקודה (point to point)-
ישנו שולח אחד ומקבל אחד. אין דרך להעביר מידע לכמה אנשים בפעילות שליחה אחת.
לאחר פתיחת הקשר, ישנה התקשרות דו כיוונית.

אמין-
הודעות לא הולכות לאיבוד, מגיעות בצורה תקינה ובסדר הנכון.

Stream-
העברת מידע רציף.

Piplined-
בכדי לשפר את יעילות הפרוטוקול, הודעות נשלחות במקביל.
בשל כך קיימים המנגנונים- TCP Congetion, Flow Control.
כלומר, ישנה בקרת עומס וזרימה שנשלט לפי גודל החלון בכדי להתאים את שליחת ההודעות לתנאי הרשת.

Full Duplex Data-
זרימת מידע דו כיוונית על גבי אותו חיבור.
ישנו- Maximum Segment Size - MSS.
ב-TCP עובדים עם Segment וה- Stream של המידע מחולק ל-Segment.
לכן, ה- MSS הינו גודל ההודעה (Segment) המקסימלי.

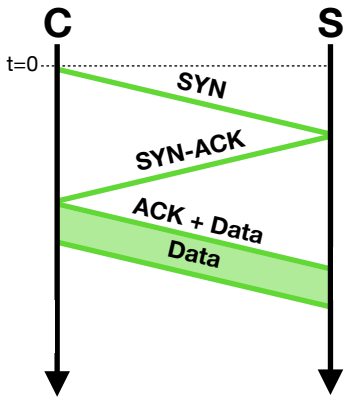
Conection Oriented-
יש צורך בביסוס הקשר לפני שליחת מידע.
הדבר מתבצע בעזרת 3HS שמאתחלת את מצבי השולח והמקבל לפני העברת נתונים בניהם.

Flow Controlled-
מתאם את קצב שליחת הנתונים גם עם תחנת היעד ולא רק ל]י תנאי הרשת (השולח לא ישלח יותר מידע ממה שהמקבל מסוגל לקבל).

הפרוטוקול רץ במחשבי הקצה ולא בנתבים שבדרך.

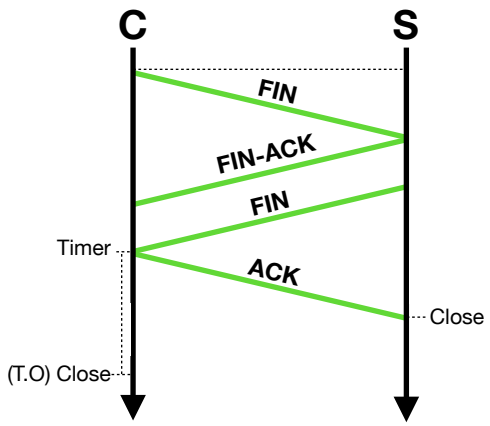
TCP יוצר ערוץ תקשורת אמין מעל שכבת הרשת המספקת ערוץ תקשורת לא אמין.

ניהול קשר TCP



תהליך פתיחת הקשר - 3HS

1. תחילה שולח הלקוח הודעת SYN שפירושה תחילת הקמת קשר.
2. השרת שולח בחזרה הודעת SYN-ACK המאשרת את קבלת הודעת ה-SYN ומאשרת את פתיחת הקשר מצידו. (לאחר הודעה זו, הלקוח יכול להתחיל לשלוח נתונים.)
3. הלקוח שולח הודעת ACK המציינת את סיום מיסוד הקשר.



תהליך סגירת קשר-

1. הלקוח שולח הודעת FIN לשרת שפירושה הוא הרצון של הלקוח לנתק את הקשר.
2. השרת מקבל את הודעת ה-FIN ושולח הודעת ACK שמציינת את קבלת בקשת הניתוק. בשלב זה השרת עדיין יכול לקבל מידע.
3. השרת שולח הודעת FIN המבקשת לנתק לנתק את הקשר גם כן, ומנקודה זו לא מועברים עוד נתונים בין התחנות.
4. הלקוח שולח הודעת ACK המאשרת את בקשת השרת לניתוק הקשר.
5. הלקוח מפעיל Timer באורך הגדול פי 2 מ-Timer הודעות רגילות.
6. לאחר סיום ה-Timer, הלקוח סוגר את הקשר ואילו השרת סוגר בעת קבלת ה-ACK.

הגדרות-

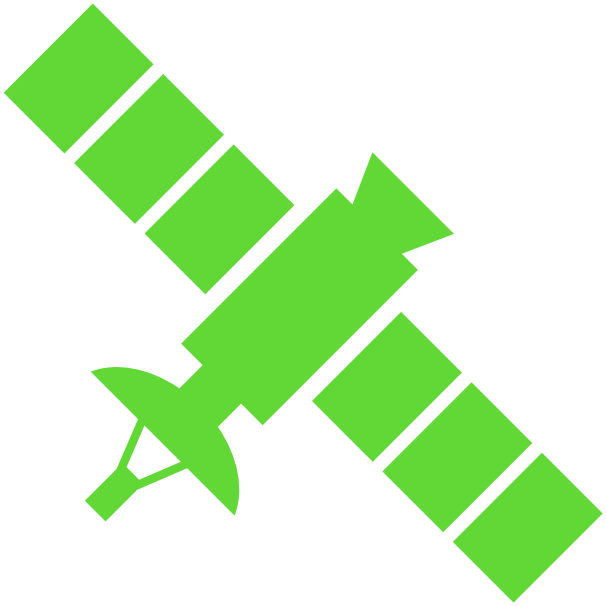
1. **Sequence Number-** המספור נקבע לפי מיקום ה-Segment בתוך ה-Buffer הכללי של ההודעה. (המספר הסידורי של ה-bit הראשון באותו Segment)
2. **מיספור ACK-** ACK עם מספר ה-Segment שחסר ב-Buffer.

TCP תחנה שולחת-

התחנה השולחת מקבלת את המידע משכבת האפליקציה ומחלקת את המידע לסיגמנטים (Header + Data) שגודלם נקבע לפי MSS (maximum segment size).
התחנה השולחת תשלח את ההודעות בחלון שלא נשלחו עוד, והן ישלחו לפי הסדר.

Timer יופעל בשידור ההודעה הראשונה בחלון.
ה-Timer יאופס בקבלת ACK המאשר את קבלת הודעה תקינה ובחלון עדיין קיימות הודעות שנשלחו ולא אושרו.
גם עבור הודעות SYN, FIN מופעל Timer.

בקבלת ACK תקין (ACK שמאשר הודעות בחלון) אני מקדם את החלון ומשדר את ההודעות החדשות שנכנסו לחלון

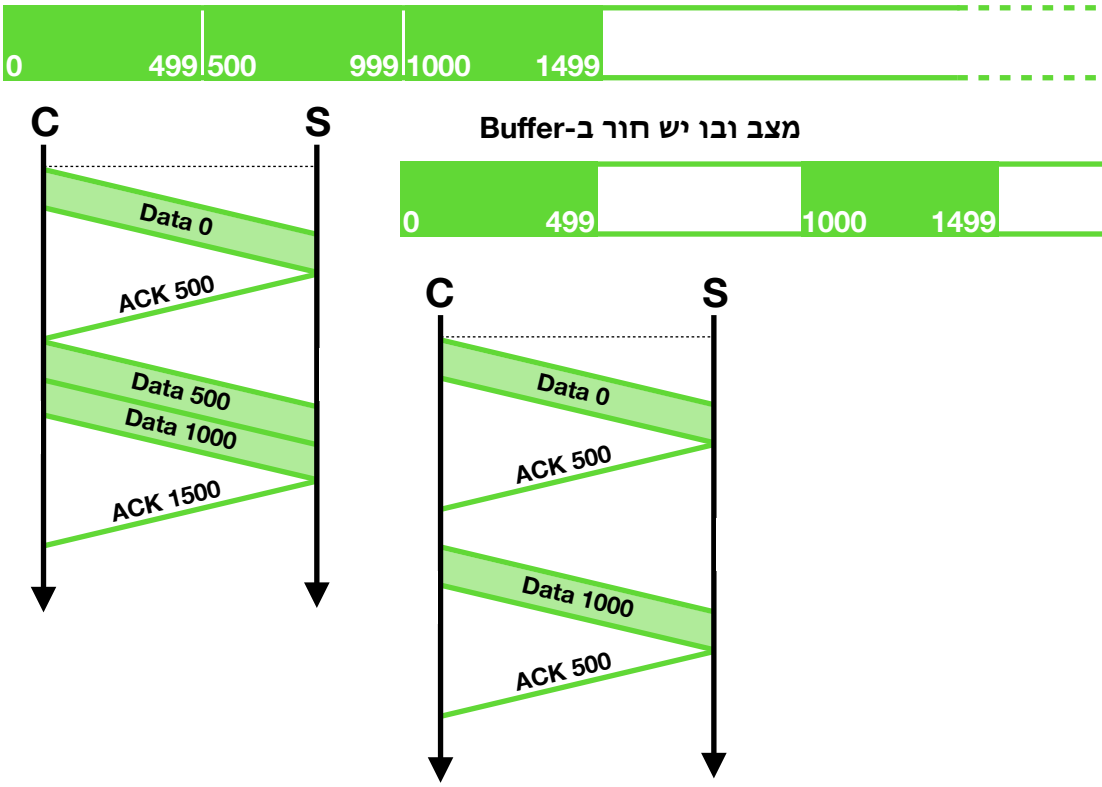


TCP תחנה מקבלת-

תפקידה לקבל מידע, לייצר רצף מהמידע המתקבל ולהעבירו לאפליקציה.
מכיוון שגודל החלון בפועל הוא יחסית גדול ושליחת ACK על כל הודעה תגרום להצפה ברשת, התחנה המקבלת שולחת ACK פעם כן ופעם לא.

במידה והתקבלה הודעה תקינה, לפי הסדר המצופה (ההודעה ברצף), וכל ההודעות שהגיעו קודם לכן כבר מאושרות (כלומר, כבר נשלחו עבורם ACK-ים ואין חורים ב-Buffer של הצד המקבל) אז משהים ה-ACK לפרק זמן של 500 ms.
אחרת, יישלח ACK עם האינדקס החסר ב-Buffer.
במידה ואכן ה-ACK מושהה, ובתוך 500 ms לא התקבלה הודעה חדשה, ה-ACK המושהה ישלח לתחנה המשדרת. הרי אם כן הגיעה הודעה חדשה אז היא תגרום לשליחת ACK שיאשר גם את ההודעה הקודמת.

מצב תקין



מצב ובו יש חור ב-Buffer

window

גודל חלונות ההזזה אינו קבוע, ומשתנה עם הזמן ע"י הצדדים, כאשר כל צד מגדיל או מקטין את החלון בהתאם לכמות המידע שהוא מוכן לקבל. בנוסף, משתנה גודל חלון ההזזה גם בהתאם לרוחב הפס של הרשת (כמות המידע שניתן לשלוח בזמן נתון), ובכך מאפשר לשני הצדדים לבצע בקרת זרימה.
בכדי לשלוט על גודל החלון ובכך למנוע מקרה ובו שצד אחד משדר יותר מידע ממה שהצד השני מסוגל לעבד, קיימים שני מנגנונים אשר באים לפתור בעיה זו.
window = min (congestion, flow) כאשר congestion זה חלון בקרת העומס ו flow זה חלון בקרת הזרימה.

מנגנון בקרת זרימה - Flow Control-

מנגנון זה מאפשר להגביל את השולח על מנת לא להציף את הצד המקבל.
בכל ACK, שולחת התחנה המקבלת גם את גודל הזיכרון הפנוי אצלה ב-Buffer.
התחנה המשדרת מגבילה את גודל החלון שלה כך שלא ישלח מידע מעבר לגודל הזיכרון הפנוי בצד המקבל. כלומר, לא ניתן להגדיל את גודל החלון מעבר למה שנקבע ע"י ה-Flow Control (גם אם אמורים להגדיל), אך יתבצע Sliding Window.
בצד המקבל, כאשר מתקבלות הודעות ברצף, ההודעות מועברות לאפליקציה (מתחילת ה-Buffer לפי הרצף).

מנגנון בקרת עומס - Congestion Control-

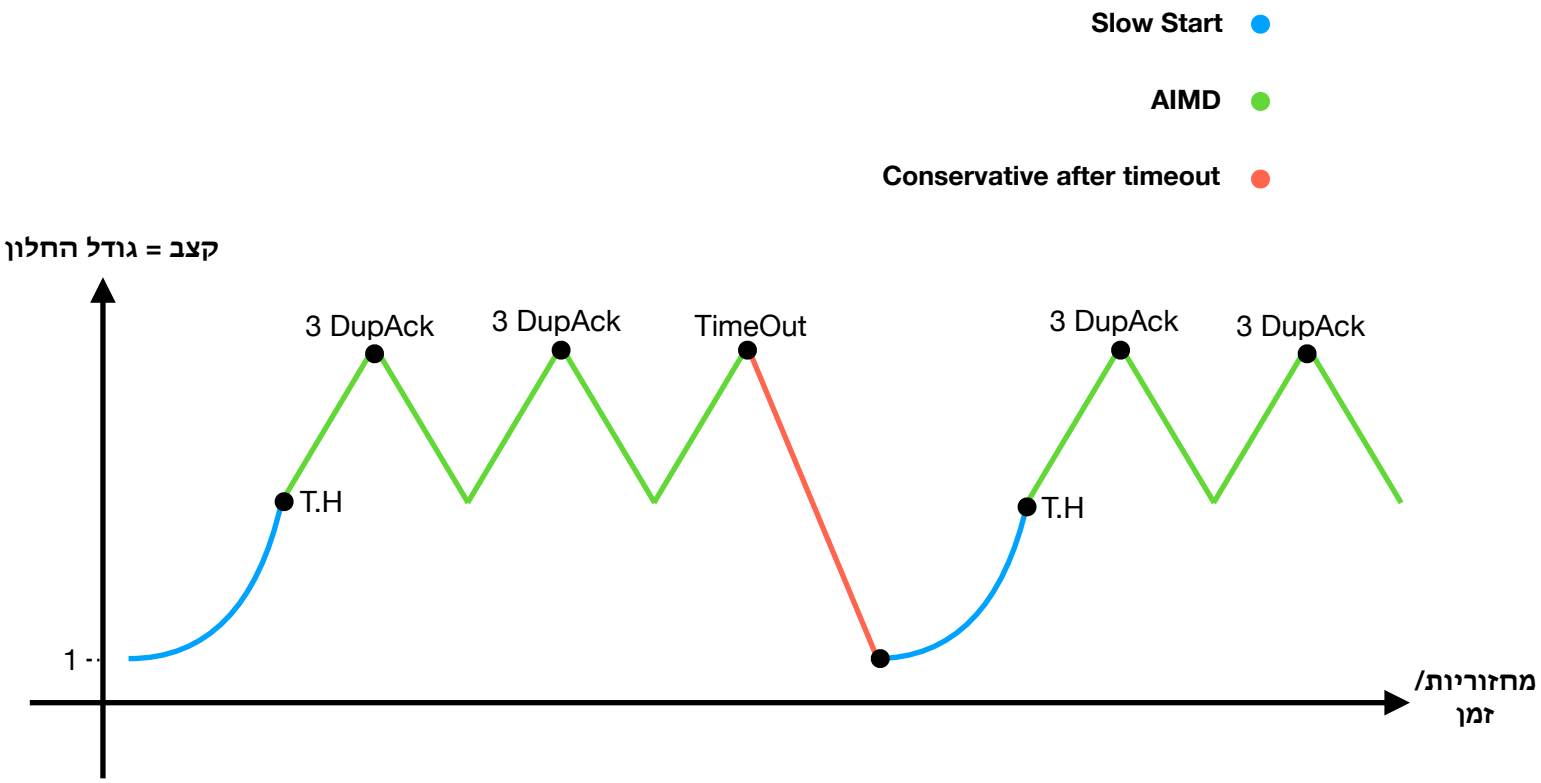
מתאים את קצב העבודה של הפרוטוקול (גודל החלון) לקצב העברת הנתונים של הרשת.
מכיוון שאין לנו דרך לדעת את הקצב של הרשת, יש מנגנון של ניסיון וטעייה.
למנגנון יש 2 שלבים:

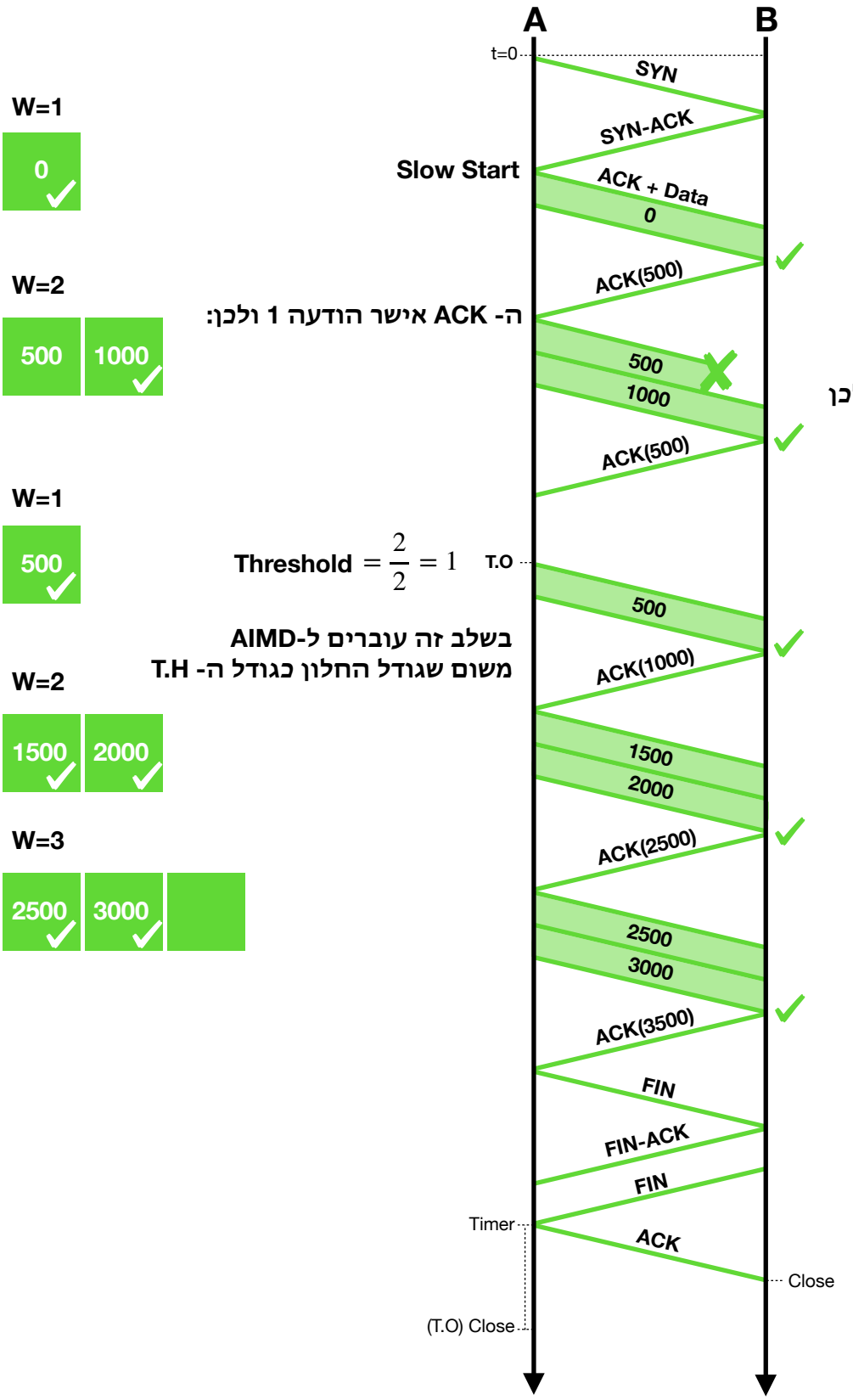
1. **Slow Start-** גידול אקספוננציאלי אשר מתחיל לפעול בתחילת העבודה או לאחר TimeOut. תחילה, גודל החלון נקבע ל-1. לאחר מכן עבור כל ACK שמתקבל, גודל החלון יגדל לפי מספר ההודעות שה ACK מאשר בצורה אקספוננציאלית (2,4,8,16...) תהליך זה מתקיים עד אשר מגיעים ל- Threshold (ערך סף- מחצית מגודל החלון לפני TimeOut או בהתחלה לפי המספר שהוגדר).
2. **AIMD-** עוזר להיצמד לקצב של הרשת, גידול לינארי המתבצע לאחר 3 DopACK או לאחר סיום תהליך Slow Start. החלון יגדל ב-1 לאחר קבלת אישור על כל ההודעות בחלון. במצב ולא מתבצע אישור לכל ההודעות בחלון, יתבצע רק Sliding Window. תהליך זה מתקיים עד TimeOut או 3 DopAck. כאשר יש 3 DupAck, משדרים את ההודעה החסרה ולאחר קבלת ACK על ההודעה החסרה מקטינים את החלון בחצי וממשיכים להגדיל את החלון כרגיל ב-1.

שידורים חוזרים של הודעות-

שידור חוזר יחול באחד משני המצבים הבאים:

1. **TimeOut-** במקרה זה, גודל החלון יקטן ל-1 וההודעה שנמצאת בחלון שלא אושרה תשודר שוב. לאחר TimeOut קביעת גודל החלון תיעשה ע"י Slow Start. כאן ה-Threshold ייקבע כמחצית מגודל החלון שהיה לפני ה-TimeOut.
2. **3 Dup ACK-** במקרה זה, תשודר ההודעה החסרה. לאחר קבלת ה-ACK שמאשר את ההודעה החסרה נקטין את גודל החלון למחצית מגודל החלון האחרון. כאן ה-Threshold ייקבע כמחצית מגודלו. בנוסף, לאחר 3 Dup ACK, קביעת גודל החלון תיעשה ע"י AIMD.





ה- ACK אישר הודעה 1 ולכן:

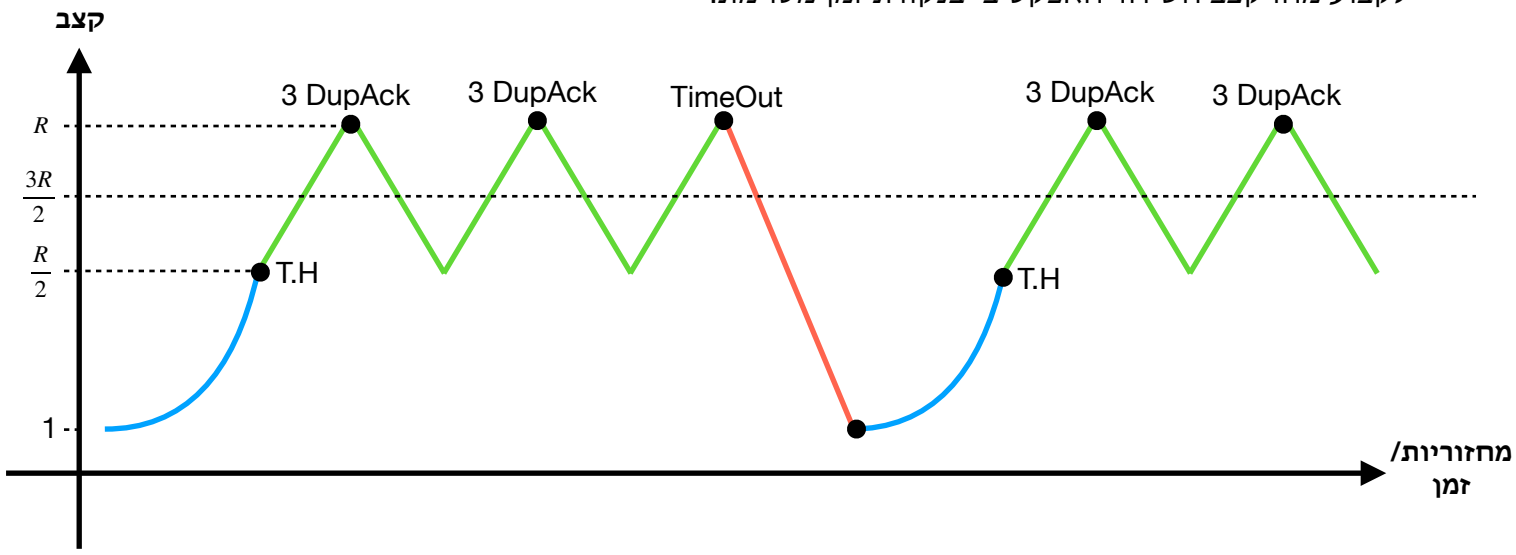
$$\text{Threshold} = \frac{2}{2} = 1$$

בשלב זה עוברים ל-AIMD
משום שגודל החלון כגודל ה- T.H

משום שב- W=1 כל החלון אושר,
ניתן לעשות Sliding Window
ולהגדיל את החלון.
אבל, ב- W=2 לא כל החלון אושר ולכן
ניתן לעשות רק Sliding Window

נצילות TCP

קצב השידור האפקטיבי תלוי בגודל החלון, ובפרוטוקול TCP גודל החלון משתנה כל הזמן ולכן לא ניתן לקבוע מהו קצב השידור האפקטיבי בנקודת זמן מסוימת.



כאשר אנו ב- 3 DupAck, אנו מנצלים את כל רוחב הפס (עובדים בקצב השידור של הרשת).
משום שברגע שנעבור את קצב השידור של הרשת נתחיל לאבד חבילות.

כאשר אנו מגיעים ל- 3 DupAck אנו מקטינים פי 2 את גודל החלון,
לכן קצב השידור האפקטיבי קטן ל- $\frac{R}{2}$

קצב העבודה הממוצע הינו- $\frac{R+\frac{R}{2}}{2}=\frac{3R}{4}$

ולכן הנצילות הינה- $u=\frac{\tilde{R}}{R}=\frac{\frac{3R}{4}}{R}=\frac{3}{4}=75\,\%$

כלומר, TCP מגיע במקסימום ל- 75% מקצב השידור של הרשת.

TCP Fairness

TCP הינו הוגן (Fairness).

מטרת ההגינות-

במידה וקיימים K תהליכים/חיבורים של TCP שחולקים את אותו צוואר הבקבוק של קו תקשורת בעל רוחב פס (קצב שידור) של R, אז כל תהליך יקבל $\frac{1}{K}$ מרוחב הפס R.

כלומר, לכל תהליך יהיה $\frac{R}{K}$ רוחב פס.

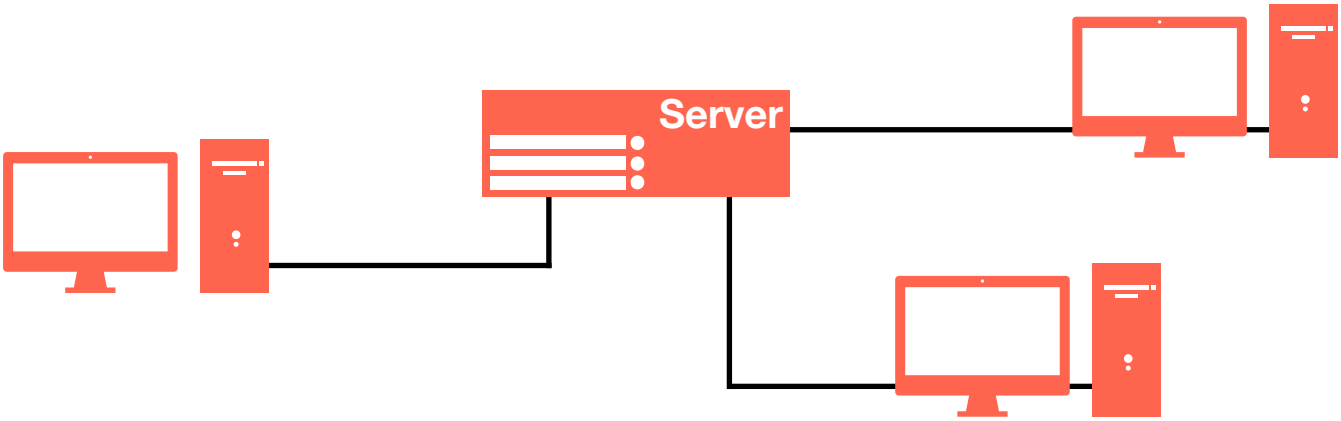
גם כאשר גודל החלון משתנה, כל החיבורים מושפעים בצורה שווה.
ברגע שרוחב הפס הכולל עולה על R, אחד או יותר מהשולחים מזהים עומס (ע"י קבלת 3 Dup ACK) ומורידים את קצב השידור בחצי. במקרה זה כל האזור יציב.

UDP אינו שומר על הגינות משום שהוא ירצה לנצל בכל סיטואציה את כל רוחב הפס, גם אם יאבד הודעות כתוצאה מכך.

- הקשר בין Go Back N, Selective Repeat ו- TCP הינו בכך שהם שלושה פרוטוקולים שונים לאותה מטרה עם מנגנונים דומים. מטרתם לייצר ערוץ תעבורה אמין מעל ערוץ לא אמין, כאשר התוספות של כל פרוטוקול הינם על מנת להגדיל את יעילות הפרוטוקול.
- TimeOut ב-TCP מחושב לפי דגימות של RTT.
- יש צורך במנגנון congestion control משום שלא ניתן לדעת את קצבי שידור הרשת בין המקור ליעד משום שהיא משתנה כל הזמן.
- יש צורך במנגנון flow control משום שאנו רוצים להימנע ממצב שהיעד זורק הודעות עקב תור עמוס.
- טיפול ב-TimeOut נחשב חמור משום שלא היה ניתן להגיע למצב של 3 DupAck. כלומר, מצב הרשת קשה. לעומת זאת, במצב של 3 DupAck נחשב פחות חמור משום שמצב הרשת בעייתי אך עדיין ישנו קשר עם הצד השני והוא מקבל חלק מההודעות.
- בעת סגירת קשר ישנו צורך בצד השולח להמתין TimeOut משום שבמידה וה-ACK שנשלח לשרת אינו מגיע בצורה תקינה, השרת ימשיך לשלוח הודעות FIN ללקוח. אך במידה והלקוח כבר סגר את הקשר, הוא לא יוכל לקבל עליהם את ה-FIN ולשלוח ACK חדש.
בשל כך, השרת ימשיך לשלוח ללקוח הודעות FIN מבלי לקבל עליהם ACK ולכן לא יוכל לסגור את הקשר. לכן הלקוח חייב להמתין TimeOut בכדי לוודא שהודעת ה-ACK הנשלחת לשרת אכן הגיע.

שכבת הרשת

בכדי להבין את תפקידה של שכבת הרשת תחילה יש להבין את מהותה, קודם למדנו על שכבת התעבורה, והבנו שהיא מאפשרת לנו לשלוח הודעה ממחשב אחד למחשב אחר. כמו שלמדנו, שכבת התעבורה עשויה לדאוג לכך שלא יהיה כישלון בהעברת המידע בין הצדדים. אם כך, מדוע צריך את השכבה השלישית?
כדי לענות על השאלה הזו, נתחיל מלחשוב על רשתות ישנות, שהיו קיימות לפני כ-30 שנים. דמיינו עולם ללא טלפונים ניידים וללא רשת האינטרנט שאתם מכירים כיום. רשת סטנדרטית בתקופה הזו הייתה יכולה לכלול כמה מחשבים בודדים, שחוברו באמצעות כבלים וקופסה קטנה.



בעולם כזה, בו רשתות כוללות מספר מצומצם של מחשבים בלבד, קל יחסית לגרום למידע לעבור מצד לצד, אך רשתות אלו אינן דומות כלל וכלל לרשתות שאנו מכירים היום. המחשב שלכם, שנמצא כאן בישראל, כמו גם הסמארטפון שלכם, נמצאים באותה רשת כמו שרת Google שנמצא בארצות הברית, והיא רשת האינטרנט. על מנת להעביר מידע בין מחשבים (ובכלל בין כלל הישויות כמו נתבים או שרתים) ברשת האינטרנט, עלינו לעבור בדרך בהרבה רכיבים שונים.
שכבת התעבורה, עליה למדנו, מניחה כי אפשר להעביר חבילת מידע בודדת ממחשב א' למחשב ב'. שכבת הרשת היא האחראית לתהליך זה.

מטרת שכבת הרשת היא להעביר חבילות מידע מישות אחת אל ישות אחרת-

כשישות אחת מעוניינת לשלוח מידע לישות שניה, המידע הזה מחולק בתורו לחבילות מידע. כשאנו מתייחסים ל"חבילות מידע" בשכבת הרשת, אנו קוראים להן בשם Packets, ובעברית- חבילות (או "פקטות"). היות שרוב המידע שעובר באינטרנט מועבר באמצעות השכבה השלישית, אנו מכנים לרוב כל מידע כזה בשם "פקטה", כפי שראיתם עד כה.
שימו לב שכל נקודת קצה בפני עצמה אינה מכירה את מבנה הרשת הכולל. כלומר, מחשב א' (המקור) לא יודע איזה רכיבים נמצאים בינו לבין מחשב ב' (היעד). הוא "מבקש" משכבת הרשת לשלוח את החבילה עבורו, ובאחריות שכבת הרשת להבין את מבנה הרשת.

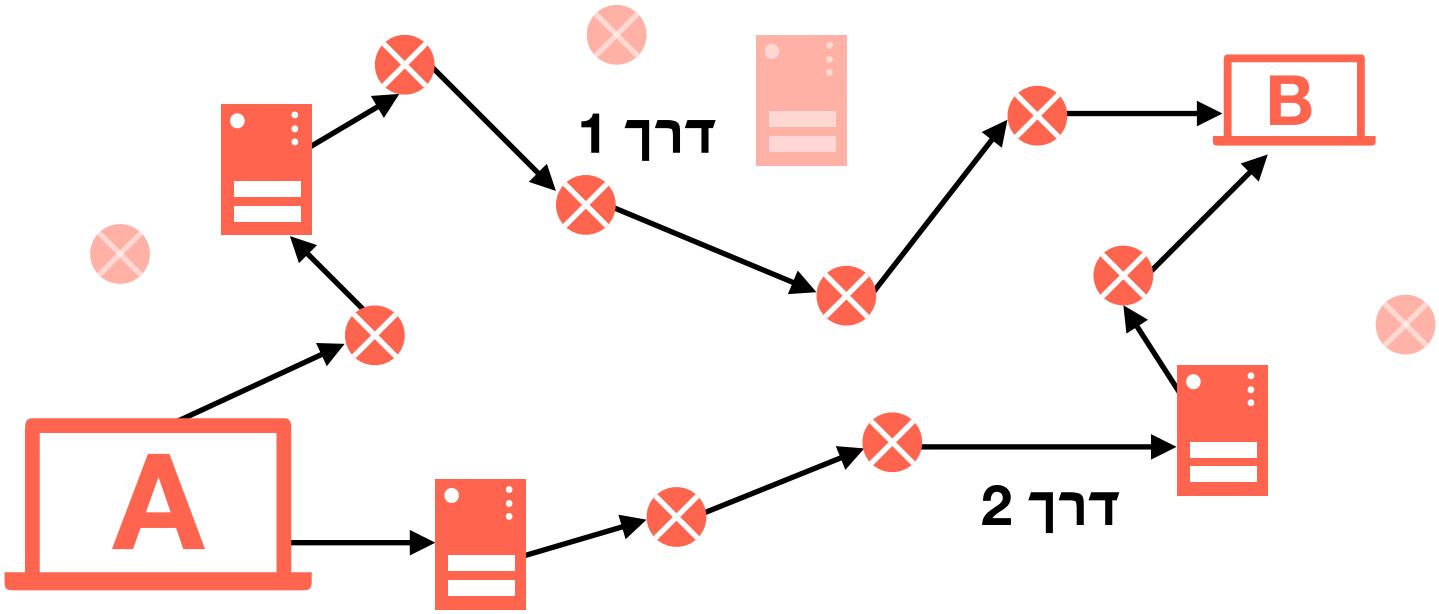
אתגרים העומדים בפני שכבת הרשת-

בפני שכבת הרשת עומדת משימה לא קלה בכלל. חבילה שמגיעה ממחשב א' למחשב ב' עוברת בדרך קשה ומפותלת. בין השאר, שכבת הרשת עשויה להתמודד עם:

1. **חומרות שונות**- יתכן שבדרך בין מחשב א' למחשב ב' יהיו רכיבים שונים לחלוטין, כשאחד מהם הוא שרת עצום ואחד מהם הוא קופסה קטנה.
2. **תקנים שונים**- יתכן שהתקשורת בין מחשב א' למחשב ב' תעבור בלוויין בחלל, לאחר מכן באמצעות כבל רשת סטנדרטי, לאחר מכן באמצעות WiFi ובחזרה.

Routing

מטרה נוספת חשובה של שכבת הרשת היא ניתוב, הכוונה היא החלטה על הדרך שבה יש להגיע מנקודה א' לנקודה ב'. הדבר דומה למציאת דרך נסיעה ברכב, למשל כמו שעושה האפליקציה Waze. על האפליקציה להבין מה הדרך שבה על הרכב (או במקרה שלנו - חבילת המידע) לעבור כדי להגיע מהמקור אל היעד.



כאן מוצגות שתי דרכים שונות בהן יכולה לעבור חבילת מידע ממחשב A למחשב B. אם נחזור לעולם המושגים של Waze, החיצים מסמנים למעשה כבישים בהם הרכב יכול לעבור. כמובן שניתן לבחור בהרבה דרכים אחרות ואין מניעה מכך.
על שכבת הרשת להחליט באיזה דרך להעביר כל חבילת מידע שהגיעה אליה, והיא יכולה לבחור בכל דרך שתרצה.

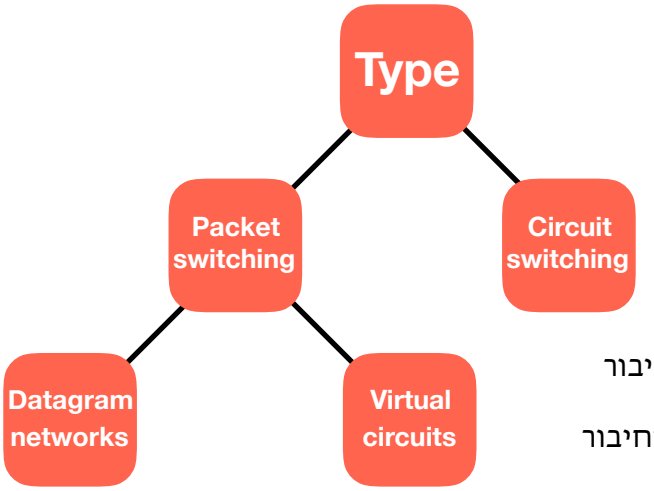
מה צריכה שכבת הרשת לדעת בכדי להחליט כיצד לנתב-

1. על שכבת הרשת להכיר את מבנה הרשת. אם שכבת הרשת תדע על כל הרכיבים שנמצאים בתרשים, היא תוכל להחליט על דרך מלאה אותה יש לעבור בכדי להגיע ממחשב A למחשב B. בדיוק כמו שעל Waze להכיר את כל הכבישים במדינה.
2. על שכבת הרשת לדעת האם קיים בכלל חיבור בין המקור ליעד. באם לא קיים כרגע אף חיבור בין מחשב A למחשב B, עליה להודיע למחשב A שהיא אינה מסוגלת להעביר את חבילת המידע שלו.
3. שכבת הרשת צריכה להבין מהי הדרך הכי מהירה. שוב, בדומה ל-Waze, המטרה של השכבה היא לאפשר לחבילת המידע להגיע בדרך המהירה ביותר.
4. באחריות שכבת הרשת להבין אילו דרכים אסורות. כפי שיתכן וכביש מסוים חסום כרגע, או שאי אפשר לעבור בו בגלל סכנת מפולת, כך גם בעולם הרשת - ישנם נתיבים אשר לא ניתן לעבור בהם.

פרוטוקול שכבת הרשת קיים בכל מחשב ובכל נתב. תחילה בצד השולח, מועברת הודעה משכבת האפליקציה לשכבת התעבורה, שכבת התעבורה יוצרת מההודעה מקטעים (Segments).
לאחר מכן, שכבת התעבורה מעבירה את המקטע לשכבת הרשת של המחשב השולח, וזו יוצרת ממנה מנה / חבילת מידע (Packet).

אם כן, קיימות 2 שיטות עיקריות להתקשרות והעברת נתונים עליהן דובר במבוא- והם מיתוג מעגלים ומיתוג חבילות,

Circuit switching (מיתוג מעגלים)- לפני העברת חבילות מידע בין תחנת המקור לתחנת היעד יש להקים "מעגל" (או ערוץ תקשורת) ייעודי להתקשרות זו . כל עוד ההתקשרות ממשיכה, מידע אחר לא יכול לעבור באותו ערוץ תקשורת.
Packet switching (מיתוג מנות/חבילות)- חבילות נשלחות כאשר מצטברת כמות מסוימת של נתונים, גם אם אלו לא מייצגים את כלל הנתונים שיש להעביר. בשיטה זו אין מקצים מראש מסלול פיזי בין המקור ליעד, וכל תשדורת נשלחת במסלול המתאים לאותו רגע ומנותבת בין צמתים.



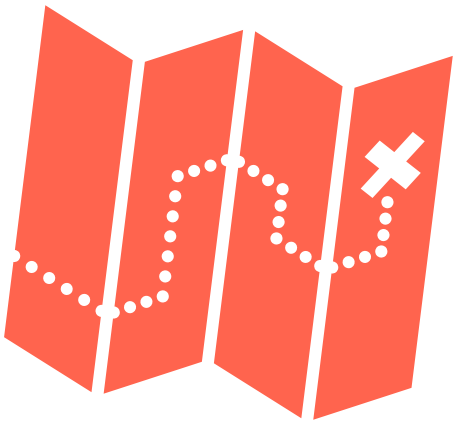
בנוסף, מיתוג חבילות ניתן לחלק לשתי שיטות

Virtual circuits-
מיתוג מעגלים וירטואלי היא שיטה בה איחדו את 2 השיטות הקודמות, חלוקה לחבילות + יצירת מסלולים.

החיבור הווירטואלי מורכב מקביעת המעגל מקצה לקצה, כלומר, נתיב בין המקור ליעד וכל חבילה נושאת זיהוי של חיבור וירטואלי (VC) ולא של מחשב היעד.
כל נתב במסלול של מקור-יעד, שומר את החיבור עבור כל חבילה שעוברת בטבלת הניתוב.

המקור ששולח את המסר יכול לבקש מהרשת להקים VC בינו לבין היעד. הרשת בוחרת את המסלול שבו יעבור המסר, ונותנת לכל ערוץ מס' VC. ברגע שהמנה מגיעה עם מס' VC לאחד הנתבים, היא צריכה להחליף מס' VC שיתאים לערוץ הבא. תפקיד הנתב להחליף בעזרת טבלת הכניסות את מס' ה-VC של המנה המגיעה מהערוץ הנכנס במס' VC חדש שמתאים לערוץ שבו המנה יוצאת. הטבלה מתעדכנת עם כל יצירה של VC, וכש-VC מגיעה לסיומה, הכניסות המתאימות נמחקות.

Datagram networks-
אינו מבוסס קישר. אין צורך בלשמור משאבים מאחר שאין נתיב ייעודי.
כל חבילה יכולה לעבור בכל נתיב ובכל נתב, אשר עליו להחליט על ידי שינוי דינמי של טבלת הניתוב שלו לאן לשלוח את החבילה.
מכיוון שכל החבילות חופשיות לבחור נתיב כלשהו, הן חייבות להיות קשורות אחת לשניה עם מידע מתאים על המקור ועל נתוני השכבה העליונה.
הנתונים יכולים להגיע ליעד בכל סדר, כלומר, הם לא צריכים להגיע לפי הסדר שבו הם נשלחו.
רשתות Datagram אינן אמינים כמעגלים וירטואליים, אך קל ויעיל, גם מבחינת עלות ליישום.



IP datagram format

32 bits

גרסת הפרוטוקול	אורך הפתיח	סוג השירות	אורך החבילה
מספר הזיהוי			סיביות חלוקה / סדר הרכבת המידע
זמן החיים		פרוטוקול שכבה מעל	Checksum של הפתיח
כתובת ה-IP של השולח			
כתובת ה-IP של היעד			

1. אורך כולל של החבילה (Data + Heade) - 16 bit.
2. שולח החבילה יכול לציין את סוג השירות שהוא מעוניין שהחבילה תקבל - 8 bit
3. אורך הפתיח המינימלי הינו 20 bytes, האורך מבוטא בכפילות של 4 בתים ולכן אורך מינימלי של פתיח יהיה 5. אורך מקסימלי של פתיח הינו 60 bytes - 4 bits.
4. גרסת פרוטוקול IP - 4 bits.
5. מציין את מיקומו של הפרגמנט בחבילה המקורית, בעזרתו ניתן לזהות את הסדר שבו יש להרכיב את המידע - 13 bits.
6. דגלים, משמשים כאשר חבילה מחולקת. למטרות פרגמנטציה וצירוף מחדש - 3 bits.
7. מספר מזהה משמש כאשר חבילות עוברות פרגמנטציה (כלומר, מחולקות לכמה חלקים). שדה זה משמש לצורך הרכבה מחדש של הודעות שחולקו, כלומר, המספר המזהה מצביע על חבילות שהן למעשה חבילה מקורית אחת - 16 bits.
8. מאפשר להבטיח שפתיח ה- IP נטול שגיאות - 16 bits.
9. מציין את הפרוטוקול אליו עוברת החבילה בשכבה מעל (כמו TCP, UDP, ICMP) - 8 bits.
10. ערך הנע בין 0 ל- 255 ומציין את כמות הנתבים בדרך שמותר לחבילה לעבור. עם כל תחנה שהחבילה עוברת בדרך, המספר קטן ב- 1. כאשר הערך נהיה 0 לפני שהחבילה הגיעה ליעדה היא מושמטת ולא מועברת הלאה.

-IPv4

גרסה 4 של פרוטוקול האינטרנט, או בקיצור IPv4, היא הגרסה הנפוצה של פרוטוקול IP, ומהווה את הפרוטוקול הבסיסי של רשת האינטרנט. כתובת IPv4 מורכבת מ-32 סיביות ומיוצגת באמצעות 4 מספרים עשרוניים המופרדים בנקודה. כל מספר מהווה מקבץ של 8 סיביות וגודלו נע בין 0 ל-255.

-IPv6

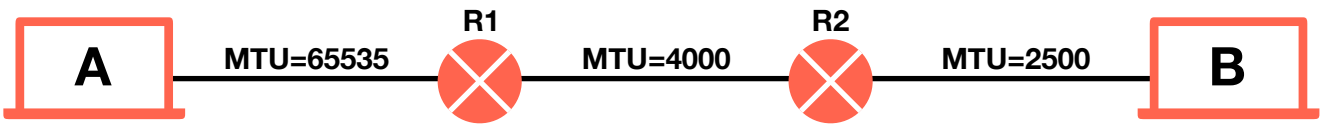
גרסה 6 של פרוטוקול ה-IP היא הגרסה העוקבת לגרסה 4 של הפרוטוקול (IPv4).
כתובת IPv6 מורכבת מ-128 סיביות: 64 הסיביות הראשונות משמשות לזיהוי תת-הרשת, ו- 64 הסיביות האחרונות משמשות כמזהה ממשק (interface ID).
הייצוג של כתובות IPv6 נעשה בדרך כלל באמצעות סדרה של 8 מספרים בני ארבע ספרות בבסיס הקסדצימלי שמופרדים בנקודתיים וכל אחד מהם מייצג 16 סיביות. לשם קיצור ניתן להשמיט את האפסים המובילים וכן רצף אחד של אפסים.
לדוגמה, הכתובת 2001:db8::ff00:42:8329 היא קיצור של הכתובת 2001:0db8:0000:0000:0000:ff00:0042:8329.
ישנן 2^{128} כתובות כאלו.
ב- IPv6 לא ניתן לבצע פרגמנטציה ולכן האחריות על כך עברה מהנתבים למחשבי הקצה.
גודל ה-Heafer הינו 40 bytes.

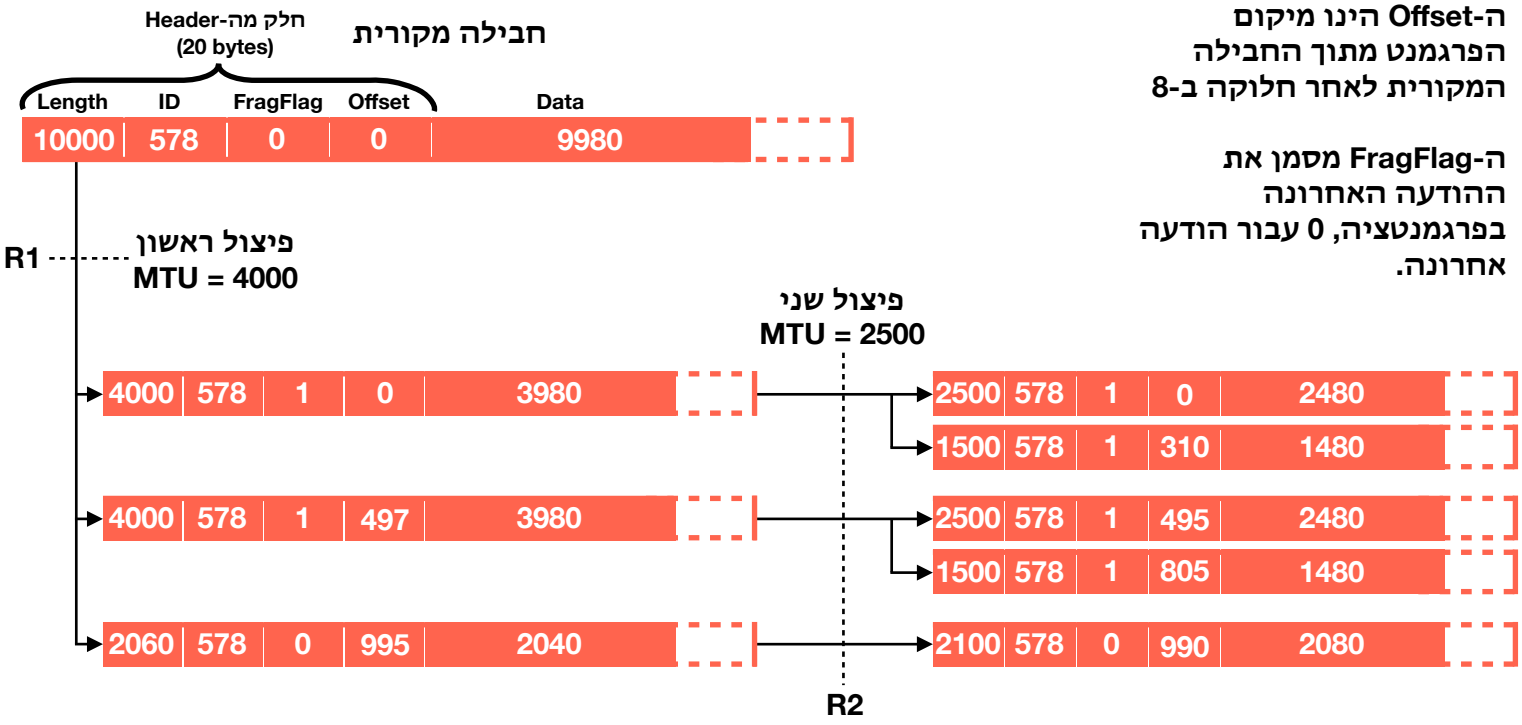
Maximum Transmission Unit - MTU

מציין את גודל המנה המקסימלי (בבתים) שניתן להעביר. ערכי ה-MTU יכולים להיקבע מראש (לדוגמה, ברשת Ethernet) או ברגע ההתקשרות. ככל שה-MTU יותר גבוה, כך רוחב הפס מנוצל בצורה יעילה יותר. כשה-MTU גבוה מדי, הזמן הדרוש להעברת מנה אחת גורם לעיכוב בהעברה של המנות הבאות אחריה (לאג). לדוגמה, מנה של 1500 בית - המנה הגבוהה ביותר המותרת בשכבת הקו של פרוטוקול Ethernet ומכאן המנה הגבוהה ביותר ברוב אזורי האינטרנט - תחסום מודם של 14.4k למשך כשנייה אחת.

IP Fragmentation and Reassembly-

תהליך של פרוטוקול IP אשר מפצל חבילות למנות קטנות כך שאותן מנות יוכלו לעבור כאשר ה-MTU קטן מגודל החבילה המקורית. על התחנה המקבלת להרכיב מחדש את החבילה ולהעביר אותה לשכבת הבאה. ההרכבה מחדש נועדה לקרות בצד המקבל, אך בפועל ניתן לעשות זאת על ידי נתב ביניים.
לדוגמה, נרצה להעביר הודעה בגודל 10000 בצורה הבאה:





ה-Offset הינו מיקום הפרגמנט מתוך החבילה המקורית לאחר חלוקה ב-8

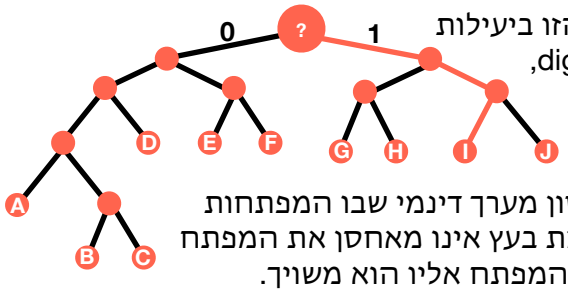
ה-FragFlag מסמן את ההודעה האחרונה בפרגמנטציה, 0 עבור הודעה אחרונה.

IP Address Lookup: Binary Tries

במבוא דיברנו על Subnet, על הנתב ועל אופן העבודה שלו, כעת נבין כיצד הנתב בוחר בפועל אילו הודעות לשייך לאיזה Subnet מתוך טבלת הניתוב שלו. על הנתב אם כן לקחת את כתובת ה-IP שניתנה לו בחבילה ולהתאים אליה את אחת מכתובות ה-Subnet הנמצאות אצלו בטבלה, אך עם כמה שמשימה זו נשמעת לא מורכבת מדי, על הנתב לעשות זאת במהירות ויעילות מרבית ולכן גם המעבדים של אותם נתבים שתוכננו לעבודה זו הם חזקים וממשוכללים יותר מהמעבדים של הנתב הביתי הרגיל.
אך עם זאת רשת האינטרנט היא גדולה ומסובכת ורק ממשיכה לצמוח, ועל כן ויכולות להיות בטבלה כמויות גדולות של כתובות נוסף על כך שעליו להעביר את המידע במינימום עיכוב.

1100

	Subnet
A	00001
B	00010
C	00011
D	001
E	0101
F	011
G	100
H	1010
I	1100
J	11110000



יש כמה שיטות ואלגוריתמים בכדי לבצע את המשימה הזו ביעילות ואחד הנפוצים שבהם נקרא- Tries המכונה גם digital tree, radix tree או prefix tree, הוא מבנה נתונים בצורת עץ בינארי המאפשר חיפושים מהירים של תת-מחרוזת כלשהי של מחרוזת נתונה, ויישומים נוספים הקשורים למחרוזות, כלומר הוא סוג של עץ חיפוש המשמש לאחסון מערך דינמי שבו המפתחות הם בדרך כלל מחרוזות. שלא כמו עץ חיפוש בינארי, צומת בעץ אינו מאחסן את המפתח המשויך לצומת זה, במקום זאת, מיקומו בעץ מגדיר את המפתח אליו הוא משויך.
לכל הצאצאים של צומת מסוימת יש קידומת משותפת של המחרוזת המקושרת לצומת, והשורש קשור למחרוזת הריקה. מפתחות בדרך כלל משויכים לעלים, אם כי כמה צמתים פנימיים עשויים להתאים את למפתח הרצוי.
אצלנו העץ בנוי כך שכל יציאה מקודקוד היא בהתאם לביט שאנו מסתכלים בכתובת ה-IP, כך לדוגמה אם אל הנתב תגיע כתובת המשויכת ל-Subnet 1100, נגיע אל קודקוד I כמתואר בתרשים.

Dynamic Host Configuration Protocol - DHCP

דיברנו לא מעט על כתובת IP, אותה כתובת לוגית בשכבת הרשת המשמשת כל ישות ברשת בכדי להזדהות, אבל איך רכיב מקבל כתובת IP? ישנן מספר דרכים לקבל כתובת IP, הנפוצות ביותן הן:

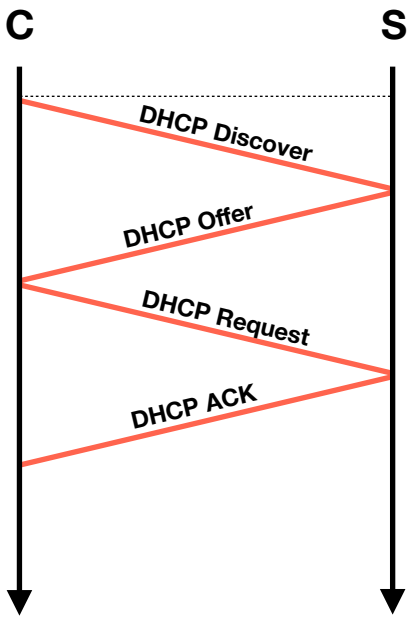
הקצאה סטטית-

על מנת לבחור כתובת IP בצורה סטטית, המשתמש נכנס להגדרות הרשת במחשב ומכניס ידנית את הכתובת הרצויה. על מנת שאפשרות זו תעבוד, עליו לבחור בכתובת שלא קיימת כבר ברשת.

הקצאה דינמית-

מתבצעת בדרך כלל באמצעות הפרוטוקול DHCP, פרוטוקול תקשורת המשמש להקצאה של כתובות IP ייחודיות למחשבים ברשת מקומית (LAN). בנוסף לכתובת ה-IP, שרת DHCP בדרך כלל יספק למחשב גם נתונים כמו ה-Subnet mask, כתובת שרת ה-DNS וכתובת שער הגישה (Gateway), כך שהמחשב יוכל להתחיל לתפקד ברשת ללא צורך בנתונים נוספים.

נאמר ואנו מחברים מחשב חדש לרשת. המחשב אינו יודע את כתובת ה-IP שלו, ולכן עליו לברר אותה. הדבר הראשון שהוא יעשה לשם כך, הוא שליחת הודעה בשם **DHCP Discover**. בבקשה זו, המחשב למעשה פונה לרשת עם כתובת IP מקור (0.0.0.0) ומספר מזהה נוסף בכדי שמי שמקבל את ההודעה ידע למי להחזיר תשובה (transaction ID) ומבקש: "האם יש כאן שרת DHCP שיכול להקצות לי כתובת IP?" בקשה זו נשלחת כמובן ב-Broadcast (255.255.255.255), כלומר לכל הישויות ברשת. המטרה היא ששרת DHCP שיראה את הבקשה יוכל לענות אליה, בעוד ישויות אחרות יתעלמו מבקשה זו. כעת, נאמר וישנם שני שרתי DHCP ברשת. שניהם יוכלו לענות למחשב המבקש כתובת IP עם הצעה. להודעה זו קוראים **DHCP Offer**. בהצעה, השרת כותב למחשב איזו כתובת IP הוא יכול לקבל, וכן פרטים נוספים כגון משך הזמן אותו הוא מבטיח להקצות את כתובת ה-IP הזו למחשב המבקש ולא לאף ישות אחרת ברשת. הודעה זו נשלחת אף היא לכתובת Broadcast - שכן אין עדיין כתובת IP למחשב שאמור לקבל את הבקשה. כעת המחשב קיבל שתי הצעות שונות לכתובות IP. עליו לבחור באחת מהן, ואז לשלוח בקשה לקבל באמת את הכתובת הזו. הודעה זו נקראת **DHCP Request**. בחבילת הבקשה, הלקוח מציין מי שרת ה-DHCP ממנו הוא רוצה לקבל את הכתובת, כמו גם את הפרטים אשר השרת הציע לו. גם הודעה זו נשלחת ב-Broadcast, וזאת על מנת ששאר השרתים ידעו שהמחשב בחר בשרת הספציפי הזה, ולא ישמרו עבורו את הכתובת.
לבסוף, השרת צריך להחזיר למחשב אישור סופי שהוא אכן הופך להיות שרת ה-DHCP שלו. להודעה זו קוראים בשם **DHCP ACK**. בהודעה זו, השרת חוזר על הפרטים שהוא נותן ללקוח. החל מעכשיו, הלקוח יכול להשתמש בכתובת ה-IP שניתנה לו באמצעות שרת ה-DHCP.
לשימוש ב-DHCP יתרונות רבים. בין השאר, הוא מאפשר לחסוך בכתובות IP בכך שהוא מקצה כתובות ללקוחות רק כשהם זקוקים להן, ולא כל הזמן. בנוסף, באמצעות DHCP ניתן לתת פרטי קונפיגורציה נוספים ללקוחות מלבד לכתובות IP (למשל- מי שרת ה-DNS שישמש את הלקוח).

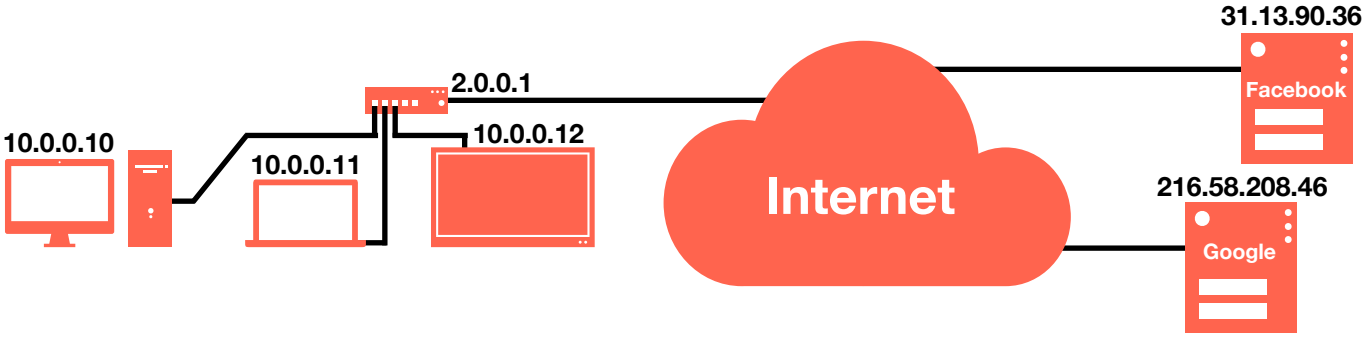


NAT

החל מסוף שנות ה-80' נוצרה בעולם בעיה אמיתית ומוחשית, נגמרו כתובות ה-IP. מסיבות שונות, נוצר מצב שבו למרות ש-IPv4 מספק כמעט 4.3 מיליארד כתובות שונות, התבצעה הקצאה לא יעילה שלהן ולא נותרו כתובות IP שניתן היה להקצות לרכיבי רשת חדשים שהזדקקו להן. בתחילת שנות ה-90', כשהאינטרנט זכה לגדילה מהירה מאוד, המחסור בכתובות ה-IP החל לפגוע בספקיות אינטרנט שפשוט לא יכלו להקצות כתובות IP ללקוחות שלהן.
נוצר אפוא צורך למצוא פתרון מהיר לבעיה. לפתרון הזה, קוראים NAT (Network Address Translation תרגום כתובת רשת) במילים אחרות, NAT זהו Subnet בעל כתובות לא חוקיות (פרטיות).

הרשת הביתית מחוברת אל האינטרנט, באמצעות הנתב בעל כתובת ה-IP, שקיבל מה-ISP, 2.0.0.1.
בנוסף, ישנם השרתים של Google ו-Facebook אליהם ירצו המחשבים ברשת הביתית לגשת. עד אשר החל השימוש ב-NAT, היה צורך לספק כתובת IP ייחודית לכל אחת מהישויות. כלומר שהמחשב הנייח, הלפטופ ואפילו הטלוויזיה החכמה, יזכו כל אחד לכתובת IP אמיתית וייחודית בעולם. דבר זה חיובי מהרבה בחינות, אך במציאות בה אין כבר כתובות IP לתת - הדבר לא יתכן. אי לכך, נוצר הרעיון של NAT.
לפי רעיון זה, כל הישויות בתוך הרשת הביתית קיבלו כתובות פרטיות, כלומר כתובות שיזהו אותן בתוך הרשת בלבד, ולא בעולם החיצוני. כתובות אלו אינן ניתנות לניתוב, כלומר, נתב באינטרנט שרואה חבילה שמיועדת לכתובת שכזו עתיד "לזרוק" אותה.
לשם כך, הוגדרו שלושה טווחים של כתובות פרטיות:

- 10.0.0.0/8 - בטווח זה ישנן 16,777,216 כתובות.
- 172.16.0.0/12 - בטווח זה ישנן 1,048,576 כתובות.
- 192.168.0.0/16 - בטווח זה ישנן 65,536 כתובות.



בצורה זו, הרשת הביתית "בזבזה" רק כתובת IP אחת, זו של הנתב שלב, ולא ארבע כתובות כמו שהיא הייתה צורכת לפני השימוש ב-NAT. אך כיצד כל זה עובד?
בשלב הראשון, מחשב א' ישלח הודעה ממנו (כתובת המקור: 10.0.0.10) אל Google (כתובת היעד: 216.58.208.46). את החבילה הוא ישלח אל הנתב.
בשלב השני, הנתב יקבל את החבילה, ויחליף בה את כתובת המקור לכתובת שלו. כלומר, החבילה עכשיו תשלח מכתובת המקור 2.0.0.1, אל כתובת היעד 216.58.208.46.

Port Src	Port Dest	IP Src	IP Dest
2222	80	10.0.0.10	216.58.208.46

IN		OUT	
IP	Port	IP	Port
10.0.0.10	2222	2.0.0.1	1234
10.0.0.11	2222	2.0.0.1	1245
10.0.0.12	2222	2.0.0.1	1567

Port Src	Port Dest	IP Src	IP Dest
1234	80	2.0.0.1	216.58.208.46

בשלב השלישי, השרת של Google יקבל את החבילה. שימו לב: השרת של Google כלל לא מודע לכתובת ה-IP של המחשב, או לעובדה שישנה ישות רשת מאחורי הנתב. הוא מודע אך ורק לישות הרשת בעלת הכתובת 2.0.0.1, וזהו הנתב של הרשת. כעת, Google יעבד את הבקשה של המחשב, וישיב עליה - אל הנתב. כאמור, מבחינתו של Google, הוא קיבל את הבקשה באופן ישיר מן הנתב.

בשלב הרביעי, הנתב יקבל את התשובה מהשרת של Google. החבילה תכיל את כתובת המקור של השרת של Google (כלומר 216.58.208.46), וכתובת היעד של הנתב 2.0.0.1. כעת, הנתב יבין שמדובר בחבילה המיועדת למעשה אל מחשב א'. אי לכך, הוא יבצע החלפה של כתובת היעד. כלומר, הוא ישנה את כתובת היעד מ- 2.0.0.1 ל-10.0.0.10, ויעביר את החבילה למחשב א'.

Port Src	Port Dest	IP Src	IP Dest
80	2222	216.58.208.46	10.0.0.10

בשלב החמישי, מחשב א' יקבל את הודעת התשובה מ-Google. מבחינת מחשב א', החבילה הגיעה מכתובת ה- IP של Google (שהיא 216.58.208.46) היישר אל הכתובת שלו (10.0.0.10), ולכן מבחינתו מדובר בתהליך "רגיל" לכל דבר.

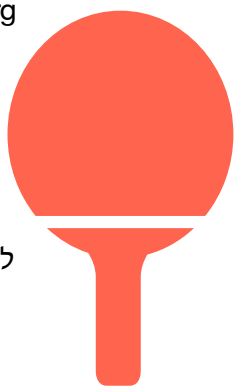
היתרונות של NAT הם משמעותיים. האפשרות לחבר מחשבים רבים לאינטרנט באמצעות חיבור אחד חוסכת הרבה כסף לארגונים וחברות, וגם חוסכת בכתובות אינטרנט. יתרון נוסף הוא בכך שניתן להסתיר את מבנה הרשת הפנימית מפני אנשים מבחוץ כגון האקרים או ספקית האינטרנט.
אחד החסרונות העיקריים הוא הקושי בשימוש בשירותים הדורשים יזימת חיבור TCP מהרשת החיצונית אל תוך הרשת הפנימית, או פרוטוקולים ללא-מצב (stateless) כגון UDP. ללא מאמץ מפורש מצד הנתב לתמוך בפרוטוקולים כאלה, חבילות המידע לא מגיעות ליעדן ולא נוצר החיבור. מצד שני יש הרואים בכך סוג של חומת-אש פשוטה, ולכן משתמשים שאינם רוצים לחשוף את שירותי הרשת במחשב שלהם רואים בכך יתרון. אחת הדרכים להתגבר על חוסר הנגישות מבחוץ אל מחשבים בתוך NAT היא Port Forwarding המכונה לעתים "תיעול" (מהמילה "תעלה", יצירת נתיב), היא פעולת ההעברה של פורט ממכונה אחת לאחרת. הדבר מאפשר למחשבים מרוחקים, אשר מחוברים לרשת האינטרנט, למשל, להתחבר למכונה מסוימת המחוברת לרשת פרטית, שאינה נגישה בדרכים רגילות. לדוגמה, תיעול השערים 6881 עד 6889 מהנתב למחשב האישי, יאפשר שימוש בפרוטוקול העברת הקבצים ביטורנט.
דרך נוספת היא בעזרת פרוטוקולים כאלה ואחרים שאחד מהם הוא STUN, פרוטוקול סטנדרטי שמאפשר למארח לדעת את ה-IP של הרשת הציבורית אם הוא נמצא בנקודת קצה, ומאפשר לאפליקציות שמתקשרות בהודעות בווידאו או בקול לעבור את ה- NAT.

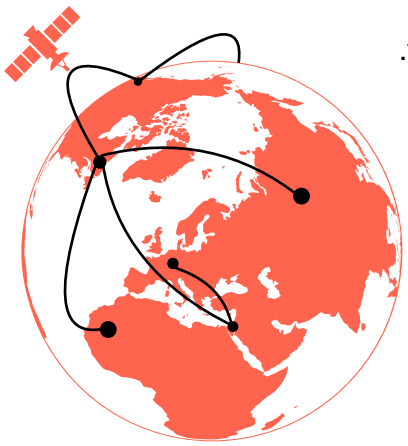
ICMP

Internet Control Message Protocol הוא פרוטוקול הנועד לעזור למצוא תקלות ברשת ולהבין את מצב הרשת.
הכלי המוכר ביותר המשתמש בפרוטוקול ICMP הוא הכלי ping, כלי זה נועד בכדי להבין האם ישות מסוימת "למעלה" - כלומר דולקת, עובדת ומגיבה לשליחת הודעות אליה. לחלופין, הוא יכול לוודא שיש תקשורת מהמחשב שעליו אנו מריצים את הפקודה אל הרשת אליה הוא מחובר.

-Ping

פינג היא חבילת נתונים בפרוטוקול ICMP הנשלחת ממקור מסוים ליעד מסוים ברשת לפי כתובתו. המטרה העיקרית לה היא משמשת היא בחינת תקינות התקשורת בין נקודת המקור לנקודת היעד. אפשר להשתמש בפינג לאתר אינטרנט ידוע על מנת למצוא האם יש תקשורת לרשת האינטרנט, למשל הפקודה (כמעט בכל מערכת הפעלה בת זמננו):
ping he.wikipedia.org, יכולה לשמש לבדיקת התקשורת לאינטרנט מהמחשב האישי. חיסרון של שיטה זו הוא שאתר עלול לא להיות זמין עקב עומס או תקלה טכנית, או שהוא חוסם את אפשרות משלוח הפינג אליו.
ההודעה שהמחשב שולח נקראות ECHO REQUEST ומכונה פינג, הודעה זו מכילה SeqNum ו- TimesTamp (המציין מתי נשלחה ההודעה) והתשובה שהוא מקבל נקראת ECHO RESPONSE המכונה פונג (pong), שמכילה את ה-Data מהודעת ה-Request ובשל כך ניתן לחשב את ה-RTT בין המקור ליעד. אם לא נשלחת הודעת Respomse אז ניתן להסיק שהיעד בלתי ניתן להשגה.





-Traceroute

זוהי שיטה למציאת הדרך שבה עוברת חבילת מידע בין מקור ליעד. שיטה זו עושה שימוש באחד השדות ב-Header של IP, שנקרא Time To Live, או בקיצור - TTL. השדה הוא באורך בית אחד, כלומר שהערך שניתן בו יכול להיות בין 0 ל-255.
תחילה נשלחת הודעה בעלת TTL=1, נתב המקבל חבילת מידע מפחית את הערך בשדה ה-TTL לפני שהוא מעביר את החבילה הלאה ליעדה. במידה והנתב רואה כי TTL=0, נשלחת הודעת Time Exceded בחזרה לשולח. כל הודעה כזו מכילה את כתובת ה-IP של הנתב ובכך ניתן לזהותו. ובדרך זו נשלחת כל פעם הודעה עם TTL יותר גבוה עד הגעה ליעד המבוקש, אשר מחזיר הודעת Echo Reply או עד שמתקבלת הודעת שגיאה כמו Port Unreachable. לבסוף התקבלה לנו "מפה" של כל הנתבים והצמתים בינינו לבין היעד.

גילוי MTU בעזרת ICMP-

בכדי לקבוע מהו ה-Path MTU המקסימלי בין מקור ליעד, תחילה נאתחל ב- Header של ה-IP את שדה ה-Flag ב-Do Not Fragment. כעת נגדיר את גודל ה-Length כגודל המנה המקסימלית שיכולה לעבור בנתיב נתון, מבלי שזו תצטרך להתפצל לתת-מנות, כעת נשלח הודעת Request אל היעד המבוקש. כאשר החבילה מגיעה לצומת שה-MTU שלה נמוך מגודל החבילה, נשלחת בחזרה חבילת ICMP עם ה-MTU של הצומת. התהליך נמשך עד שמגיעים לשרת היעד.
הבעיה בשיטה זו היא, שכיום מספר גדול של שרתים חוסמים את התעבורה של חבילות ICMP כדי למנוע התקפת מניעת שירות. בנוסף לכך, בפרוטוקול IP הנתיב משתנה באופן דינמי בעקבות אירועים שונים, ולכן ה-Path MTU עשוי להשתנות.

Routing Algorithms

ניתוב פירושו קביעת דרך, מסלול, למעבר של מידע וחבילות נתונים ממקום למקום ברשתות תקשורת נתונים, במצבים שבהם אין קו תקשורת ישיר בין נקודת השליחה לנקודת היעד. לדוגמה, ברשת האינטרנט, מכיוון שבדרך כלל אין קו תקשורת ישיר בין נקודות המוצא והיעד, יש צורך להשתמש ב"שרשרת" של תחנות ביניים שיקבלו את המידע מהתחנה הקודמת ויעבירו הלאה לתחנה הבאה, הניתוב מסמל את הדרך ותחנות התקשורת השונות שבהם יעברו הנתונים מנקודת המוצא\שליחה עד לנקודת היעד.

יכולת לנתב חבילת מידע ליעד הסופי ברשת, מחייבת ידיעה מסוימת של מבנה הרשת. ברשתות מחשבים קטנות טופולוגית הרשת פשוטה בדרך כלל ומנוהלת בצורה ידנית ואילו ברשתות גדולות יותר הטופולוגיה בדרך כלל מורכבת מאוד עקב שינויים שמתבצעים בתדירות גבוהה והופכים את משימת הגדרת טבלאות ניתוב בצורה ידנית לכמעט בלתי אפשרית.
ברשתות גדולות כמו רשת האינטרנט, שהטופולוגיה שלה מבוססת על מערך של תתי רשתות אוטונומיות, שמחוברות הדדית ובכך יוצרות את רשת האינטרנט הגדולה, כל תת-רשת (AS) מנהלת מערכת ניתוב עצמאית ובנוסף מנהלת גם מערכת ניתוב חיצונית עם תת-רשתות שכנות לה ובכך יוצרת חיבור לרשת הגלובלית.

סוגי ניתוב

כיצד אנו מלמדים את הראוטר להגיע לרשתות מרוחקות? יש שתי שיטות להשגת מטרה זו:

1. **ניתוב סטטי (Static Routing)-** ניתוב זה מוגדר ע"י מנהל הרשת ואינו משתנה. מנהל הרשת מגדיר בראוטר שבכדי להגיע לרשת מסוימת יש צורך לצאת ביציאה מסוימת. בשיטה זו הניתוב אינו משתנה אלא קבוע (סטטי) ונקבע ע"י המשתמש. גם אם טופולוגית הרשת משתנה, הניתוב אינו משתנה כי הוא קבוע. נניח וכבל המחובר ליציאה שהוגדרה נקרע או הפסיק לעבוד, הראוטר לא יוכל להעביר את החבילה הלאה, על מנהל הרשת להגיע ולקבוע ניתוב חדש.

2. **ניתוב דינמי (Dynamic Routing) –** ניתוב זה מוגדר באופן אוטומטי ע"י פרוטוקולי ניתוב דינמיים. מנהל הרשת מעביר את הראוטר למצב של פרוטוקול מסויים ומפרסם רשתות, ברגע שההליך מתחיל ע"י מנהל הרשת מכאן הכל אוטומטי. הראוטרים ממפים את הטופולוגיה ומחפשים דרכים יעילות וטובות להגעה לרשתות יעד מרוחקות.

שתי גישות לפרוטוקולי הניתוב הדינמי

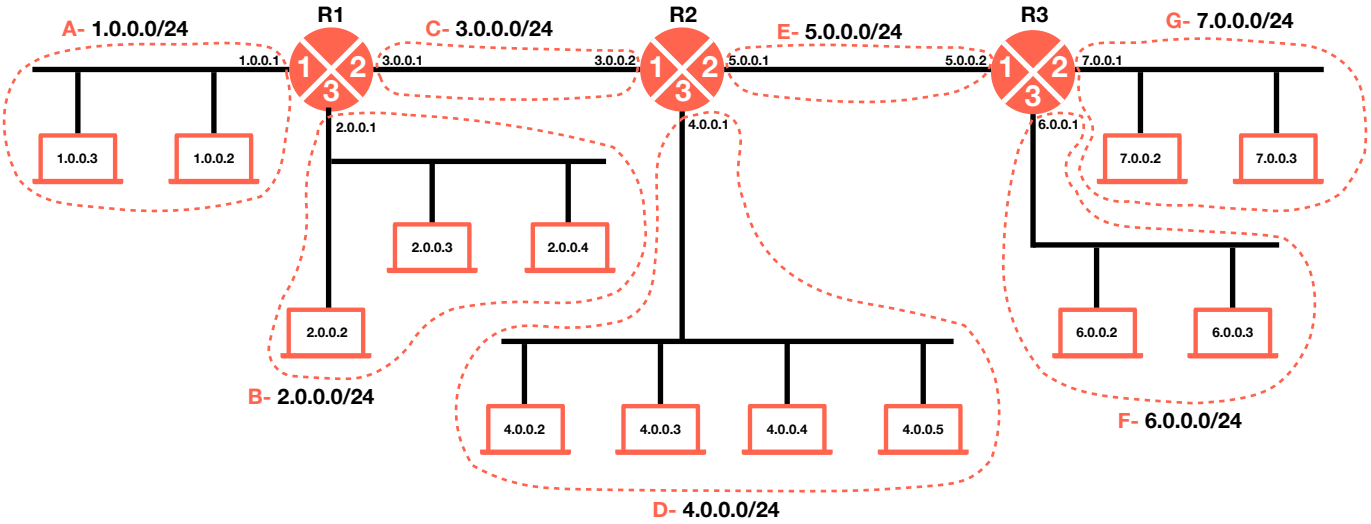
1. **Link State (מצב חיבור)-** פרוטוקולי ניתוב Link-state הם אחת משתי הגישות העיקריות של פרוטוקולי ניתוב המשמשים רשתות מיתוג. הפרוטוקול מבוצע על-ידי כל צומת מיתוג (נתבים) ברשת. הקונספט הבסיסי של ניתוב Link-state הוא שכל צומת בונה מפת קישוריות לרשת, בצורה של גרף, המציגה אילו צמתים מחוברים לאילו צמתים אחרים. לאחר מכן, כל נתב מחשב את הנתיב הבא הטוב ביותר ממנו לכל יעד אפשרי ברשת. אוסף של המסלולים הטובים ביותר יהיו לבסוף לטבלת הניתוב של אותו הנתב. ניתן לתאר אלגוריתמים של Link-state בכך שכל נתב בעצם "מספר לעולם על השכנים המחוברים אליו"

יתרונות- חישוב הניתוב היעיל ביותר מחושב בראוטר עצמן.
מחלק את הרשת לאזורים, כל ראוטר באזור מסויים מחזיק את המפה של האזור בלבד, ראוטרים הנמצאים בגבולות בין אזורים מחזיקים את המפות של שתי האזורים.

חסרונות- מחזיק את כל טופולוגית הרשת – מעמיס על הראוטר.

פרוטוקולים בשיטה זו- OSPF, IS-IS.

דוגמה ל-Link State-



תחילה, בכל פרוטוקול ניתוב בין אם Link State ובין אם Distance Vector, על מנהל הרשת להגדיר בצורה ישירה לכל נתב אילו Subnets מחוברים אליו בכל רגל.
לאחר פעולה זו, הנתבים מתחילים בעבודה אוטומטית על ידי אלגוריתם החיפוש לקביעת המסלולים הרצויים.

טבלת ניתוב R3

Subnet	Interface	Next Hop
5.0.0.0/24	1	-
6.0.0.0/24	3	-
7.0.0.0/24	2	-

טבלת ניתוב R2

Subnet	Interface	Next Hop
3.0.0.0/24	1	-
4.0.0.0/24	3	-
5.0.0.0/24	2	-

טבלת ניתוב R1

Subnet	Interface	Next Hop
1.0.0.0/24	1	-
2.0.0.0/24	3	-
3.0.0.0/24	2	-

א. שלב האתחול- כל נתב שולח הודעת "hello" בקווים המחוברים אליו על מנת לגלות לאילו רשתות ונתבים הוא מחובר אליהם בקשר ישיר. כל נתב שמקבל את ההודעה מחזיר תושבה

ב. כל נתב יוצר טבלת Link State אשר בה כל הנתבים והרשתות שגילה בשלב האתחול.

R2 Link State
R3 —> 5.0.0.0/24
R3 —> 6.0.0.0/24
R3 —> 7.0.0.0/24
R3 —> R2

R2 Link State
R2 —> 3.0.0.0/24
R2 —> 4.0.0.0/24
R2 —> 5.0.0.0/24
R2 —> R1
R2 —> R3

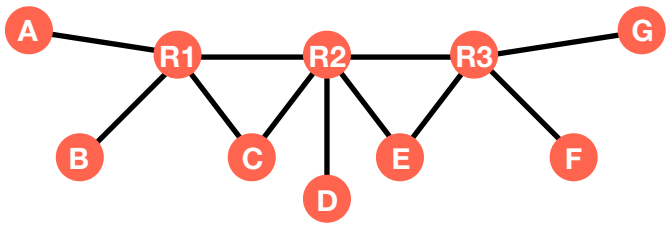
R1 Link State
R1 —> 1.0.0.0/24
R1 —> 2.0.0.0/24
R1 —> 3.0.0.0/24
R1 —> R2

ג. כל נתב שולח אל הנתבים השכנים לו את את טבלת ה-Link State שלו. נתב המקבל טבלה, מעביר אותה גם כן אל השכנים שלו.

ד. כך נוצרת טבלה הכוללת את כל הקשרים והיא זו שמתורגמת לגרף. חשוב לשים לב, הקודקודים של הגרף הינם הנתבים וה-Subnets של הרשת, ולא המחשבים עצמם.

טבלת הקשרים הנוצרה אצל כל נתב

יעד	מקור
A	R1
B	R1
C	R1
R2	R1
C	R2
D	R2
E	R2
R1	R2
R3	R2
E	R3
F	R3
G	R3
R2	R3



ה. מריצים את אלגוריתם דייקסטרה מכל נתב, ולפי הנתונים שיתקבלו נבנית טבלת הניתוב.

טבלאות הניתוב לאחר הרצת האלגוריתם-

טבלת ניתוב R3

Subnet	Interface	Next Hop
1.0.0.0/24	1	5.0.0.1
2.0.0.0/24	1	5.0.0.1
3.0.0.0/24	1	5.0.0.1
4.0.0.0/24	1	5.0.0.1
5.0.0.0/24	1	-
6.0.0.0/24	3	-
7.0.0.0/24	2	-

טבלת ניתוב R2

Subnet	Interface	Next Hop
1.0.0.0/24	1	3.0.0.1
2.0.0.0/24	1	3.0.0.1
3.0.0.0/24	1	-
4.0.0.0/24	3	-
5.0.0.0/24	2	-
6.0.0.0/24	2	5.0.0.2
7.0.0.0/24	2	5.0.0.2

טבלת ניתוב R1

Subnet	Interface	Next Hop
1.0.0.0/24	1	-
2.0.0.0/24	3	-
3.0.0.0/24	2	-
4.0.0.0/24	2	3.0.0.2
5.0.0.0/24	2	3.0.0.2
6.0.0.0/24	2	3.0.0.2
7.0.0.0/24	2	3.0.0.2

2. Distance-Vector (וקטור המרחק)

פרוטוקול ניתוב distance-vector קובע את המסלול הטוב ביותר עבור חבילות נתונים בהתבסס על מרחק. פרוטוקול מבוסס distance-vector מודד את המרחק על ידי מספר הנתבים שעל חבילת הנתונים לעבור בכדי להגיע ליעד, כל נתב שיש לעבור בו נחשב ל- "hop" אחד. קיימים פרוטוקולי distance-vector הלוקחים בחשבון נתונים וגורמים אחרים המשפיעים על התנועה במסלול נתון כך שיתכן שהמידע יכול לעבור יותר מהר דרך 5 נתבים מאשר 4 נתבים.
כדי לקבוע את המסלול הטוב ביותר, כל נתבי רשת שבהם הפרוטוקול מיושם מחליפים מידע אחד עם השני, בדרך כלל טבלאות ניתוב בתוספת כמות ה- "hop" עבור יעדי הרשת. distance-vector דורש מהנתב להודיע לשכניו לטופולוגית הרשת על שינויים מעת לעת. חישוב המסלול הטוב ביותר נעשה על ידי שימוש באלגוריתם בלמן-פורד ובאלגוריתם פורד-פולקרסון.

א. תחילה כאמור האדמיניסטרטור ממלא בטבלאות הניתוב את הקשרים הישירים לכל נתב.לאחר מכן, כל נתב יוצר distance-vector עם הקשרים הישירים בעלי משקל 1, בשל כך בפרוטוקול distance-vector יש רק Subnets ולא נתבים.

R1	{ (A,1) (B,1) (C,1) }
R2	{ (C,1) (D,1) (E,1) }
R3	{ (E,1) (F,1) (G,1) }

ב. כעת כל נתב מעביר את וקטור המרחק שלו אל השכנים שלו, להבדיל מ- Link State כאן כל נתב שולח אך ורק לשכן שלו, כלומר אותו נתב שקיבל וקטור לא ממשיך להעביר אותו הלאה. כל נתב שקיבל וקטור בודק האם הוא משפר את הוקטור הקיים אצלו, אם מביא לו יעדים חדשים או מעדכן מחירים קיימים

ג. נוצרת טבלה הכוללת את כל ה-Subnets וכל שורה בטבלה תייצג distance-vector של נתב. גם אם שני נתבים שכנים, הם לא יכירו את ה-distance-vector אחד של השני עד אשר הם ישלחו אותם.

	A	B	C	D	E	F	G
R1	1	1	1	-	-	-	-
R2	-	-	1	1	1	-	-
R3	-	-	-	-	1	1	1

שלב אתחול

ד. בכל איטרציה יישלח לנתב מסויים ה-distance-vector של כל שכניו מהאיטרציה הקודמת. הדבר יסתיים כאשר בין 2 איטרציות לא יהיו עדכונים.

איטרציה 1

	A	B	C	D	E	F	G	
R1	1	1	1	2_{R2}	2_{R2}	-	-	**R1 מקבל את הוקטור של R2**
R2	2_{R1}	2_{R1}	1	1	1	2_{R3}	2_{R3}	**R2 מקבל את הוקטור של R1 ושל R3**
R3	-	-	2_{R2}	2_{R2}	1	1	1	**R3 מקבל את הוקטור של R2**

איטרציה 2

	A	B	C	D	E	F	G	
R1	1	1	1	2_{R2}	2_{R2}	3_{R2}	3_{R2}	**R1 מקבל את הוקטור של R2**
R2	2_{R1}	2_{R1}	1	1	1	2_{R3}	2_{R3}	**R2 מקבל את הוקטור של R1 ושל R3**
R3	3_{R2}	3_{R2}	2_{R2}	2_{R2}	1	1	1	**R3 מקבל את הוקטור של R2**

באיטרציה הבאה לא מתבצעים שינויים בטבלה ולכן כאן התהליך יסתיים.
קיבלנו מצב ובו כל נתב יודע את המרחק ואת התחנה הבאה בדרך אל יעד מסויים אך אינו מודע (בניגוד ל-Link State) לכל המסלול.

מה קורה כשיש עדכון-
כאשר העדכון הוא לטובה, כלומר הורדת מחיר של קשת או הוספת קשת חדשה (Subnet), האלגוריתם יתמודד בצורה מהירה וטובה עם העדכון, וכעבור מספר לא רב של איטרציות נגיע לנתונים הנכונים.
אך כאשר מעדכנים שדה בטבלה למספר גדול יותר או מנתקים Subnet נוצרת בעיה של Count To Infinity.

ישנן 2 דרכי התמודדות-

1. **Poison Reverse-** לא נשלח לשכן מסלולים שמלכתחילה עוברים דרכו. כלומר, כאשר אני מעביר וקטור לשכן שלי אני מנתק את כל השדות שהמסלולים שלי תלויים בו. למשל אם R1 שולח ל- R2 את ה-distance-vector שלו וב- Subnet D רשום 2_{R2} (בוקטור של R1) אז R1 לא ישלח את השדה הזה. זהו פתרון שלא עובד בכל סיטואציה (למשל במעגלים ברשת)
2. **הגדרת Limit-** הגבלת מספר הקפיצות, כלומר הגבלת המסלול הכי ארוך ברשת. לאחר המס׳ שהוגדר יימחק המסלול. לכל שדה ישנו הגבלה זו, כלומר אם שדה מסויים עובר את ההגבלה, רק הוא ינותק וערכו ישונה לאינסוף.

יש לשים לב כי בעיית Count To Infinity נוצרת משום שכאשר לאחר עדכון אני שולח את ה-distance-vector שלי לשכנים, במידה וישנם שדות שהשכן תלוי בי, הוא מחויב לעדכן לפי ה-distance-vector שנשלח גם אם העדכון הוא לרעה.

Hierarchical Routing

עד כה למדנו כיצד הנתב בוחר באיזה מסלול לשלוח את ההודעה וכיצד הוא בכלל מכיר וממפה את אותם מסלולים, אך כמו שכבר הבנו, רשת האינטרנט היא דבר עצום ודינמי, אם לדוגמה נתב עובד ב-Link State יהיה עליו ללמוד ולהכיר כמות גדולה עד בלתי אפשרית של נתבים, נוסף על כך קימות המון חברות וארגוני אינטרנט שונים ואם חברה אחת עובדת ב-Link State ואילו אחרת ב-Distance-Vector איך הם יעבדו ביחד, איך הם נתבים ידעו לדבר אחד עם השני?
למדנו שרשת האינטרנט מורכבת מ-ISP, ספקי שירות אינטרנט, בכל מני רמות (tier 1, tier2..) וכל אחת מהן היא חברה פרטית עם חומרה שלה וטכנולוגיה שלה, כל ארגון כזה שומר את מבנה הרשת שלו, את סוג הנתבים ואת הפרוטוקולים בהם הוא משתמש באופן חסוי מסיבות של אבטחת מידע וסודות מסחריים. אם כן רשת האינטרנט מחולקת לאוטונומיות Autonomous System (AS), כל ארגון או לחלופין ״דומיין״ נקרא AS.

ישנן שתי רמות לאלגוריתמי ניתוב-

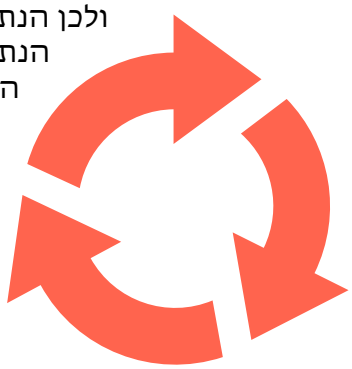
1. **Intra AS/Intra Domain Routing(ניתוב פנים ארגוני)-**
הנתבים בתוך AS צריכים להכיר רק את פרוטוקול ה- Intra AS של אותו AS.
פרוטוקולים- RIP/OSPF.
2. **Inter AS/Inter Domain Routing (ניתוב בין ארגוני)-**
אלגוריתם ה- Iinter AS צריך להיות אחיד לכל הרשת מהסיבה שהוא משלב את כל ה-AS ורץ בין כולם, בכל AS חייב להיות לפחות נתב אחד שמכיר את הפרוטוקול הנ״ל אחרת אותו AS לא יוכל לתקשר עם שאר הרשת. נתב זה הינו Border Gateway שמחבר בין AS שונים.
פרוטוקולים- BGP.

Routing Information Protocol - RIP

פרוטוקול ניתוב המתבסס על ספירת צעדים (Hop Count) על מנת לבצע החלטות ניתוב.
פרוטוקול פשוט, Open-Source המבוסס על Distance-Vector.
נתב המשתמש ב־RIP מנהל רישום של כל הנתבים אותם הוא "מכיר", הרשתות המחוברות אליהן, וכמות הצעדים בכל נתיב לכל יעד. כל הודעה ב-RIP מציגה עד 25 יעדים שניתן להגיע אליהם. כל נתב מבקש עדכונים לגבי שינויים בטופולוגית הרשת (ה-DV) מהנתבים המחוברים אליו כל שלושים שניות, וכך הוא נשאר מעודכן לגבי שינויים בנתיבים המובילים אל היעד, ומקבל מידע על נתבים חדשים שחוברו אל הרשת. אורך המסלול המקסימלי ברשת (כמות הנתבים בדרך אל היעד) אינו MaxHop=15, בשביל לייעל את הנתב משתמשים ב-UDP.

התאוששות מכישלונות ברשת-

במידה ונתב לא מקבל הודעה משכנו לאחר 180 sec, ההנחה היא שהקשר נותק (קו תקול/נפל) ולכן הנתב מעדכן את ה-DV שלו כך שהקו מנותק וכל מי שעבר דרך הנתב מנותק גם כן. הנתב שולח לשכניו את את ה-DV המעודכן. במידה וה- Hops=15 מעדכנים את השדה המתאים לכך. בנוסף משתמשים ב-Poison Reverse למניעת לולאות אינסופיות כאשר מבצעים עדכונים.



RIP פותח כחלק מהתשתית הראשונה של האינטרנט, כפרוטוקול הניתוב הראשון, והיה במשך שנים רבות פרוטוקול הניתוב הנפוץ ביותר באינטרנט. עם התפשטות האינטרנט חישובי עלות, רוחב פס ומרחקים פיזיים הפכו למכריעים יותר ויותר בהחלטות הניתוב ו־RIP נדחק לטובת פרוטוקולים חדשים יותר כמו ה־OSPF או ה־IGRP.

Open Shortest Path First - OSPF

פרוטוקול ניתוב היררכי, תלוי מצב (Link-state) להעברת נתונים בין ראוטרים שונים הנמצאים באותה מערכת אוטונומית. בבסיס הפרוטוקול עומד אלגוריתם דייקסטרה, המשמש לחישוב עץ המרחק הקצר ביותר. OSPF משתמש בעלות העברת הנתונים (מספר המציין את גודל רוחב הפס) לצורך חישוב המרחק, ותמיד יבחר את הנתיב הזול ביותר להעברת חבילה מהמקור אל היעד.
בפרוטוקול זה קיימת אבטחה כך שכל ההודעות מאושרות (למניעת חדירה של הודעות זדוניות).
ניתן לייצר כמה מסלולים הכי קצרים, כלומר בטבלת הניתוב ניתן להזיק יותר ממסלול אחד לכל יעד.
ניתן לקבוע לפי Matric כיצד לחשב את אורך המסלול (לא בהכרח לפי קפיצות), מחלקים את הרשת לאזורים וישנם 2 רמות:

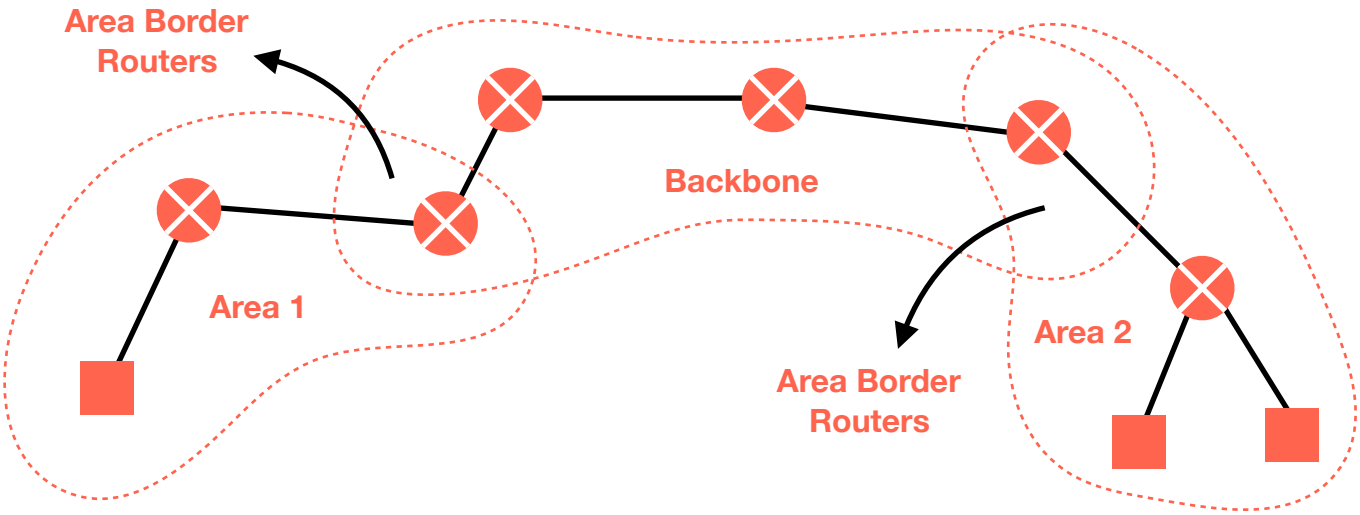
רמה1- Area-

נתבים שהם ברמה אחת ובאזור אחר לא מכירים אחד השני גם אם קיים בניהם קשר כלשהו.

רמה 2- Backbone-

ברמה זו נמצאים נתבים אך ורק ברמה ה-2 או נתבים שנמצאים גם ברמה 1 וגם ב-2.

עבור כל אזור מריצים Link-State כך שכל נתב מכיר את כל הרשת שבאותו אזור. לאחר מכן מריצים Link-State על הנתבים שנמצאים באזור המשותף (נתבים שנמצאים גם ברמה 1 וגם ב-2).
נתב שנמצא ברמה 1 ו-2, כאשר הוא מריצים עליו Link-Stste במסגרת רמה 2, הוא משתף את הקשרים מבחינת Subnets של האזור שלו.



Border Gateway Protocol - BGP

פרוטוקול ניתוב המהווה את ליבת מערכת הניתוב של רשת האינטרנט. BGP הוא פרוטוקול ניתוב מבוסס קשר (Path vector). נתב שפועל באמצעות BGP מנהל טבלה של הרשתות המחוברות אליו, והקשרים ביניהן לבין רשתות אחרות, ומבצע החלטות ניתוב על בסיס הקשרים בין הרשתות ומדיניות המוכתבת בצורה ידנית על ידי מנהל הרשת.

BGP הוא פרוטוקול ניתוב בין AS- autonomous systems, הוא Path Vector כלומר, ה BGP בונה מסלול שלם לכל ניתוב.

הניתוב של BGP מתבסס על המסלול, מדיניות ה- AS, הסכמים בין AS שונים, ביטחון ועוד מגוון חוקים.

מאחר שכל AS רשאי להריץ פרוטוקול ניתוב פנימי שונה אזי אין משמעות אמתית למחיר של מעבר ב AS כלשהו מאחר שהמטריקה של כל אחד שונה, לכן מפורסם רק "יכולת הגעה" אל היעד.

BGP עובד בשני מישרים:

1. **e-BGP (external)-** מעביר מידע מ-AS אחד לשני (שכנים), e-BGP רץ בין Border Gateway שונים.
2. **i-BGP (internal)-** תפקידו להעביר מידע מה-Border Gateway של אותו AS אל כל הנתבים שבתוך ה-AS. נוסף על כך עליו לקבוע עבור נתבים לאיזה Border Gateway לגשת בעבור מסלול מחוץ ל-AS (עבור כל תת רשת חיצונית).

BGP בנוי כך שכל זוג נתבי Border Gateway שמחוברים בין AS שונים, יוצרים קשר אחד עם השני בחיבור TCP.

BGP עובד בשיטת Distance-Vector ולכן בין נתבים אלו מועברים הודעות DV ובכך הנתבים לומדים לאיזה Subnet ניתן להגיע דרך הנתב השכן.

הודעות BGP כוללות את היעד של ה-Subnet שאליו אני יכול להגיע, ה-Border Gateway שדרכו ההודעה עוברת ואת ה-AS שדרכם ההודעה עוברת.

AS Path-

מכיל את כל ה-AS שההודעה עוברת דרכם, על מנת שנדע מהו המסלול הנבחר כדי להגיע לתת הרשת הבאה.

Next Hop-

מביא את ה- Border Gateway הבא שאליו אני רוצה להעביר את ההודעה, כולל את כל ה-AS שעוברים בדרך ליעד.

שכבת הקו

בפרק הקודם למדנו על שכבת הרשת, וכעת אנו מבינים שחבילות מידע שעוברות בין שתי נקודות קצה עוברות בדרך כלל בין מספר רכיבים בדרך (למשל נתבים). לכל אורך הפרק הקודם, הנחנו שניתן להעביר חבילה בין ישות אחת לישות אחרת כשאלו צמודות זו לזו. עם זאת, פעולה זו אינה כה פשוטה. במהלך הפרק הקרוב נבין את מטרתה של שכבת הקו, נלמד על פרוטוקול Ethernet, כמו גם פרוטוקול ARP, ונבין את האתגרים עמה מתמודדת שכבת הקו.

אם כן ראינו ששכבת הרשת אחראית להעביר חבילה בין מחשב א' למחשב ב'. כמו כן, הבנו שהיא אחראית על המסלול שבה החבילה תעבור. השכבה השנייה, שכבת הקו, אחראית על התקשורת בין כל שתי ישויות הקשורות זו לזו באופן ישיר. כל עוד הישויות מחוברות זו לזו באופן ישיר, הטיפול של העברת הודעה ביניהן שייך לשכבה זו. בשכבת הקו אין הבנה של הדרך המלאה שהחבילה עוברת מהמקור אל היעד כמו בשכבת הרשת, אלא אך רק בין ישויות סמוכות.

מטרת שכבת הקו היא להעביר מידע בין שתי ישויות המחוברות זו לזו באופן ישיר

המשמעות של חיבור ישיר היא שמידע יכול לעבור בין הישויות מבלי לעבור בישות אחרת בדרך. חיבור ישיר יכול להיות בצורות שונות. יתכן ומדובר בחיבור קווי- משתמשים בכבל פיזי בכדי לחבר ישות אחת לאחרת. למשל, נאמר שמחשב א' מחובר לאחד הנתבים הקרובים אליו בכבל. חיבור אחר יכול להיות חיבור אלחוטי- לדוגמה, יתכן והנתב של מחשב א' מחובר לנתב הבא באמצעות WiFi.
שכבת הקו צריכה לדאוג לכך שחבילות המידע יצליחו לעבור מישות לישות בצורה אמינה. השכבה צריכה להתמודד עם תקלות שיכולות להיות בקו, עליהן נפרט בהמשך.

שכבת הקו מספקת לשכבת הרשת ממשק להעברת מידע בין שתי ישויות המחוברות באופן ישיר

באופן זה, שכבת הרשת לא צריכה לדאוג לסוגיות הקשורות לחיבור בין שתי תחנות. את שכבת הרשת לא מעניין אם הישויות מחוברות בכבל, בלוויין, ב-Wifi, או באמצעות יוני דואר. היא רק אחראית להבין מה המסלול האופטימלי. כמו ש-Waze רק אומרת לרכב באיזו דרך לעבור, ולא מסבירה לנהג שהוא צריך לתדלק, ללחוץ על הגז, לאותת ולעצור ברמזור או להולך רגל. בזה יטפל הנהג, או במקרה שלנו- שכבת הקו.

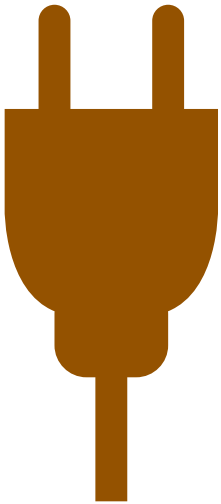
איפה ממומשת שכבת הקו?

המימוש של שכבת הקו "נמצא" בכל ישות ברשת- וספציפית, בכרטיס הרשת של הישות. כך למשל כרטיס Ethernet יממש את פרוטוקול Ethernet, וכרטיס WiFi יממש את פרוטוקול WiFi. כך כרטיסי רשת שונים מתקשרים זה עם זה. עם זאת, כרטיס רשת Ethernet לא יכול לתקשר ישירות עם כרטיס רשת של WiFi.

כיום נהוג לחלק את שכבת הקו לשתי שכבות נפרדות:

- **Logical Link Control (LLC)-** אחראית על חלוקת הנתונים שמגיעים מהשכבות העליונות למסגרות, ועל הטיפול בכל מסגרת בנפרד. חבילות נתונים בשכבת הקו מכונות "מסגרות" (frames).

- **Media Access Control (MAC)-** קיימת כאשר קיים מדיום משותף לכמה נקודות קצה- כאשר הרשת אינה רשת נל"ן (נקודה לנקודה) מלאה (ברשת נל"ן כל נקודת קצה מחוברת בקו ייעודי לכל נקודת קצה אחרת), ונכנסת לפעולה כאשר שני מחשבים או יותר באותו מתחם התנגשות (collision domain) מעוניינים לשדר בו-זמנית. במקרה כזה, יופעל פרוטוקול לטיפול בשכבת ה-MAC. דוגמאות לפרוטוקול MAC:
  - פרוטוקול ALOHA אשר מהווה בסיס לפרוטוקול שכבת ה-MAC של מערכות Ethernet.
  - שימוש במנגנון מרכזי אשר מפקח על כלל התעבורה ברשת ונותן "רשות. דיבור" לאחד המכשירים. אלגוריתם זה ממומש למשל במערכות מבוססות Token Ring.



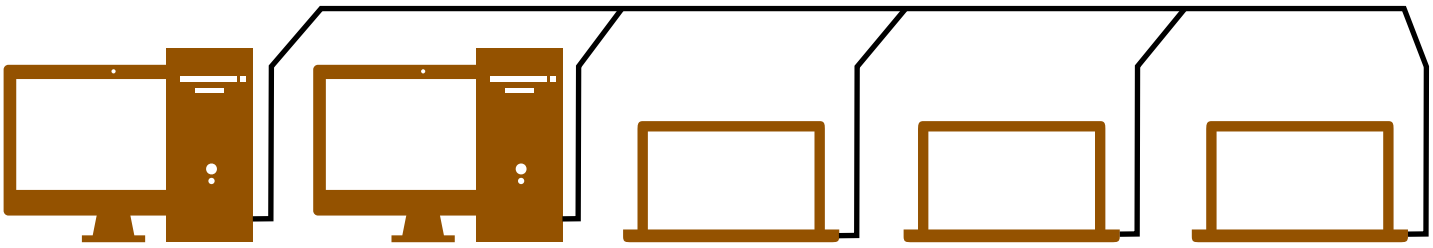
Multiple Access protocols

כאשר לוקחים קו ומחברים אליו כמה מחשבים, עם כמה שהדבר נשמע פשוט לביצוע יכולות להיות כמה וכמה בעיות. אם נרצה להעביר אינפורמציה ממחשב א׳ למחשב ב׳, ובדיוק באותו זמן, מחשב ג׳ שגם הוא מחבור לאותו קו מנסה לשדר את המידע שלו, נוצרת התנגשות. הדבר דומה למכשיר ווקי-טוקי, כאשר שני מכשירים מנסים ״לדבר״ בו זמנית, המידע מתנגש ואף ישות לא תצליח להעביר את המידע שלה ליעד. בנוסף כאשר נרצה להעביר מידע ממחשב אחד למחשב אחר שנמצא על אותו קו, נרצה שהמידע יגיע רק למחשב שלו רצינו לשלוח, ולא לשאר המחשבים שמחוברים על אותו הקו.

מסיבות אלו קיימים פרוטוקולים Multiple Access שתפקידם כאמור הוא לתת מענה לכמה מחשבים שמחוברים על קו אחד, לדבר ולתקשר ביחד תוך הימנעות מהבעיות המדוברות.

קריטריונים ליעילות פרוטוקול Multiple Access-

1. כאשר ישות אחת רוצה להעביר מידע, היא תוכל לשלוח בקצב R.
2. כאשר M מחשבים רוצים להעביר מידע במקביל, כל אחד יוכל לשלוח בקצב ממוצע R/M.
3. מבוזר- שלא יהיה נתב מיוחד לתיאום שידורים, מהסיבה שהוא יכול לקרוס ואז כל הקו מושבת. כלומר שהפרוטוקול ירוץ על כל יחידה בנפרד.
4. פשוט, אמין וללא תקלות.



MAC - Multiplexing Access Control - בקרת ריבוב

ריבוב הוא צירוף של שני אותות או יותר משני ערוצים או יותר ושידורו במדיום פלט יחיד.
ניתן לחלק ל-3 סוגי פרוטוקולים:

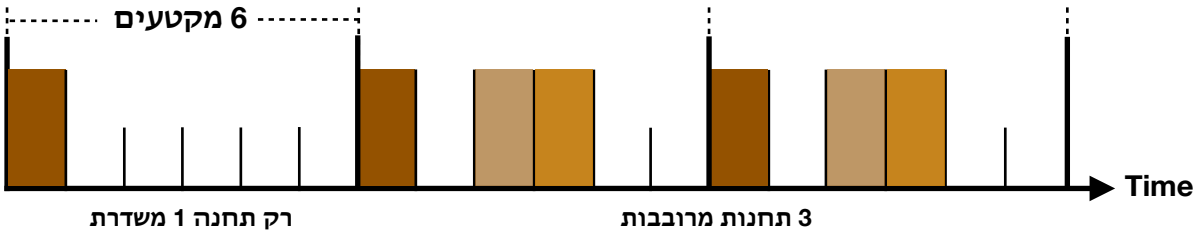
1. חלוקת ערוצים (Channel Partitioning)- חלוקת הערוץ ל-״חלקים״ קטנים, מחיצות, לדוגמה חלוקת תדירויות, חלוקת זמנים ועוד. לכל צומת ישנה בלעדיות על כל חלק. במידה וכל הצרכנים רוצים לשדר ישנה חלוקה שווה.

2. גישה אקראית (Random Access)- לאף צומת אין בלעדיות על הערוץ/קו, הגישה רנדומלית, כמובן שההסתברות להתנגשויות גדולה ולכן צריך מנגנון שיבצע התאוששות מהתנגשויות.

3. בתורות/החלפות (Taking turns)- ניתן לתאם פעולות בערוץ המשותף בכדי להימנע מהתנגשויות. ישנו ״אסימון״ ואנו מגדירים את סדר העברתו.
כאשר האסימון מגיע ללקוח, ניתן לשדר, וברגע שהלקוח מסיים לשדר האסימון מועבר ללקוח הבא בתור. הדבר נותן ניצולת קו גבוהה ביותר אך בעל נקודות כשל.

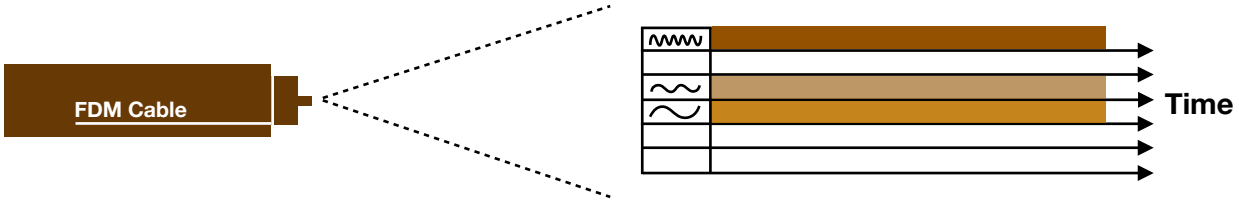
Time Division Multiple Access - TDMA

נמנע עם סוג 1- חלוקת ערוצים.
ריבוב חלוקת זמן, מחלק את הזמן לחלונות-זמן קצרים (באנגלית timeslots). כל חלון זמן מוקדש להעברה של המידע שהתקבל בערוץ-כניסה מסוים. למכשירים הפועלים בשיטת ה-TDMA יש מנגנון פנימי הגורם להם לשדר ולקלוט מידע רק בחלון-הזמן המוקצה להם, ולא בחלונות זמן שמוקצים למכשירים אחרים, כל תחנה מקבלת מקטע בגודל (זמן) קבוע. מקטעי זמן אשר לא מומשים מתבזבזים ולא ינוצלו. כלומר הנצילות בפרוטוקול לא אופטימלית, רק במידה וכל התחנות מעוניינות לשדר יחד, רק אז ישנה נצילות אופטימלית.
קיים קושי בסנכרון מדיוק של השעונים אצל כל תחנה.



Frequency Division Multiple Access - FDMA

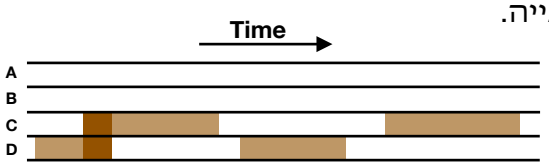
נמנע עם סוג 1- חלוקת ערוצים.
ריבוב חלוקת תדר, מחלק את ספקטרום (טווח) התדרים לתחומי-תדר קטנים יותר- אותם מכנים אפיקים. כל לקוח מקבל תדר אחר. רוחב הפס זהו תחום התדרים שניתן להשתמש לטובת שידור.
בין כל שני ערוצים סמוכים מגדירים מרווח ביטחון שמיועד למנוע הפרעה הדדית בין שתי תתי הערוצים.
תדר אשל לא נמצא בשימוש, מתבזבז ולא ינוצל. במידה ותחנה טחת מעוניינת לשדר- הנצילות לא מקסימלית, רק במידה וכל התחנות משדרות יחד ישנה נצילות אופטימלית.



- בפרוטוקולים מהסוג השני- גישה אקראית, כאשר לתחנה יש חבילה לשלוח, היא תשלח את החבילה ברוחב הפס המלא של הערוץ בקצב R ללא תיאום בין שאר התחנות על הקו. כאשר שתי תחנות או יותר משדרות יחדיו נוצרת "התנגשות" ופרוטוקולי MAC דואגים לזהות את ההתנגשויות ו"להתאושש" מהם.

ALOHA

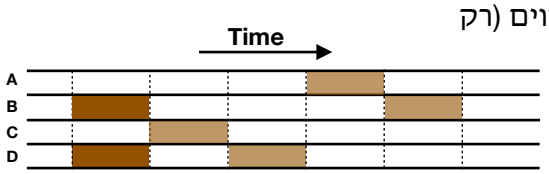
נמנע עם סוג 2- גישה אקראית.
פרוטוקול זה מציע דרך ייחודית בפשטותה לאפשר למספר משתמשים במקביל להשתמש בערוץ תקשורת משותף. פרוטוקול ALOHA שימש בבניית רשת מחשבים אלחוטית, מבוססת חבילות הראשונה בעולם.
פותח על ידי נורמן אברמסון בשנת 1970 באוניברסיטת הוואי, מכאן גם השם.
שיטת ALOHA סימנה התפתחות משמעותית בהנדסת תקשורת אלחוטית ובהנדסת רשתות תקשורת.
הרעיון הכללי מאחורי ALOHA הוא פשוט: כל משתמש יכול לשדר על הערוץ מתי שהוא חפץ. במידה ויש התנגשות, המסגרות ששודרו נהרסו ויש לשדרם שנית במועד מאוחר יותר. מועד השידור החוזר נקבע בצורה אקראית כדי למנוע התנגשויות חוזרות. התנגשויות ניתנות לגילוי לאחר השידור, מאחר שכל התחנות מאזינות אחת לשנייה.
בנוסף, מניחים שהערוץ נטול רעשים, כלומר נפילת חבילה מתרחשת רק בעקבות התנגשות עם חבילה אחרת.



בכל פעם שלשתי מסגרות יש חפיפה בזמן השידור על הערוץ, תהיה התנגשות. למעשה, בהינתן שתי הודעות שמשודרות, מספיק שרק הביט האחרון של ההודעה הראשונה יתנגש עם הביט הראשון של ההודעה השנייה כדי ליצור התנגשות. עובדה זו יכולה לגרום לכמות גדולה מאוד של התנגשויות החל מעומס מסוים בכמות השליחות מלקוחות. הפרוטוקול מגיע לנצילות מקסימלית של 18%.
במידה ותחנה אחת רוצה לשדר, הנצילות אופטימלית. במצב בו כולם רוצים לשדר- ישנה התרסקות עקב ההתנגשויות.

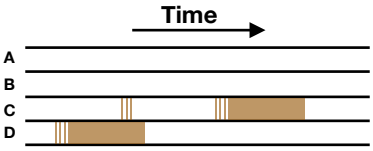
Slotted ALOHA

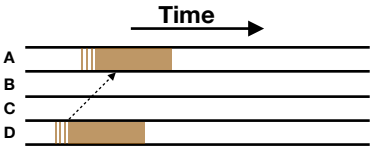
בעקבות פרוטוקול ALOHA המקורי, עוצבו במשך השנים פרוטוקולים מבוססי ALOHA נוספים לכן ניתן לומר שכיום ALOHA למעשה מייצג משפחה של פרוטוקולים. Slotted ALOHA (ALOHA מחורץ) הוא פרוטוקול אשר מבוסס על הפרוטוקול שראינו, בהבדל אחד. במטרה להגדיל את נצילות הערוץ ולחסוך שידורים חוזרים, מוסיפים ל-ALOHA מנגנון סנכרון. סנכרון זה פירושו שצרכן יכול לשדר הודעה רק בזמנים מסוימים, במרחקים קבועים ושווים (רק בחריצים). עקב סנכרון זה נמנעות התנגשויות רבות ותפוקת הערוץ עולה משמעותית ממה שראינו ב-ALOHA הקלסי, כי עכשיו הודעה שמגיעה לשידור תחכה עד להגעה לחריץ הבא ולא תתנגש בהודעות המשודרות בחריץ הנוכחי. מה שגורר נצילות מקסימלית של 37%.



Carrier Sense Multiple Access - CSMA

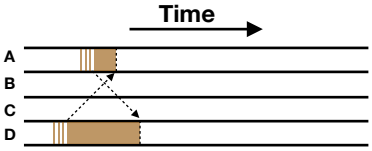
עוד שכלול של פרוטוקול ALOHA הוא פרוטוקול CSMA, לפני תחילת השידור קודם מאזינים לערוץ האלגוריתם מוודא שערוץ התקשורת פנוי ואין תחנה אחרת שכבר משדרת באותו חלון. במידה והערוץ תפוס התחנה תמתין עד אשר הוא מתפנה, אם הוא פנוי היא תשדר את המסגרת.
אם שתי תחנות מתחילות לשדר במקביל, האלגוריתם אינו מזהה כי קרתה התנגשות אלא מסתמך על מנגנוני בקרת השגיאות של שכבת התעבורה כדי לקבל את החבילות שנשלחו. הסיבה שבגללה עדיין יכולות להיות התנגשויות היא מכיוון שגם לפעולת ההאזנה יש פרק זמן מסויים. במידה ותחנה מאזינה, קיימת אפשרות שהיא לא תזהה הודעות על הקו למרות שבפועל תחנה אחרת כבר שולחת בו אך מידע זה עדיין אינו הגיע אליה.





-CSMA/CD

Carrier Sense Multiple Access with Collision Detection או CSMA/CD הוא הרחבה לאלגוריתם CSMA בעל מנגנון לזיהוי התנגשויות.
היישום הנפוץ ביותר של CSMA/CD הוא תקן ה-Ethernet. החידוש כאן הוא האזנה לתווך התקשורת גם תוך כדי שידור מסגרת. במקרה של התנגשות, שתי התחנות המעורבות מזהות התנגשות והן מפסיקות לשדר את המסגרת, ממתינות זמן אקראי ומשדרות את המסגרת שוב. בשל כך נחסך זמן ורוחב פס לא מנוצל.



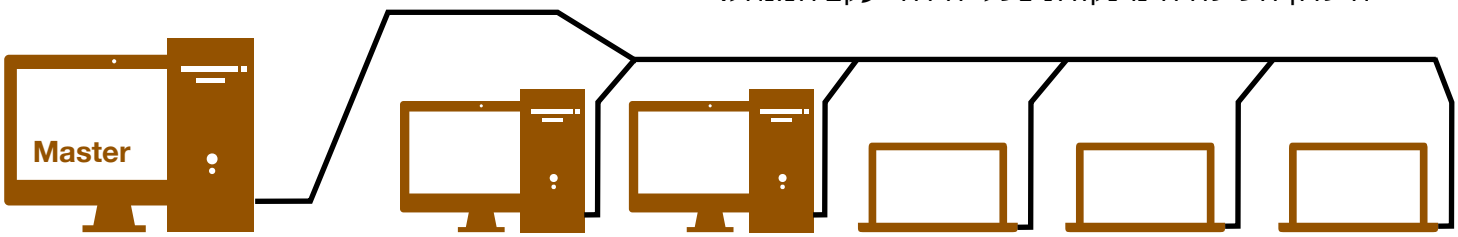
-CSMA/CA

Carrier Sense Multiple Access with Collision Avoidance או CSMA/CA הוא הרחבה לפרוטוקול CSMA בעל מנגנון למניעת התנגשויות הפועל ברשתות אלחוטיות (למשל WIFI).
הפרוטוקול עובד בשתי תצורות:
1. בקשת אישור שליחה- התחנה מאזינה לערוץ, ואם הוא פנוי היא שולחת אות שמודיע לשאר התחנות על כוונתה לשלוח מסגרת, לאחר שליחת האות היא ממתינה לאישור מהתחנה המקבלת ורק אז שולחת את המידע. שאר התחנות הנמצאות באותו מתחם התנגשות מאזינות למידע המועבר בין התחנות ויודעות כאשר הערוץ אמור להיות תפוס.
2. ללא בקשת אישור שליחה- כאשר התחנה מעוניינת לשדר היא בודקת אם הערוץ פנוי. אם הוא פנוי, היא מתחילה לשדר באופן מיידי תוך שהיא בודקת שלא נוצרה התנגשות עם תחנה אחרת. אם התחנה מזהה התנגשות היא זונחת את החבילה ומשדרת אותה מחדש.

- מהסוג ה-3, תורות/החלפות, ישנם 2 פרוטוקולים עיקריים.

Polling

ישנו צומת שולט (מנהל, Master) אשר מנהל את התור על ידי כך שהוא מזמין את התחנות לשדר לפי תור. המנהל דוגם, סוקר (מכאן השם- Poll) את כל התחנות אחת אחת ובודק האם יש מה לשדר, במידה ויש תחנה שמעוניינת לשדר, המנהל מאפשר לה שידור. בשיטה זו ישנה ניצולת של הקו משום שכל תחנה שמעוניינת לשדר, מקבלת כמעת את כל רוחב הפס (הניהול של המנהל הוא בעל תקורה נמוכה יחסית). במידה וכולם רוצים לשדר- רוחב הפס מתחלק שווה בשווה, כלומר כל תחנה מתוך N תחנות מקבלת N/R מרוחב הפס. חשוב לציין כי כל תחנה מקבלת מהמנהל זמן קצוב לשידור.
חיסרון השיטה הינו נקודת כשל יחידה- עקב המנהל.

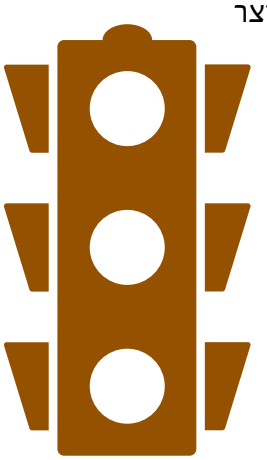


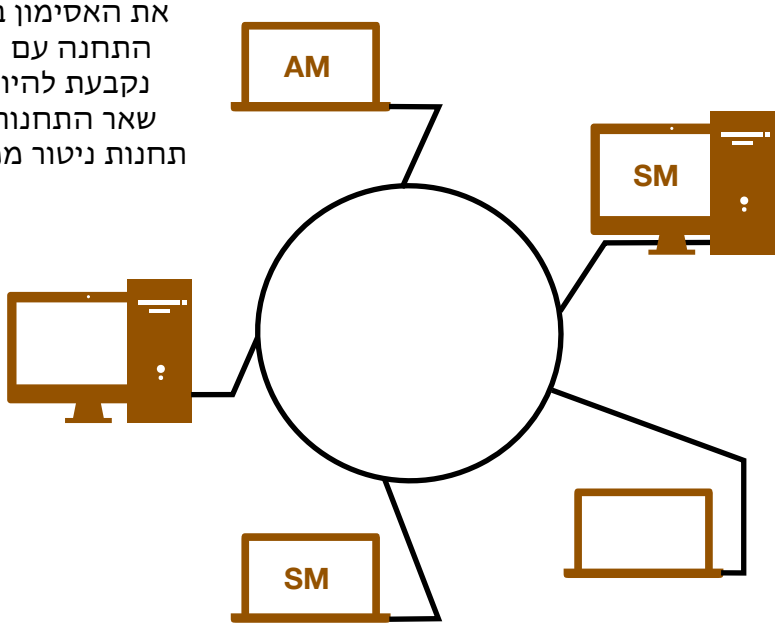
Token ring

Token ring (בתרגום מילולי- טבעת אסימון) הוא תקן לרשתות מקומיות. רשתות Token ring עושות שימוש באסימון בגודל 24 סיביות המועבר בין המחשבים ברשת, חידוש השיטה הוא שהתור נעשה ללא מנהל כאשר רק מחשב שברשותו האסימון יכול לשדר. האסימון מועבר בין המחשבים בסדר מעגלי קבוע. ברגע שהמשתמש מסיים לשדר את החבילה הוא מעביר את הבעלות על הקו על ידי העברת ה-Token לבא בתור אחריו. כאשר משתמש אחד רוצה לשדר הוא מקבל כמעט את כל רוחב הפס, ובמידה וכל המשתמשים מעוניינים לשדר החלוקה מתבצעת שווה בשווה- R/N.
הסכם זה גורם לכך שאין התנגשויות ברשת, מצד שני עלולה להגרם הרעבה של תחנות (עלול להגרם מצב בו תחנה שצריכה לשדר לא מקבלת אסימון ולכן לא משדרת).

אופן הפעולה-

בכל רשת Token ring קיימת תחנה אחת שמוגדרת כ"תחנת ניטור פעילה" Active Monitor- AM, תחנה זו אחראית על ניהול הצטרפויות והתנתקויות מהרשת ועל ניטור האסימון- התחנה מוודאת שקיים אסימון ברשת, וזאת מכיוון שאם עקב רעש על הקווים האסימון נפל, הרשת תקפא (לא ייתכנו שידורים ללא אסימון). בנוסף לתחנת הניטור הפעילה קיימות ברשת "תחנות ניטור ממתינות" Standby Monitor- SM שייכנסו לפעולה במידה וייווצר נתק עם תחנת הניטור הפעילה. בחירת תחנת הניטור הפעילה מתרחשת כאשר מתקיים אחד ממקרים הבאים:
נמצא שסיגנל מסוים אבד בטבעת.
תחנה אחרת בטבעת לא מצאה את תחנת הניטור הפעילה, או לאחר זמן מסוים שעובר מבלי שאחת התחנות קיבלה את האסימון ב-7 השניות האחרונות.
התחנה עם ה-MAC הגבוה ביותר ברשת נקבעת להיות תחנת הניטור הפעילה.
שאר התחנות ברשת נקבעות להיות תחנות ניטור ממתינות.





פרוטוקול Ethernet

Ethernet היא טכנולוגיה לתקשורת נתונים ברשתות מחשבים מקומיות (LAN) הראשונה והדומיננטית ביותר כיום. כרטיס רשת Ethernet נחשב רכיב זול מאוד לרכישה, הטכנולוגיה פשוטה זולה ומהירה יותר מטכנולוגיות דוגמת Token ring.

LAN - Local Area Network-

רשת תקשורת מקומית היא רשת מחשבים המתפרסת, בשונה מרשת עירונית (MAN) או רשת אזורית (WAN), על אזור גאוגרפי מוגבל (עד אלפים בודדים של מטרים רבועים), בדרך כלל בתוך בניין אחד, או בניינים סמוכים. רשת כזו מתאפיינת ברוחב פס גבוה יחסית ובזמן השהיה נמוך יחסית.

בתצורה הראשונית היה קצב התעבורה ברשת 3Mbps, והיא כללה שדה של 8 ביט בלבד לכתובות המקור והיעד (לעומת 48 היום). עם התפתחות טכנולוגיית המחשוב והאפשרות לעבוד במהירויות גבוהות יותר, נוצר מצב בו הרשת לא מצליחה לספק את הצורך בקצב נתונים מהיר ומספר גדול של מחשבים, ועל כן טכנולוגיית האתרנט הורחבה בשנות התשעים ל- Fast Ethernet. מיד לאחר הפיתוח של ה- Fast Ethernet הופיעה גרסה שנקראת Ethernet 10/100, אשר מאפשרת תמיכה של כרטיס הרשת בשתי המהירויות (גם 10 וגם 100 mbps). פיתוחים נוספים של הטכנולוגיה כדי לתת מענה לקצבי תקשורת מהירים ולארגונים גדולים מאוד התפתחו לקראת סוף המאה ה-20 - זוהי טכנולוגיית Gigabit Ethernet.

הכבל כולל בתוכו ארבעה זוגות חוטי נחושת השזורים בזוגות בלתי-מסוככים. הכבל קיים בשני סוגים: UTP - Unshielded Twisted Pair ו-STP - Shielded Twisted Pairs, שההבדל ביניהם הוא שכבת סיכוך נוספת המגנה על הכבל מהפרעות אלקטרומגנטיות.
בשני הקצוות של הכבל קיים מחבר הנקרא, RJ45 (דומה למחבר טלפון, אך מעט גדול יותר). מחבר זה מתחבר ל Transciever (מקלט-משדר) ברכיב הרשת.

מבנה חבילת Ethernet

חבילה בפרוטוקול Ethernet מורכבת מחמישה חלקים: מבוא, פתיח, תוכן, סוגר וגובל. אורכה של חבילה נע בין 84 ל-1542 בתים בהתאם לגודל שדה התוכן.

מבוא- המבוא (Preamble) מורכב מסדרה של 7 בתים המכילים את הרצף 10101010 ואחריהם בית המכיל את הרצף 10101011 המסמן את סוף המבוא. מטרת המבוא היא לסנכרן את תחילת השידור באופן פיזי.

פתיח- הפתיח הבסיסי מורכב מ-14 בתים לפי הפירוט הבא:

כתובת יעד	כתובת מקור	EtherType

- כתובת יעד- (שישה בתים) - כתובת ה- MAC של רכיב היעד.
- כתובת מקור (שישה בתים) - כתובת ה- MAC של רכיב המקור.
- EtherType (שני בתים) - שדה המציין את סוג הפרוטוקול של הנתונים אותו ההודעה מעבירה כתוכן (לדוגמה, עבור תוכן בפרוטוקול IPv4 יהיה ערך 0x0800 בעוד לתוכן בפרוטוקול ARP יהיה ערך 0x0806). אם הרשת עושה שימוש בפרוטוקול רשת מקומית וירטואלית (802.1Q, VLAN, אותו כבר נכיר), המתג אליו מחובר המחשב מוסיף לפתיח שדה "תווית VLAN" בן ארבעה בתים. שדה זה נושא את מזהה הרשת הווירטואלית לה שייך המחשב.

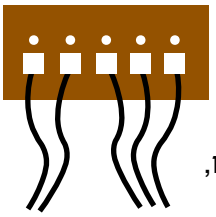
תוכן- שדה התוכן בחבילת Ethernet מכיל בין 46 ל-1500 בתים.

סוגר- הסוגר של חבילת Ethernet מכיל שדה בן ארבעה בתים המורכב מסיכום ביקורת (checksum), מספר האימות של הפתיח והתוכן.

גובל- הגובל (Interframe gap) הנו סדרה של 12 בתים המשודרים בסוף החבילה ומשמשים להפרדה לקראת תחילת השידור הבא.

HUB (רכזת)

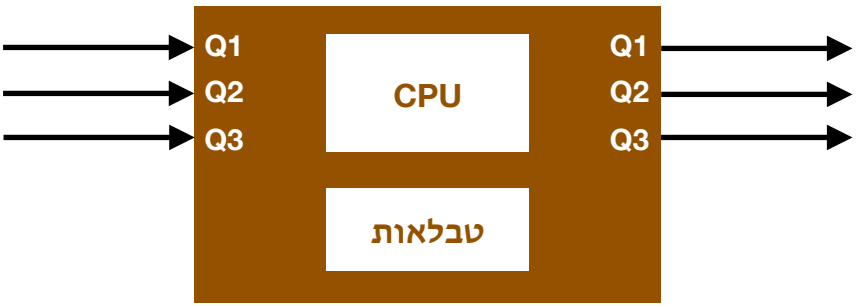
רכזת היא רכיב ברשת מחשבים המחבר בין שני מקטעים או יותר של אותה הרשת ומאפשר להגדיל את המרחק בין קצותיה. ניתן לומר כי רכזת היא מפצל בעל יותר משתי כניסות.
הרכזת נכנסת לפעולה כאשר מתקבלת תשדורת באחד מקצותיה, היא מנקה את התשדורת מהפרעות, מחזקת את האות אם יש צורך בכך ולאחר מכן מעבירה אותו הלאה אל כל שאר קצוות הרשת.
רכזת היא רכיב של שכבה הפיזית, בהיותה בשכבה הפיזית, הרכזת לא מבצעת פעולות מתוחכמות עם חבילות הנתונים שמגיעות, היא לא מנתחת את תוכן החבילות כדי לייעל את שליחתן, והיא איננה מודעת למקור החבילה או ליעדה.



רכזת היא רכיב פשוט יחסית ברשתות תקשורת, אשר לא מנהל את התעבורה העוברת דרכו. כל חבילת נתונים שמגיעה אליו דרך כניסה מסוימת מועברת אל שאר הכניסות של הרכזת. בגלל שהחבילות מגיעות ממקורות שונים שאינם מתואמים, ובגלל שהחבילה מועברת אל שאר הכניסות של הרכזת ללא הבחנה, לא פעם נוצרת התנגשות בין החבילות השונות - עובדה שמפחיתה מיעילות הרשת. הצורך של אמצעים שמחוברים לרכזת לזהות מצבים של התנגשות ולטפל בהם יוצרת הגבלות על כמות הרכזות שאפשר לחבר אחת לשנייה: ברשתות 10BASE-T ניתן לחבר עד 4 רכזות בין כל 2 תחנות, וברשתות100BASE-T ניתן לחבר רק 2 רכזות זו לזו בין כל 2 תחנות.

Switch (מֶתֶג)

הוא רכיב ברשת מחשבים המחבר בין צמתים שונים ברשת, בין אם הם מכשירי קצה (כגון מחשבים) ובין אם הם מרכיבי רשת בסיסיים (כגון רכזות). מספר הכניסות במתג יכול לנוע בין כניסות בודדות, במקרה של מתג ביתי, ועד למאות כניסות, במקרים של רשתות תקשורת גדולות.
המתג הקלסי מבצע העברה של נתונים בין הציוד שמחובר לכניסות השונות שלו באופן סלקטיבי על-פי כתובות MAC של היעד המבוקש של כל חבילת נתונים שנשלחת דרכו.
מתג "לומד" את כתובות ה-MAC של הרכיבים המחוברים לכל כניסה שלו על ידי קריאת נתוני הבקרה בחבילות המידע המגיעות אליו ושמירתן בטבלה פנימית הנקראת טבלת MAC (קיצור של Memory Addressable Content). כאשר חבילה מגיעה אל המתג הוא בודק מה היא כתובת ה־MAC אליה מיועדת החבילה: אם הכתובת מוכרת למתג בטבלת MAC הוא יעביר את החבילה אך ורק אל הכניסה שתוביל את החבילה ליעדה, אם הכתובת לא מוכרת המתג יעביר את החבילה לכל הכניסות פרט לזו שממנה היא התקבלה



MAC	Interface
11-AD-CC-47-35-11	1
11-AD-CC-47-35-12	2
11-AD-CC-47-35-13	3

עבור כל חוט תקשורת המתג משדר ב-2 כיוונים, input ו- output. ובכל אחת מהיציאות והכניסות האלו ישנו תור, כך בעצם המתג מתמודד עם הבעיה של התנגשויות על הקו, הוספת מעבד וניהול התורים.
המעבד אחראי להעביר חבילות מהתור של הכניסה את התור המתאים של היציאה.
המתג מבצע תהליך למידה עצמי בכדי לדעת אילו מחשבים מחוברים לאיזה Interface. הוא עושה זאת על ידי שמירת כתובת ה-MAC של הודעה מסוימת ושליחת ההודעה ב-broadcast, ולאחר שמתקבלת הודעה חזרה מהיעד הוא יכול ללמוד לאן מחובר כל מחשב מהיציאות שלו.
לטבלה זו יש גודל מסויים וברגע שהטבלה חורגת מהגודל, היא מתחילה "לזרוק כניסות" (Floading)
מתג הינו "Plug & Play" ואינו מקבל כתובת IP.

ישנן שלוש שיטות בהן יכול לתפקד מתג להעברת נתונים-

Cut Through (קיצור דרך)-

המתג קורא את הפתיח של חבילת הנתונים, וברגע שהוא יודע לאן עליו לשלוח את החבילה הוא מעביר אותה הלאה ללא בדיקה נוספת של תוכן החבילה. דרך זו היא המהירה ביותר, אך היא לא מאפשרת לוודא שחבילת הנתונים היא תקינה, ולכן הנתונים יעברו את כל הדרך שוב במקרה ונוצרה שגיאה בחבילת הנתונים.

Store and forward (שמור ושלח)-

המתג יקרא את כל חבילת הנתונים, יוודא את תקינותה ורק אם היא תקינה יעביר אותה הלאה. שיטה זו אטית יותר מהקודמת מכיוון שהמתג מקבל את כל תוכן החבילה לפני שהוא מעביר אותה הלאה, אך היא מבטיחה שרק חבילות נתונים תקינות יעברו את המתג. שיטה זו דורשת שמירה זמנית (caching) של החבילות בזיכרון המתג.

error free cut-through (קיצור דרך ללא שגיאות)-

כמו בטכנולוגיית cut through המתג קורא את הפתיח של חבילת הנתונים, וברגע שהוא יודע לאן עליו לשלוח את החבילה הוא מעביר אותה הלאה. אך בשיטה הזו המתג ממשיך לבדוק את התקשורת, ובאם הוא מגלה שגיאה בחבילת נתונים הוא מעביר את החיבור ממנו הגיעה החבילה למצב Store and forward ובודק כל חבילה שמגיעה מאותו המקור לפני שהוא מעביר אותה, כאשר מספר חבילות מאותו המקור הגיעו תקינות החיבור חוזר לפעול בשיטת Cut Through. שיטה זו מבטיחה את הטוב שבשני העולמות - כל עוד הרשת מתפקדת ללא שגיאות המתג מספק מהירות העברה מקסימלית, וברגע שמתגלה שגיאה הוא דואג לנפות חבילות נתונים שגויות.
בכדי למנוע מעגלים מתג עושה שימוש בפרוטוקול STP.

שם	שכבה	כתובות	מטרה ודרך פעולה
HUB (רכזת)	פיזית (1)	לא מכיר כתובות	יוצר קשר בין מספר ישויות. מעביר כל מידע לכלל הישויות המחוברות אליו.
Switch (מתג)	קו (2)	MAC Addresses	יוצר קשר בין מספר ישויות ברמת הקו. מעביר כל מסגרת רק למי שהמסגרת מיועדת אליו, באמצעות טבלה הממפה כתובת MAC לפורט פיזי.
Router (נתב)	רשת (3)	IP Addresses	מקשר בין רשתות ומחשבים ברמת ה-IP. מחליט על הדרך הטובה ביותר לנתב חבילה ממקור אל יעד. לרוב פועל בעזרת טבלאות ניתוב דינמיות.

The Spanning Tree Protocol - STP-

פרוטוקול העץ הפורש הוא פרוטוקול תקשורת שנועד למנוע לולאות ברשת מקומית המגושרת באמצעות גשרים או מתגים בחיבור ל-Ethernet. הפעולה הבסיסית של STP היא למנוע לולאות. עץ פורש מאפשר לכלול קישורים עודפים (redundant links) כדי לספק גיבוי נתיבים במקרה ונתיב פעיל מושבת, ללא סכנת לולאות וללא הצורך באפשור/ביטול קישורים בצורה ידנית. בתקן 802.1d-2004 נשמר רק לצורך תאימות לאחור והוחלף בפרוטוקול העץ הפורש המהיר The Rapid Spanning Tree Protocol (RSTP).

הפרוטוקול מייצג את טופולוגית הרשת של רשת מקומית באמצעות גרף בלתי מכוון שבו כל צומת מייצג גשר או מתג וכל קשת מייצגת קישור. המטרה של הפרוטוקול היא ליצור טופולוגית רשת של עץ פורש באמצעות חישוב משותף (כלומר חישוב מבוזר) של הרכיבים בשכבת הקו.

אחת מהבעיות שצצות ברשת מקומית המחוברת במתג היא שאם מדובר בארגון או חברה, והמתג שולח הודעה ב-broadcast, יכול להיות שההודעה מכילה מידע רגיש ולא נרצה שלכל אחד שמחובר לרשת תהיה האפשרות לקבל את אותה הודעה, מסיבה זו פותחה טכנולוגיית VLAN

-Virtual Local Area Network - VLAN

רשת תקשורת מקומית וירטואלית, היא מקטע- Segment, ברשת מקומית שמוגדר מבחינה לוגית כרשת אוטונומית.
פרוטוקול VLAN מגדיר טכנולוגיה המאפשרת לחלק רשת LAN מקומית, המרוכזת על ידי מתג אחד או יותר למספר תת-רשתות "וירטואליות".
הבידול לרשתות שונות נעשה פיזית, על ידי חסימת העברת אותות במתג בין הפורטים השונים שלו אשר אינם משויכים לאותו ה-VLAN. החסימה מאפשרת את חלוקת הרשת המקומית למספר רשתות שונות ונפרדות (Segments) אשר יכולות לתקשר ביניהן כרשת WAN.

יישום של רשתות מקומיות וירטואליות נעשה כיום בעזרת מתגים "חכמים", המאפשרים קיבוץ לוגי של מספר קצוות המחוברים אליהם. מספר מתגים המחוברים ביניהם מתפקדים לצורך העניין כמתג בודד, ומאפשרים קיבוץ קצוות מכל רחבי מתחם השידור (המוגבל בדרך כלל על ידי נתבים) והפרדתם למתחם שידור נפרד. המתג מנהל רשימה של כל כתובות הMAC השייכות לרשת הווירטואלית, ומפריד אותם משאר הקצוות ברשת. על ידי הפרדה זו ניתן למשל לחלק לקצוות אלו כתובות IP השייכות לתת רשת מסוימת, להחיל מדיניות אבטחה (על ידי הגדרת הרשאות שונות למי ששייך לרשת הווירטואלית ולמי שלא) ועוד.

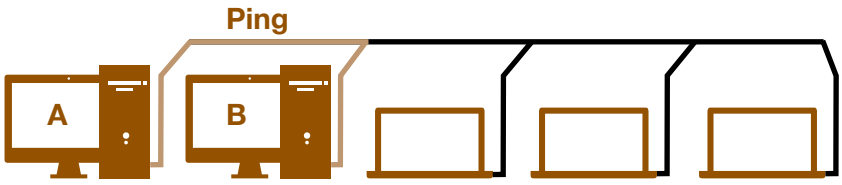
-MAC Address

כתובת MAC- Media Access Control address או בתרגום חופשי: כתובת בקרת גישה למדיה, היא מזהה ייחודי המוטבע על כל רכיב תקשורת לתקשורת נתונים בעת הייצור. כתובת ה-MAC מוטבעת בדרך כלל בכרטיס הרשת של המחשב ו/או במודם. כתובות MAC נחשבות כחלק משכבת הקישוריות ונקבעות על ידי ארגון ה-IEEE. בעבר הכתובות בשכבת הקישוריות נקבעו ונוהלו באופן ידני על ידי מנהלי הרשת, אך פרוטוקול Ethernet הגדיר תקן לקביעת כתובות MAC ייחודיות. תקן זה הוטמע בהדרגה בפרוטוקולים נוספים בשכבת הקישוריות וכיום הוא משמש ברוב הפרוטוקולים בשכבה זו (לדוגמה Token Ring ו-ATM).

כתובת MAC בפורמטים MAC-48 ו־EUI-48 מורכבת מ-48 סיביות, מה שמאפשר כ-2 בחזקת 48 (281 טריליון) כתובות שונות. ה-IEEE מקצה לכל יצרן חומרה קידומת ייחודית בת 24 סיביות שבה עליו להשתמש בכל רכיבי התקשורת שהוא מייצר, מה שמותיר לכל יצרן כ-16 מיליון כתובות אפשריות. נהוג לכתוב את הכתובת כ-6 זוגות של ספרות הקס-דצימליות, המופרדים בנקודתיים או מקפים. לדוגמה: 00-0E-A6-A7-63-F7.

Address Resolution Protocol - ARP

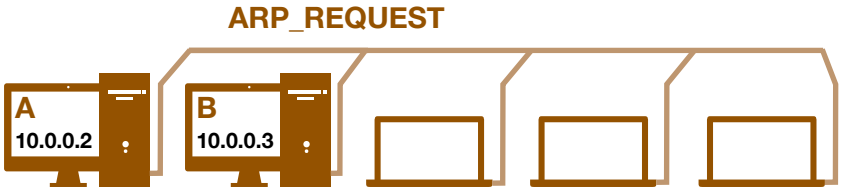
עד כה תיארנו את פרוטוקול Ethernet וכיצד הוא עובד. אך עדיין, משהו חסר. הביטו בשרטוט הרשת הבא:



כלל המחשבים כאן נמצאים על תווך משותף. כלומר- מסגרת הנשלחת לכתובת Broadcast תגיע אל כולם, ואין צורך בישות נוספת (כגון נתב) כדי להעביר הודעות ממחשב אחד למחשב אחר. במקרה לפנינו, המחשב שנקרא A רוצה לשלוח הודעת ping (כלומר Echo-Request) אל המחשב B. המחשב A יודע את כתובת ה-IP של מחשב B, למשל, באמצעות שימוש בפרוטוקול DNS. אך דבר זה אינו מספיק- על מחשב A לדעת גם את כתובת ה-MAC של כרטיס הרשת של מחשב B!

אך מדוע הדבר כך? מדוע לא מספיקה כתובת ה-IP?
ראשית, היזכרו במודל השכבות. חבילת ה-Ping שתשלח ממחשב A צפויה להיות בנויה משכבת ה-Ethernet בשכבה השנייה, מעליה שכבת IP ולבסוף שכבת ICMP. על מנת לבנות את מסגרת ה-Ethernet, על המחשב A לדעת מה כתובת היעד של המסגרת, כלומר מה הכתובת של כרטיס הרשת של B. שנית, על כרטיס הרשת של המחשב B להבין שהמסגרת מיועדת אליו. את זאת הוא עושה באמצעות הסתכלות בשדה כתובת היעד של המסגרת. כרטיס הרשת אינו מבין כתובות IP, הוא כרטיס מסוג Ethernet ומכיר כתובת MAC בלבד. על כן, הכרטיס צריך לראות כתובת Ethernet.

אי לכך, על המחשב A להבין מה כתובת ה-MAC של המחשב B. לשם כך נועד פרוטוקול ARP (Address Resolution Protocol- פרוטוקול תרגום כתובות. פרוטוקול זה ממפה בין כתובות לוגיות של שכבת הרשת לכתובות פיזיות של שכבת הקו. במקרה שלנו, המחשב A מעוניין למפות בין כתובת ה-IP הידועה לו של מחשב B, לבין כתובת ה-MAC של כרטיס הרשת של המחשב B. בשלב הראשון, המחשב A ישלח שאלה לכל הרשת (כלומר - לכתובת Broadcast עם כתובת ה- FF:FF:FF:FF:FF:FF MAC) שמשמעותה: למי יש את הכתובת הפיזית של המחשב עם כתובת ה-IP של B? נאמר שכתובת ה-IP של מחשב B הינה 10.0.0.3, והכתובת של המחשב A הינה 10.0.0.2:



ניתן לומר ש-A בעצם שלח אל כל המחשבים שמחוברים לאותו הקו את ההודעה "אני מחפש את הכתובת הפיזית של מחשב עם כתובת ה-IP 10.0.0.3"

בשלב זה, כלל המחשבים מקבלים את ההודעה. מי שצפוי לענות להודעה זו הוא המחשב B, אשר יודע את הכתובת הפיזית שלו. כשהמחשב שכתובתו היא 10.0.0.3, במקרה שלנו מחשב B, יקבל את החבילה הוא יחזיר חבילת ARP המיועדת אל מחשב A בלבד המכילה בין השאר בשדה כתובת החומרה של השולח את כתובת החומרה שלו.

כך מחשב A מגלה את כתובת ה-MAC של כרטיס הרשת של מחשב B. כעת, יש ברשותו את כל המידע שהוא צריך בכדי לשלוח את חבילת ה-Ping:

- כתובת ה-IP שלו עצמו (מחשב A) - שכן הוא יודע מה הכתובת שלו, למשל באמצעות DHCP.
- כתובת ה-MAC שלו עצמו (מחשב A) - שכן הוא יודע מה הכתובת שלו, שהרי היא צרובה על הכרטיס.
- כתובת ה-IP של מחשב B - שהוא גילה, למשל, באמצעות DNS.
- כתובת ה-MAC של מחשב B - שהוא גילה באמצעות פרוטוקול ARP.

מטמון (Cache) של ARP-

גם עבור פרוטוקול ARP מערכת ההפעלה שלנו שומרת מטמון (Cache), במטרה לא לשאול שוב ושוב את אותה שאלת ARP. כפי שכבר הבנו, המחשב שלנו זקוק לתקשר עם הנתב הקרוב אליו באופן תדיר. על מנת לאפשר את התקשורת עם הנתב, עליו לדעת מה כתובת ה-MAC שלו. לא הגיוני שלפני כל חבילה שהמחשב יעביר לנתב הוא יבצע שאילתת ARP ויחכה לתשובה - תהליך זה יקח זמן רב מידי. לכן, סביר שכתובת ה-MAC של הנתב תישמר במטמון.

סיכום תכנותי

Unix

יוניקס היא מערכת הפעלה מרובת משימות ומרובת משתמשים, הנחשבת ליעילה ואמינה.
דרישות המערכת הנמוכות מאפשרות להריצה גם על מחשבים חלשים יחסית.
יוניקס שוכתבה ושימשה השראה למשפחה גדולה של מערכות הפעלה, ביניהן לינוקס, המהווה את הליבה של רוב שרתי האינטרנט ומרבית הציוד לניהול רשתות והעברת מידע ומערכת ההפעלה של חברת אפל, הנקראת OS X ובמידה פחותה גם מערכות ההפעלה לטלפונים אנדרואיד iOS. יוניקס נכתבה, ברובה, בשפת תכנות עילית - שפת C, דבר שאפשר לה להיות מוסבת לארכיטקטורות מעבדים שונות, ולשרוד עם הזמן. זאת בעוד, שמערכות הפעלה מזמנה, שנכתבו ברובן בשפת סף (אסמבלי), בדרך כלל לא שרדו את המעברים לארכיטקטורות מחשבים חדשות.

במרכז מערכת ההפעלה ישנו Hardware (החומרה) הכוללת יחידת עיבוד נתונים (CPU), הזיכרון הפנימי והחיצוני, כוננים, מדפסות ועוד. את החומרה מקיף גרעין מערכת ההפעלה ודרכו, באמצעות שפת מכונה, מתקשרים עם החומרה.

Kernel (גרעין/ליבה)-

ליבה היא הרכיב המרכזי של מרבית מערכות ההפעלה, זהו הגשר שבין תכניות המחשב לבין עיבוד הנתונים עצמו שמבוצע ברמת החומרה. אחד התפקידים העיקריים של הליבה הוא ניהול משאבי המערכת (התקשורת שבין רכיבי החומרה והתוכנה).
ליבת מערכת ההפעלה היא התכנית היחידה אשר מוכנה לריצה בכל זמן שהוא. בנוסף, הליבה היא התוכנה היחידה שיכולה לבצע אוסף פקודות בצורה אטומית (ללא שום הפרעה או הפסקה מגורם כלשהו). כל מערכת הפעלה חייבת ליבה כדי לפעול, הגרעין מאתחל תכנית אינטראקטיבית הנקראת מעטפת.

Shell (מעטפת)-

מעטפת היא רכיב תוכנה המספק ממשק למשתמשים. המונח מתייחס בדרך כלל למעטפת של מערכת הפעלה, המספקת גישה לשירותי הליבה. עם זאת, המונח משמש גם לציון יישומים ה"בנויים מסביב" לרכיב מסוים, לדוגמה דפדפנים ותכניות לדואר אלקטרוני המשמשות כ"מעטפת" למנוע תצוגה מבוסס HTML.
למערכות הפעלה יש בדרך כלל שני סוגים של מעטפות: שורת פקודה וגרפית.
מעטפת שורת הפקודה מספקת ממשק שורת פקודה (CLI) למערכת ההפעלה, ואילו המעטפת הגרפית מספקת ממשק משתמש גרפי (GUI). בשתי הקטגוריות, היעוד העיקרי של המעטפת הוא לשפעל או "לשגר" תכניות אחרות, אך עם זאת, יש למעטפות לרוב גם יכולות נוספות כגון הצגת ספריות וקבצים.

הגרעין ממיין את הפקודות המגיעות אליו מהמעטפת על ידי המשתמשים השונים, ומקצה את המשאבים לפי סדר עדיפות הנקבע מראש. למשתמש אין גישה ישירה לגרעין.
הגרעין מאפשר גישה למשאבי המחשב למשתמשים השונים ולמשימות השונות.
ניתן להסתכל על הגרעין והמעטפת כזוג תבניות משולבות המהוות את מערכת ההפעלה UNIX.
קיימת תכנית גרעין אחת לכל המשתמשים, תיתכן מספר תכניות מעטפת כשלכל משתמש קיימת לפחות מעטפת אחת.

Utility software (תוכנת שירות)-

תוכנת שירות היא תוכנת מערכת שנועדה לעזור לנתח, לייעל או לתחזק מחשב.
תוכנת שירות מתרכזת לרוב בבחינת פעילות תשתיות המחשב (כולל חומרה, מערכת הפעלה, תוכנה, ואחסון מידע). תוכנת שירות, בדומה לתכנת מערכת הפעלה, היא סוג של תוכנת מערכת המבדילה אותו מיישום מחשב.

Open Source (קוד פתוח)-

קוד פתוח משמש בעולם התוכנה לציון תוכנה שקוד המקור שלה פתוח ונגיש לכל מי שחפץ בו והוא חופשי לשימוש, לצפייה, לעריכת שינויים ולהפצה מחודשת לכל אחד ואחת. שיטת פיתוח כזו מאפשרת בעצם לכל מי שחפץ בכך לקחת חלק בפיתוח התוכנה ולתרום לשיפורה. מושג קרוב במשמעותו (ויש אומרים שזהה במשמעותו) הוא "תוכנה חופשית". לעיתים כוללים את שניהם ביחד כ"תוכנה חופשית וקוד פתוח"

יש להבחין בין קוד פתוח לבין קוד נגיש – מצב שבו ניתן לראות את הקוד אך אסור לבצע בו שינויים. מצב זה אינו נופל תחת הקטגוריה של קוד פתוח.

פקודות ב- Unix הן למעשה תכניות, ולכן הקשת שם פקודה היא בעצם בקשה למערכת ההפעלה להריץ תכניות מסוימות, המאוחסנות במקום מסוים.

המערכת מבדילה בין אותיות קטנות לגדולות ומזהה קבצים על פי מיקומם בהיררכיית הקבצים, ולכן אין כל מניעה כי לשני משתמשים יהיו שמות קובץ זהים, או שלמשתמש אחד יהיו שמות קובץ זהים השייכים לקבצים שונים אשר ממוקמים במקומות שונים בהיררכיית מערכת הקבצים.

Compiler (מהדר)-

מהדר הוא תכנית מחשב המתרגמת משפת מחשב אחת לשפת מחשב אחרת. המהדר הקלסי מקבל כקלט תכנית הכתובה בשפה עילית ומתרגם אותה לתכנית בשפת מכונה. מהדר תואם בדרך כלל לסוג מעבד מסוים בלבד, ולגרסת שפה מסוימת, ותוצריו לא יהיו תואמים למעבד אחר. פעולת המהדר קרויה הידוּר.

ב-Unix המהדר נקרא gcc.

מערכת ההפעלה אינה מבדילה בסיומות של קבצים, אך הקומפיילר (המהדר) מבדיל.

במידה ונרצה לקמפל קובץ בשפת c עליו להסתיים ב-c.

לאחר פקודת gcc נוצר קובץ הפעלה (execute). במידה ולא מבטלים, נוצר גם קובץ out שבו הקובץ המקומפל.

Linux

לינוקס היא משפחה של מערכות הפעלה המבוססות על ליבת לינוקס (ליבת הלינוקס נוצרה על ידי הסטודנט הפיני לינוס טורבאלדס בשנת 1991 מתוך שאיפה לאפשר שימוש במערכת ההפעלה יוניקס גם על מחשבים אישיים). מערכת לינוקס שכוללת רכיבים וספריות ממיזם גנו נקראת לעיתים גנו/לינוקס (GNU/Linux).

הפיתוח הראשוני בשנות ה־80 התמקד בגנו ובמערכת הגרפית X11. בשנות ה־90 המוקדמות, מפתחי ליבת לינוקס ואנשים נוספים החלו לעבוד על לינוקס.

לינוקס היא דוגמה חשובה לפיתוח תוכנה חופשית וקוד פתוח. קוד המקור של ליבת לינוקס זמין לשימוש, לשינוי ולהפצה בחינם לכל אחד. יש מקרים שמערכת הפעלה שלמה מורכבת מתוכנות חופשיות או מתוכנות קוד פתוח.

ישנן אין ספור הפצות לינוקס- כאלה המיועדות לשרתים וכאלה המיועדות למשתמשים פרטיים.

אנדרואיד היא דוגמה למערכת הפעלה המבוססת על ליבת לינוקס. היא רצה בעיקר על טלפונים חכמים ומחשבי לוח, ומפותחת על ידי גוגל.

ממשק שורת הפקודה-

אחד הממשקים הנפוצים ללינוקס הוא ממשק שורת פקודה. לרוב משתמשים בבאש (מעטפת פקודה למערכות UNIX ומערכות דמויות יוניקס מאת פרויקט GNU).

הפקודה רצה מול מסוף מחשב מסוג זה או אחר: בסביבה הגרפית: לרוב מדמה מסוף. אפשר לעבוד מול המחשב המקומי או להתחבר למחשב אחר (לרוב דרך SSH).

ממשק שורת הפקודה נחשב למינימליסטי, הוא לא צורך משאבים רבים מהמחשב. הוא גם לא יוצר תעבורה רבה ולכן יכול לעבוד בצורה יעילה גם בחיבורים אטיים.

יתרון נוסף של ממשק שורת הפקודה הוא שקל מאוד לשלב פקודות שונות לפקודות מורכבות ואף לתסריטים פשוטים ובכך ליצור אוטומטיזציה של העבודה.

לינוקס, מעצם תכנונה כמערכת דמוית יוניקס, כוללת ממשק שורת-פקודה יעיל ועשיר.

פקודות בסיסיות

1. **pwd-** כאשר פותחים לראשונה את המסוף, נמצא בספרייה הביתית של המשתמש. בכדי לדעת איזה ספרייה נמצאים, ניתן להשתמש בפקודה "pwd". פקודה זו נותנת לנו את הנתיב המוחלט, כלומר את הנתיב שמתחיל מן השורש. השורש הוא הבסיס של מערכת הקבצים לינוקס. הוא מסומן על ידי קו נטוי קדמי (/). הנתיב של משתמש הוא בדרך כלל משהו כמו "/ home / username".

2. **ls-** השימוש בפקודה "ls" הוא כדי לדעת אילו קבצים נמצאים בספרייה שבה נמצאים. ניתן לראות את כל הקבצים מוסתרים באמצעות הפקודה "ls-a".

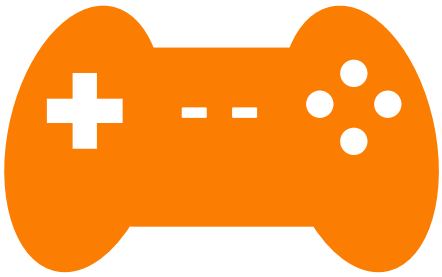
3. **cd-** נשתמש בפקודה "cd" כדי לעבור לספרייה. לדוגמה, אם נמצאים בתיקייה הביתית, וברצוננו לעבור לתיקיית ההורדות, נוכל להקליד "cd Downloads". זכרו, פקודה זו "רגישה" לאותיות גדולות, ויש צורך להקליד את שם התיקייה בדיוק כפי שהיא. אבל יש בעיה עם פקודות אלה, תארו לעצמכם תיקייה בשם "Raspberry Pi". במקרה זה, כאשר אתה מקליד "cd Raspberry Pi", המעטפת תתייחס אל הארגומנט השני של הפקודה כמו אל פקודה נפרדת, כך נקבל שגיאה האומרת כי הספרייה אינה קיימת. הפתרון הוא שימוש בקו נטוי לאחור. כלומר, במקרה זה יש להקליד- "cd Raspberry\ Pi". אם אתה פשוט נקליד "cd" ואז enter, נגיע אל ספריית הבית. כדי לחזור מתיקייה לתיקייה לפני כן, נוכל פשוט להקליד ".. cd". שתי הנקודות מייצגות חזרה.

4. **mkdir & rmdir-** נשתמש בפקודה mkdir כאשר צריך ליצור תיקייה או ספרייה. לדוגמה, אם נרצה ליצור ספרייה בשם "DIY", אז נקליד "mkdir DIY". זכרו, כפי שאמרנו קודם, אם נרצה ליצור ספרייה בשם "DIY Hacking", אז נצטרך להקליד "mkdir DIY\ Hacking".
השימוש ב- rmdir הוא כדי למחוק ספריה. אבל rmdir יכול לשמש רק כדי למחוק ספרייה ריקה. כדי למחוק ספריה המכילה קבצים, יש להשתמש ב- rm.

5. **rm-** נשתמש בפקודה rm כדי למחוק קבצים וספריות. הפקודה "rm -r" היא בכדי למחוק רק את הספרייה. שימוש בפקודה rm מוחק הן את התיקייה ואת הקבצים שהיא מכילה.

6. **touch-** הפקודה touch משמשת ליצירת קובץ. הקובץ יכול להיות כל דבר, החל מקובץ txt ריק ועד קובץ zip ריק. לדוגמה, "touch new.txt".

7. **man & help—-** כדי לדעת יותר על פקודה וכיצד להשתמש בה, נשתמש בפקודה man. הפקודה תציג את ה"מדריך" של הפקודה. לדוגמה, "man cd" מציג את המדריך למשתמש של הפקודה cd. הקלדת שם הפקודה והארגומנט "help" תציג את הדרכים שבהן ניתן להשתמש בפקודה (למשל, cd -help).



8. **cp-** נשתמש בפקודה cp להעתקת קבצים דרך שורת הפקודה. לפקודה זו יש שני ארגומנטים: הראשון הוא המיקום של הקובץ להעתקה, השני הוא המיקום ברצוננו להעתיק אליו.

9. **mv-** נשתמש בפקודה mv בכדי להעביר קבצים דרך שורת הפקודה. אנו יכולים גם להשתמש בפקודה mv כדי לשנות את שם הקובץ. לדוגמה, אם אנחנו רוצים לשנות את שם הקובץ "text" ל- "new", אנו יכולים להקליד "mv text new". לפקודה זו יש לספק שני ארגומנטים בדיוק כמו הפקודה cp.

10. **locate-** הפקודה Locate משמשת לאיתור קובץ במערכת Linux, בדיוק כמו פקודת החיפוש ב- Windows. פקודה זו שימושית כאשר איננו יודעים היכן נשמר או מה שמו בפועל של הקובץ. שימוש בארגומנט i- עם הפקודה יגרור התעלמות מאותיות קטנות או גדולות. אז, אם נרצה קובץ שיש לו את המילה "hello" בשם, הפקודה תחזיר את רשימת כל הקבצים במערכת המכילים את המילה "hello" כאשר נקליד "locate -i hello". אם אנחנו זוכרים שתי מילים, ניתן להפריד אותם באמצעות כוכבית (*). לדוגמה, כדי לאתר קובץ המכיל את המילים "hello" ו "this", נוכל להשתמש בפקודה כך: "locate -i *hello*this".

פקודות ביניים

1. **echo-** הפקודה "echo" מסייעת לנו להעביר נתונים מסוימים, בדרך כלל טקסט לקובץ. לדוגמה, אם נרצה ליצור קובץ טקסט חדש או לחלופין להוסיף לקובץ טקסט שכבר נוצר, יש רק צורך להקליד את הפקודה כך: "echo hello world >> new.txt". אין צורך להפריד את מילים באמצעות קו נטוי אחורי, כי אנחנו שמים שני סוגריים משולשיים כאשר אנו מסיימים את מה שאנחנו צריכים לכתוב.

2. **cat-** נשתמש בפקודה cat כדי להציג את תוכן הקובץ. זה בדרך כלל משמש להצגת תכניות באופן קל.

3. **nano, vi, jed, pico-** nano ו- vi הם עורכי טקסט שכבר מותקנים בשורת הפקודה של לינוקס. הפקודה nano הוא עורך טקסט המציין מילות מפתח עם צבע ויכול לזהות את רוב השפות. ו- vi הוא פשוט יותר מאשר nano. באפשרותנו ליצור קובץ חדש או לשנות קובץ באמצעות עורך זה. לדוגמה, אם נצטרך ליצור קובץ חדש בשם "check.txt", ניתן ליצור אותו באמצעות הפקודה "nano check.txt".

4. **sudo-** פקודה בשימוש נרחב בשורת הפקודה לינוקס, sudo מייצג "SuperUser Do". כך, אם נרצה להריץ פקודה כלשהי עם הרשאות מנהל או שורש, נוכל להשתמש בפקודה sudo. לדוגמה, אם ברצוננו לערוך קובץ כמו viz. alsa-base.conf, אשר צריך הרשאות שורש, נוכל להשתמש בפקודה - sudo nano alsa-base.conf. ניתן היכנס אל שורת הפקודה של השורש באמצעות הפקודה "sudo bash" ולאחר מכן להקליד את סמסמת המשתמש. אפשר גם להשתמש בפקודה "su" כדי לעשות זאת, אבל אז צריך להגדיר את סיסמת השורש לפני כן. בשביל זה, ניתן להשתמש בפקודה "sudo passwd" (לא מדובר בשגיאה, זה passwd) ולאחר מכן הקלד את סיסמת השורש החדשה.

5. **df-** השימוש בפקודה df יראה לנו את שטח הדיסק הזמין בכל אחת מהמחיצות במערכת. אפשר פשוט להקליד df בשורת הפקודה ואז ניתן לראות כל מחיצה ואת השטח בשימוש/הזמין שלהם באחוזים וב- KBs. אם נרצה את המידע מוצג במגה בייט, נשתמש בפקודה "df-m".

6. **du-** נשתמש ב- du כדי לדעת את השימוש בדיסק של קובץ במערכת. אם נרצה לדעת את השטח בדיסק שנמצא בשימוש עבור תיקייה מסוימת או קובץ ב- Linux, נוכל להקליד את הפקודה df ואת שם התיקיה או הקובץ. לדוגמה, אם נרצה לדעת את שטח הדיסק המשמש את התיקייה מסמכים ב- Linux, נוכל להשתמש בפקודה "du Documents". ניתן גם להשתמש בפקודה "ls-lah" כדי להציג את גודל הקבצים של כל הקבצים בתיקייה.

7. **tar-** נשתמש ב- tar לעבוד עם tarballs (או קבצים דחוסים בארכיון tarball) בשורת הפקודה. יש לו רשימה ארוכה של שימושים. tar יכול לשמש כדי לדחוס ולפתוח סוגים שונים של ארכיוני tar כמו לדוגמה: .tar.bz2 ,.tar.gz ,.tar, וכו'. הפקודה tar פועלת על בסיס הארגומנטים שניתנו לה. לדוגמה, "tar -cvf" ליצירת ארכיון .tar, -xvf בכדי "לחלץ" ארכיון tar, -tvf כדי לרשום את תוכן הארכיון, וכו'.

8. **zip, unzip-** נשתמש ב- zip בכדי לדחוס קבצים לתוך ארכיון zip, ובכדי לחלץ קבצים מארכיון zip.

9. **uname-** נשתמש ב- uname כדי להציג את המידע על המערכת הפצת לינוקס שלנו פועל. שימוש בפקודה "uname -a" מדפיסה את רוב המידע אודות המערכת. פעולה זו מדפיסה את תאריך השחרור, הגרסה, סוג המעבד וכו'.

10. **apt-get-** נשתמש ב-apt כדי לעבוד עם חבילות בשורת הפקודה וב- apt-get כדי להתקין חבילות. דבר זה דורש הרשאות שורש, לכן יש להשתמש בפקודת sudo. לדוגמה, אם ברצוננו להתקין את עורך הטקסט Jed, אנחנו יכולים להקליד את הפקודה "sudo apt-get install jed". באופן דומה, ניתן להתקין כך כל חבילה. רצוי לעדכן את המאגר בכל פעם שמנסים להתקין חבילה חדשה. ניתן לעשות זאת על ידי הקלדת "sudo apt-get update". ניתן לשדרג את המערכת על ידי הקלדת "sudo apt-get upgrade". אנחנו יכולים גם לשדרג את distro על ידי הקלדת "sudo apt-get dist-upgrade". הפקודה "apt-cache search" משמשת לחיפוש חבילה. אם נרצה לחפש אחד, ניתן להקליד "apt-cache search jed".

במערכות הקבצים של לינוקס ישנה חלוקה של המשתמשים לשלוש קבוצות:
u- המשתמש (user) שיצר את הקובץ.
g- הקבוצה (group) הקבוצה אליה משתייך בעל הקובץ.
o- אחרים (other) כל משתמשי המערכת.

ניתן להגדיר הרשאות שונות לכל אחת מהקבוצות כאשר גם ההרשאות מחולקות לשלוש:
r- קריאה (read) כלומר קריאת תוכן הקובץ כולל העתקת הקובץ אלי.
w- כתיבה (write) כלומר כתיבה ושינוי תוכן הקובץ כולל מחיקתו.
x- ביצוע (execute) כלומר ביצוע או הרצת התכנית/פקודה.

מעל לחלוקה זו יושב מנהל המערכת (משתמש root) ויכול להתעלם מההרשאות של הקובץ (גם קובץ שאף אחד לא יכול לקרוא, root יכול). היחיד שיכול לשנות הרשאות (חוץ מ-root) הוא המשתמש שיצר את הקובץ, הוא מוגדר כ"בעלים" של הקובץ.

כאשר נרשום ls -l נקבל את הרשימה המפורטת של הקבצים. הפלט של הפקודה הינו רשימת קבצים בציון סוג הקובץ, רשימת ההרשאות של הקובץ, בעל הקובץ, הקבוצה אליה משתייך בעל הקובץ, גודל הקובץ, תאריך אחרון בו כתבו את הקובץ לדיסק ואת שם הקובץ.

בחלקו השמאלי של הפלט- האות הראשונה מסמלת את סוג הקובץ.
d תיקייה או ספרייה.
- קובץ.

תשע האותיות הבאות מסמנות את ההרשאות של הקובץ.
ההרשאות מקובצות בקבוצות של 3:
שלישייה ראשונה- הרשאות עבור המשתמש (u)
שלישייה שנייה- הרשאות עבור הקבוצה (g)
שלישייה אחרונה- הרשאות עבור כלל המשתמשים (o)

התו "-" מסמן שאין הרשאה ומיקומו של התו מסמן למי ומהו סוג ההרשאה

בהצלחה