

סיכון תקשורת מחשבים – סרגי דבורהן

תקשורת בין יחידות קצה ביתית – Home Access

DSL – Digital Subscriber line

תקשרות על קו של חברת הטלפון כבל נחושת מודם DSL משתמש בקו הטלפון המקורי כדי להעביר מידע, המודם מתרגם את המידע הדיגיטלי לאנלוגי בתדר גבוה אשר מתרגם חזרה לדיגיטלי בעזרת ה **DSLAM** אשר נמצא במרכז הבקרה של חברת הטלפון.

קידוד התדרים :

- ערוץ הורדת מידע – 50KHz – 1 MHz (מהירות סטנדרטית Mbps 12-24)
- ערוץ העילאת מידע – 4KHz – 50 MHz (מהירות סטנדרטית Mbps 1.8 – 2.5)
- ערוץ הטלפון הדו ציוני – 2Hz – 4KHz

מקום ה DSL (הלקוח) צריך להיות בין 5 ל 10 מיל ממרכז הבקרה של חברת הטלפון.

אינטרנט בכבלים (של חברת כבלים לטלויזיה)

מסופק באמצעות כבל סיב אופטי אל "הצומת השכונית" – Fiber Node , ומשם לכל בית בכבל קוakisיאלי.

בכל שהמערכת מתקשרת בעזרת 2 סוגי כבלים, נהגים לקרוא לה – HFC – Hybrid Fiber Coax

כל Fiber Node יכול לתמוך ב 500 – 5000 בתים במוצע.

מערכת זו דורשת חיבור של התקן חיצוני – "מודם הcablim" אשר מחובר בד"כ למחשב הבית בחיבור Ethernet.

במרכז הבקרה של חברת הcablim ישנו – CMTS אשר תפקido לתרגם את המידע האנלוגי המועבר באמצעות הcablim למידע דיגיטלי חזרה (בדומה ל- DSLAM).

הגדרת מהירות סטנדרטיבית :

- הורדה – עד ל 42.8 Mbps
- העילאה – עד ל 30.7 Mbps
-

FTTH – Fiber To The Home

העברת סיב אופטי אל כל בית, 2 ארכיטקטורות קיימות :

AON – Active Optical Network

PON – Passive Optical Network

בכל בית ישנו ONT – Optical Network Terminator אשר לחבר סיב אופטי ייחודי לבית הלקוח אל מול מפצל אופטי (שכוני) אשר מקבל עד 100 חיבורים (בתים), מאחד אותם לסיב יחיד אשר מחובר ל OLT – Optical Line Terminator אשר במרכז הבקרה.

Packet Switching

נתב (Router), מקבל "חבילות" מידע (Packets), עליו לקבע בזמן אמתஇeo חבילת העبور ולאחר. היהות יוכל להיות עיקוב בהעברת מידע לנטים, לנטים יש נתכים FIFO וכאשר חבילה תקועה בנתב הרבה זמן, היא תעבור לתור FIFO, על מנת שבחילות טריות יותר יצליחו לעبور מהר יותר. עיקוב של חבילות מנהלות על ידי כמה קווים של FIFO - LIFO כאשר השאייה היא להוציא חבילות טריות יותר. (זה קורה ב Packet Switching כאשר נתב מסוים עמוס).

בכל שנתב צריך לעשות מילוי החלטות בשניה (לאיזה קו להוציא מהנתב לנtab אחר רלוונטי), דרוש מבנה נתונים מהיר שיקשר בין היעד שעלה החבילה להגעה אליו לבין איזה קו (נתב אחר) קיבל את החבילה. בכל נתב ישנה טבלה (מבנה נתונים) כזו.

מייתוג מעגלים (Circuit Switching) – החיבור בין המוצא אל היעד נעשה בחיבור ישיר, כלומר אין נתבים בדרך שצרכיהם לעשות החלטות, אלא הכל מתבצע "בזמן אמת", בזמן ידוע ומוקצב מראש. (לא משתמשים ברגע זה ברשות). שיטה זו מוגבלת בכמות ה"משתתפים" של קו.

Packet Switching (רשת האינטרנט):

- כמות גדולה של מידע ומשתמשים
- לא תמיד החלטות הניטוב "טבות", מה שיכול לגרום לעיקוב בהעברת המידע
- כאשר אין העברות מידע על הקו, הקו פניו ונitin לשימוש (מייעל את השימוש)

Circuit Switching (מייתוג מעגלים):

- חיבור ישיר, כלומר המידע הגיע בוודאות בזמן ידוע
- כמות קטנה יותר של מידע ומשתמשים
- אם הקו מחובר, גם אם אין העברת מידע, הקו תפוס ואי אפשר לפנות אותו לטובות משתמש אחר. (ניצול גרוע של רוחב הפס).

ISP – Internet Service Provider –
IXP – Internet Exchange Provider –

Packet Delay

עיקוב בשליחת נתונים מנתב לנtbל תלי ב – 4 גורמים:

- Propagation Delay - דילוי התקדמות מנתב לנtb (מהירות זרימת הנתונים)
- Transmission Delay – רוחב פס. העיקוב תלי באורך החבילה (L – בביטים) לעומת רוחב הפס (R)
- Queueing Delay – עומס שנשאר לקו כורם לתור של העברת החבילות (קצב הכניסה לעומת קצב היציאה)
- Nodal Processing – קבלת החלטות של המtb, בדיקת החבילה והיעד שלה.

העיקוב של חבילה, מהרגע שנכנסה למtb הראשון עד שהגיעה למtb היעד הסופי :

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- החבילה נכנסת למtb, עוברת בו עיבוד נתונים וקבלת החלטות (Processing), לאחר מכן נמצאת בתור (אם קיים) שנוצר ערב עומס בקו השידור (Queueing) לאחר מכן מושמת על הקו (הביטים) (Transmission) ולבסוף שהחבילה יכולה על הקו שידור, היא נעה בקבב בתלות של מրחק היעד ומהירות הזרמת הנתונים על הקו, זה משפיע על סוף קבלת כל החבילה ביעד (Propagation).

Packet Loss

ברגע שיש עיקוב בהעברת חבילות המידע, החבילה מגיעה לבאפר (Queue). המקום בבאפר הוא מוגבל ולכך ברגע שמגיבים לנפח אחסון מסוימלי של החבילות המועברות, יתכן איבוד מידע, ככלומר שחבילות מידע "זרקו" (בד"כ לפי הרעיון שההודעה המיושנת/פחות חשובה תימחק). ישנו כמה תורות (באפרים) במקביל, והם מתחלקים לפי זמן המתנה לחבילה בתור או לפי חשיבות החבילה. צוואר הבקבוק בהעברת מידע הוא בד"כ בכונסה או ביציאה, ככלומר, נגיד כשהמידע יוצא משרת אל יחידת קצה, אז היהות ורוחב הפס שלו מיוחובייה ייחידה או השרת יכולם להיות נמוכים, אך יכול להיווצר "פקק" בהעברת המידע. בדר"כ במרכז הרשת backbone אין בעיה כזאת להיות ורוחב פס שם מספיק רחב.

מודל 5 שכבות

Internet protocol stack

:Protocols

אפליקציה – תוכנה שiodעת לעבור "לדבר" בשפה מקובלת שתומכת ברשות (FTP, SMTP, HTTP) – היא מכילה את המידע שיועבר לידי

Transport – קישור בין אפליקציות יחידות הקצה, אחראי על המידע שעובר (UDP, TCP).

:Data Flow Layers

Network – מחברת בין יחידות הקצה על מנת שיכלו "לדבר" (כל תחבורה לtransport (routing protocol, ip)

Link – טכנולוגיות התחבורה של תוכן ה-transfer, שהוא עובר בדרך מיחידת הקצה של המקור אל היעד (כליום המסלול בין 2 נקודות קרובות – Wi-Fi, Ethernet, סיב אופטי) שכבת ה-network בוחרת לאיזו יחידה סמוכה (router) עליה לgesת על מנת להעביר את המידע בשכבות transport ו-az שכבת ה-link מבצעת את הקישור בין 2 הנתבים בסוגי תקשורת שונים.

Physical – השכבה שאחראית להעביר את הביטים למקום – "one the wire"

- 2 השכבות של ה-protocol נלקחו ממודל TCP/IP model (בעל 4 שכבות) , ו-3 השכבות הנותרות נלקחו מ- model OSI (בעל 7 שכבות)
- השכבות החסרות (מודל ה-OSR/250) – presentation. סיבה זו כוללת אפשרות הצפנה למידע שנשלח.

Session – דואג לגיבוי המידע במידה ויש נפילה ברשת.

שתי השכבות מוכילות ב-application .

ארQUITטורת אפליקציית רשת - client-server

חיבור בין יחידות קצה (clients) בעזרת שרתים. כלומר כל המידע עובר קודם דרך שרת, ורק אחר כך המידע מוצג ביחידת הקצה. זאת הארכיטקטורה הכי מקובלת באפליקציות רשת.

ארQUITטורת Peer To Peer (P2P)

החלפת סרברים ביחידות קצה של הלקחות. כלומר, ניצול של המשאים של שאר הלקחות לטובת יחידות קצה הנדרשים לכך. התקשרות מתבצעת בין 2 משתמשי האפליקציה.

הvisorון הוא שלעומת שרת (server) שנמצא שם תמיד על מנת לתת מענה ללקוח (יחידת קצה), יחידת הקצה שאמורה לשמש עצמי כשרת יכולה להתנתק מהרשת וכן צריך להמשיך בחיפוש אחר "השרת" שייתן את השירות.

אם כתובות ה-IP של הלקוח היא דינמית לכן הגישה אליו לא תמיד תתבצע בכתבות קבועה, לעומת זאת כתובות IP שלו קבועה.

ניהול של רשת צו הוא מסובך, להיות ואם לקווי אחד מחיבור ללקוח אחר דרך לקווי צד ג' (שימוש כשרת) ולהקווי צד ג' יתנתק, החיבור בין 2 הלקחות האחרים ינותק. לכן בעת חיבור בין 2 לקוחות בדרך כלל מփשים עוד כמה לקוחות צד ג' על מנת שיהיה גיבוי במקרה אחד יתנתק. (ex skype משתמשים בReLUIN זה).

Process communication

-

כאשר 2 אפליקציות רוצות לדבר אחת עם השנייה באותו המחשב, מערכת הפעלה משחקת את תפקיד השירות. ככלומר היא מעבירה מידע בין ה-client process ל-server process.

Socket - הממשק שאחראי על שליחת ההודעות בין אפליקציות (כמו קובץ), הוצאה שבה מבקשים ממערכת הפעלה לשולח הודעה ליחידת הקצה האחת.

Addressing processes

לכל יחידת קצה אחרת יש כתובת IP בעל 32 ביט (לא כתובות חד חד ערכית בית ושם יותר יחידות קצה מקומיות IP אפשריות). כמשמעותם לגשת לאפליקציה ביחידת קצה מסוימת, לא מספיק לציין רק את כתובת ה-IP של יחידת הקצה, אלא גם לציין לאיזו אפליקציה לגשת. בשביל זה יש מזהה בשם פורט – Port לכל אפליקציה רשות.

כשפותחים socket לשילוח נתונים צריך לציין איזה סוג תקשורת רצים (UDP/TCP), כתובת ה-IP של הלוקוט (client side), מספר פורט של האפליקציה שלו נתון ע"י מערכת הפעלה, כתובת ה-IP ביחידה אליה רוצים לתקשר, ומספר הפורט של אותה אפליקציה איתה רוצים לתקשר.

הגדרת פרוטוקול של שכבת האפליקציה App-Layer Protocol Defines

- סוגי ההודעות שאפשר לשולח מאפליקציה לאפליקציה
- תחביר הודעה
- המשמעות של הודעה (סמנטיקה) – שימושות כל שדה בהודעה
- חוקים – מתי אפשר לשולח הודעה, מתי להשתמש בפקודות, איך משתמשים בהם.

פרוטוקולים פתוחים – פרוטוקוליםם פתוחים הם פרוטוקולים שכולם יכולים להשתמש בהם והם פתוחים לציבור. הקושי הוא לעדכן אותו לטובה כיוון שיש הרבה משתמשים לפרוטוקול, ולעדכן שהוא שימושי בין הרבה לקוחות זה קשה ולוקח הזמן.

פרוטוקולים ייחודיים – לא הרבה משתמשים, ניתן לעדכן כיוון שהוא פרוטוקול פרטי וכל מיש משתמש בו צריך להישאר מעודכן כי התלות היא בחברה שהפרוטוקול שלו לא תלוי בכמות המשתמשים.

אילו שירותים שכבת האפליקציה צריכה מהשכבה שתحتיה, שכבת ה-? Transport

חלק מהאפליקציות דורשות אמינות – ככלומר ש-100% מהמידע יעבור לייעד. לעומת זאת קיימות אפליקציות שיכולים לסבול הפסדי מידע בתעבורה.

Timing – ככלומר שהמידע יעבור מהר לייעד. גם פה חלק מהאפליקציות יודעות לספוג איחור בהעברת המידע.

Throughput – כמה מידע ליחידת זמן מתאפשר. (לדוגמא בשיחת וידאו צריך MB 1 לשניה כדי לראות כמו שצרי)

לעומת זאת הורדת קובץ הזמן בהפרש קטנים לא משפיע.

מהצרכים האלו נובעים 2 הפרוטוקולים : TCP/UDP

Internet Transport Protocol Servers

UDP- שירות לא אמין בכל הדריכים של התעבורה, לעומת זאת העולות שלו נמוכה, כלומר אין הבטחה עצה שהميدע יגיע ליעדו בזמן וכו'.

TCP- אמינות גבוהה בין שליחה לקבלת המידע, סדר שליחת המידע והצורה שבו הוא נשלח. שני הפרוטוקולים הללו אינם מצלינים את המידע שMOVVER בערatan.

SSL- שירות הצפנה שניתנת על ידי מערכת הפעלה למידע המועבר, בעלות של זמן וגודל המידע. דרך הצפנה יחסית שלשה.

HTTP (Hyper Text Transfer Protocol)

פרוטוקול שימושי אפליקציות שרצות לפנות לשרת. פותח במטרה להעביר עמודי web ממוקם למקום.

hyper text markup language -HTML

uniform resource locator -URL

Hyper text -tekst שבתוכו ישנו קישור למקומות אחרים.

דרך פרוטוקול-HTTP מתאפשרת גישה לדפי web, כלומר הבקשה לדף מסוים נשלחת מהלקוח (client) אל השרת (server), והדף מוצג ביחידת הקצה. אחריות הפרוטוקול היא שליחת אובייקטים.

ישנם מצבים שמתבצע זיהוי מסוימת דפדף נשלחה הבקשה והתשובה לבניית בזורה דינמית בהתאם לדפדף הלוקוח. לעיתים גם יש תכניות מוכנות לכמה סוגים דפפניים ואם הלוקוח משתמש לדפדף לא בניין נשלחת ברירת מחדל שלשה.

הגישה לשרת ה-HTTP מתבצעת על ידי תקשורת TCP לפורט 80 בשרת. לאחר שהлокוח מקבל את התשובה לדרישה שלו חיבור ה-TCP נסגר. בשרת לא נשמר מידע לגבי בקשות של הלוקוחות. (*HTTP is stateless*) כלומר הפרוטוקול הזה לא תוכנן לאחסן מידע לגבי בקשות קודמות.

חיבור HTTP Persistent – ניתן לשלוח מספר אובייקטים בחיבור TCP יחיד בתקשרות בין לקוח לשרת. לעומת זאת חיבור HTTP-persistent, שם לאחר כל חיבור ה-TCP נסגר ככלomer להעברת מספר אובייקטים נדרשות מספר חיבורים.

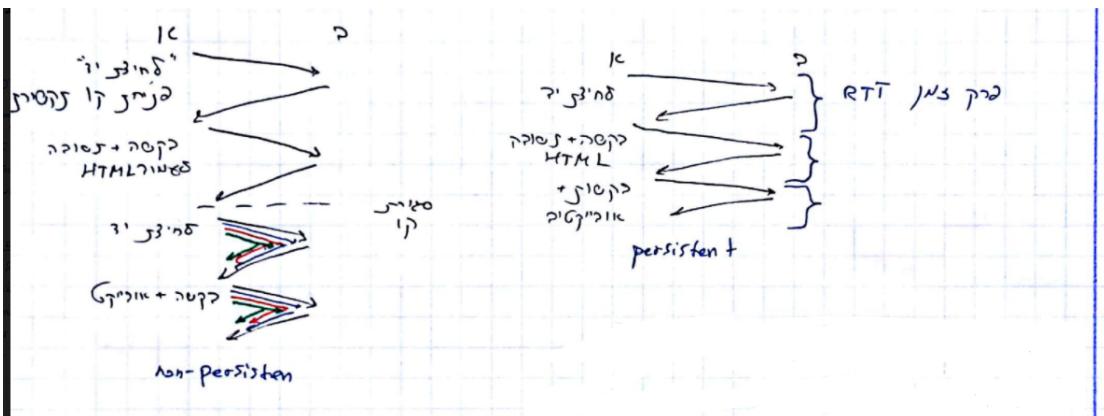
בגישה הלא עקבית לקבלת כל אובייקט, יש לפתוח תהליך בקשה מחדש (לחיצת יד->בקשה->קבלת) היota והתהליך של לחיצת היד. כלומר תחילת התקשרות מול השרת, לijken יחסית הרבה זמן, והגישה הזאת בה השרת שואף לסיים תקשורת עם הלוקוח כמה שייתר מהר על מנת "להיפטר" ללקוח אחד ולעבור לטיפול של הבקשה הבאה.

כדי להתמודד עם מצב זה יש את הגישה העקבית, היota ולא ניתן ב프וטוקול ה-HTTP לבקש כמה אובייקטים בהתא אחד, מקבלים את הבקשות, כלומר שלוחים את הבקשות לפתחות קו התקשרות עבור כל אובייקט בהפרש זמן מסוימים (ככיו לפתיחת אותם בו זמן מסוימת במקביל). היota וכל שרת יודע לטפל במקביל בכמה בקשה ולא אכפת לו אם הבקשות הן מאותו לקוח או ללקוחות שונים. לפני הבאת האובייקטים יש להפעיל קודם את עמוד ה-HTML ורך לאחר מכן להביא את מספר האובייקטים הנדרש. ברור כי בגישה הלא עקבית ייקח פרק זמן

(round trip travel time – RTT)

מעבר הבאת עמוד ה-HTML ועוד אותו פרק זמן כפול מספר האובייקטים הנדרש.

בגישה הלא עקבית אחרי שקיבלנו את HTML, יש לפתח בקשה חדשה כדי לקבל את האובייקטים הנדרשים, שכן נדרש פעמיים התהילך של "לחיצת יד". בפעם הראשונה בשביב עמוד HTML (ואז קו תקשורת נסגר) ובפעם השנייה עברו כל אובייקט (במקביל, וכן ניתן להניח כי הזמן עברו פתוח קו עברו כל אובייקט הוא זניח). בגישה עקבית (Persistent), נשלח בקשה _לחיצת יד_) לפתיחת קו התקשורת ולאחר שהתקבל עמוד HTML (בפרק זמן RTT), הקו לא נסגר עד קבלת כל האובייקטים, וכך נשמר בשלב "לחיצת יד" נוסף עברו פתיחת קו לקבלת אובייקט.



בקו תקשורת שעבוד בגישה העקבית *shout time*, ככלمر לאחר זמן מסוים הוא יסגר, גם הוא יהיה בשימוש.

פרק הזמן RTT - הזמן שלוקח להודעה לצאת מהלקוח עד השרת והדרך חזרה, לרוב ההודעות הן קטנות ("לחיצת יד"), הבאת HTML, אובייקטים קטנים) لكن כולם תהליך ייקח זמן RTT מסוים מהתחלה עד הסוף. עבור אובייקטים גדולים, ייקח זמן RTT לביט הראשון של האובייקט להגיע לשרת מהלקוח, וכן הביט הבא אחריו יגיע בזמן transmission אחרי הביט שלו, ככלmr עבור אובייקט גדול ייקח לו:

$$RTT + \text{transmission delay} = \text{number of bits}$$

HTTP Request Method

- **שנム 2 שיטות להעביר מידע לדפסון:**
 - שיטה של העלאת מידע שהוא כמעט לא שימושית הדפסנים לא תומכים בה.
 - הבקשת געשית ע"י כתיבת שורת-h-URL אשר מפרטת לאיזה כתובות רצחים לגשת, מה לעשות באורך העמוד, עם איזו פונקציה ליצור את הדף וכו'.
- **GET** - מיועדת לקבלת אובייקט שנמצא על השרת, בכתבתו שנייתנה בתחילת ההודעה. בקשות GET הן הנפוצות ביותר ברשת האינטרנט.
- **HEAD** - מבקשת מהשרת לשלוח את כל שדות הכותרת שהוא נשלחם לבקשת GET, אך בלי האובייקט עצמו. השיטה נועדה, בין השאר, לאפשר לבדוקה של קישורים שבורים או זמני שינויים של אובייקטים מבלי לבקש את כל האובייקט.
- **PUT** - מבקשת מהשרת לשמור את גופו ההודעה המצורף לבקשת בטור אובייקט, שכתובתו היא הכתובת שנייתנה בתחילת הבקשתה.

- DELETE - מבקשת מהשרת למחוק את האובייקט שכתובתו מצוינת בתחילת הבקשה.

- 1.1 HTTP – עובד בשיטת **Resistant**, וכתיבת הבקשה נדרש לציין כמה זמן **server** ישמר נתונים עבור הליקון (Keep-Online).

HTTP Response Message

שורט התשובה של הפרטוקול כולל: סוג הפרטוקול, מספר ASCII ותשובה בתור מחוץ (Status line). After the response there is a header line (Head-line).

User-Server state: Cookies

כל דף שנוצר עבודה עם cookie file שומר ב-file cookie במחשב של הליקון פרטימ על האתר. ה- cookie file הוא מסד נתונים ששומר את שם האתר ומספר מזהה של הליקון. כאשר האתר מזיהה שהיקון ניגש לאתר וליקון אין מספר מזהה הוא מעניק לו מספר זיהוי חדש, תחת מספר זהה ישמרו הפרטים על הליקון. הסיכון בשימרת cookies הוא אוסף מידע ע"י אתרים מכירוט.

FTP: file transfer protocol

פרטוקול זה נועד להעביר קבצים בין הליקון לשרת ובין שרתים.

- כאשר מבקשים מספר קבצים, כדי שקובץ גדול לא "יתקע" את שאר הקבצים, אז בשיטת **-חוט persistence**, יפתח השירות כמה קווים במקביל ולאחר העברת הקבצים, כל קו יסגר בהתאם. FTP - הוא לא State Less Control traffic (Port 21) וקיים אחרים להעברת הקבצים פיזית בקו non-data-, persistence traffic (Port 20).
- עיקרי רוחב הפס משמש להעברת קבצים. קו נסוף שעוקף את קו data הוא קו **data persistence**. שהוא נועד לפקודות וביקשות, בקו זהה (Port 21) ניגש הליקון לשרת FTP באמצעות קו זה חוסך את בדיקת הקו ב-data-persistence, אלא מגיע לשירות באמצעות קו נוסף. גישה זו של קו אחד לפקודות וקו נוסף לקבצים נקרא "**out of band**".
- בעצם זה קו בקרה שנוטן לשות תקשורת עם קו RTP, ברגע שהשירות קיבל פקודה של העברת קובץ הוא יפתח קו data וישלח לו את הקובץ ויסגור את הקו. גישה שעובדת ללא קו בקרה הנוסף נקראת: "**on band**".

Electronic mail – SMTP (simple mail transfer protocol)

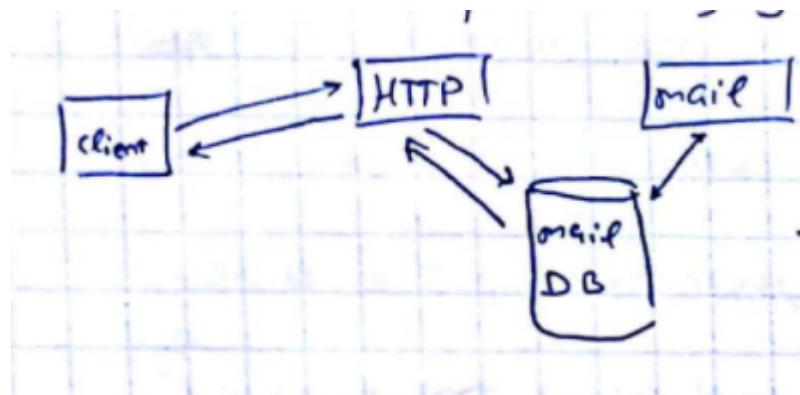
שליקון שלוח מייל הוא נשלח ל-mail server שלו הליקון עובד. משם (לאחר שהשרת העביר את ההודעות שהיו בתור לפני המייל הנוכחי), המייל נשלח אל השירות המייל של הצד השני (הליקון שהשני שלו נכתב הדוא"ל) ואז כשהמקבל יתחבר לשרת הדוא"ל שלו הוא יראה את המייל.

- SMTP – פרוטוקול שמשמש לדיוור בין שירותי מייל.
- Port 25 – הוא הפורט בו יושב SMTP בTCP.
- פרוטוקול זה מורכב מ 3 שלבים (1) handshake, (2) שליחת הודעה, (3) סגירת קו.
- הודעות מייל בSMTP מועברות כשלילים (ASCII) ולכן נדרש לתרגם מ 7 ביטים לע"מ Unicode וחזרה, על מנת לקרוא את הודעת המייל.

- ב-SMTP קיימים תוספות, תלויים (MIME) על מנת לשפר ולהוסיף אופציית לתרגם של טקסט, קובץ, תמונה וכו'. ושלילה של קובץ במיל' צריכה להיות MIME ידוע מראש כאשר הדפסן של הצד מקבל משתמש בדיק באותו MIME לקידוד.

ישנו 2 פרוטוקולים לקליטת מייל:

- POP 3 (Post Office Protocol) – תומך במשיכת הודעה ומחיקתה, הפרוטוקול לא מוצפן. בנוסף, הפרוטוקול זהה לא תומך בתיקיות, כלומר כאשר תיבת הדואר נפתחה במקום אחר היה צריך להוריד את כל המail'ים ברשימתו. הוא שומר את כל ההודעות בצד הלוקוח כשר-3 POP מושתמש ב-download-and-delete.
 - IMAP – שומר את כל המail'ים בשרת, תומך בתיקיות ושומר late של הלוקוח: שמות תיקיות, מספר זיהוי של ההודעות וכו'.
- כיום תקשורת בין שולחן-מקבל הודעה נעשית: מהלוקוח נשלחת הודעה לשרת HTTP ושרת זה ההודעה נשלחת למסند נתונים של שירות SMTP אשר מעדכן אם נכנסה הודעה או נלקחה הודעה. لكن משיכת הודעה יכולה להתבצע ע"י HTTP או IMAP או POP3 כשר ניגשים לתיבת הדואר דרך mail.web.
- תקשרות בין שירותי המייל תבוצע תמיד דרך SMTP, אלא אם 2 השירותים הם של אותו ה"בעל".
(GOOGLE)



כלומר, HTTP יכול להחליף את SMTP בשלב של שליחת הודעת המיל' מהלוקוח אל שירות המייל, ושלב קבלת ההודעה (המידה ומשתמשים בmail.web) במקום POP3 או IMAP. אבל בתקשרות בין שירותי המייל ע"י SMTP דרוש.

Web Caches (Proxy Server)

שירות proxy הוא בעצם שירות בין הלקוח לבין שירות הראשי אליו הלקוח פונה. במקרה זה מידע (אתר, תמונת וכו') נשמרם בשירות proxy על מנת ליעיל ולזרז את תהליך העברת וקבלת נתונים. כך שלמעשה יכול המידע שהלקוח בิกש מאთר מסוים ייקח פחות זמן להביא, במידה ואותו מידע שמור על proxy. כמובן שמשתמש בשרת proxy, מבחינת פניות אליו, ימחק מהשרת ויתפנה הזיכרונו. היוות שיש אובייקטים שמתוחדים ממשתנים שירות proxy צריך לדעת להתחעדן על מנת שתתערבורה של המידע תהיה יעילה. רוב האובייקטים שודיעו שהם עתידיים להתחעדן בשלב מסוים,នושאים אותם מידע שיכול את זמן העדכן וכן שרת proxy יכול לעדכן את המידע כך שהיא זמין ללקוח. היוות ששרת proxy חוסף פניות של לקוחות ישירות לשירות המידע זה לטובתם של שירותי המידע לשЛОוח מידע לגבי עדכן האובייקט.

האסטרטגיה השנייה של עדכן המידע היא get conditional. במידה ולא ידוע متى האובייקט יתעדכן, שירות proxy יגיש לשרת וישאל אותו אם אותו עודד מבודק השתנה. במידה וכן proxy יעדכן את המידע הקיים וישלח אותו ללקוח, במידה ולא proxy יתנו ללקוח את המידע הקיים עצמו. מכיוון שההודעה של כן או לא יותר מהירה מהעברת כל הנתונים המבוקשים.

שירות proxy ישלח בקשה (GET) לקבלת העמוד בתוספת שאלה: <date>: if-modified-since: אם עדכן אז יקבל proxy את העמוד החדש אחרית יקבל שהמידע עדין בתוקף: "304 Not Modified".

DNS- domain name system

(*שאיפית 62-2)

תפקיד מערכת DNS הוא לתרגם את שמות הכתובות של אתרים האינטרנט (URL עד ה-"/' הראשון בכתובת), לRFC ה-IP של אותו אתר (שרת).

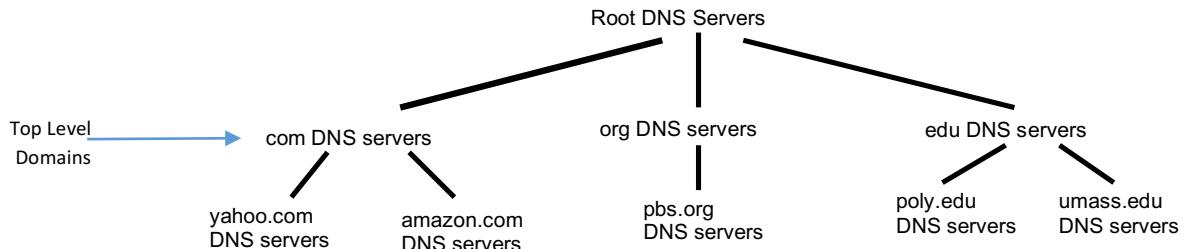
הweeneyון של שמיית השמות של האתרים לעומת כתובות ה-IP שלהם מביא אותו ישר מחשבה על טבלה כלשהי (hash-table) שמקשר יישור בין מספר לבן שם האתר, אך היה יש המונ שמות של אתרים קשה לשומר את כל השמות בשרת כלשהו שידע לתת את הסנסכרון זהה, היה והגישה לטבלה כזו תבצע אלף פעמים בשנייה (בגלאל שאנשים מסביב לכל העולם מנוטים לגשת לאותו האתר באופן הרגע), וגם אם ניתן (וניתן) ליצור טבלה עצמת הגישה אליה של לקוחות מכל העולם תיצור עמוס בשרת, ועיכוביים בקבלת המידע גם בגלאל המרחק הפיזי של אותו שרף לבין הלקוחות מסביב לעולם (רווח פס של הלקוחות ועוד..). חיסרונו נוסף הוא תחזוקת השרת זהה.

שרתים עם נגישות גבוהה אליויהם (כמו גוגל), מפיצלים את השירותים שלהם תחת אותה כתובות, על מנת לא ליצר עמוס בשרת בודד. שינוי הכתובת – יעד מתבצע בעזרת מערכת DNS. העמוס על השרת העמוס ירד בזכות זה שהשרת יעד עבר להיות שרף אחר, וכך ניתן לאזן את העמוס בפניות לשרתים.

היות וכותבות האתרים מתעדכנים, משתנים עם הזמן, ולא משתלים להחזיק מסד נתונים המאגד את כל הכתובות בעולם, מפזרים את מסדי הנתונים למסדים אזוריים. כך לדוגמא לגבי אתרים ישראליים, ישמיר מסד נתונים של אתרים ישראליים בישראל, על מנת שהוא יהיה לרלוונטי לתוךן שלו וגמם הגישה לאתרם היישראליים כנראה לרוב תבוצע בישראל וכן הגישה תהיה מהירה יותר.

היררכיה של DNS:

בראש ההיררכיה יעמוד שרף DNS ROOT, ממנו יצאו פיצולים לשרתים DNS כאשר תפקידו של כל שרף זה הוא ל"הכיר" את כל האתרים בסיומת משותפת:



כך בעצם כאשר מנוטים לגשת לאתר מסוים, ניגשים קודם ל-SERVER ROOT והוא יידע להפנות את הבקשה לשרת ה-DNS המתאים. כדי לא ליצר עמוס בשרת ה-ROOT DNS , קיימים 13 שרתים ROOT DNS מפוזרים המכירים את כל ה-NODE LEVEL TOP (השרתים שמאגדים סיוומיות משותפות). חיים קיימים הרבה יותר שרתים אלו (כ-200). ברגע שגילינו שרף שמכיר את כל האתרים שמסתיימים ב-.com. נגיד, אנחנו נפנה מעכשיו רק אליו (במקרה שנדרשת גישה לאתר שנגמר ב-.com). כדי לנו פניות מיותרות ל-ROOT DNS, וכך גם לגבי שאר הסיוומיות. לרוב כל שרת ה-NODE LEVEL TOP, גם לא יודיעם להכיל את כל האתרים הקיימים באוטה הסיוומות, لكن יוצרם מין מערכת היררכית דומה לזה הראשית, אשר מפצלת בין אפשרויות הסיוומיות השונות (לדוגמא: שרף ל-.il.ac.il, שרף ל-.co.il, שרף ל-.gov וכו'). וכך תחת כל שרף ראשי יותר, ישנו שרפים מנויים וככל שיורדים בשרתים המשניים ניגשים לשרתים שיודעים לנשל מידע ספציפי יותר. כך למשל שרוצים לגשת לכתחובת המכילה : *.cs.huji.ac.il , ניגשים ל-ROOT DNS שニיגש לשרת שתחתיו שמכיר את הכתובות שמסתיימות ב-.il, שרף זה יגיש לשרת שתחתיו שטוף בסיוומות: il.ac.il , והלאה ייגש לשרת של cs.huji.ac.il ולבסוף לשרת הספציפי של il.cs.huji.ac.il

היות והרבה גישות מסוימות איזור נעשות לאוטומטית, הקיימו שרת DNS Local Name Server (LNS). שתפקידו לשמר את השאלות אשר יוצאות מהלוקה (HOST), על מנת לגשת לאתר מסוים. כמובן הוא מין שרת פרוקסி של ספק האינטרנט, אשר מכיל את כתובות האתרים אל מול כתובות IP שלהם, וכך בעצם נמנע הצורך לגשת כל פעם מחדש למספר האינטרנט שלנו, על מנת שילך ויחפש את אותה הכתובת שביקשנו. אם שרת-h-LNS מכיר באותו אתר שביקשנו לקבל, הוא יחזיר לנו את הכתובת IP של אותו האתר (בצורת דף אינטרנט). אם הוא לא מכיר את השם של האתר הוא ירך ויחפש את הכתובת בשרתים המתאים (לפי היררכיה המתואמת לעמלה). על מנת לאזן בעומס של הפניות לשרת-h-LNS ישנו שרת ראשי ושרת משנה, שבעת עומס על השרת הראשי, נובע דרך השרת המשני.

כדי לקבל תשובה, שרת-h-LNS פונה לשרתים DNS בכמה דרכים:

- I. **פניה איטרטיבית:** שרת-h-LNS פונה לשרת אחד DNS אחד, האם הוא מכיר את הכתובת. שיטת הגישה הזאת מודול נוחה לשרתים אליו פונה שרת-h-LNS, כיון שהוא מקבל שאלתה ומחזיר תשובה מיידית (בין אם זה הכתובת עצמה או בין אם זה שם השרת עליי ציר לגשת).
- II. **פניה רקורסיבית:** שרת-h-LNS פונה לשרת DNS כלשהו, אם אותו שרת לא מכיר את הכתובת הוא יפנה לשרת DNS אחר, וכך עד שתתקבל תשובה לגבי האתר. התשובה תחזור לשרת המבקש עד ששרת-h-LNS יקבל תשובה לגבי הבקשה שלו ויחזר אותה להלוקה. בגישה זו כל שרת אליו פנו, אם הוא לא מחייב בכתובת, מנסה לתשובה מהשרת הבא שהוא פנה אליו וגם צריך להחזיר אותה לשרת הפונה אליו.

שיטת הפניה נקבעת עפ"י המרחק בין הפונה (הലוקה) לבין מיקום השרת, כאמור אם לקוח מישראל רוצה לגשת לשרת שנמצא בתל אביב, אין הבדל בין 2 השיטות לכארה, היות והבדלי המרחקים הם קטנים. לעומת זאת אם לקוח מחול' מנסה לגשת לשרת בתל אביב, עדיף שהשרותים יעבדו בשיטה רקורסיבית, היות והשרותים (בישראל) מדברים בין עצם בקרה רקורסיבית עד שיקבלו תשובה ויחזרו אותה להלוקה, וכך בעצם הלוקה פנה פעמי אחד לשרת ישראל' ומנסה עד שהשרותים בישראל ידבו אחד עם השני עד לקבלת הכתובת. אחרת אם הלוקה היה מחפש את הכתובת בצורה איטרטיבית, הוא היה מעמיס על הקו הבינ'ל של ישראל היות והוא נדרש לגשת כל פעם לשרת שונה בישראל עד שיגיע לעד'ו.

DNS: caching, updating records

כאשר נסופים דומיינים, שירותי חדשים, נדרש לעדכן זאת בשרת DNS הרלוונטי. כמובן צריך לגשת לשרת DNS ולעדכן את המודד נתונים שלו (אחרי שהתקבל אישור לפתח הדומיין). כך גם כאשר אתרים נסגרים. לכל רשומה כזאת של כתובות ישנו תוקף, עד מתי הוא יהיה תקין ונitin' יהיה לאחזר כתובות IP לשם האתר המסוים. הבעייה נוצרת כאשר משנים כתובות IP של אתר מסוים לפחות סוף התוקף שלו, כך שהכתובת הזאת שומרה בשרת DNS הרלוונטיים, וכשייה נסיוון גישה לאתר, לא ניתן יהיה לאתר אותו. כדי לפתור את הבעיה הזאת ישנו מגנון אשר דואג לעדכן את כל מי שפנה לאתר הספציפי, בפניה הbara'ה שלו, שהכתובת שונתה, בנוסף מתן הכתובת החדשה. על מנת ששרת אחד לא יצטרך לפנות לאלפי לקוחות שונים שניגשו אליו וקיבלו התchieיות על התוקף, השרת יפנה לשרת DNS וهم יdaggo לשנות איפה שchar, לפי היררכיה של שרת DNS.

DNS records

RR format: (name, value, type, ttl)

בעת פניה לשרת DNS (כasher כותבים את השם של האתר בשדה ה-URL בדף), מעבירים בעצם שאלתה, כל שאלתה מכילה 4 שדות, בעצם פונים קודם לשרת-h-LNS ורק הוא יפנה לשרת DNS שימושי.

TTL – (time to live) עד מתי התשובה שמתقبلת בתוקף. לדוגמה שירותי אשר צפויים להיות עומסים, מאזורים את העומסים בכר שהם מפנים את הלוקחות לשרתים אחרים של אותה הכתובת, והttl מגדיר לכמה זמן תקפה ההפניה הזאת.

TYPE – סוג השאלה/תשובה.

TYPE=A: כאשר פונים לשרת שלוחים לו את השם של השירות: NAME, ואם שרת DNS יודע מה הכתובת IP של השירות המבוקש יחזיר שרת ה-DNS את שם השירות בשדה ה-NAME, את כתובות ה-IP בערך ה-VALUE, סוג התשובה בערך ה-TYPE (במקורה זהה A) וכמה זמן תקפה הכתובת הזאת לפני ה-TTL.

NS=TYPE: כאשר שרת DNS לא יודע את כתובות ה-IP של השירות המבוקש, הוא מחייב תשובה מסוג NS, כאשר בשדה ה-NAME יהיה מצוין השם של האטר המבוקש, ובשדה ה-VALUE יהיה השירות DNS שלו מומלץ לפנות, כי יכול להיות שהוא מכיר את כתובות ה-IP המבוקשת.

TYPE=CNAME: כאשר שרת ה-DNS מזיהה שמנסים לגשת לשרת מסוים אשר יש לו שירותים אזוריים (נגיד שרת IBM אשר לו יש שירות שומרת את מדינת ישראל בעברית), لكن תחזיר תשובה מסוג CNAME אשר תמליץ לחפש את השירות המקומי, כאשר בשדה ה-NAME יופיע השם של השירות האיזורי המתאים, ובשדה ה-VALUE יופיע השם המקורי – האמתי של אותו השירות הראשי .

TYPE=MX: בסוג שאלות זה משתמשים בהקשר של דוא"ל. כאשר רוצים לשלוח מייל, פונים אל שירות DNS בשאלה , מי משרת את שירות המייל שלו אני רוצה לשלוח את המייל. תשובה מהסוג הזה תחזיר את שם השירות המבוקש בשדה ה-VALUE .

- כל שירות (של חברה/מודד וכו') חייב להחזיק תשבות מסוג A וטסוג NS. ככה כאשר ירצו לפנות לכתובת של האטר הספציפי זהה, תשובה מסוג A תדע להחזיר את כתובות ה-IP של אותו השירות, ותשובה מסוג NS תדע להפנות לשרת ה-DNS המתאים שיודע להחזיר את כתובות ה-IP של השירות המבוקש. אם אותה חברה רוצה גם להחזיק כתובות מייל, אז היא צריכה להחזיק תשובה מסוג MX שתודיע לנתן את השירות מייל הנדרש.

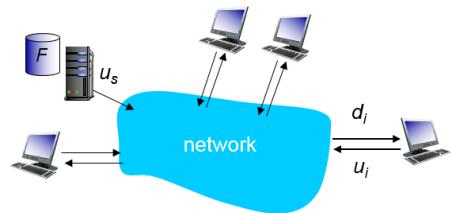
Attacking DNS

אחד הדרכים לתקוף שירות DNS היא שיטת **DDOS**, ככלומר שליחת הרבה הודעות באותו זמן לשרת ה-DNS, דבר שיגרום עומס על השירות ויגרום האטה/ של פועלות השירות. אך גישות לחלק מהשירותים שנעים דרך שירות ה-DNS הספציפי שמוטקף יהיו בלי אפשרויות, היוות ולא ניתן לתרגם את שם האתר לכתובת IP.

דרך נוספת היא, להיות ולמערכות ה- DNS יש אפשרות עדכון, לאתרם החדש שקיימים, אך אם מפברקים היוצרים של אתר חדש, כדי לעדכן את השירות ה-DNS הרלוונטיים יש לשלוח הרבה מאוד מידע ברחבי השירות, לכן זה יכול ליצור עומסים בקוו הרשות בעולם.

P2P architecture

שליחת קובץ בשיטת שרת-לקוקו לעומת P2P :
 נגיד ונרצה להפיץ קובץ בגודל F מהשרת אל N יחידות קצה (לקוקות). נדרש לשלוח את הקובץ N פעמים מהשרת אל כל לקוחות פרטיים. ככלומר:



$$time \text{ to } distribute F \\ to N \text{ clients using} \\ client-server \text{ approach} \quad D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

כאשר u_s זה רוחב הפס של השירות. d_{min} יכול הורדיה של הלוקו.

לעומת שיטת P2P, מעלים לפחות קובץ אחד לרשות המשרת, ולפחות ללקוח אחד חייב להוריד את הקובץ במלואו. היות וארכיטקטורה זו תומכת בשיתוף קבצים וכoch העיבוד של כל יחידות הקצה המחויבורת לאוטו השירות, אז ככל שייתר ללקוחות יתחברו, יגדל רוחב הפס של ההעלאת הקובץ, ככלומר כל לקוח יכול לתרום הعلاה של חלק מהקובץ לטובת לקוח אחר שרוצה להוריד את אותו הקובץ. כך בעצם מתחלק העומס המשרת שהיא נדרשת לשלוח לכל לקוח את הקובץ. כאשר לקוח רוצה להוריד קובץ הוא ניגש לכמה יחידות קצת כדי להוריד חלקים מהקובץ ובוסף משלים את המידע.

פרק 3 Transport Layer -3

Transport services and protocols

- הودעה בשכבה התעבורה נקראת סגמנט – segment .

שכבה התעבורה – transport – היא שכבה לחיבור בין האפליקציות בשני ייחידות קצה. מתפקידיה לזרזות את התוכניות השולחות ואת התוכניות המקבלות את ההודעה. שכבה הרשות – network – היא שכבה המעביר את ההודעה בין שני ייחידות הקצה. על שכבה ה-TRANSPORT להזמין איזו מבין התוכניות הרצויות על המחשב ('יחידת הקצה') צריכה לקבל את ההודעה הספציפית. זאת היא עשויה בעזרת פורטים שונים, מספר הפוטט מתබל בהודעה מהשולח. בנוסף לשורותים (או אפליקציות למיניהם) אשר ממתינים לבקשתם בפורטים מסוימים, יפנו למערכת הפעלה וידעו אותה פורט הם ממתינים. פרוטוקולי שכבה זו עובדים רק ביחסות הקצה.

בצד השולח – שכבה זו חותכת את המידע לחטיכות קטנות בגודל אופטימלי לשילחה, החתיכות הקטנות הללו נקראות סגמנטים – segments . ואולם היא מעבירה לשכבה התקשרות. בצד מקבל כל אותן הסגמנטים מחוברים זהה להודעה המקוריית בעזרת ביטים של מידע שבחובים בכל סיגמנט.

פרוטוקולי שכבה ה-TRANSPORT

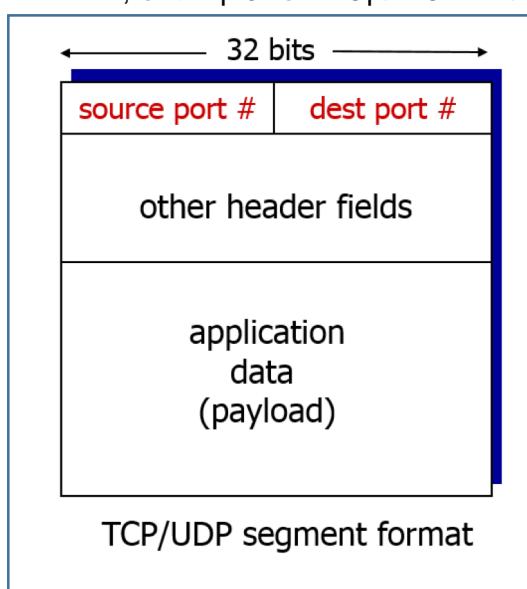
TCP – פרוטוקול זה מבטיח אמינות על תוכן המידע, ועל זה שהמידע באמת הגיע אליו. כמובן הוא יודע לזרות שבוש בהודעות. בפרוטוקול זה יש מערכת תשובות, הכולרת על כל הגעת מידע לעד התקבל הודעה לצד השולח, עד שהודעה זו לא נשלחה (במידה והמידע הולך לאיבוד בדרך), ישלח המידע שוב . וגם תהיה תשובה על כך שההודעה התקבלה תקינה. יתרון נוסף הוא שMRIה על סדר ההודעות, הכולרת בסדר שבו נשלחו ההודעות על אותו קו TCP כך הן תתקבלנה. (לעומת זאת אם נשלחו מסרף הודעות על קו TCP שניים כמו בגישה ה-PERSISTENT אין קשר בין חבילות המידע בקישורים השונים) .

מנגנון נוסף שפרוטוקול זה מכיל הוא ניתוח עומס על הרשות, הכולרת ברגע שיש עומס על הרשות , השילוח תהיה איטית יותר על מנת להקטין את העומסים וכדי להקטין את ה- queue delay . ישנה בקרה נוספת הנקראת control flow , במקורה והצד מקבל לא יודע להתמודד עם כמות גודלה של נתונים ברגע נתון, הוא מודיע על כך ושילוח הנתונים נדרש להיות בקצב מואט, על מנת הצד מקבל לא יצוף במידע.

הפרוטוקול ה-TCP ישנה בקרה – connection setup , של החלטת יד, לפני שפותחים קו תקשורת בין 2 הצדדים, בודקים את העומסים הל הקו, רמת ההצפה ועוד פרמטרים שיהיו רלוונטיים לשילוח וקבלת תקינה של המידע .

UDP – בפרוטוקול זה אין אמינות הן שכלל התוכן הגיע לידי, והן שהוא בכלל הגיע או בסדר הנכון. כל מה שהוא יודע לתת זה לזרות את התהילה/האפליקציה המסוימת אתה נדרש לתקן. בפרוטוקול UDP ,

נשלחת ההודעה (לא תיאום קו מראש כמוTCP), עם הפורטים הנדרשים. זה נעשה באמצעות SOCKET , שהוא בעצם הממשק בין שכבה האפליקציית ובין שכבה ה- TRANSPORT . כמובן עירקון הפעולה הוא בקשה-תשובה, ורק לאחר מכן מתפנים לבקשת הבאה. لكن הייעוד שלו הוא עבר דיאלוג קצר .



MULTIPLEXING/DEMULITPLEXING

– מזהה מייזו יחידת קצה ההודעה יצאה.
Multiplexing – מזהה לאיזו יחידת קצה ההודעה אמורה להיות מועברת.
Demultiplexing בפועל בכל הודעה שנשלחת ישנו זיהוי מי השולחomi והמקבל (בשכבה הרשות – network), וגם לאיזה פורט פונת בצד המקלט.

פרוטוקול TCP נבנה מראש לשכבה ה-TRANSPORT ומיעוד לעבוד מעל שכבת ה-IP (שכבה ה-network),
בגלל זה נהגים לקרוא לזה ip/tcp .

Connection-oriented demux

כאשר תהליך כלשהו רץ (בשכבה האפליקציה) הוא מודיע לשכבה ה-transport שהוא פעיל בפורט מסוים. לכן כשייחידת קצה מסוימת רוצה לגשת לפורט של אותו התהליך שרצ בשרת, הוא שולח בקשה לתקשרות לאותו הפורט. אבל רוב הסיכויים שאותה האפליקציה/תהליך באמצעותו עובדה, لكن משכפלים את אותו התהליך אשר יטפל באורה בקשה, ויש תהליך נוסף שממתין לפניה לאותו הפורט. אם תהיה עוד פניה אז המערכת תפתח תהליך חדש שייטפל בבקשתו וርח חיללה. כמובן יש לתהליך אחד שמטרתו לנtab את הבקשתה (המאזין לביקול) לתהליך שמטפל בבקשתה (וכאלו יכולם להיות כמה, תליי כמה בקשות יש לאוטו פורט) כמובן נוצרים העתקים של אותו התהליך למטרות שונות. וכך גם אם תורן כדי הבקשתה נדרש לפנות ליחידת קצה אחת ישוכפל אותו התהליך שייטפל בתקשרות מול יחידה אחרת. כאשר התקשרות נגמרה (ה-SOCKET נסגר) רוב הסיכויים התהליך גם ייסגר.

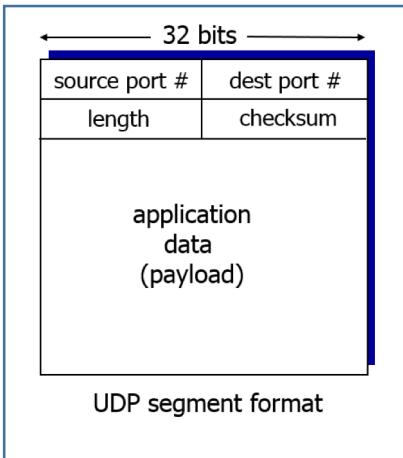
היות ופתיחה של התהליך (process) חדש דורש הרבה אמצעים, באותו התהליך יפותחו כמה threads שיעשו שימוש בבדיקה את המתואר לעיל. כמובן כל פעם שתהיה התקשרות חדשה לאותו הפורט, יהיה thread שייטפל בתהליך אחד, בעת התקשרות נוספת thread נוסף ייגש לטפל בבקשתה.

UDP: user datagram protocol

סגןנט – חתיכת מידע נוספת באנוף ל-header שיפורט את ומספר הפורט של השולח והמקבל.
כמו שכבר ציינו ב프וטוקול UDP אין אמינות, אין הבטחה לגבי סדר קבלת הסגמנטים, האם יגיעו בכלל ליעד והאם יגיעו תקינים.

היתרון שלו הוא שהוא connectionless כלומר אין את התהליך "לחיצת-יד" על מנת לבסס קו תקשורת בין השולח למקבל (דבר היכול ליצור עיכובים). כל סגןנט אשר נשלח ב프וטוקול זה מעובד אינדיידואלי, ללא קשר לשאר הסגמנטים. באנוף לזה header הוא מאד קצר.

משתמשים בו בעיקר ב-Streaming multimedia, אשר לא רגשים לאיבוד מידע קטנים בדרך, קצב שליחת ההודעה). באנוף אפליקציות DNS ו-SNMP משתמשים בפרוטוקול זה. כיוון שההודעות באפליקציות אלו בדרך כלל קטנות, לכן נחסר פתיחה הקו שדורש הרבה בתים.
על מנת לגשר על הפער של האמינות, נמשח את הפתרון בעיות האלו ברמת האפליקציה.



Udp segment format:

– אורך הסגמנט בבתים כולל header הוא 64 קילובייט (bit 16²), לשדה ה-length מוקצים 16 ביטים.

– בודק שההוودעה שהתקבלת היא באמת ההוודעה שנשלחה.

Checksum:

אפשר למקבל לזרות שהביטים בהוודעה שהתקבלת השתבשו, כזכור הסבירות שההוודעה המקוריית תשتبש היא נמוכה. זה נעשה על ידי חישוב פונקציה מסוימת על ההוודעה המקוריית, את החישוב זהה גם הצד המקבל יודע. להוודעה מסוימת מתאימים מספר בן 16 ביטים. ככלומר שדה checksum מכיל את אותו המספר בן 16 ביטים שמכיל את תוצאת החישוב על תוכן ההוודעה (application data). כאשר ההוודעה מתקבלת עושים חישוב בעזרתה פונקציה ידועה ומשווים את התוצאה, אם התוצאה זהה לערך ה-checksum אז ננראת שההוודעה שהתקבלת תקינה.

במצב ההפוך רצוי, מתקבל שההוודעה נתנה ערך מסוים בחישוב של הפונקציה, והוודעה אחרת לגמרי נתנה את אותה התוצאה (היות וכמות ההודעות הרבה יורט גודלה מכמות התוצאות – כמוות התוצאות היא 64kb), לכן יש מצב שההוודעה שהתקבלת היא לא ההוודעה שנשלחה, אבל הסיכוי שמצוב כזה יקרה הוא נמוך. במצב זה לא ניתן יהיה לגלות את הטעות בטוווח הביטים של 16², כדי לקחת מקדם בטיחות אפשר לחת טווח ביטים יותר גדול. (בנס"א משתמשים ב1024 ביטים למטרה זו).

במידה והצד השולח לא רוצה לעשות את חישוב הבדיקה (או שלא חשוב תוכן ההוודעה), ניתן לשולח בשדה checksum 16 אפסים. במצב זה המქבל יודע שאין צורך בבדיקה תקינות ההוודעה. במקרה לא המქבל מזיהה טעות בתוכן ההוודעה, הוא יזרוק את ההוודעה. لكن ההוודעה תעבור לאפליקציה כמו שהיא.

Connection oriented transport: TCP

בפרוטוקול זה כאמור ישנה אמינות, checksum שבודקת תקינות ההוודעה. ובנוסף אם ההוודעה "חולכת לאיבוד" ככלומר לא מגיעה ליעדה, ההוודעה נשלחת שוב (ישנה כמות מסוימת של ניסיונות חוזרים לשילוח ההוודעה, ברגע שmaguiim לכמות זו ההוודעה פשוט לא תישלח). בעובדה עם פרוטוקול זה, קיימ סדר בהצעת הסגמנטים לפי סדר שליחתם.

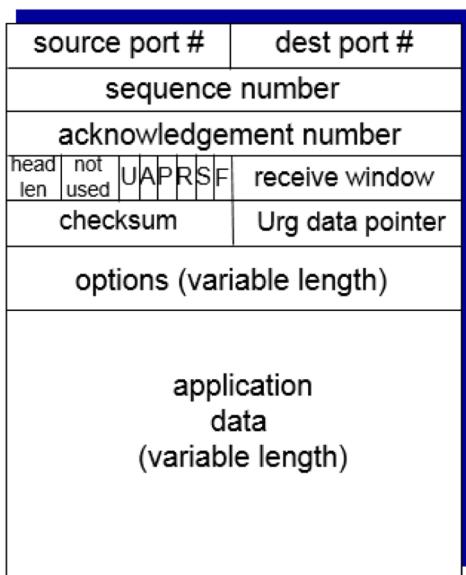
ב-TCP ניתן לקרוא byte-stream, ככלומר את ההוודעה שמתקבלת ניתן לקרוא עפ"י כמות בתים מוגדרת ללא שום קשר איך ההוודעה נשלח. (ככלומר אם כל סגמנט מכיל 200 בתים, והצד שקורא, קורא את ההוודעה כל 120 בתים, אין שום בעיה עם זה).

בהעברת הוודעות בשיטת TCP, לפני שליחת המידע מצד השולח אל היעד, נפתח קו תקשורת (hand-shake), ובניגוד ל-UDP ישנה הנחה שברגע שהמיידע נשלח קיימ קו פתוח בין 2 הצדדים, והצד המქבל "ודע" שעליינו לקבל הוודעה.

– בקרת "הצפה", מניעת מצב שבו השולח שולח יותר מידע ממנו שהצד המქבל מסוגל לקבל. דבר אשר עלול לגרום לעומס. היה ובקרים רבים הייחודיים המקבלות הם יחידות תקשורת "חלשות" כמו מחשב ברכב, חיישני מים – IoT וכו'. וכן במקרים כאלה, בזמן תהליך ה"חיצת-יד" הצד המქבל מודיע את הצד השולח בגודל המקסימלי שהוא יכול לקבל בבת אחת.

– תפקידי זהות שיש עומס על הרשת, ככלומר שההוודעות מתעכבות בתווים יותר מיד, ובuczם במקרה זהה מתבקש הצד השולח לשולח פחות מידע וכן אומנם ישלח פחות מידע בבת אחת, אבל הוא יתקבל בעיכוב נמוך יותר.

32 bits

– TCP header

Sequence number: מספר לפי הסדר של סגמנטים, היות

יכול להיות מצב שהסמנט הראשון הגיע אחרי הסגמנט השני, שכבת ה-transport צריכה לדעת להרכיב את ההודעה לפי סדר שליחתה. לכן לכל סגמנט יוצא ישנו מספר סידורי כלשהו שמהווה את מיקומו בסדר השילוח של הסגמנטים. **בד"כ מספר זה יהיה אותו מספר ממן מתחילה הסגמנט, כדי**

שתיה נקודת תחילת.

Acknowledgement number: הודעת האישור. השדה זהה

הוא חלק מנתנית פדבק על מידע שהתקבל מצד השולח מהצד שמקבל כרגע, כדי למנוע הרבה הودעות אישור בין הצד השולח לצד במקבל, שלוחים אישורים על הודעות קודמות שהתקבלו באותו הזמן. זה חוסך תעבורות מיותר בראשת של הודעות אישור. **מצין עד איזה בית התקבל אישור על כל**

שהסמנט הגיע לעדנו.

Checksum: תוצאה חישוב של פונקציית the-hsum

שמתבצעת על המידע עצמו. (אם תוצאה החישוב היא 000...16 אפסים, שלוחים 111...16 "אחדים" היות וכאשר שדה זה הוא 16 אפסים זה אומר שאין צורך לבדוק תקינות ההודעה).

Receive window: מודיע כמה זיכרון פנוי יש לצד מקבל לקבלת ההודעה, כולל מצב "צפה" (הזכיר הפני) של הצד מקבל.

Options: מילה אופציונלית, בד"כ משתמשים בה בזמן "לחיצת-יד" על מנת להגדיר את גודל ההודעה שתשלחנה. מילה זו אופציונלית לשימושים שונים של TCP. השימוש במילה זו צריך להיות "מוסכם" על שני הצדדים, השולח וה מקבל, נכון להיום המוסכמה של שימוש במילה זו, כאמור, הוא בזמן פתיחת קו התקשורת.

Head len: גודל HEADER (שדה בן 5 ביטים), כולל מספר המילים בהדר. בד"כ יופיע הערך הבינאיי של 5 (0010100), כולל ב-HEADER הסטנדרטי ישנו רק 5 מילים בהדר, בזמן "לחיצת-יד" יש 6 מילים (00110).

Not used: 5 ביטים אשר לא בשימוש כביכול. ישנו שימושים שונים בשדה זה אבל לפי הפרטוקול אין לסגור על הביטים האלה היות ולפי הפרטוקול אלו ביטים לא שימושיים.

Urg data pointer: סימון שהחלק מסוים מהסמנט הוא דחוף, כולל אותו יש להציג ראשון. החלק הדחוף תמיד יתחל בראש ההודעה, ושדה זה יסמן את סוף החלק הדחוף בהודעה מתחילה. שדה זה תקף כאשר הביט urgent הוא 1.

6 ביטים – UAPRSF

- U: מסמן שההודעה דחופה. הפרטוקול לא מגדר התיעחות ספציפית להודעה

דחופה. עובד עם Urg data pointer.

- P-push: אם הביט הזה הוא '1', זה מסמן על כך שהוא רצים לחכות לשכבת האפליקציה תמסור את ההודעה בעת הגעתה, אלא ברגע שההודעה תגיע לשכבת ה-transport ("תדחוף" בכח אל האפליקציה מקבלת).

A-acknowledge: השדה הזה מודיע האם להתייחס לשדה ה-

S-synchronize: ביט זה מייצג האם אנחנו רצים לפתח קו תקשורת (בשלב "לחיצת-יד").

בתגובה לבקשת פתיחת הקו, נשלחת הודעה מהצד מקבל, syn-ack, כולל ש הצד מקבל מאשר את פתיחת הקשר, ועל הדרך הצד מבקש לפתח קו תקשורת עם הצד השולח.

R-reset: אם ביט זה הוא '1', זה אומר שצריך לסגור את קו התקשורת. ברגע שהאפליקציה

שמיועדת לкриיאת המסר נסגרת, מערכת הפעלה מזזה את זה, והיא שולחת הודעה

למערכת הפעלה בצד השני (זה ששולח את המידע) שהקשר נזתק, כולל זאת ההודעה

האחרונה שתיהיה בקו הנוכחי ולכן צריך לסגור את קו התקשורת. ("גיטוק בפנים" – סיגרת

קשר אגרסיבית). רעיון השדה הזה במקור היה למכבים שביהם מועברים יותר מ-4 GB (

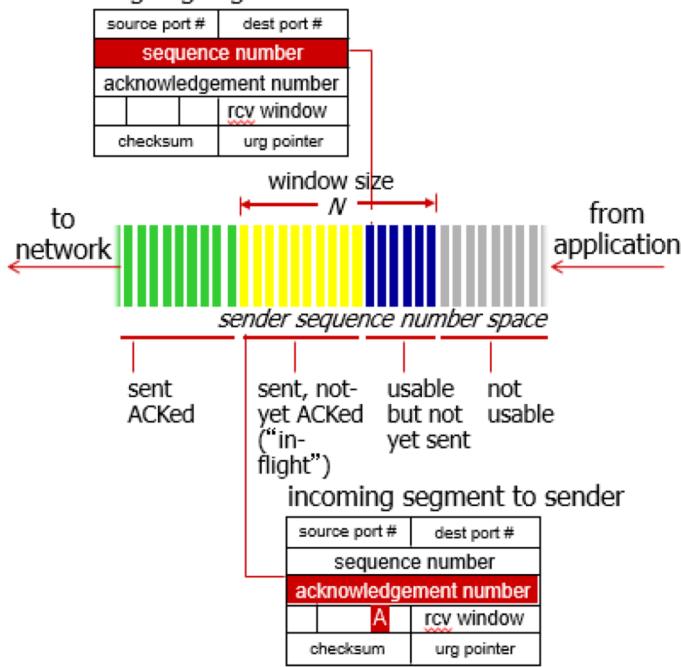
2^832 – קלומר בשדה של sequence number (יגמרו האפשרויות), לכן תפקידו של שדה זה היה להודיע על כך שנדרש לאותה מחדש מחדש את קו התקשורת, קלומר לסגור את הקשר כדי שמיד אחר כך יפתח קשר חדש.

- F-finish: מסמל את סוף הבקשות. קלומר ברגע שהצד השולח לא נדרש לשЛОוח עוד מידע, הוא שולח לצד מקבל את הביט זהה, אך הוא עדין ממתיין להודעה זהה מהצד מקבל. אם נותרו עוד הודעות שהצד מקבל מעבר לצד השולח, הוא "יקשיב" וגם "יענה" עד שתתקבל ממנה הודעה נוספת לוסף תקשורת מהצד מקבל.

Reliability Tools – מנגנוני אמינות בפרוטוקול TCP

- Checksum: בדיקת תקינות ההודעה שהתקבלה.
- Acknowledgement: הודעת אישור על קח שההודעה שנשלחה הגיעו ליעדה באופן תקין. אם לא התקבלה הודעה אישור בפרק זמן סביר (תלוי במרחק/ רוחב פס), אז נשלח שוב את ההודעה. ישנו כמה אפשרויות לבצע שליחת אישור:
 - אישור על כל הודעה שמתකבלת
 - אישור של כמה הודעות , קלומר אישור על כמה הודעות בבבאת אחת.
 - אישור על הודעה שהתקבל או לדוח גם על הודעה לא תקינה.
 - ב-TCP בחרו לשולח אישור על כמה הודעות בבבאת אחת.
- Retransmission: מתריע על איבוד הודעה בראשית, או שהאם ההודעה הגיעו עם שגיאות. קלומר אם סגמנט הגיעו ליעד עם שיבוש או לא הגיע בכלל הצד השולח ישלח את ההודעה שוב. ישנו כמה אפשרויות לבצע שליחת מחדש:
 - אם לא התקבלו אישור על כמה סגמנטים, לשולח רק את הסגמנט הראשון בתקווה שהוא זה שלא הגיע.
 - לשולח את כל הסגמנטים שעבורם לא התקבל אישור קבלה, לשולח אותם מחדש.
 - ב-TCP בחרו לשולח רק את הסגמנט הראשון מחדש.
- Timer: קובע את משך הזמן הסביר להמתין לאישור קבלת ההודעה. נקבע ע"י הזמן RTT, קלומר הזמן שלקח מאז שלחנו בקשה פתיחה קו עד שקיבלנו אישור לפתיחת הקו (syn-> syn-ack). קלומר לפניו שתשלוח ההודעה בשנית הצד השולח יחבר זמן RTT ועוד קצר לפני הערכות סטטיסטיות. הערכת זמן ההמתנה יעשה בעזרת ממוצע של כמה RTT שקיים בקו, קלומר בכמות של שליחת וקבלת הודעות. ישנו כמה אפשרויות להפעיל את הטימר:
 - הפעלת טימר על כל סגמנט שנשלח.
 - או הפעלת טימר על כל התקשורת, קלומר כל עוד התקשורת פתוחה הטימרעובד מפתיחה ועד לסגירתה.
- Sequence number: עוזר לשכנת ה- transport המקבלת לסדר את הסגמנטים שגיעים אליו בסדר שליחתם.

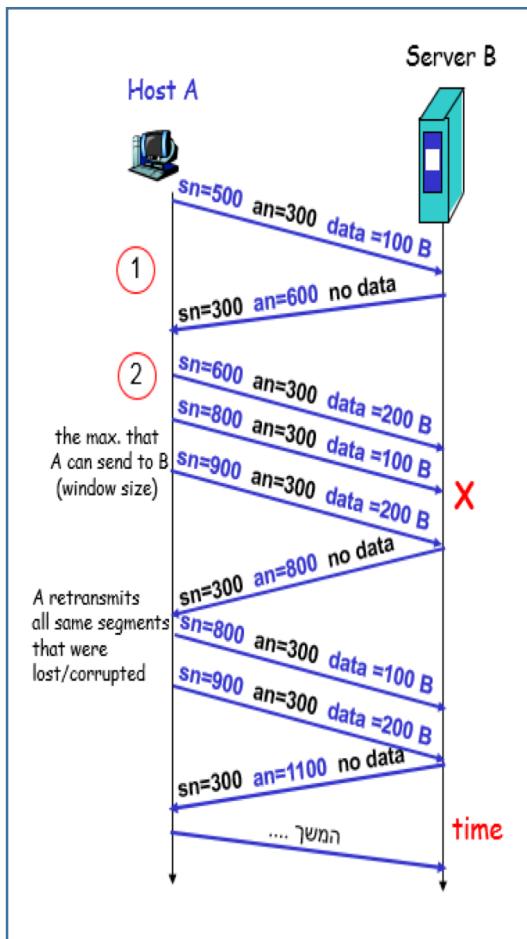
outgoing segment from sender



TCP seq. numbers, ACKs – sender view

כאשר באים למש TCP, הגיעו למסקנה שהזמן
שմבdziים על המתנה אישור על כל סגמנט שנשלח
הוא זמן מיותר, שכן שלוחים סגמנטים במקביל מבלי
לחכות לאישור על קבלת הסגמנט הקודם. וכך עד
שהתקבל אישור על קבלת סגמנט את אותו סגמנט
צריך לזכור, על מנת לאפשר לשולח אותו שוב
במקרה הצורך. אבל ישנה הגבלה – "חלון" בתים,
כלומר כמה בתים ניתן לשולח כדי שלא תהיה הצפה-
עומס. חלק מהחלון השילחה, אלה כבר סגמנטים
שנשלחו וחלקם הם סגמנטים שטרם נשלחו. המידע
הבא שאמור להישלח ימתין עד שהחלון יתפנה,
כלומר המידע שנשלח יאשר שהगיע לעדוי, וכן יהיה
לשולח את המידע הבא בתור. סגמנטים עליהם
התקבל אישור שהגיעו לעדום לא נשמרים יותר
לשילוח חוזרת (הסגמנטים הירוקים באירור).
סגמנטים אוטם אני שומר לשילוח חוזרת הם אוטם
הסגמנטים שעדיין לא התקבל אישור לגבייהם
ונמצאים בחלון (הסגמנטים הכהובים).

דוגמא – שkopiyot 3-62:



- HOST A שולח הודעה בגודל 100 בתים, החל מבית 500 ($an=500$), וגם שולח שעד בית 300 (לא כולל) שהתקבל מ-B HOST תקין ($an=300$).
- לאחר מכן מ-B HOST מתקבלת הודעה, שעד בית 600 הכל תקין ($an=600$) ולא מתקבלת הודעה בעלת תוכן (0 בתים), ולכן ההודעה שמתתקבל מ-B HOST היא מבית 0 + 300 בתים ($an=300$).
- לאחר מכן מ-B HOST שולח הודעה מבית 600 ($an=600$) בגודל 200 בתים וכמוון החל מבית 300 הכל תקין.
- במקביל נשלחת הודעה מבית 800 ($an=800$) בגודל 100 בתים וגם שההודעה מ-B HOST (שהיא ריקה) מבית 300 הכל תקין.
- ההודעה הנ"ל לא התקבלה.
- במקביל נשלחת הודעה מבית 900 ($an=900$) בגודל 200 בתים וגם שההודעה מ-B HOST (שהיא ריקה) מבית 300 הכל תקין.
- לאחר מכן נשלחת הודעה מ-B HOST (עדין ריקה) מבית 300 ($an=300$), וגם שעד בית 800 הכל תקין ($an=800$) – להיות וההודעה החל מבית 800 לא התקבלה ואי אפשר לאשר את שאר ההודעות.

- נשלחות שוב במקביל ההודעות החל מאותה ההודעה שלא הגיעה, ההודעה החל מבית 800 (800=אחס) בגודל 100 בתים ובמקביל ההודעה החל מבית 900 (900=אחס) בגודל 200 בתים.
- הסגמנטים שנשלחים מחדש חייבים להיות באותו גודל אשר היו כנסו לשכלהו בפעם הראשונה. لكن אי אפשר לאחד את שתי ההודעות הנ"ל לסגמנט אחד לשילוחה.
- ולבסוף נשלחת ההודעה מה-B HOST שעדי בית 1100 הכל תקין, ושוב ההודעה ריקה החל מבית 300 בגודל 0 בתים.

TCP round trip time (RTT), timeout

חישוב של הזמן אותו נדרש להמתין מאז שנשלח הסגמנט ועד שהתקבל האישור על הגעתו, עד לשילוח הסגמנט מחדש. דוגמיה בכל פעם את זמן ה-RTT שלוקח מאז שליחת הסגמנט עד לקבלת אישורו (ACK) ועושים את הממוצע ביניהם. הפונקציה לחישוב זמן זה היא פונקציה וקורסיבית כתולות ב-RTT של כל שלב קודם:

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

typical value: $\alpha = 0.125$ ♦

זהו הזמן של הדגימה האחרונה sampleRTT

הממוצע של הדגימות הקודמות estimatedRTT

כאמור לזמן זה מוסיפים בנוסף מקדם בטחון (טטיית התקן) :

$$|\text{DevRTT}| = (1 - \beta) * |\text{DevRTT}| + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

typically, $\beta = 0.25$

ולבסוף הזמן שנמדדין עד לשילוח הסגמנט מחדש הוא:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

TCP ACK generation

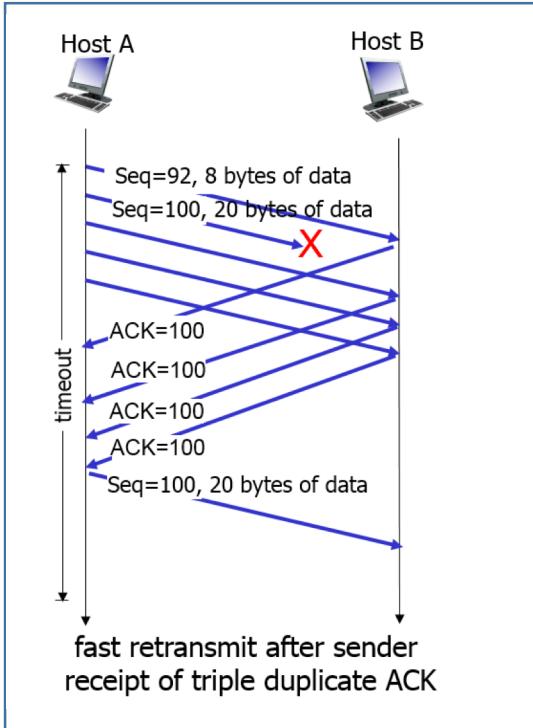
כאמור לא תמיד רוצים לשולח ACK על כל סגמנט, אלא נמתין כי אולי תתקבל עוד הודעה ואז נוכל להוציא הודעת אישור על 2 הודעות בלבד. לכן נהוג להמתין 500 מיל' שניות עד שמוצאים ACK על ההודעה. אלא אם כן ייצאת הודעה בכל מקרה.

סבירה נוספת בה לא נמתין 500 מיל' שניות, אלא נוציא ACK כאשר הצלבו כבר 2 אישורים, כך שלא צריך להמתין ולצבור הודעות נוספות כדי לשולח עליהם ACK. אם ההודעות המתקבלות מגיעות מרוחקים יותר מ-500 מיל' שניות אז תצא הודעה ACK לכל הודעה בנפרד.

כאשר שולחים כמה הודעות בהפרש זמן קטנים מאוד, ואחת ההודעות באמצע לא הגיעה ליעדה, נמתין 500 מיל' שניות ועוד נוציא אישור. לדוגמה - במקרה שההפרש בין ההודעה הראשונה והשלישית הוא יותר מ-500 מיל' שניות, נשלח הודעה ACK על ההודעה הראשונה. היהות וההודעה השנייה "הלנה לאיבוד" והגיע כבר ההודעה השלישית נוציא ישר הודעה ACK שעדי סוף ההודעה הראשונה הכל בסדר, וכך"ל לגבי הודעה הרביעית וכו'. נאמר שההודעה השלישית שהגיעה, כאשר השנייה חסורה, היא הגיעה – *out of order*. ברגע שהצד השולח מקלט כמה הודעות כאלה ברצף אותן, הוא מזיהה שהייתה בעיה בסגמנט השני שהלך לאיבוד, ואז עוד לפני *time-out* הצד השולח ישלח שוב את הסגמנט שכונראה הלך לאיבוד. כאמור ברגע שהגיע סגמנט שמחוץ לסדר שליחתו, הצד מקבל יוציא הודעה על כך שההודעה שלפניו בסדר, כמו פעמים, ואז נזהה שכונראה שיש בעיה.

TCP fast retransmit

במצב שהוזכר קודם, במידה ואנחנו שולחים הרבה סגמנטים אחד אחרי השני (היות ו-TCP מאפשר זאת, כי זה ייעיל), וסגןט אחד לא התקבל, אנחנו נקבל הודעה אישור ACK על הסגמנטים הקודמים לסגןט שאבד, וכן עוד לפני ה-timeout נשלח שוב את הסגמנטים החל מהסגןט האבוד.



TCP flow control

תפקידה של בקרת הזרימה הוא למנוע שליחת מידע בגודל אשר גדול יותר מהזיכרון של הצד מקבל. Win size – כמות הזיכרון שהוקצתה לטובת קבלת הנתונים אצל הצד מקבל. לדוגמה אם הצד מקבל הקצה 5000 בתים, ונשלחה הודעה באורך 2000 בתים, בהודעת אישור מהצד מקבל, ישלח גם עדכון של הזיכרון הפנוי הנותר, במקרה זה 3000 בתים נוספים. הזיכרון נשאר תפוס (שמור) עד שהאפליקציה תקרה את המידע הנדרש, אז כבר אין צורך לשמור את ההודעות התקינות שנקרו ע"י האפליקציה. ע"י פינוי הזיכרון ניתן למחזר את אותם התאים ולשמור את ההודעות הבאות שיתקבלו. האפליקציה כאשר היא קוראת את המידע שהגיע, היא תמיד תקרה את הבטים שהגיעו הראשונים (fifo).

כאשר מגיע סגןט out of order, הצד מקבל ישמר אותו בכל זאת, אך עדין יזהיר שיש לו מספיק מקום בזכרון כדי לקבל מידע אליו שאותו הסגןט לא הגיע וכמוון שהודעת אישור תשלח עד הבית של הודעה שהתקבלה תקינה ברגע עד להודעה "אבדה". (כלומר במידה והגיעו תקינים 3000 בתים הראשונים, הם 1000 בתים הבאים התעכבו, ואז הגיעו עוד 1000 בתים החל מבית 4000, הודעה התקינות תשלח עד לבית ה-3999).

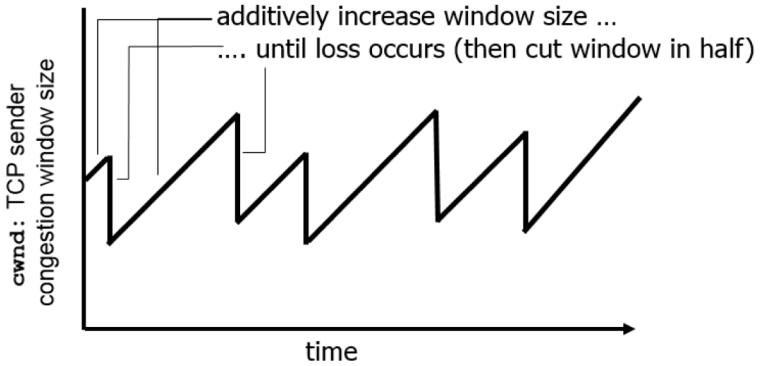
Connection management

תהליך "לחיצת היד" ב-TCP הוא תהליך תלת-שלבי, ככלומר כצד הלקוח רוצה לפתח התקשרות מול השרת, נשלחת הודעה הראשונה – SYN (כאשר הבית של SYN שווה ל-1), מצד השרת נשלחת הודעה אשר כוללת אישור – ACK וגם בהודעה זהה בית ה-SYN יהיה שווה ל-1, ככלומר השרת מבקש בחזרה לפתח התקשרות מול הלקוח ולבסוף, הודעה השלישית שנשלחת מלקוח כולל הודעה ACK.

סיגרת קו התקשרות באופן "נק'", נשלחת הודעה מלקוח אשר בה הבית של FIN הוא 1, ככלומר מודיעים על הכוונה לסגור התקשרות, השרת יחזיר הודעה שבה בית ה-FIN יהיה 1, לאחר מכן הלקוח ישלח הודעה אישור ACK על כך שההשתתפות מודעת לזהות שטוגרים קו. השרת ימתן פרק זמן מסוים (ב"כ RTT) על מנת לסגור התקשרות עם הלקוח. ב"כ השרת הוא זה שיזום את סיגרת קו, היוט ווים לשולח את כל המידע שהלקוח ביקש ובמקרה צזה כל מה שנאמר לעיל מתבצע הopor.

בקורת עומס – TCP congestion control

המטרה היא למנוע/להקטין עומס ברשת (רשת או בנתב וכו'), וזאת ע"י הורדת קצת שליחת המידע המקוריים שונים ברגע שזוהה עומס בדרכ. למעשה אין צורך שכל המקוריות יירידו את קצב השילוח אלאRob המשתמשים, היוות ויש משתמשי UDP וגם כאלה שלא מימשו בקרת עומס ב-TCP, שכן אין אפשרות להגעה אליהם. וזאת כמובן שהעומס עולה/יריד לא באופן ליניארי לעוממת delay queue, שכן מספיק להוריד קצת שליחת של רוב המשתמשים ורק delay queue ירד ממשמעותית.

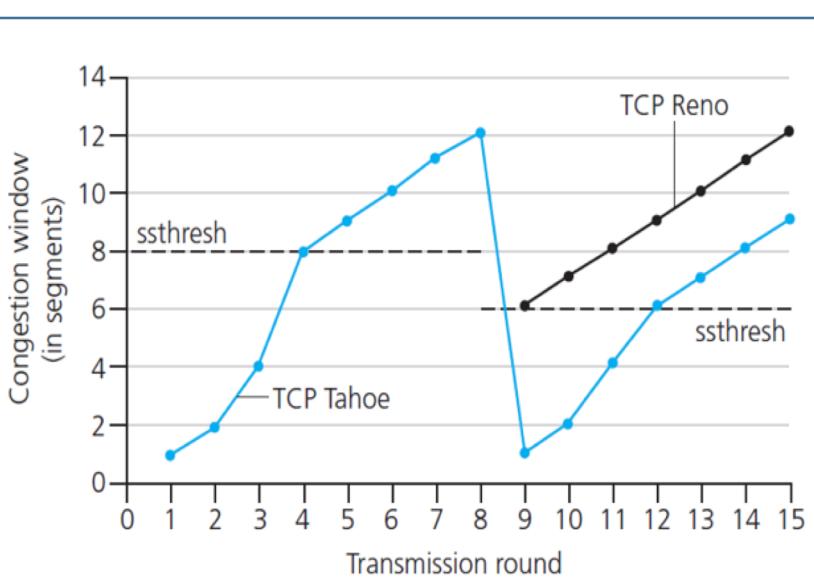


ב-TCP קצב השליחה (חлон השליחה – *cwnd* – congestion window) המידע מוגבר כל פעם, כל עוד לא>Zomega. ברגע שזomega עומס יורד קצב השליחה למחרות אופטימלית (בד"כ בחצי) ומהמהירות זו ממשיכים להגבר את קצב השליחה עד לזמן העומס הבא. קצב שליחה – כמה מידע ליחידת זמן (RTT) ניתן לשולח.

TCP slow start

בתחילת התקשרות מול השרת נתחיל את שליחת המידע בקצב איטי (נגד סגמנט אחד ל-RTT). ונגבר את קצב השליחה בשלהה הבאה (נגד ל-2 סגמנטים) וכך הלאה מכפילים את כמות הסגמנטים בכל שלב. עם כל הכפלת כמות הסגמנטים עולה ניצולות הקו, ברגע שנגיע לניצול של כ-80%, ה-queue delay יגדל יותר מידי. וכן יוצר עומס, עומס זה (בגישה TCP RENO) מזווהה ע"י 3 הודעות ACK זוחות (threshold) המעדות על קבלת סיגמנט out of order, ככלمر סגמנט כלשהו אבד בדרך, או שסגמנט מתעכב) וכך בעצםណע להוריד את המהירות.

כלן עד לקצב שליחה מסוים - threshold, נעלם את קצב השליחה בקצב אקספוננציאלי, מעל לקצב-hold-marketing מהירות מקצב השליחה בו זזהה עומס.



בבול זה יקבע BD"כ ע"י threshold, נעלם את קצב השליחה בקצב אקספוננציאלי, מעל לקצב-hold-marketing מהירות מקצב השליחה בו זזהה עומס.

אסטרטגיית TAHOE (שקופית 108-3)

אסטרטגיית שליחת מידע זו עובדת על העיקנון שצוין קודם. השליחה מתחילה לאט, סגמנט אחד ל-RTT, CMON שኒצול הקו פה הוא לא אופטימלי, שכן בשלהה הבאה גודל את CWND ל-2 סגמנטים ל-RTT, ו-*cwnd* גדל אקספוננציאלית. ברגע שקצב השליחה הגיע למספר מסוים – threshold שבו ניצול הקו כבר הוא טוב יחסית אבל גם כזה שלא יגרום לעומס. החלק מקצב-hold-marketing השלים סגמנט אחד יותר מהקדם. עד שנגיע לקצב שליחה שבו נזהה שיש עומס גבואה בקו, נוריד את קצב השליחה לסיוגמנט אחד ל-RTT והכל חוזר חלילה. קצב-hold-marketing, נקבע עפ"י מחזית מהקצב שבו זזהה עומס בפעם הקודמת (כלומר אם בפעם הקודמת הגענו לקצב של 12 סגמנטים ל-RTT, אז בפעם הבאה, ה-*cwnd* יגדל אקספוננציאלית עד שנגיע ל-6 סגמנטים ל-RTT שהוא בעצם threshold-hold-marketing). קלומר עוביים מדיניות של slow start עד threshold, למדיניות של CA-congestion avoidance.

אסטרטגיית RENO (שקופית 3-109)

אסטרטגיה זו דומה ל-TAHOE, אך לאחר זיהוי העומס קצב השילחה לא ירד לסוגמנט יחיד ל-RTT אלא ירד ל- threshold החדש (מחצית מקצב השילחה שבו זזהה עומס), ומשם קצב השילחה יעללה ליניארית. שיטה זו טוביה במקרה שהזיהה "עומס קל" וכן אין צורך להוריד את קצב השילחה בהתאם. עומס "קל" הכוונה שקיבלנו 3 הודעות אישור על סוגמנט מסוים (כלומר, אבד סוגמנט בדרך או שהוא מטעב). וכך ניתן להוריד את קצב השילחה קצר. אבל אם זזהה עומס "כבד" בראשת איזה התנהלות היא כמו של שיטת TAHOE. עומס "כבד" יזזהה ע"י כר שלא נשלחו הודעות אישור – ACK ברגע שהגיע הזמן – timeout.

TCP throughput

הצד השולח מחזיק משתנה שנקרא CWIN – גודל המידע שהוא מוכן לשЛОוח בהתאם לעומס בראשת. כאשר נתקיים בעומס בראשת יורד גודל ה-CWIN בחצי ומשם עולה חזרה לגודל CWIN עד העומס בפעם הבאה (באופן גס). ככלمر רוב קצב השילחה של מידע נוע בין CWIN-ל-($cwin/2$) لكن בממוצע קצב השילחה הוא $RTT - \frac{1}{4}cwin$.

פרק 4 Network Layer -4

תפקידה להעביר הודעה מיחידה אחת לשניה באמצעות שכבה הערוך (link layer). שכבת הרשת, לעומת השכבות הקודמות (שכבת האפליקציה, שכבת התעבורה – transport), נמצאת גם על נתבים בראשת. בעצם שכבה זו אחראית על קיבעת המסלול מהmoza אל היעד וגם להעביר פאקטים מכניות הרואוטר אל היציאה המתאימה לרואוטר המתאים. קביעת הרואוטר המתאים נעשית בד"כ בטבלאות ניתוב אשר מוגנות בנתבים. עוד דרך לקביעת הנתב המתאים היא לבחור במסלול הזרול/הקרצ'ר ביותר. אבל הדרכן הנכונה היא לדעת את המיפוי של הרשת הקרובה ובכך ישנה יכולת לחשב את המסלול האופטימלי בלי לertz Zeitן על "לשאול" ולקבל "תשובה" נתבים סמוכים.

כאמור בכל נתב ישן טבלאות ניתוב אשר מסדרות כתובות IP לעומת היציאה המתאימה באותו הנתב. ככלומר כתובות ה-IP של היעד שכבת הרשת יודעת לאיזו יציאה של הנתב להוציא את ההודעה, היא פונה לשכבה ה-link של אותה יציאה כדי שתעביר אותה לנット הבא.

הבעיה היא לכטוב את הטבלאות נכון, כך שלא תהיה התנגשות ולא ניכנס למעגלים של העברת המידע. ככלומר שלא יהיה מצב שאוthon ההודעה תעבור דרך נתב מסוים יותר מפעם אחת. הטבלה הזאת גם צריכה להיות מסדרת נכון כך שהשליפה תהיה מהירה וגודלה לא יהיה גדול מדי.

בניגוד לשיטת ה-packet switching (שליחים פאקטים לראים לאו דוקא זהים, כדי להגיע לעד), ישנה שיטת ה-circuit switching. שיטה זו דוגלת בתכנון מסלול מתאים מהmoza אל היעד להעברת המידע, בנוסח העשיה "שריון" של רוחב פס וכל מה צריך על מנת שההודעה תשלח כמו שצריך. אבל שיטה זו לא קיימת באינטרנט הרחב.

השיטה היוצרת מומלצת היא virtual circuit switching. גם כאן, לפני שנשלח המידע, נעשית בדיקת מסלול אולטימטיבי מהmoza אל היעד. אבל בניגוד ל-circuit switching הקלטי, רוחב הפס נשמר להעברת המידע בין 2 היחידות, במדיה והוא לא מנוצל, הוא ישמש להעברת מידע אחר מוקור אחר, ככלומר רוחב הפס לא "תבדבץ" לחנים (בניגודטלפון אשר שם נשמר לו לשיחת הטלפון בד"כ 40 kbit , וגם אם יש שטיקה בין 2 הצדדים עדין אוטם 40 kbit לתפוסים למטרות שהם לא מנוצלים). لكن פתיחת קשר בין 2 צדדים לוקח בד"כ יותר זמן מאשר העברת מידע עצמו, כיוון שהנתבים עוסקים בלבד במסלול האופטימלי ו"לשמור" אותו לטובות אותה תעבורה. אבל גם שיטה זו לא קיימת באינטרנט. (לפי ההבטחות דור 5 של רשת הסולר אפשר את שיטת ה-virtual circuit switching). IPv6 מאפשר עבודה בשיטה זו וישנם מעט מקומות שימושים בשיטה זו).

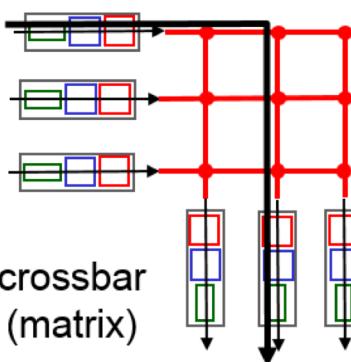
קיימים עובדים ב- packet switching הקלטי, עם טבלאות ניתוב. ככלומר אין תכנון מסלול מראש.

- הודהה בשכבת הרשת נקראת פרגמנט – fragment .

Router architecture overview

ישנם 3 טכנולוגיות המאפשרות העברת מידע מכניות הרואוטר אל היציאה המתאימה.
– Memory – קבלת ההודעה מהמכניסה אל הזיכרון הראשי של הנתב ומהזיכרון אל היציאה המתאימה.

Bus – כל הכניסות והיציאות יושבים על קו תקשורת מסוית, כשרצחה להעביר הודעה מהכניסה אל היציאה,



נקם את ההודעה על קו התקשרות (bus) והיציאה המתאימה תשולף את ההודעה מהקו. אבל הבעיה היא שהקו זהה הוא צואר בקבוק, כלומר העברת של הודעה אחת מהכניסה אל היציאה ולן שאר ההודעות מחכות עד לשילפה של ההודעה הוקדמת. לכן קו ה-*bus* צריך להיות מאד מהיר.

Crossbar (matrix) – חיבור קווי בין כל כניסה לכל יציאה. במצב זה לא נוצר עומס על העברת ההודעות של היות ואין הודעה שמתינה לשילפה (כמובן שאם כניסה הרבה הודעות שמיידות לכניסה אחת, יוצר עומס מיוחד ביציאה הזאת).

Datagram forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00010100 00000000 through 11001000 00010111 00010100 11111111	2
otherwise	3

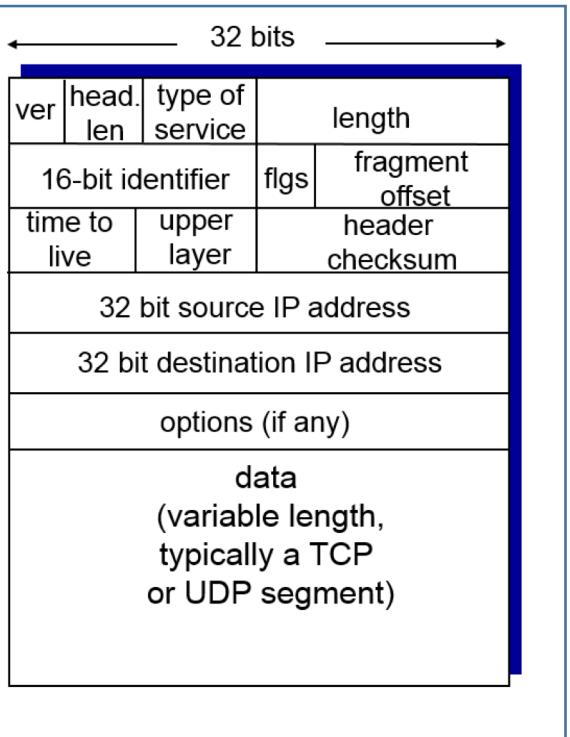
כתובת IP מורכבת מ- 32 ביט, لكن יש בעולם 2^{32} , בערך 4 מיליארד, כתובות אפשריות. פיזית אי אפשר להכיל בכל נתב טבלה בגודל זהה. לכן טבלת הניתוב שמצויה בראוטר תכיל את הקבוצות הרלוונטיות אל מול היציאות והנתב אמור לזהות בכל הודעה לאיזו קבוצה הוא שייך וכך הוא ידע לאיזו יציאה לשלוח אותה, כאמור לאיזה ראטור מבין הקרובים אליו זה אמרור להגיעה.

בטבלת הניתוב, הקבוצות מחולקות עפ"י כמה ספורות ראשונות של כתובת ה-IP (מודגר שונה לכל קבוצה), כאמור הראטור בודק לפי הרישא של כתובת ה-IP, ניגש לקבוצה המתאימה שבתווחה שלא נכללה כתובת ה-IP הרצiosa ולפי זה הוא יידע לאיזו יציאה להוציא את ההודעה.

Longest prefix matching

חישוב בטבלת הניתוב אחר כתובות היעד (כדי לדעת לאיזו יציאה של הנתב נדרש להעביר את ההודעה הנכנסת) יכול לדרוש הרבה זמן, שכן נדרש אלגוריתם בעל זמן ריצה מאוד קצר.

כתובות IP לעיתים יכולות להשתנות, כלומר ישנה קבוצה של כתובות IP שהיא מיועדת למקום מסוים (נגיד קבוצת הכתובות של ישראל), אבל יכול להיווצר מצב שבו ישנים שרתיים בישראל שקנו כתובות IP ממדינה אחרת. בכל פעם שימושו בעולם ינסה לגשת לאחת הכתובות הקניות הללו, ההודעה תישלח קודם כל לאוותה מדינה ממנה נקנתה הכתובת. לכן במקרים כאלה נדרש לעדכן את הראותים המרכזיים במרכז הרשות שקבעצת הכתובות הנ"ל שהיו משויות פעם למدينة אחרת, ביום משויות לישראל.



IP – Internet protocol, datagram format

גודל ההדר הוא 20 בתים.

: גרסת הפרטוקול IP. שדה בגודל 4 בתים.

Head len. – header length: גודל ההדר, ישנה אופציה ב-IP להרחב את ההדר בהודעה.

Type of service: מכיל בתוכו את סוג המידע שבו יש לטפל (TCP ו-UDP וכו').

Legth: 16 ביט, אשר מסמנים מהו אורך ההודעה (data).

המילה השנייה בהדר נועדה למצוב שבו הראטור מחלק את ההודעה לכמה הודעות קטנות (בגלל שבאותה יציאות שלו אין אפשרות להעביר את ההודעה בשלמותה). لكن המילה הזאת מתיחסת לאם ההודעה מוחלקת או בשלמותה, באיזה חלק מתוך כל החלקים הקטנים מדובר וכו'.

Time to live: שדה זה מואתחל בד"כ כ-127, וכל קפיצה- כאשר ההודעה עוברת לראטור הבא, מקטינים אותו

ב-1. ככלור אם ההודעה מסתובבת בלופ אינסופי בין הראוטרים, שדה זה יגיע בסוף ל-0 וכן במצב צזה ההודעה תמחק ונדרש להודייע לכל הנטים שיש תקליה וצריך להסתנכרן. ככלור שדה זה אומר כמה "קפיצות" יש להבצע בין נטים עד שתגיעו ליעד. מטרתו היא לזהות מugal של שליחת הודעות בין הנטים ולפטור זאת.

Upper layer: מצין את הפרטוקול שנמצא מעל ההודעה.

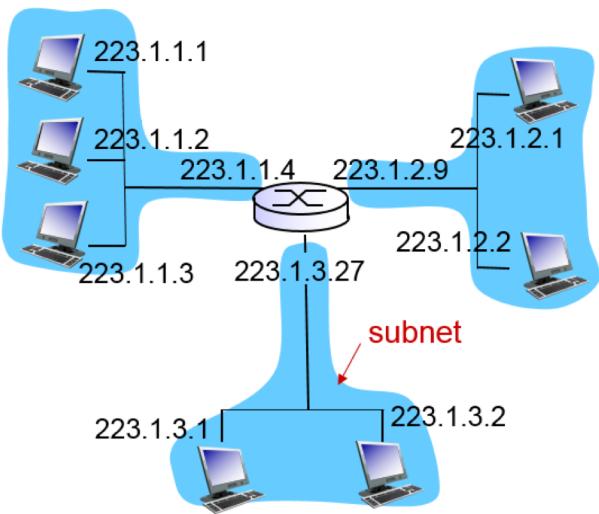
Header checksum: שדה לבדיקת תקינות ההדר. ככלור תקינות כל השדות בהםם הכתובות יעד ומוצא. כל ראייר שלו תגיע ההודעה צריך לחשב את ה-checksum מחדש על מנת לוודא שהדר תקין.

Fragment offset: מצין החלק מסוימת בティים לקרוא את ההודעה במידה והפרגמנט מוחלק לכמה חלקים. ככלור ההודעה הראשונה נגיד החל מאפסט 0 באורך 1500 בתים (20 בתים מיועדים להדר ועוד 1480 בתים להודעה) וכן ההודעה הבאה תתחילה מפרקמנט 185 (1480/8 = 185).

IP fragmentation, reassembly

כאמור יכול להיווצר מצב שבו הראייר קיבל הודעה בגודל מסוים, אך זו היציאה של הראייר שלו נדרש לשולח את ההודעה לא מסוגל להעביר ההודעה בגודל זהה, אלא קטנות יותר. וכן נדרש לחלק את ההודעה לכמה הודעות קטנות יותר – fragmentation.

כמו כן ראייר שמקבל כמה הודעות מפוצלות, יכול להרכיב אותן חזרה להודעה אחת בגודל הראשוני ולהמשיך לשולח. בעיקרו הראייר שיעשה זאת הוא יהיה הראייר שמהווה צואר בקבוק כלומר, בוודאות כל חלקו ההודעה יעברו בהכרח דרכו ולא יהיה מצב שחלק מההודעה המפוצלת תעבור דרך נטים אחרים מפה ואילך, וכך ירכיב את כל ההודעות ויבחר את ההודעה המורכבת הלאה, אבל זה לא מתקבל כמובן להיות חלק מההודעות תגענה דרך נטים אחרים, שכן ההרכבה מחדש רק ע"י יחידת הקצה שהוא יעד. היות והראיירים עובדים בשיטת store and forward על כל הודעה קטנה, שהיא חלק מהודעה גדולה, שתגיע לראייר תשלח לדרכה גם אם שאר ההודעות הקשורות אליה עוד לא הגיעו, כי כל הודעה צאתה היא הודעה בפני עצמה מבחינה הנטו.

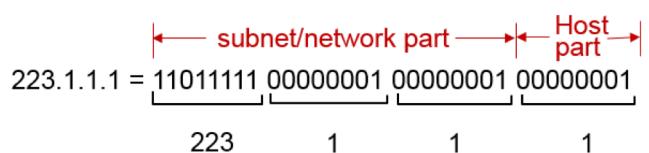


IP Addressing

- **Subnet part** – הביטים הראשונים בכתובת ה-IP (MSB), מאפיינים את תת הרשת שבה נמצאות יחידות הקצה. (באירור (223.1.2.2)).

- **Host part** – הביטים הנמוכים בכתובת ה-IP אשר מצינים את יחידות הקצה.

לא נהוג לתת סימנות של 111... או 000... ליחידות הקצה. הסימנת 000... בד"כ שומר למספר תת הרשת. לעומת זאת ישנו 2^k-2 כתובות שאפשר להקצות כליחיות הקצה בתת הרשת. כמובן שגם לראוטר של אותה תת הרשת נדרש להקצות כתובות IP.



IP Address structure

The network prefix: בכל תת רשת ישנו k ביטים קבועים שהם אופייניים לאוותה תת רשת (כמו שניתן לראות באירור מעלה 2 – 223.1.3 – כלומר $2^k=24$ ביטים קבועים). ישנו שני דרכים לציין את מספר הביטים הקבועים לכל תת רשת:

- לכתוב את כתובות ה-IP סלאש מספר הביטים הקבועים : 223.1.3.0/26
- הדריך השנייה היא ע"י מסיכה : 1111 1111 1111 1110 0000 ... 11 1100 = 255.255.255.192

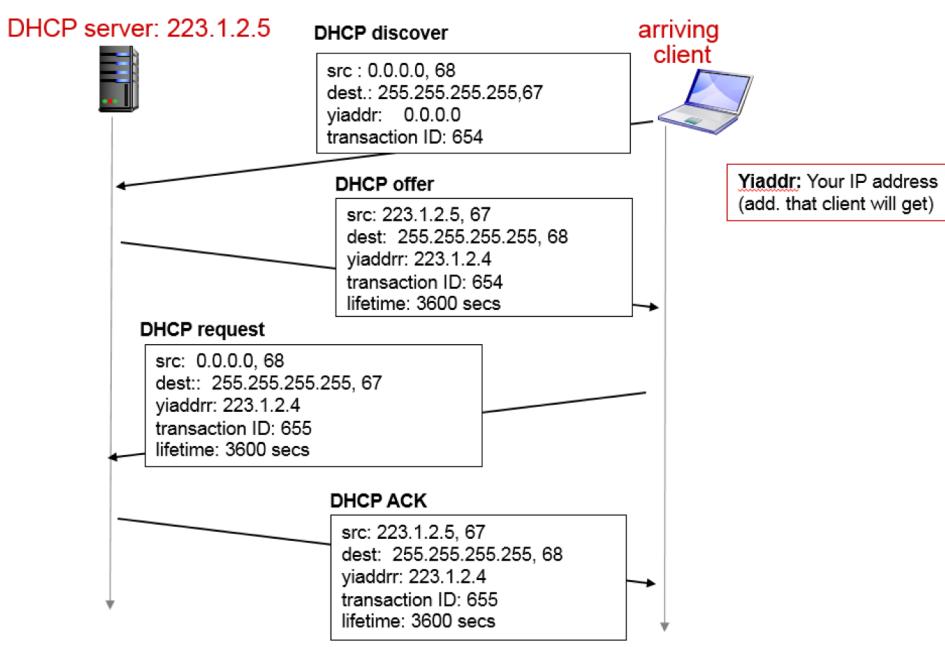
IP addressing : CIDR – classless InterDomain Routing

קיים גוף שנקרא (ICANN) Internet Corporation for Assigned Names and Numbers, הגוף הזה אחראי על חלוקת כתובות IP וניהול DNS. כך בעצם ספקיות האינטרנט – ISP, מקבלות את בלוק הכתובות IP שהן מחלקות ללקוחות שלהם בשיטה שכותבה לעלה.

קורה שקובוצת כתובות IP מחליפה מיקום/יעד (לדוגמה, ישראל קנחה קבוצת כתובות IP ממדינה אחרת), נדרש לשנות את היעוד שבעל טבלאות הניתוב על מנת שמעכשו "העולם" ידע שטוחה הכתובות הזה שייר לישראל.

איך ניתן לקבל כתובות IP כליקו?

לרוב, אנו משתמשים רגילים לא צרכיים כתובות IP קבועה כמו שירותי ידועים, אלא צרכיים כתובות IP מזדמנת – דינמית, (לדוגמא התחברות דרך wifi). בשביל זה קיימת הקצתה כתובות IP דינמית, וזה מיושם ע"י DHCP – dynamic host configuration protocol. שירות DHCP מספק כתובות דינמית ללקוח שהתחבר לרשת. כשיחידת קצה מתחברת לרשת, היא מבקשת כתובות IP, שירות DHCP מזיהה את הבקשה, הוא מפיץ לכלל הרשת (בעזרת פניה לנכונות IP שומרה לתפוצה כללית באותה הרשת 255.255.255.255) את הכתובת IP שהוא יכול להקצות. כשיחידת הקצה מבקשת כתובות IP היא מפרטת סוג של חתימה- ההזדהות על מנת שהשרת ידע לאיזו יחידת קצה הוא מקצה את ה-IP ולזמן מוגבל.



בכל רואוטר ורשת WIFI ישנו שרת DHCP וכך בעצם כל פעם שימוש מתחבר לרשת WIFI מסויימת הוא מקבל כתובת IP דינמית לזמן מוגבל. לאחר שהזמן עבר, במידה והלוקה רוצה להמשיך להזיק בכתובת הוא מבקש הארץ. כל רשת מתנהלת שונה, ישנו רשותות שלושת שדרותות תשלום עבור הארץ.

בעזרת שיטת עבודה של הקצאת כתובות IP דינמיות, ניתן למחזר כתובות IP. בכל התחברים צו שרטתי ה-DHCP נותנים בנוסף כתובות של שירותי DNS ו-LNS. וגם את המסיכה של אותה תת הרשת שאליה מתחברים.

המסיכה ניתנת לצורך זיהוי כתובת היעד (שאליה רוצים לפנות – אתר שאליו רוצים להיכנס). לדוגמה: כתובת היעד : 192.23.17.25 , הכתובת שנייה לлокו ע"י DHCP : 192.23.16.175/23 , כולם 23 ביטים קבועים. אם כתובת היעד היא חלק מתת הרשת אליה אנו מחברים, אנחנו נשלח שירות לעיד (בלי לעבר נתבים חיצוניים), אחרת נדרש לרשום לרואוטר שיוציא את ההודעה אל מחוץ לתת הרשת. כדי לגלוות האם היעד נמצא באותה תת הרשת צריך לעשות : כתובת היעד & (לוגו) המסיכה של רשת המקור , ובנוסף כתובת המקור (הכתובת שלו) & המסיכה של רשת המקור. אם התוצאות שוות זה אומר שהיעד נמצא באותה תת הרשת עם המקור, אחרת יש להוציא לרשת חיצונית.

8 הביטים הלפni האחרונים של כתובת היעד – (17) = ...00010001 ...

... ישנו 9 ביטים חופשיים – (254) = 11111110...

התוצאה של פעולה & על המספרים האלה ניתן : 0010000 = 16 כלומר ניתן לראות שרשת היעד היא 192.23.16.0 , כלומר היעד נמצא באותה תת הרשת של המקור.

(4-61) Private IP Addresses

Class	RFC 1918 internal address range
A	10.0.0.0 to 10.255.255.255
B	172.16.0.0 to 172.31.255.255
C	192.168.0.0 to 192.168.255.255

כתובות IP פרטיות
שניתנות למחשבים
בארגונים (ב"כ).
ישנו סיווגים שונים
לכתובות-IP בטוווחים
השונים:

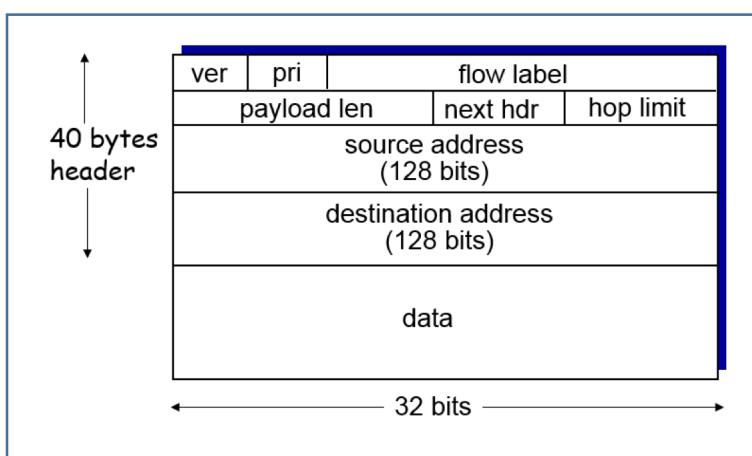
כתובות אלו לא נטמכות ע"י האינטרנט, וכך ייחdet קצה בעלת כתובת צו רוצה לתקשר עם יחידה באינטרנט , נדרש לתרגם את הכתובת IP הפרטית ע"י (network address translation NAT) שתתאים לרשת הציבורית. כאשר הודעה יוצאת מכתובת פרטית, כתובת השולח הפרטית מתרגמת לכתובת של אותה רשת שהיא כן מזוהה באינטרנט וכן גם כשנכנתה הودעה מהינטרנט לרשת הפנימית. בכל הودעה, בעת ההמרה

של הכתובת נשמר גם מספר הפורט של היחידה שלחה את הודעה, ומספר פорт זה נשמר בראוטר או בשרת NAT שנמצא על הראוטר. אל מול מספר פорт שהראוטר מקצה שהוא ייחודי לאוותה ייחידת הקצה ממנה באה הודעה היוצאת.

ספקיות האינטרנט – ISP בעלות , לצורך העניין, 100 אלף כתובות IP חוקיות באינטרנט, אך יש להן מיליון לקוחות, כדי לספק לכל לקוח כתובות IP ייחודית, מתיקנות שרת NAT וכן בעצם הן יוצרות רשותות פנימיות בעלות כתובות IPפרטיות להן וכמוון לכל לקוח כנראה ישים כמה ייחידות בבית אשר מחוברות לראוטר הביתי. וכן בעצם מנוהלות רשותות פנימיות בתוך רשותות פנימיות. לכן כל הודעה שיוצאה מלפטופ שמחובר ב-WIFI לראוטר הביתי, אשר בעל כתובות IPפרטית של התת רשת הביתית, מתורגם ביציאה מהבית על ידי שרת-h-NAT לכטובת ה"חיצונית" שהיא הוגדרה בעצם ע"י ספק האינטרנט, אך כתובות זו היא עדין כתובותפרטית של ספק האינטרנט וברגע שההודה יוצאה מתחם הרשות של אותו ספק האינטרנט מתורגם ע"י שרת-h-NAT של ספק האינטרנט לכטובה IPפומבית שידועה לאינטרנט. כל זאת נעשה כפי שצווין קודם ע"י התאמת מספרי פורטים. מספר הפורטים מוגבלים לכ-60 אלפי מספרים, שכן אנו מוגבלים לפחות מ-60 התקשרותות פתוחות (היות וכל ייחידת קצה יכולה להציג שירותים של דוא"ל וכו'). לכן שרת-h-NAT יכול "להחביא" עשרות עד מאות ייחידות קצה מאחורי אותה כתובות IPפומבית.

אם ייחידת קצה רוצה לשמש כשרת, יכולם קבוע שכולם מכירם, אפשר לשתף פעולה עם ה-NAT והמערכת הפעלה של אותה ייחידת קצה וכן בעצם שרת-h-NAT יודע לשומר פורט מסוים לאותה ייחידת קצה.

שרותי-h-NAT במקור תפקידם הוא להגדיל את מספר ייחידות הקצה שמחוברות ל-h-NAT (כיוון שיש יותר ייחידות קצה בעולם מכתובות IP אפשריות ב-32 ביט). כאשר ארגון עובר מספק אינטרנט אחד לאחר, חוץ מזה שהוחלפה הכתובת הפומבית של הרשות לכטובת שניתנה ע"י הספק החדש, לא השתנה כלום. יכולם הכתובות הפרטיות נשארו אותן כתובות היות ואלו כתובות פרטיות בתת רשות של ספק האינטרנט ושרת-h-NAT יודע להעביר הודעהו לאותן הכתובות.



IPv6 datagram format

: גרסת ה-IP Ver

: **Flow label** תכונן כדי לאפשר virtual circuit ב-IP. בעצם מזהה את מספר ה-flow (ההודעה במהלך התקשרות פתוחה), נועד על מנת לזהות לאיזה התקשרות ההודהה קשורה ונitin להתחילה לחשב את היציאה עוד לפני שהראוטר סיים לקרוא את כל ההדרה.

: **Payload len** כמות המידע.

: **Next hdr** מצין כמה מטרק ה-data הוא המשך ההדר בסדרות נוספים אופציונאלים.

.TTL- time to live :**Hop limit**

היתרון של IPv6 על פני IPv4 הוא ההדר הקצר יותר והמחסום ב checksum שזה גורר פחות "ביזבוז" זמן על חישובים ובנוסף שדה ה-flow label, כמספר ה-flow נקבע מראש "לחיצת יד" וכן בעצם כל עוד ההתקשרות פתוחה מספר ה-flow קיים, ישנו פחות מספרי flow מאשר יציאות אפשריות בראוטר, יכולם לא צרי לחשב לאיזו יציאה להוציא את הודעה כל פעם אלא זה קבוע מראש בפתיחת ההתקשרות.

Tunneling

תקשורת בין ראותרים שעובדים ב-4VPI לבין ראותרים שעובדים ב-6VPI אשר "mdbri'im" בין עצם, עושים זאת מהר ובעילות. אך כדי ידרשו לעבור דרך ראותר שעובד ב-4VPI, ההודעה "טייעף" ב-4VPI ותשלח לראותר, דבר אשר יאט את התקשורת במעט, עד שתגיעה שוב לראותר של 6VPI וההודעה תמשיך להישלח במהירות גדולה יותר. לעומת מעבר מ-4VPI ל-6VPI עדין תאופサー התקשורת בין ראותרים שעובדים עדין ב-4VPI.

VPN – virtual private network

כשרוצים להעביר מידע רגי'ש (נגד שמרצה רוצה להעלות ציונים לשרת המכלה, ציונים – מידע רגי'ש, מביתו ולא אחד המחשבים של תת הרשת של המכלה). במחשב בביטו הוא יבקש שהມידע יצא דרך יציאת ה-VPN, כאשר יש פניה ל-VPN מתבצעת הצפנה של המידע, נשלח באינטרנט המידע המוצפן אל שרת ה-VPN של המכלה, בשרת מתבצע פינוח ההודעה ונשלחת אל השרת הייעודי של העלאת הציונים. לעומת זאת העניין בהתקשרות זו המחשב של המרצה מתנהג כמו מחשב פנימי בתוך הארגון.

פרק 5 – שכבת הערוץ (Link Layer)

הודעה בשכבה זו נקראת מסגרת – frame.

שכבת הערוץ נותנת שירותים לשכבת הרשת באמצעות שכבה הפיזית, כולל תפקידה להעביר את המידע (מסגרת) משכבת הרשת בין 2 יחידות. עליה לדעת להבדיל בין הודעות שונות, שירותים בשכבה זו בצורת רצף ביטים. ברמה הפיזית ביטים מתפרשים כמתאים שונים (נגד '1' זה 4 וולט ו-'1' זה 4 וולט).

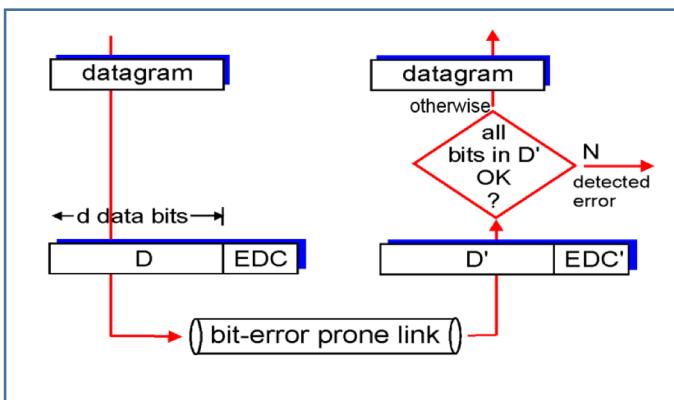
דרך לזהוי מסגרת:

- **תחילת מסגרת באמצעות קוד ידוע מראש.** בד"כ אחרי קוד זה יציין אורך ההודעה, על מנת שנדע כמה מקום להקדזה להודעה ולעתים גם יש קוד מיוחד לסוף ההודעה.
- רצף ביטים מקובל לסימון תחילת מסגרת הוא : 01111110, כאשר שכבה הפיזית מזהה תחילת מסגרת (רצף זה), היא משחילה 0 לפני ה-'1' האחרון כלומר, 011111-0-10, ואז בסוף המסגרת היא מוציאה את ה-0 המשוחל. אם אין את ה-0 המשוחל. אם אין את ה-0 זה סימן שהרצף היה חלק מהມידע ולא מסמן סוף מסגרת.

שכבת הערוץ ממוקמת על כרטיס הרשת. ברובה ממומשת בחומרה.

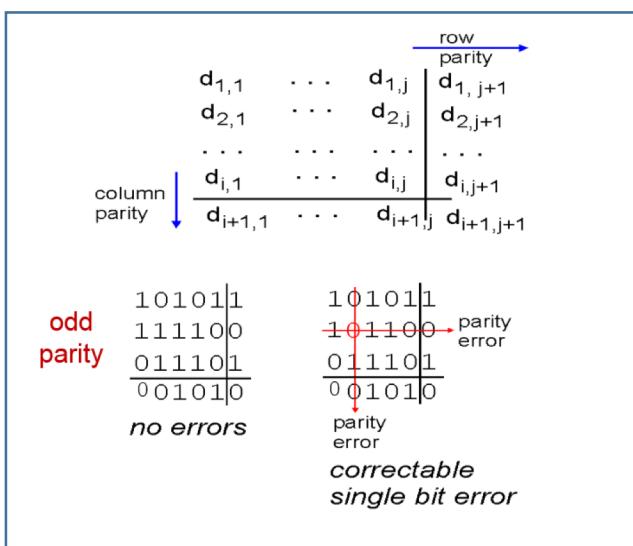
Link layer services

- **בקורת זרימה (flow control):** בرمת הערוץ קצב העברת המידע מנוהל בין 2 יחידות סמוכות זו לזו, ככלומר יחידה אחת יכולה להודיעה לשכנה שהיא משדרת בקצב גבואה מיידי. ככל שMOVEDים את קצב השידור עולה הסיכוי לפיענוח נכון יותר של המסגרת עקב רעשהם.
- **Error detection** – זיהוי שגיאות במסגרות בין 2 יחידות סמוכות זו לזו
- **Error correction** – תיקון שגיאות במסגרת ע"י מזהים של המסגרת, ללא צורך בשילוחה חוזרת של אותה המסגרת.
- **Half duplex** – יכולת לשדר ולקבל אך לא בו זמן, ככלומר אם אחד משדר הוא לא יכול לקבל ולהיפך. (לדוגמא מ.ק. 77 – מכשיר קשר צבאי).
- **Full duplex** – יכולת לשדר ולקבל בו זמן (לדוגמא הטלפונים).



Error detection

הרעין מאחריו זיהוי שגיאות בשכבה הערוץ, למידע המקורי מוסיפים עוד כמה ביטים שהם מוחאים את ה- (EDC) (error detection and correction bits). ככלומר כשהודעה מתתקבלת, היחידה מחשבת את ה- H -checksum בדומה לשדה checksum. ה- H הוא בגודל 16 ביט. בחישוב EDC לא ניתן לגלוות את מקום הטעות. על מנת לדעת איפה השגיאה משתמשים ב-bit parity.



Parity bit – parity check

מוסיפים עוד ביט נוסף להודעה המקורי, שהביט הזה מצין את מספר ה-'1', אם מספר זה הוא זוגי או ביט הזוגיות יהיה 0, אחרת 1. להיות זהה רק ביט אחד לא ניתן לגלוות את מקום הטעות. אחת השיטות לגלוות את מקום הטעות הוא להשתמש במטריצה של ביטי זוגיות. ככלומר מסדרים את המידע בשורות שורות. אם גיד גדול של המידע הוא 100 ביטים, מוסדרים את המידע ל-10 שורות לכל שורה מכילה 10 ביטים. מחשבים ביט זוג לכל שורה ולכל עמודה. בנוסף מוסיפים עוד ביט זוגיות שמחשב את הזוגיות של הטרו שנוסך והשורה שנוספה (ע"י ביט הזוגיות). ככלומר על הودעה של 2^k ביטים נוספים עוד 2^{k+1} ביטים. ככלומר על הודעה של m ביטים סה"כ נוספים עוד $1 + \lceil \frac{m}{2} \rceil$ ביטים. כאשר יש מספר זוגי של טיעויות בשורה/טור לא ניתן להזיהות טיעות כיון שביט הזוגיות ישר אוות הדבר. לכן אם יש יותר מטיעות אחת, בשיטה זו לא ניתן לתקן את הטעות, כיון שלא ניתן להזיהות את המיקום במדויק. היות והסיכוי לטיעות אחת הוא קטן, השיטה הזאת היא די טובה. במידה ולא ניתן לתקן את הטעות היחידה תודיע ליחידה השולחת שישנה טיעות במסגרת.

בשורה/טור לא ניתן להזיהות טיעות כיון שביט הזוגיות ישר אוות הדבר. לכן אם יש יותר מטיעות אחת, בשיטה זו לא ניתן לתקן את הטעות, כיון שלא ניתן להזיהות את המיקום במדויק. היות והסיכוי לטיעות אחת הוא קטן, השיטה הזאת היא די טובה. במידה ולא ניתן לתקן את הטעות היחידה תודיע ליחידה השולחת שישנה טיעות במסגרת.

Cyclic redundancy check

קוד לגילוי שגיאות בחומרה באמצעות CRC. קוד זה קל למימוש בחומרה.

- בפרקטייה לרוב ברשת האינטרנט אין מנגנונים לתיקון שגיאות אלא לזיהוי שגיאות בלבד, וכשמדובר שגיאה המסגרת "זרקת" ונשלחת מחדש.

Multiple access links, protocols

בניהול ערוץ מרובה משתמשים, קיימות התנgesיות בין הודעות שמייעות מכמה משתמשים שמשדרים בו זמןנית. לכן אפשר לנוהל את הערוץ בכמה דרכי כדי להתמודד עם הבעה.

- ניהול ערוץ במניעת התגשויות, ככלומר למנוע את האופציה שתהיה התנgesות בין כמה הודעות.
- התאוששות מההתגשויות, כאשר ישנה התנgesות ישנה גם דרך לסדר את ה Hodoutes כך שכביכול התנgesות לא קرتה.

כאשר יש דובר יחיד, היינו רוצים שהוא יקבל את כל רוחב הפס לשידור שלו, וכך גם כאשר יש יותר מדברים היינו רוצים שכל אחד מהם יוכל חלק יחסית מרוחב הערוץ.

כמו כן היינו רוצים של תהיה ייחידה מיוחדת המועדת לריכוז ההודעות ושליחתן להלאה כדי לפתור את בעיית התתגשויות.

(5-20) MAC protocols: taxonomy

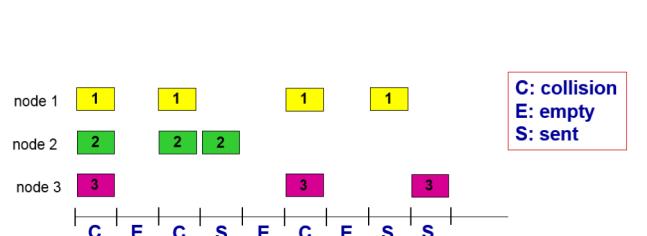
- Channel partition: חלוקת הערוץ לפי כמות המשתמשים הפוטנציאליים. השיטה לא אידיאלית היות ואם רק מישחו אחד מהמעמשתמשים רוצה לשדר עצה, הוא ישדר על החלק שהוקצה לו לשידור ולא על כל רוחב הערוץ. אך שיטה זו פשוטה וגם אין ייחודה שמנחת הודעות. בשיטה זו יתבצע חילוק מעריך שמווקצת למשתמשים אשר לא משדרים באותו רגע.

החלוקת מתבצעת בין כל המשתמשים הפוטנציאליים ליחידת זמן בשיטת TDMA - time division multiple access. ככלומר אם יש 5 משתמשים, כל אחד מהם ישדר רק 20% מהזמן וב- 80% הנותרים הוא יאדי.

חלוקת נוספת היא FDMA – frequency division multiple access – תרומות חלוקה של הערוץ לפי תדרים לכל משתמש. לרוב המשתמשים ב-TDMA.

- Random access protocol: הגישה המקובלת היום. כאשר אחד מהמעמשתמשים רוצה לשדר, הוא משדר ברוחב ערוץ מלא (R – data rate). במקרה שני משתמשים או יותר משדרים בו זמנית, תהיה התתגשויות. כאשר ישנה התתגשויות משדרים שונים, רק שהפעם בניהול נכון של שידור. ישנו כמה פרוטוקולים לניהול ערוץ עם התתגשויות. (Slotted ALOHA, ALOHA, CSMA, CSMA/CD, CSMA/CA).

Slotted ALOHA



Pros:

- ❖ single active node can continuously transmit at full rate of channel
- ❖ highly decentralized: only slots in nodes need to be in sync
- ❖ simple

Cons:

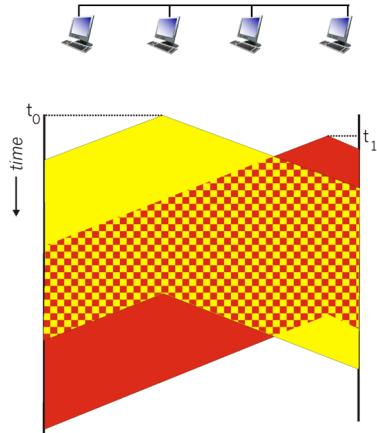
- ❖ collisions, wasting slots
- ❖ idle slots
- ❖ nodes may be able to detect collision in less than time to transmit packet
- ❖ clock synchronization

אם 2 משתמשים או יותר שולחים בו זמנית הודעה, תזוזה התתגשויות. זמן השילוח מחולק לתאים – slots לפי זמנים בהם ניתן לשולח פריטים אחד. כל הודעה נשלחת למחשב הראשי. כאשר ישנה התתגשויות המחשב הראשי מודיע על כך, ועל כל משתמש להगיריל ערך שיקבע האם ב-slot המשמש לשדר. יכול להיות מצב כך שאף משתמש לא ישדר והכו ישאר לא מנוצל. יכול גם להיות מצב שניים ישדרו שוב ביחד ותהיה שוב התתגשויות. וכך הלאה כל משתמש משדר עפ"י ערך "מוגול" האם ישדר או לא. פרוטוקול זה עובד על הרעיון שאם יש לך משהו לשדר אז תשדר, אם תへיה התתגשויות תקבל הודעה ואז תגיריל ערך שיקבע האם ב-slot הבא לשדר או לא. אם לא הייתה התתגשויות אז ממשיכים לשדר כרגע עד לתתגשויות. אחת ההנחות בפרוטוקול זה היא שהפרויימים בגודל זהה. ניצול הערוץ המקסימלי בפרוטוקול זה הוא 37%. ככלומר ב- 63% מהזמן הערוץ יהיה מבזבז. בשיטה זו לא נדרש מישחו שינוי את הערוץ.

Pure (unslotted) ALOHA

עפ"י פרוטוקול זה אין צורך במחשב מרכזי שיניהל את ההודעות ויודיע על התתגשויות קיימות. החיסרון בכך הוא שהטיכוי להתגשויות (חוסר הסync'ון) גדול. ככלומר כל משתמש ישדר ברגע שיש לו מה לשדר, מה שיגורר ירידת בניצול הערוץ פי 2 לפחות 18%. אם המשתמש לא קיבל אישור על הודעה שלו, סימן שהיתה התתגשויות והוא ישדר מחדש.

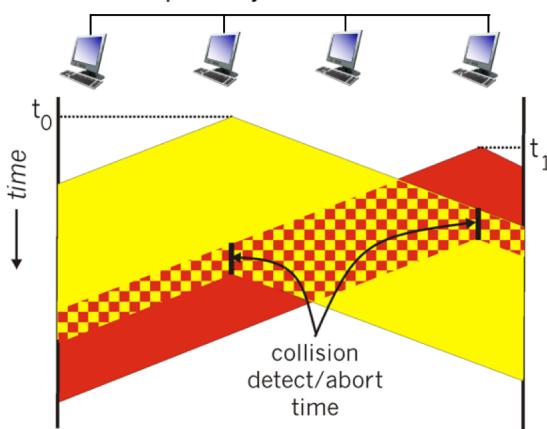
spatial layout of nodes



CSMA – carrier sense multiple access

פרוטוקול זה רעיון דומה ל-ALOHA, אם יש מה לשדר אז תshedder, שנייה אחד: לפני שאתה משדר תازין לך. אם מישו כבר משדר אז תמתין עד שהוא יסימן כדי למנוע התנגשות ואז תshedder. אך עדין יכולה להיות התנגשות, בмедиיה 2 ממשמשים ממתיינים לך פניו, ברגע שהקו התפונה שנייים מתחילה לשדר ואז ישנה התנגשות. בغال ה-propagation delay יש זמן שבו יחידה אחת התחלת לשדר והיחידה האחראית עדין לא "שומעת" את השידור על הקו, ולכן היא בטוחה שהקו פניו ותחילה לשדר גם. לכן בזמן מסוים תהיה התנגשות בין הפרויימים. בפרוטוקול זה היהות וכל יחידה יכולה רק לשדר או להאזין ולא בו זמן, שתי היחידות ימשיכו לשדר. כדי לוודא שלא הייתה התנגשות נמתין לאישור על הפרויים, אם לא התקבל האישור סימן שהייתה התנגשות.

spatial layout of nodes



CSMA/CD – collision detection

בפרוטוקול זה ההנחה היא שכל יחידה יכולה גם להאזין וגם לשדר בו זמן. במצב של התנגשות תוך כדי שידור היחידה תדע שיש עוד מישו שמשדר במקביל. במצב זה מקובל להוציא signal, collision signal, כך שכל מי שמאזיןuko ידע שהייתה התנגשות ולאחר מכן להפסיק לשדר. על מנת לזהות התנגשות תוך כדי שידור יש צורך לוודא שזמן השידור לא יהיה קטן מפעמים ה-propagation delay. זאת מכיוון שאם 2 יחידות רחוקות אחת מהשנייה על אותו ערך במרקם מקסימלי, יחידה א' מתחילה שידור בזמן t_0 , ו- $t_0 + 2t_{propagation}$ (distance/velocity = propagation delay). זאת אומרת שבזמן $t_{propagation}$ יחידה ב' תשמע שינוי שידור על הערץ. לכן הזמן המקסימלי בו יחידה ב' יכולה להתחיל לשדר הוא שואף ל- $t_{propagation}$. ועד שיחידה א' תשמע שינוי ב' שידרה עברו זמן השווה ל- $t_{propagation} - 2t_{propagation} < t_{transmission}$. לכן נדרש:

(transmission) קטן יותר מפעמים $t_{propagation}$ לא ניתן לדעת אם הייתה התנגשות בשידור, היות יחידה א' הפסקה לשדר כבר וכן מבחינה זה בסדר שיש עוד שידור על הקו. את המרחק וזמן ההתפשטות של האות ניתן לקבוע ולחשב עפ"י תקנים שונים (לרוב תקני ETHERNET), וכך ניתן לדעת את $t_{propagation}$. כמובן שעיל זמן השידור ניתן לשנות ע"י קיר שגדלים את גודל הפרויים הנשלח בכל פעם. לאחר שזיהתה התנגשות נדרש לשדר את 2 הפרויים (או יותר), השידור מחדש רנדומלית כמו ב프וטוקול ALOHA ע"י הגרלת ערך שיגדר רנדומלי כמה זמן על היחידה הזה לשידור חוזר.

Ethernet CSMA/CD algorithm

1. יצירת הפרויים לשידור.
2. אם הערוץ פניו – התחל שידור אם הערוץ תפוא (יש שידור) – המtan עד שהקו יתפונה ואז תshedder.
3. תازין לך בזמן שאתה משדר על מנת לזהות התנגשות.
4. אם היחידה סימנה לשדר בלי שזיהתה התנגשות במהלך השידור, אז הפרויים נשלח בצהלה.
5. אם בזמן השידור זיהתה התנגשות, הפסיק שידור ושלח signal jam על מנת לכל המאוזינים לערוץ ידעו שהתרחשה התנגשות (למשך סמן מספיק כך שכל היחידות על הערוץ יתעדכו).
6. לאחר התנגשות מגילים ערך K שמנוגר בטווח של $\{1, 2, \dots, 2^m - 1\}$ כאשר m מיצג את מספר התנגשויות שהתרחשו ברגע. לאחר ש- K מוגרל, היחידה השולחת תמתין $K * 512$ bit times עד שתshedderשוב. ההגרלה נעשית בכל יחידה שולחת עם מספר התנגשויות האינדיווידואלי שלה.

"taking turns" MAC protocols

שיטת POLLING : הגדרת יחידה master כ-masterמנהלת את שאר היחידות מתי הן יכולות לשלוח את המידע שלහן. לכל יחידה בנוסף ישנו חיבור אשר מסמן האם לאוֹתָה יחידה יש מה לשדר או שאין.

שיטת TOKEN PASSING : השיטה מעודדת חיבור בין היחידות הרבות בمعالג, כלומר כל הודעה שיצאת מיחידה אחת עוברת את כל שאר היחידות בمعالג וכשההודה תגיע שוב לשולח סימן שהוא עבר את כל היחידות באותו ערך. ברגע שליחידה המשדרת נגמר המידע לשידור, היחידה הבאה בתור בمعالג יכולה לשדר (כלומר אותה יחידה אשר משדרת היא בעלת ה-*token*). הבעיה היא שהיחידה שמחזיקה ב-*token* תקרוס. במצב זהה היחידה הבאה בתור תזהה שנוצר מצב של אובדן *token*, היא תמתין זמן מסוים ואז תיצור לעצמה *token* ככלומר את הסמכות לשדר. כמו כן צריך לוודא ששתי היחידות במקביל לא תייצרנה לעצמן את הסמכות זו. لكن נדרש מגנון התאוששות חזק מהקרים של בעל ה-*token*. במצב שבו מctrפפים עוד 2 יחידות למעגל, יתריעו על כך שהן מctrפפו, אבל מצב זה יכול ליצור התנגשות וידרש מגנון התאוששות מהתנגשות, لكن השיטה זו כבר לא מקובלת.

MAC addresses and ARP

בשכבה הערוץ, לכל כרטיס רשות יש כתובות קבועה מהיצרן, כתובות זו היא כתובות MAC (או LAN, physical Ethernet). מספר זה הוא בן 48 ביט. לדוגמה: AD-09-BB-76-2F-IA. בኒוגוד לשכבה הרשת, בשכבה הערוץ ההודעה נשלחת ומתקבלת עפ"י כתובות MAC. כתובות ה-MAC רלוונטיות רק לערוץ בין 2 נתבים, כלומר הכתובת של השולחת והכתובת של הנ才干ב המתקבל.

מערכת -(*protocol*) ARP יודעת לתרגם מכתובות IP לכתובות MAC. המערכת הזאת צריכה להיות מאוד פשוטה והיא נמצאת על כרטיסי רשות שיכולים להיות מואוד פשוטים כמו בובות, אוזניות וכו'. וגם היא צריכה לדעת רק את כתובות ה-MAC של אותם המכשירים המתחברים ישירות באוטה (טלפון – אוזניה וכו', מחשב – נתב וכו').

מערכת ARP מבוססת על רשומות, כל יחידה ARP מחזיקה טבלה שבה יש כתובות IP, כתובות ה-MAC שמתאימה לאוֹתָה כתובות IP והזמן שהצמד זהה רלוונטי (כלומר, אם כתובות ה-IP עתידה להשתנות זה אמרו להיות מצוין : <IP address; MAC address; TTL>). הטבלה זהה מתעדכנת בזמן אמת. כשכתובת ה-MAC לא ידועה, הפרטוקול מרים בתת הרשות שאליתה שבה הוא שואל לאיזה כתובות MAC שייכת כתובות ה-IP שאליה רוצים לגשת, וכך הטבלה מתעדכנת. כאשר רוצים לשדר שאליתה לכל היחידות שמחוברות לאוֹתָה תת הרשות משתמשים בכתובת MAC שמורה : ff-ff-ff-ff-ff-ff , ככלומר ההודעה מיועדת לכל כרטיסי הרשות בערוץ. ברגע

A (a , a) knows *B*'s IP addr. (b) & wants to know *B*'s MAC addr (β)

1. *A* sends ARP Query Message for *B*'s MAC address:

- message sent as broadcast frame on Ethernet

Src MAC	Dest MAC	Type	Source IP	Src MAC	Dest IP	Dest MAC
α	FF-...-FF	query	a	α	b	?

Ethernet Header ← → *ARP Message*

2. *B* reads the message and sends ARP reply to *A*

- reply sent as a unicast frame to *A*'s MAC address

Src MAC	Dest MAC	Type	Source IP	Src MAC	Dest IP	Dest MAC
β	α	reply	b	β	a	α

Ethernet Header ← → *ARP Message*

שהיחידה בעלת הכתובת ה-MAC הלא יודעה מקבלת את ההודעה, הוא יודע איזה כרטיסי רשות שלח את ההודעה זו, ועונה לו צזרה עם כתובות ה-MAC הרצוייה. (הדגמה בשקופית 5-46)

כאשר יחידה שולחת שאלתה לגילוי כתובות MAC רצiosa, כל שאר כרטיסי הרשות באותו הערוץ מעדכנים את טבלת ARP שלהם שאוֹתָה היחידה בעלת כתובות IP ו-MAC אשר היא צירפה לשאלתה, נמצאת איתם באותו הערוץ.

כאשר רוצים לעדכן את שאר היחידות על שנייה/ עדכון של כתובת IP או כתובת MAC (במקרה של החלפת כרטיס הרשת), אותה היחידה שבה בוצע השינוי, שולחת שאלתה לגבי כתובת ה-MAC של עצמה, כך בעצם אף ייחידה לא תענה על שאלתה זו, אבל כל היחידות יעדכו את טבל ARP שלהם על השינוי שבוצע.

בתת הרשת, הכרטיס רשות יודעים את כתובות ה-MAC רק של היחידות המוחברות לאותה תת הרשת, כאשר רוצים לפנות לכתובת IP מוחוץ לתת הרשת , מעבירים את ההודעה לנットב של תת הרשת, והנטב של תת הרשת אמור לדעת את כתובות ה-MAC של הראטור הבא, ואם הוא לא יודע אז הוא ישלח שאלתה כמו שצווין קודם.

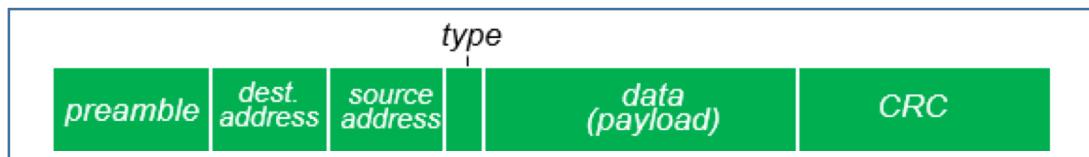
Ethernet

פרוטוקול של CSMA/CD.

Bus: כבל קוֹאָקְסִיָּאֵל' המחבר בין כמה מחשבים ביחד באותו ערוץ.

Star: כל יחידה מחוברת בחותם אל switch, וזה נותן את התחששה שכל היחידות מחוברות ביניהן באותו חוט.

Ethernet frame structure



הפריים בניו מכנה שברגע שמתחלים לקבל את ההודעה, מתחילה לחשב את ה-CRC לגילוי שגיאות, וכך שуд שההודעה תגיע לשדה CRC יהיה כבר בידי הכרטיס רשות את התוצאה של החישוב ואפשר יהיה להשווות בלי לבלבול זמן.

preamble: תחילך אשר "עיר" ומופיע את הכרטיס רשות, היות וכל הכרטיס רשות מזין בחצי אוזן לערוץ עד שמקבלת הودעה רלוונטית אליו. שדה זה הוא בן 8 בתים , כך ש-7 בתים בתבנית : 10101010 והבית האחרון הוא : 10101011, אשר מציין את סוף השדה. ככלומר הבטים האלה הם בתים לא חינוניים להודעה עצמה אלא נתונים לאלקטרונית של הכרטיס רשות זמן להידלק ולהיכנס למצב פעולה. כמובן שגם הביטים האלה נוכנים לחישוב השגיאה.

היות וכרטיס רשות הוא רכיב מאד זול, גם השעון שלו הכרטיס הוא לא מדויק, ככלומר אם כל בית יהיה משודר במשך מיליונית השניה והשעון של מקבל נגיד מזיף , ככלומר הוא מקבל בית במשך 0.9 מיליון שניות, אך יהיה חוסר סyncronon בין השידור למקבל. לכן שדה זה יודע לסנכרן את השעון של מקבל לפי שעון השידור כדי לקבל את הביטים בסדר שצרכיר.

Dest.address: כתובת ה-MAC של היעד. לפי הכתובת זו יודע הכרטיס המקבל אם הפניה נעשית אליו או לא. במקרה שההודעה מיועדת אליו הוא ממשיר לעבוד. ואם לא מיועדת אליו אז הזרזנה נגמרה מבחינתו כי לא הוא היעד.

Type: שדה של 2 בתים. אינדיקציה לפרוטוקול של השכבהعلילונה (AppleTalk, IP ...). עברו גרסאות שונות המספר ששמור בשדה זה מהheid על משמעותות שונות. אם המספר הוא 1518 אז שדה TYPE מייד על גודל ההודעה(DATA). אחרת זה מייד על סוג האפליקציה שבה משתמשים, מהו סוג-data (ARP) הודעת AppleTalk וכו').

Data: בין 64 ל-1518 בתים של מידע. מינימום 64 בתים נקבע כדי שגודל הפריים לא יהיה קטן מדי, היות ופרוטוקול ה- Ethernet עובד בשיטת - CSMA/CA, אז אנו רוצים שזמן השידור יהיה גדול מפעם-ה- propagation delay . לכן שומרים על גודל פריים מינימלי שיעמוד בתנאים. בנוסף בגלל הסיבה הזאת הפרוטוקול האינטרנט יש אורך (פיזי) של הקו מוגבל באורך. **כל שגדל קצב השליחה אורך הקו מתקצר.**

בפרוטוקול ה- Ethernet אין תחילך "לחיצת יד" בעת שליחת/קבלה פריים (connectionless). והוא גם לא שולח הודעות ACK או NACK על קבלת הودעה תקינה, שכן הוא פרוטוקול לא אמין. פריים אשר נדרש לא נשלח

פעם נוספת, והדרך לוודא אמינותו הוא רק ע"י מימוש בשכבה הגבוהה יותר, אחרת הפריים אובד. מה שכנן הוא יודיע לשולח פריים חדש רק אם הייתה התגובה בשידור זהה כבר ממומש ע"י פרוטוקול CSMA/CD.

802.3 Ethernet standards: link & physical layers

- בכל תקני-h Ethernet הפרוטוקול של MAC והפורמט של הפריים דומים.
- ניתן להשתמש במהירותים קווים שונות.
- ניתן להعبر דרך שכבות פיזיות שונות (CABLE, FIBER ..).

השני הוא קטן בין התקנים השונים בכרטיסי הרשות, אך אם שמים כמה כרטיסים באותה רשת יש לוודא שיש להם לפחות יציאה אחת זהה.

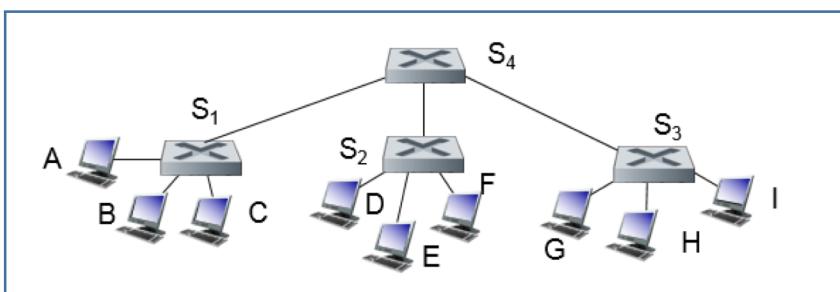
Hubs

זאת קופסה שאליה מחוברים הרבה יחידות, כל פעם שSEGUE אליו אות (ביט), הוא מעביר את האות הלאה לשאר היחידות שמחוברות אליו. הוא שיר לשכבה 1 – השכבה הפיזית. ב-*hub* פשוט ישנה התurbבות בין היחידות הנוכנסים אליו, כלומר אם כמה היחידות משדרות בו זמנייה הוא יSEND את הביטים לפי סדר כניסה אליהם. ב-*hub* יותר מתוכם הוא יידע לזרום התגובה משדרת בשיידור ולשלוח jam שמספיק את השידור. יכול להיווצר מצב שההودעה משודרת מיחידה אחת בקצב של 100 Mb/s והכרטיס רשות של היחידה המקבלת עובד ב/s 10Mb יהיו פספוסים של ביטים. לכן כל היחידות המוחוברות אמורים לעבוד באותה מהירות.

Switch

בדומה ל-*hub* היא מחברת בין כמה היחידות קצה. שייכת לשכבה 2 – שכבת הערוץ. Switch מעביר מסגרות (פריים) אשר מגיעות אליו בוגוד ל-*hub* שמעביר ביטים. בנוסף היות והוא-*switch* מקבל ומעביר מסגרות, הוא יידע להתאים קצב שליליה של המסגרת בהתאם לקו של יחידה מסוימת, כלומר לכל יחידה הוא יידע לשולח את המסגרות ב מהירות אשר מתאימה לעובדה של הכרטיס רשות של אותה היחידה. לכן בחיבור ל-*switch* ניתן לחבר יחידות בעלי כרטיסי רשות שונים זה מזה שעובדים ב מהירותים שונות. כמו כן יש להקפיד על מרחק פיזי שעומד בתקני-h Ethernet, כך שלא יהיה ארוך מדי. ה-*switch* יודע לבדוק תקינות של מסגרות ובהתאם יידע לזרוק הודעה לא תקין. כמו כן הוא יודע להוסיף התגובה עצמה. בנוסף יש לו טבלת תיבות switching בהתאם לכתובות MAC של היחידות המוחוברות אליו, כלומר בכל פעם שנשלח פריים מיחידה מסוימת הוא יידע לעדכן את הטבלה אותה יחידה נמצאת ביציאה מסוימת, כלומר הוא לומד את מיקום היחידות בהתאם ליציאות שלו. וכך בעצם במקום לשולח את המסגרת לכל היחידות המוחוברות אליו הוא יידע לנtab את המסגרת לקו הספציפי של יחידת היעד.

Interconnecting switches



ברגע שנשלחת הודעה מ-A ל-G, ההודעה מגיעה ל-S1. אם ה-*switch* זה יודיע איפה G נמצא הוא שולח לאן שציריך, אחרת הוא שולח את ההודעה לכל היציאות שלו, חוץ מהיציאה של A, כולל היציאה ל-S4.

בדומה התחילה ממשיך בכל switch עד שיגיע ל-G. וממנו שהמסלול אל A נרשם.

Wireless and mobile networks

היחס בין יחידות אלחוטיות לחותיות הוא כ- 5 אלחוטיות ל-1 חותית.

הסוגיות לטיפול ברשתות אלחוטיות :

- תקשורת בקוו אלחוטי. – wireless
- התנהלות במסגרת היחידה ממקום למקום. – mobility

Elements of a wireless network

יחידות הקצה ברשת אלחוטית מחוברות בד"כ **base station** (או access point). כלומר כל היחידות מדברות אל אנטנה אחת בחדר, והאנטנה מחוברת בד"כ קווית, אל הרשת עצמה. רשתות WIFI בד"כ מיעדות למרחקים קצרים, ורשתות סלולר לעומת זאת מייעדות לחברות למרחקים גדולים. ההבדל בין דורות הרשת הסלולרית הוא בעיקר בקצב השידור (Mbps).

ברגע שעוברים מאנטנה אחת לאנטנה אחרת (בטקשורת סלולרית נגד) צריך לנצל נכון את השידור כך שייהי מינימום שיבושים. זאת היא בעיית התניניות.

קיים רעיון של שימוש בתקשורת של אנטנות בסביבה (לא תשתית), אלא על כמות המסתמשים באוטה הרשת בסביבה, וכך בעצם כל יחידה מנצלת לטובות שידור (מציר קצת את הרעיון של P2P). לדוגמה אוזניות שיש במודיאונים, שהמשדר מדבר עם האוזניות-ב-Bluetooth, bluetooth, אין שם תשתיות תקשורת לכל האוזניות אלא מקומי.

Single hop: מצב בו מדברים אל אנטנה שמחוברת קוית לרשת, כאשר ישנה תשתיות - במצב ללא תשתיות שתי יחידות מדברות בין עזמן ללא צורך באנטנה מרכזית (Bluetooth).

Multiple hop: מצב שבו השידור עובר כמו יחידות (אנטנות) אלחוטיות עד שmagiu לרשת קוית – כאשר ישנה תשתיות.

כאשר אין תשתיות בניה, המצב הוא שהשידור עובד בין כמה יחידות קצה עד שmagiu לעד ללא שימוש באנטנות מרכזיות.

Wireless link characteristics

SNR: single-to-noise ratio, יחסאות לרעש, ככל שעוצמת האות עולה, קצב השידור ירד אך יותר קל להוציא את המידע מעוד הרעש.

BER: bit-error-ratio, ככל שה-NSR יעללה, ככל העוצמה של השידור עליה, וכך בעצם השגיאה היחסית תרד, ככל יותר מסגרות יגיעו ללא שגיאות.

Decreased signal strength: �ווית לא רלוונטי כמעט. ככל שמרתחים עוצמת האות יורדת בתקשורת אלחוטית. מה שבתקשורת קוית לא רלוונטי כמעט.

Interference from other sources: קיימת הפרעות מיחידות אחרות.

Multipath propagation: היות וגלי רדיו מוחזרים מכל אובייקט הניכר בדרכם, אותן רדיוס מגיעים לעיטים שונים מה שיכל לגרום לחוסר סyncronization בהודעה המשודרת.

במצב בו יש כמה יחידות שמשדרות יכולות להיות חסימות פיזיות שימנען מיחידה אחת לשמוע את היחידה האחרת. ובנוסף כאשר 2 יחידות משדרות בו זמן היחידה השלישית תשמע את שנייה ויהי מצב של התנגדות. אבל היות ו-2 היחידות לא שוממות אחת את השניה, לכן לא מודעת לכך שהן משדרות בו זמן נתן. מצב זה יכול להיגרם גם לא דואקן מחסם פיזי, אלא בגלל המרחק הרב בין 2 היחידות המשדרות, לכן גם לא תזהינה שמתבצע שידור במקביל.

ברשות אלחוטיות לא כדאי לעבוד על ניהול התנגשויות היות ולא תמיד ניתן לזיהות התנגשויות, אבל כן יש הימנעות מהתנגשויות. למשל ברגע שיחידה זיהתה שידור היא לא תshedר בו זמן.

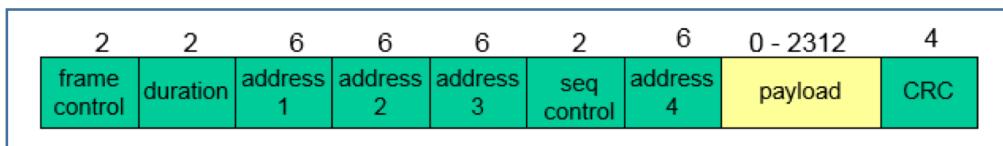
חיבור לרשת WiFi

- **הגישה הפסיבית:** יחידת קצה מחקה ל-point access שיפנה אליה, וודע לה שקיימת רשות אלחוטית זמינה. הودעה זו שימושתת מ-AP נקראת beacon frame.
- **הגישה האקטיבית:** דוגלת בכך שהיא ידית קצה תשליך מסגרת כלשה' שתגרום ל-AP לשולח beacon frame ליחידת קצה.

Collision avoidance

כשיחידה רוצה לשדר, היא תשליך הודעה RTS כדי לבדוק האם הקו נקי ואפשר לשדר, כל עוד לא התקבלה הודעה CTS היחידה לא רשאית לשדר. ברגע שהתקבלה הודעה CTS היחידה רשאית לשדר את ההודעה, בתקופה שבמקביל כל שאר היחידות שמעו את הודעה RTS ולא ישדרו אותה AP. כמובן שיכולה להיות התנגשויות גם בשילוח הודעה RTS, אבל היה זה אונתי לא שדר אותה שוב (גם כאן מגרילים ערך אשר יקבע האם היחידה תשליך הודעה RTS ומתי). כאשר המסגרת התקבלה בשלמותה ב-AP, ה-AP יוציא הודעה ACK כך שכל שאר היחידות ידעו שהוא קיבל עוד מסגרת וגם זאת הודעה אישור שההודעה הגיעה תקינה.

802.11(wi-fi) frame : addressing



: כתובת ה-MAC של השולח או AP שתתקבל את ההודעה. address 1

: כתובת ה-MAC של השולח או ה-AP אשר משדר את המסגרת. Address 2

: כתובת ה-MAC של הרואוטר אליו מחובר ה-AP. Address 3

: מספר המסגרות שנשלחות כחלק מאותה הודעה (כמו ה-seq. number). Seq. control

: משך השידור המותר ל-AP. ניתן לטובת היחידות המאזינות כדי שידעו متى הן יכולות להתחיל לבקש רשות לשדר. Duration

כאשר העוברים בין AP אשר מחוברים לאותו switch, נדרש לעדכן את הטבלאות ב-switch כדי שזה ידע לאיזה AP לשדר על מנת להגיע לאותה היחידה שעבירה. זאת הביעות של mobility.