

מבוא לרשתות מחשבים

הרצאה 1

מהו האינטרנט?

בילוניים של קשרים בין מכשירים. קצווות הרשות נקראים hosts ואלו מרכיבים אפליקציות (כמו Web, email, TCP, IP, HTTP, Skype, וכו').

ב途 כלל הזרים עובר מידע, כאשר הדרך הנפוצה ביותר להעברתו היא העברת בabiliaות ע"י ראותרים – "forward packets".

להעברה וקבלת מידע משתמשים בפרוטוקולים שונים, כגון: 802.11 IETF – Internet Engineering Task Force, קיימם ה-802.11 אחראי על הסטנדרטים של האינטרנט.

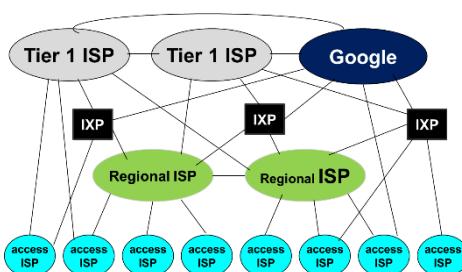
חוויות מידע בהעברת מידע:

- ❖ Bits are the units used to describe an amount of data in a network
 - 1 kilobit (Kbit) $\sim 1 \times 10^3$ bits = 1,000 bits
 - 1 megabit (Mbit) $\sim 1 \times 10^6$ bits = 1,000,000 bits
 - 1 gigabit (Gbit) $\sim 1 \times 10^9$ bits = 1,000,000,000 bits
- ❖ Seconds are the units used to measure time
 - 1 millisecond (msec) = 1×10^{-3} seconds = 0.001 seconds
 - 1 microsecond (msec) = 1×10^{-6} seconds = 0.000001 seconds
 - 1 nanosecond (nsec) = 1×10^{-9} seconds = 0.000000001 seconds
- ❖ Bits per second are the units used to measure channel capacity/bandwidth and throughput
 - bit per second (bps)
 - kilobits per second (Kbps)
 - megabits per second (Mbps)
- ❖ Bytes are units that describe a series of eight bits
 - 1 Byte = 8 bits
 - Bytes per second (Bps)



שלבים בהתפתחות האינטרנט

1. כלום מחוברים לכלום – יש קשר בין כל רשות לכל רשות אחרת. סה"כ קשרים: (2N). חיבור לכל הקשרים משמעו הכרה של הכתובות כך שכל מחשב יוכל לדעת בעצמו لأن לשלוח חבילה מסוימת שברצונו לשולח.
2. Global ISP – רשות מרכזית אחת שאליה מחוברות כל הרשותות (הביתיות). זה לא יעיל מבחינה אבטחה, כיון שיש מרכז אחד שברשותו כל המידע הקיים, וכן לא עמיד לתקלות – במקרה תקלת רשת כל האינטרנט העולמי קורט.
3. Global ISPs – מס' רשותות מרכזיות הקשורות ביניהן בצורה היררכית (מיקומיות, אזוריות, ספקים מרכזיים וכו').
4. הוספה exchange point בין רouters 'שכנים' בתוך מערכת של Global ISPs.
5. Internet Xchange Point (IXP) – הוספה מיני חיבורים (הסכמים) בתוך מערכת של Global ISPs בין רשותות בכל מני רמות, עם"ג לקצר תהליכי גישה:



Katzot הרשות

גישה לרשות

Pop = נק' החיבור של הרכיבים לרשות האינטרנט (כניסה ויציאה של חיבור). קיימ בכל רכיב. Central office – הרכיב הראשון של ספק שהמידע היוצא מהבית פוגש (ארון תקשורת למשל).



2 מודלים מרכזיים להעברת נתונים:

1. Client–Server model – כשלקוח מבקש מידע משרת מרכזי.
2. Peer to peer – חיבורם ישירם בין לaptops, כגון Skype ושיתוף קבצים.

גישה של מכשירים בבית לרשות אינטרנט נעשית דרך ראטור המחבר למודם (כיסוי לעיתים מדובר מכשיר אחד), והמודם מקשר ל'headend או ל central office.

חיבור הפיזי

מחשבים ומיכרים אחרים מתחברים לראטור באמצעות כבל או wireless.

צורות העברת תקשורת:

- Dial-up: שימוש בטלפון הביתי לצורכי העברת מידע באינטרנט בזמן שאין בשימוש, ובמקרים לא אפשריים שימוש בטלפון בו-זמנית. במקרה זה השתמשו בעבר.
- DSL – Digital Subscriber Line – שימוש בתשתית הטלפון הקווית להעברת חברות המידע באינטרנט. במקרה זה, לצד הלקוח יהיה splitter בטלפון בו-זמנית. במקרה זה השתמשו בשירותISP – כאשר יש קצב שונה בין שליחת מידע (כמו גם בין העלה והורדת ADSL – כשר יש קצב שונה בין שליחת מידע (כמו גם בין העלה והורדת של קבצים). רוב האינטרנט יכולים להשתמש במקרה זה.
- Cable network – שימוש בתשתית הטלויזיה לצורכי האינטרנט. במקרה זה, מוסך לקוחות יתחברו ל headend (= סיב) אזרחי והוא יתחבר ל- ISP. ככל שהסיב יהיה קרובה יותר למחשב, כך האינטרנט יהיה מהיר יותר.
- Ethernet – Enternet access network – אינטרנט של חברות/מוסדות גדולים (כמו האוניברסיטה), בו קיימת תשתיית מקומית הכוללת מחשבי רשת, ובין כל התקשרות הפנימית קיימים רכיב אחד שמחובר לISP כדי שהרשת תהיה מוקשרת לאינטרנט חיצוני.

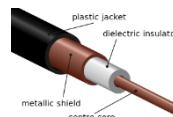


הcabl עצמו – בניו באחת משתי הצורות:

- Twisted Pair – בניו בצורת חוטים מלופפים, כמו ביסל. הליפוף עוזר בכך שאם ישנה הפרעה בשדה מגנטי מסוים – ההשפעה תהיה יחסית זהה בכל החוטים ולא יהיה פער גדול.



- Coaxial cable – צורה אחרת בה יש חוט מרכזי וסביבו חוט נוסף (עוטף אותו), עם שכבת בידוד ביניהם.



צורת התקשרות

ה-hosts מחלקים את המידע לחבילות להעברה, כל חבילה תופסת את כל הcabl (אחין) בזמן שהוא עוברת. כאשר חבילה מגיעה לנtab כלשהו בדרך, היא מתחילה את דרכה לנtab הבא רק לאחר שהגיעה לנtab הנוכחי בשלמותה.

Packet switching

החבילות מנוטות בעצמן דרך חיבורו התקשרות, וייתכן שהן יגיעו לעד לא לפ' הסדר. כמו כן ייתכן שחבילות שונות היוצאות מאותו היעד ינותו בצורה שונה לאותו היעד, זאת בהתאם לתקילות ועומסים. routing קבוע לכל חבילה את המסלול שלה וכותב אותו בheader של החבילה. ה-forwarding שלוח כל חבילה למסלול המתאים (מעביר את החבילה מהinput של אותו Router לoutput שלו).

דוגמא מאד ל – *waze* !! וכן יותר היום. זמן העברת T ביטים בקצב R : $\frac{L}{R}$

יתרונות: טוב להעברת מידע רב, משאיר את התשתייה להיות משותפת לכולם, פשוט ואמיש.
חרונות: כבל יכול להיות עמוס, חבילות יכולות להיבזב/להגיע באחור, מצרי פרוטוקולי שליטה.

Circuit switching

מוגדר מראש כל המסלול עבור מקור ויעד מסוימים, ומסלול זה יהיה פרט依 רק לקשר המסוים זהה. ייתכן שהמסלול יכול רק מס' חוטים בודדים מתור כבל גדול בין 2 נtabים. במקרה זה, אין חשש לאיובן חבילות, עומסים או הגעה לא לפ' הסדר.

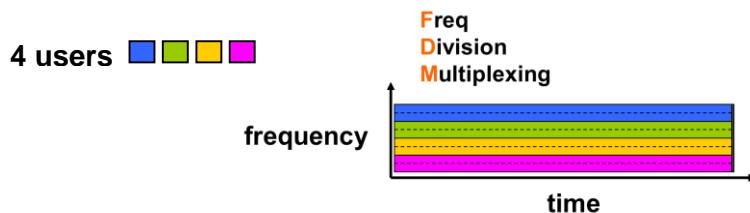
חסרון בולט: בקשר דו-כיווני, ייתכן ואחד המסלולים (-אחד הכוונים) שימוש רק מעט אך מוקצה לזמן רב בשל החזקת הקשר לשילוח מידע בכיוון השני (מציר שיחת טלפון הין אדם פטפטן לאדם שתפקיד...), כמו כן דורש זמן של הקמת העוזץ.

דוגמא מאד ל – מסלולי רכבות!!

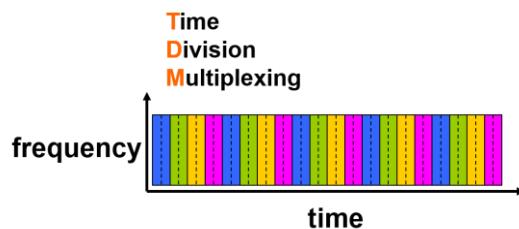


בפועל, הקצאת הcabl נועשית **לפי זמן או לפי תדר:**

FDM = הקצאה **לפי תדר** – כל תדר מוקצה למשתמש אחד לאורך כל זמן התקשרות.



TDM = הקצאה **לפי זמן** – הcabl מוקצה בכל זמן ללקוח אחר, והואusable בכל התדרים.



4 קритריונים לאופטימיזציה:

1. Power – מתחבطة בגודל השרתים, ועוד דברים פיזיים

2. Delay – 4 מרכיבים לעיכוב:

Process.1 – זמן העבודה בראוטר, כולל קביעת מסלול אופטימי.

Queue.2 – תור לשילוח החבילות, קיימן בכל ראותר

Transmission.3 – קיבולת הلينק, כמה זמן לוקח להעביר חבילה.

מוגדר עפ"י הנוסחה: $\frac{\text{packet size}}{\text{rate}}$, כאשר rate = רוחב הפס (כמה יכולם לעבור בו בזמןית)

Propagation.4 – התפשטות - העברת בית בודד אחד. תלוי באורך הcabl

וב מהירות. מוגדר עפ"י: $\frac{\text{distance}}{\text{velocity}}$.

3. Throughput – תלוי בכמה מידע יכול כל cabl להעביר בביטחון אחת.

4. איבוד חבילות - בד"כ קורה בכניסה לתור, כשחbillות באות להיכנס לראוטר והטור מלא.



הרצאה 2

Internet Layers

- Application – המשמש, הנאה שירותי הרשת – גלישה, שליחת מייל וכו'.
- Transport – סוגים שונים של משלוחים = פרוטוקולי העברת מקצה לקצה. זיהוי שגיאות. מטפל גם בעומסים ווחbillות שאבדו בדרך.
- Network – אחראי על הניתוב, מציאת המסלול. מפרק את החבילות לקטנות יותר בהתאם לצורך. משתמש בכתובות ה-IP.
- Link – ה'קו' – לחבר בין 2 נק' סמוכות, לעיתים גם בין רשתות מקומיות. מארגן את המידע למעבר לנק' הבאה בדרך, ומטפל בשגיאות.
- Physical – משלוח פיזי, העברת הסיביות על החוט.

השכבות הן חלוקה היררכית לתחתי-בעיות, מאפשרות מודולריות ו POSSIBILITY לשינויים, מאפשרות משלוחים שונים בנסיבות שונות ללא השפעות רבות על המערכת. חסרונות: דרוש עירוב של שכבות, תקללה בשכבה מסוימת תזקעת את כל החבילות, עבודה כפולה שנעשית במס' שכבות (כגון בדיקת תקינות החבילה).

ישנו גם מודול 7 השכבות – נקרא Open Source Interconnection – OSI. במודול זה 4 השכבות התחתונות זהות, אך בין שכבת ה-transport לשכבת application יש 2 שכבות נוספות:

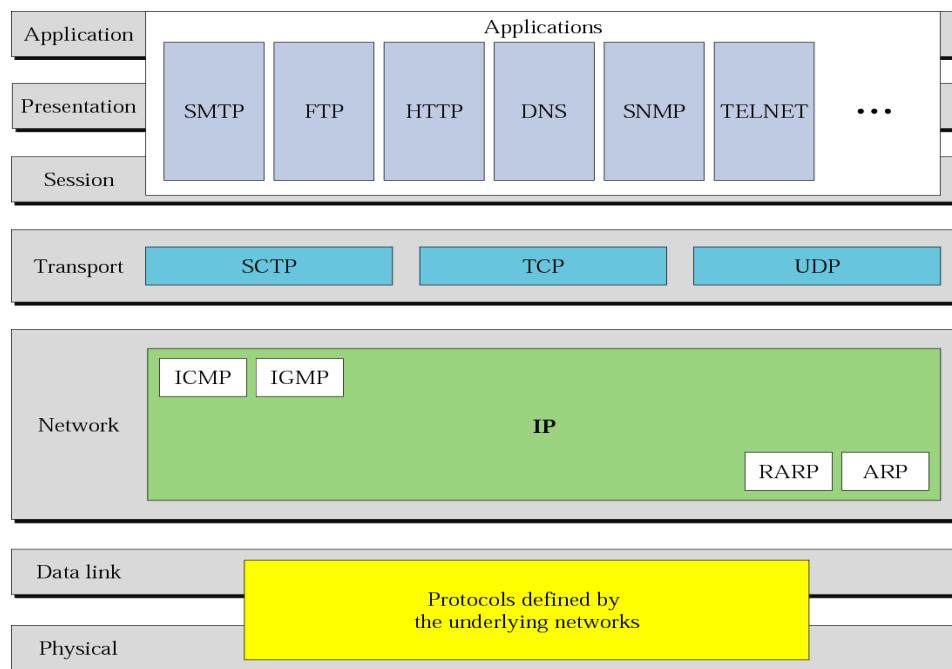
- עטיפת המידע – תרגום, פענוח/הצפנה, ציוז/הרחבה
- החלטה כיצד תהליכי מדברים בזמנים וallow כתובות יישמו לשם כך.

במודול 5 השכבות, התפקידים של 2 השכבות הנ"ל נכללים בשכבת האפליקציה.

במעבר חבילה, מידע מסוים רוצה להישלח משכבת האפליקציה בקצת אחד של הרשת לשכבת האפליקציה בקצת אחר. כדי להישלח, המידע עובר מהשכבה העליונה לתחתונה במחשב השולח, אשר כל שכבה מוסיף לשכבה header מסוים שמיועד לשכבה המקבילה הצד מקבל. השכבה הפיזית של הצד השולח תעביר את החבילה יחד עם כל התקורה שלה (=אוסף כל headers) לשכבה הפיזית הצד מקבל, ושם החבילה תעלתה במעלה השכבות כאשר כל שכבה מפרקת את header המיועד לה.



פרוטוקולי השכבות:



Application Layer

איך לתוכנת אפליקציה מוביל לחשב על העברת המידע באופן פיזי, תוך כדי שימוש על סקלביות ומודולריות.

הרכבת האפליקציה נעשית בשכבה העליונה (החיצונית) בלבד, ו מבחינת הנטבים מדבר בחבילות שעוברות כמו כל מידע אחר. לנטים יש גישה רק למידע המתווסף ב-Headers של 3 השכבות התחתונות.

צורה נפוצה לשימוש היא באמצעות **שרות-לקוח**. השרת צריך להיות זמן כל הזמן ובעל כתובת קבועה. הל庫ון מצדיו יכול להתחבר/להתנתק לטיורוגן, לשנות כתובות וכו'. במודל זה לקוחות לא מדברים אחד עם השני אלא רק דרך השרת.

מודל נסוף – P2P – ללא הגדרת שרת אלא חיבורים בין לקוחות. כל משתמש חדש מביא אליו דרישות משלו אף גם משאביים נוספים. השרת דינמית מאוד ולכן ניהול מסובך. ה-IP משתנים לכלל הלקוחות.

קיימים מודולי הכלאה בין השניים – כמו Skype, בהם יש שרת מרכזי ובנוסף קשרי עמיתים.



תקשורת בין תהליכיים

תהליך = תוכנית שרצה בתוך host.

בתוך מחשב, תהליכיים שונים מתקשרים בתקשורת פנימית. אך בין מחשבים שונים התקשורת שלוחים **הודעות** אחד לשני. גםemodel ה-**client**, מוגדר סוג של client ו- server ע"י אבחנה בין 2 המחשבים הינה יוזם הקשר.

כדי לתקשר, תהליכיים יוצרים **socket** = 'פתיחת דלת', כאשר שאר השכבות דואגות שהמידע ייעז בהודעות מה socket של client ל- socket של ה server.

זהו ה server נשעה ע"י **כתובת IP** ו- **port**. IP הוא כתובת המחשב ולכנן לתהליכיים שונים יש אותו IP, ויש צורך להוביל ביניהם – זהו ה port.

פרוטוקול – מצין את סוג ההודעה (כמו post/get למשל) והמבנה שלה, וכן כללים לגבי שגיאות ומענה להודעות.

ישנם פרוטוקולים פתוחים לציבור (כמו HTTP) וכאלו הסגורים לחברות מסוימות (ושוב, Skype).

מה האפליקציות דורשות משלוח החבילות?

חלקו דורשות העברת המידע **בשלמות**, חלקו מאפשרות איבוד נתונים בדרך. נ"ל לגבי השהייה מידע (שליחת מייל למשל יכולה להתעכב בשניה, בעוד שבמשחק אונליין מול יריב זה יכול לגרום להפסד...). כמו כן יש אפליקציות שדורשות הצפנה לעתים.

פרוטוקולים השונים נותנים מענה לדרישות אלו – זה מהו מein שימוש בשירותי משלוח שונים, ועל החבילה עצמה נמtag את השירות המבוקש.

TCP – פרוטוקול יקר אך אמין.

בפרוטוקול זה יש חיוויים על הגעת המידע באופן תקין, הוא נותן בקרות זרימה ועומס. הוא לא מספק מהירות, תפוקה וabetחה.

UDP – פחות אמין, פועל בשיטת "שגר ושכח". אך הוא מהיר וזול.

רוב האפליקציות משתמשות ב TCP גם אם הן יכולות להשתמש ב UDP בשל אבטחה – UDP פחות מזוינה ולא מאובטח.

היום, יש פרוטוקולים שבנויים על TCP אך מאובטחים יותר, כגון TLS.



Web & HTTP

Web page - מרכיב HTML בסיסי הכלול מס' אובייקטים (files) שלכל אובייקט URL משלו. האובייקטים יכולים להיות: file, JPEG, Java applet, audio file HTML וכו'.

HTTP – Hyper Text Transfer Protocol

פרוטוקול נפוץ שמספרת כיצד משתמש (לקוח) ושרת מתקשרים ביניהם עם"נ להעבר מידע. **Client**: דףן ש牒קש בקשות, ואת מה שמקבל – מציג כאובייקט web. **Server**: מגיב לבקשת ע"י שליחת אובייקטים (או הודעות שגיאה).

ה- HTTP עובד בשיטת TCP בשכבה הרכיבית. הלקוח client יוצר socket ל server ב **port 80**.
ה-server מתחבר לבקשת ההתחברות והודעות http מועברות.
ה server ב http לא זוכר מידע על בקשות קודמות clients. מצב זה נקרא "stateless".

Persist VS No-Persist

ב persist , דרך תקשורת אחת פתוחה של חיבור TCP client-server יכולים לעבור אובייקטים רבים.
ב No-persist , כל העברת אובייקט דורשת תקשורת משלה, ז"א שאחרי כל אובייקט שמועבר התקשורת סגרת ונפתחת מחדש בבקשתו של האובייקט הבא.
ה persist של http default זה

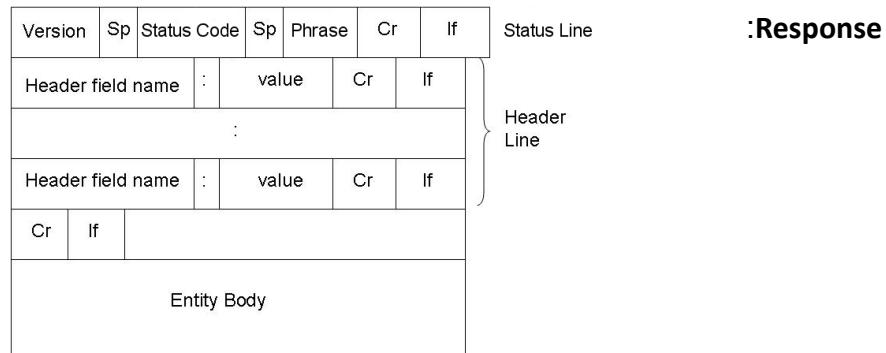
Request / Respond ל 2 http

method	Sp	URL	Sp	Version	Cr	If		Request line	:Request
Header field name		:		value	Cr	If		Header Line	
		:							
Header field name		:		value	Cr	If			
Cr	If								
Entity Body									



** \n If = סוף שורה,

.GET, POST, HEAD, PUT, DELETE :request של ה method



Status code עיקריים:

- התקבל, המידע בהמשך הודעה. OK - 200
- קיימת כתובת חדשה לאובייקט והוא מצוינית בהמשך הודעה 301 - הועבר לצמיות
- ה server לא הבין את הודעה Bad Request - 400
- המסר לא נמצא בשרת Not Found – 404
- גרסת HTTP לא תומכת בסוג בקשה שנשלחה. - 505

טוופים – בחלוקת לפי הספירה הראשונה:

- מידע	1xx
- הצליח	2xx
- נוטב מחדש	3xx
(לא ברור/לא נמצא/נשלחו יותר מדי בקשות וכו')	4xx client error -
server error -	5xx



Cookies

שימושים לזיהוי clients בכניסה לשירותים מסוימים, כגון: shopping carts, mail, תגיות וכו'.

כיצד זה מתבצע? – השומר ב"קצה החיבור" – ב- sender או ב- receiver. CNSNLLHT הודיעת http היא מכילה את state של אותו cookie.

איפה מופיע ה cookie ?:

Header line .1 של תשובת ה http

Header line .2 בבקשת הבאה של http

Cookie file .3 שנשמר ב host של user, ומונוהל ע"י הדף של user.

.web server Database .4 שנשמר ב

ה cookie מוגדר בכניסה הראשונה לשרת מסוים, ונשמר אצל הלוקה לפחות פעמיים הבאים שייכנו לשרת.



הרצאה 3

Web proxies

= "שרותי עזר" שנועדו להפחית את העומס מהשרת, אליו שיש מעין "צוואר בקבוק" בשל המס' הרב של הבקשות ברחבי העולם. ה- proxies אוספים מס' בקשות מחשבים ושולחים אותם יחד לשרתים הראשיים. נבחן כי ה proxy מתפקיד כשרת עבור לקוחות, אך כ client עבור השירותים.

יתרונות:

- לקוחות מקבלים מידע בצורה מהירה יותר, כיוון שהproxy שומר אצלם מידע שUMBOKASH פעמים רבות.
- מorigד עומס מהשרת.
- קרוב ללקוחות באופן פיזי וכן בקטע הקצר שבין הלוקחות לעproxy אمنם מועבר מידע רב, אך בקטע הארוך מהproxy לשרת עבור מידע מועט בשל ארגונו ע" proxy, ולכן התגובה עיליה יותר.

ללקוחות שהם בעצם חברות גדולות (כמו האוניברסיטה), עליה להיות proxy ביציאה מהרשת של האוניברסיטה שיחובר לשרתים בחוץ, ואפיו הוספה proxies בתוך הרשת עצמה.

אתגרים:

- חייב לוודא שכל החבילות עוברות דרכו
- זמן עבודה מיותר שלעיטים נעשה בעproxy – אם הוא חיפש מידע/cache והמיד לא קיים תחזוקת cache באופן יעיל
- Security – ה proxy אוסף מידע על הלוקחות וכן פוגע בפרטיות!! (מצד שני, מונע מגישה של השירות למידע על הלוקה כי אין ביניהם גישה ישירה).

** proxy יכול לשמש גם cache filter ולחסום גישה לאתרים מסוימים (- אינטרנט מוגן)

Cache coherency

כיצד מודאים שהמידע שיש ב-cache מעודכן?

יש אפשרות להוסיף cache 'תנאי' שיתווסף לבקשת הקט proxy לשרת, שבודק אם המידע שבידיו מעודכן. אם כן – נחסכת העברת המידע, ומוחזר מהשרת לעproxy רק עם תשובה חיובית. אם לא – המידע יישלח.

אבל, אם ה-cache לא מעודכן מספיק – אז proxy רק מעכבר את הבקשה. לצורך כך יש Bloom Filter.



Bloom Filter

שאילתא מבוססת hash שעונה במתירות בcn/לא לשאלת 'האם המידע נמצא בcache'. כדי לשמור על עדכניות המידע, השרתים שלוחים מדי פעם bloom filter מכועז עם מידע ספציפי שהתעדכן.

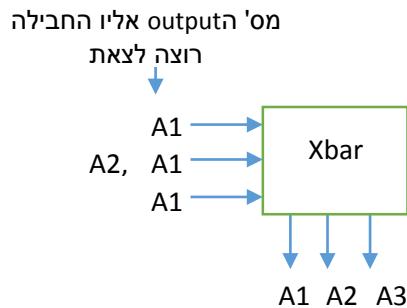
Pipeline

גרסת 1.1 http://1.1 מאפשרת pipeline, אך צריכה לשומר על סדר החבילות (חיב' שהמידע עברורה לא נמצא בcache וצריכה לעבור לשרת – מול חבילה שהגעה אחרת אף מסתפקת בcache בלבד וכן מוכנה לחזור ללקוח עוד לפני החבילה הקודמת....)
גרסת 2.0 http://2.0 – גם מאפשרת pipeline אך ייעלה יותר.

בפועל, רק האובייקטים שבדף מועברים בpipeline, אך לא הבקשה לדף.
בעיה שנוצרת בpipeline: במקרים של 'צומת', חבילות שרוצות לפנות לכיוון א' בצומת יכולות להיתקע מאחוריו חבילות שעומדות בתור לפניה לכיוון ב' בצומת, זאת למחרות שפניה א' פניה לחולוטין ויצא שהחbillות עומדות במקום לחינם.
פתרונות: הרחבת הציר שלו, כך שהיא נתיב לכל פניה בצומת.

בעיית ה Head of Line blocking

בכל רגע נתון, יכולה חבילה אחת להיכנס מכל input של xbar וחבילה אחרת בלבד יכולה לצאת בכל output. בעיה: במקרה של (לדוגמא) 3 חבילות של כל מהן ב- שונה ורוצות להיכנס לאותו ה – כיון שאין יכולות להיכנס אליו במקביל, חלוקן נאלצות לעמוד במקומות עד שאותו ה יתפנה, וייתכן שהן תוקעו אחריה בתור ב חבילות שסנתיניות ל אחר!

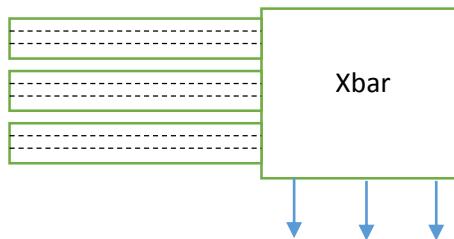


פתרונות: Virtual Output Queue

העברת התור לנק' היציאה של ה outputs, כך בדוגמה של מעלת 3 החבילות מסוג A1 יעברו במקביל לתור שימוקם ביציאה לoutput מס' A1, וחבילה A2 תוכל לעבור ליעדה.

השיטה הנ"ל היא שיטה שאליה שוופים, אך בעיתית כיוון שעדין יש זמן עיבוד חבילה לפני שמעבירים אותה מ לoutput ולכן במקרה יש המתנה של חבילות בתור של ה outputs.

פתרון נוסף: לפצל כל למס' נתבים, לפי כמות הדטטים של אותו xbar.



אבל גם זה פתרון בעיתתי, כי הוא תלוי במס'outputs, ויצא שיש n^2 מסלולים (תורים) ל.

פתרון (אחרון....) : להגביל כל קבוע > 1 של נתבים. למשל, עבור 1,000, חלקם – לחלק כל ל-10 נתבים.

DNS = Domain Name System

שרות היררכי ובו database כל כתובת URL לכתובת IP. למה היררכי?

- למקשה של תקלה בשרת
- לשחרר/להפחית עומסים
- יותר קל לתחזק ולעדכן
- שרת אחד אינו scale.

כפייליות כתובות:

ישנן כתובות URL שימושים רבים מזינים, אך אין הכתובות המקוריות – לדוג'-
משתמשים רבים מזינים את הכתובת www.gmail.com, בעוד שכתובת הURL המקורית
הינה www.mail.google.com. כיוון שהכתובת הלא-מקורית שימושה מאוד, DNS ירצה
לשומר גם אותה ולא להסביר לכל בקשה צוזו שהכתובת לא נמצאת.
בנוסף לכך, ישנים שרתיים שמחזקים מס' כתובות IP עבור URL בודד, וגם במקרים אלו על
הDNS לטפל.



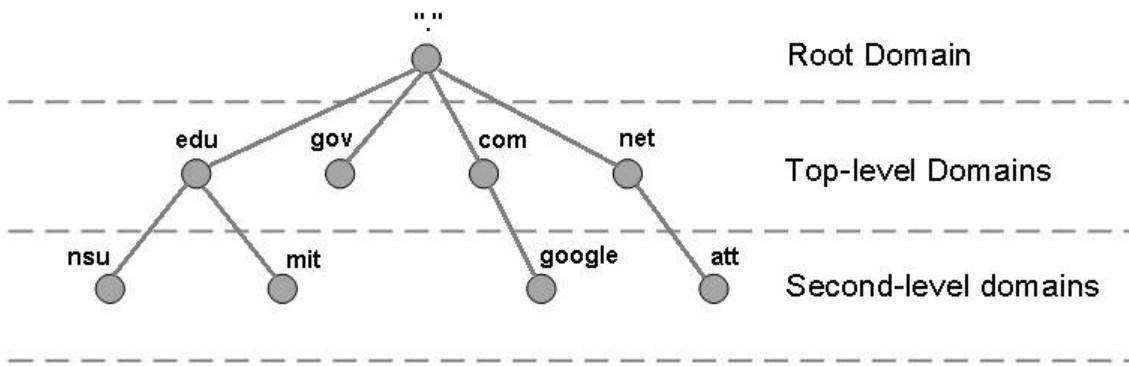
כפתרון לכך, מוגדר ה Authoritative DNS

שרות המחזיק מס' כתובות IP לכל URL, גם כדי לתרמן בין עממים. ה DNS Authoritative שופנה אליו כתובות IP אקראית מתוך רשימה IP של אותו URL, או מחזיר לכל local DNS שפונה אליו כתובות IP אקראי, והוא- DNS local יבחר להשתמש בIP הראשוני ברשימה, ואם השירות עמוס- יעביר לו הבא וכו'.
חלופין מחזיר את כל כתובות IP אך בסדר אקראי, ובנוסף לדף המבוקש, לעיתים לכל אובייקט בדף יש URL משלה. אובייקטים אלו נשלחים כבקשה ע"י browser ללא התערבות המשתמש.

2 דרכי להפחית מס' הגישות הלא-הכרחיות לביצוע DNS Query:

1. שימוש בעroxod, שלעיתים יחזיק בעצמו cache ולייעיתים רק יצמצם חבילות
2. שימוש בcache DNS.local

היררכיות database של ה DNS



Root – שרתים מרכזיים, יודעים רק את כתובות השרתים של ה TLD. (קיימים רק 13 כאלה בעולם).

TLD = Top Level Domain – כל שרת מחזיק את הכתובות של כל השרתים שבאותה הסיוומת.

SLD = Second Level Domain – השרתים עצם.

cash-client מבקש כתובת IP (לפי URL), root מפנה אותו לDNS Server האחראי על הכתובות מסויימות בסיוומת המבוקשת. אותו שרת בשכבה ה TLD יבצע שאלתה ויפנה את הליקוח ל **Local DNS** של השירות המבוקש, ורק אותו local ימסור ללקוח את IP שחייב.



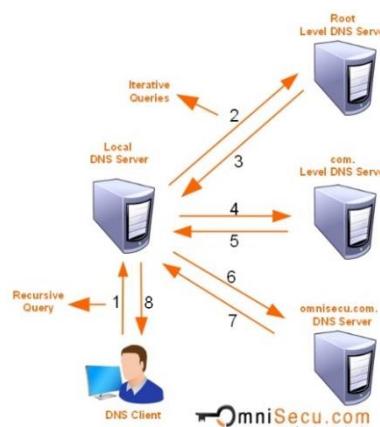
ה אל Local DNS מוקם ומוגדר ע"י השרת של החברה והוא חלק מההיררכיה, ומשמש מעין proxy בעל cache משלהו, שיכל להיות לא מעודכן.

כש משתמש קצה (ה browser בבית, למשל) מבקש מה-Local DNS שלו לקבל כתובת IP, הוא משתמש Iterative VS Recursive באת המידע באמצעות שתי הדריכים: Local DNS מחפש את המידע באמצעות שתי הדריכים:

- Iterative

ה Local DNS מבקש כל פעם את המידע מהשרת אליו הוא עצמו מופנה. ז"א, ה Local פונה ל Root, וה Root מחזיר כתובת של TLD. ה Local ב עצמו פונה לשרת ב TLD, והוא מוחזיר לו את כתובת ה Authoritative. ה Local פונה לאוטומטית והוא מוחזיר לו את ה IP המבוקש, והוא Local יכול לספק אותו ל browser.

** השירותים הם כמו "פקידים עצניים", שאומרים: אני לא יודע את התשובה, קח מס' טלפון ותשאל אותו....



- Recursive

כל שירות DNS שמקבל בקשה – פונה לשרת המתאים שיכל לספק תשובה, וմבקש ממנו לחפש לו את התשובה ע"י פניה לשרת הבא שתחתיו.

** כל שירות מתפקיד מעין "פקיד חוץ" – שלא מביא לך טלפון אלא מתקשר בעצמו ובסוף יביא לך תשובה....

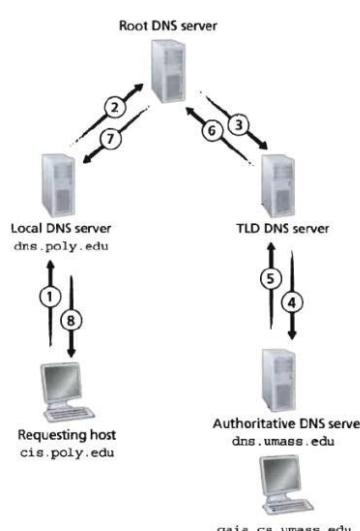


Figure 4. Recursive queries in DNS



כדי לשמר על אמינות המידע, לכל URL ששימור ב cache יש timeout שסופר מתי הפעם הבאה שיש לעדכן את המידע, ומחושב לפי צפי לשינויו IP / URL. שעון זה נקרא = TTL = Time To Live.

DNS RR = Resource Records

משמעות פרוטוקול של ההודעות המוחזרת לבקשת לפן.

הפורמט הוא אחיד, ומהצורה: .Name, Value, Type, TTL

כאשר: name = כתובת URL המבוקשת, value = כתובת הIP המוחזרת, type הוא אחד מהබאים:

• A = תקין.

דוג': (www.google.com, 1.213.56, A, 3600s)

• NS = מודיע שאותו URL הוחלף לאחד חדש. מציין הודעה מהצורה: (old URL, new URL, NS)

ומיד אחרת מכרף הודעה מסוג A עם המידע.

דוג': (abc.com, dns.abc.com, NS)

(dns.abc.com, 2.89.513, A, 3600s) - ומיד -

• CNAME = כשמה URL בצורה חד-משמעות אך זהו אינם URL המקורי. מציין מידע מהצורה: (typed URL, origin URL, CNAME)

ומיד אחריו הודעה A רגילה.

דוג': (walla.com, walla.co.il, CNAME)

(walla.co.il, 50.134, A, 3600s) - ומיד -

• MX = זהה למקורה של CNAME, אךushman בכתובות של אתרי מייל. (gmail.com, mail.google.com, MX) דוג': (mail.google.com, 70.89.1, A, 3600s) - ומיד -



הרצאה 4

P2P – Peer To Peer

מודל קשה יותר לניהול כיון שאין שרת מרכזי אלא קב' דינמיות.

نبיט בזמן העלאת קובץ F מהשרת להורדה ל-N ללקוחות :

במודל client-server – השרת מעלה N פעמים קובץ בגודל F, וכל לקוח מוריד את הקובץ בקצב שלו.

$$\text{זמן הכלול הוא: } \max \left\{ \frac{(N*F)}{u_s}, \frac{F}{d_{min}} \right\} \leq D . \text{ כאשר:}$$

u_s = זמן העלאה של השרת, Upload Server = u_s

d_{min} = זמן ההורדה של הלוקות הcy AiTi. Download Minimum = d_{min}

במודל P2P, לעומת זאת – השרת מעלה את הקובץ פעם אחת, והлокות הcy מהיר יכול כבר להוריד את הקובץ ולהתחליל להעלות אותו במקביל לשרת עבור שאר הלוקוחות. לאחר שיש 2 שמהווים source, לקוח נוספת שסימן להוריד יכול להפוך גם הוא ל-source נוסף! כך מס' ה servers' הולכים וגדלים וזמן העלאת הקובץ ע"י השרת הוא כבר לא N פעמים, אלא $\frac{N*F}{u_s + \sum_{i=1}^n u_i}$, ושאר ההעלאות ע"י הלוקוחות – $\frac{F}{u_s + \sum_{i=1}^n u_i}$.

$$D \geq \max \left\{ \frac{F}{u_s}, \frac{N*F}{u_s + \sum u_i}, \frac{F}{d_{min}} \right\}$$

נבחן כי במודל P2P, ככל ש-N גדול יותר (מס' הלוקוחות) – כך גם מס' המעלים גדול יותר – $\sum u_i$.

יתרונות P2P:

- מהיר יותר להורדה (=זמן ההגעה של הקובץ ללקוח)
- עלויות נמוכות יותר
- מיד לאחר העלאת הקובץ ע"י השרת – יש 'ביטחון' מפני תקלות בקובץ כי יש עותק נוסף.
- פחות עומס על מקום אחד



BitTorrent

משתמש שברשותו קובץ מסוים רוצה להעלות קובץ לרשת של לקוחות הנקראת *swarm* (או *torrent*). משתמש זה, שמהווה מקור – נקרא *seed*, מחלק את הקובץ לחבילות קטנות ומחיל להעלות אותן אחת-אחרת. העמידים המוקשרים ברשת מקבלים חבילות ועם סיום קבלת חבילה אחת הם כבר יכולים לשתף את מה שהם קיבלו, כאשר במקביל הם ימשיכו לקבל את שאר החבילות. גם עתה שכבר סיימם להוריד יכול להמשיך ולשתף ברשת.

כשלקוח רוצה להצטרף לקב' קיימת, הוא מקבל מהשרת את רשימת העמידים המשתפים בקובוצה, והלקוח מצטרף אליהם ובוחר מהם לקבל את המידע (אלגוריתם המסתמך על מהירות קבלת החבילות). ברגע ההצטרפות, עם"ג שהלקוח יידע להשלים את כל הקובץ – הוא מקבל מחלק מהחברים רשימות של איזה חבילות כבר קיימות אצלם. בנוסף, הלקו החדש יחפש אחר חבילה hei' נDIRA' (הכי פחות משותפת) כדי להוות מקור נוסף לחבילה זו.

כדי שימושים יעלו את הקבצים ולא יהיו סנובים, מתבצעים 'תגמולים'. למשל – המשתמש החדש מעלה חבילותഴרה לאלו שלחו לו (Nazcor כי לא בטוח של אלו שלחו לו יש את כל החבילות, וכי הלקוח החדש קיבל אותם ממשתפים אחרים...).

כמו כן, כדי שלא יוצר רק 'מעגל' אחד של משתפים וכל שאר הלקוות יהיו 'מנודים' – מעבר לחברים המתוגמלים, הלקו החדש בוחר איזשהו עמית משאר חברי הקבוצה ומקצתה לו תקשורת מהירה, וכך הקשר ביןיהם יתחזק והמctrfr החדש ישלח חבילות לאותו עמית, והעמית כמובן ישלח חבילותഴרה למctrfr החדש...

...ובקצרה:

מצטרף חדש לקובוצה מתחבר לרשימת משתפים פופולריים וגם מתוגמל אותםഴרה, בעוד הוא מhapus אחר חבילה hei' נDIRA' ומנסה להשיג אותה, ולטיזום פותח תקשורת מהירה עם חבר מנודה כדי לחזק את הקשר.

כדי להבטיח שהחבילות הן באמת חבילות של הקובץ ולא spam כלשהו שמתבזה, ה*seed* יוצר Torrent Descriptor ש מכיל hash cryptographic של חבילות הקובץ.



FTP = File Transfer Protocol

פרוטוקול המאפשר העברת המהירה ויעילה בין שרת ללקוח (כאשר שרת יכול להיות 'עמית'). ה-FTP פותח 2 קשרי TCP: אחד להעברת המידע, ואחד להעברת הבקרה (כגון: סיסמה, path, מידע נוספת לקשר לאבטחה וכו'). בזרה זו קשר המידע יעל יותר כיון שימושו של port בלבד ולא את שאר המידע הנלווה. קשר הבקרה נשאר פתוח עד לסוף העברת המידע. ה-port המשומש לכך הוא 21.

בפרוטוקול זה השירות שומר אצלו מידע נוסף תחת "state" לכל לקוח, וכך נוצר עומס על השירות.

Email, SMTP = Simple Msg Transfer Protocol

3 רכיבים מרכזיים בהעברת מייל:

- **User Agents** – תוכנים מקומיים אצל המשתמש (כגון: outlook, iphone וcdf)
- **שירותי מייל** – משמשים גם כ-client וגם כ-server – כיוון שיש שליחת וקבלת של מיילים
- **תקשרות בין שירותים מייל – SMTP**

הטוכן המקומי מחזיר תור של מיילים לשילוח וכו' שומר אצלו מיילים שהתקבלו עד שהשלקו פותח אותם (עד ש"מושך" אותם מהטוכן).

3 שלבים להעברת המייל:

1. **פתיחת קשר וידוא המען והנמען**
2. **שליחת המייל**
3. **סיום**

SMTP משתמש ב-TCP ב-**port 25**. הפקודות נשלחות ב-ASCII והתשובות – כ-**status code** או **phrase**.

שלבי העברת המייל – מן הפקודה אצל השולח ועד לקריאה אצל מקבל:

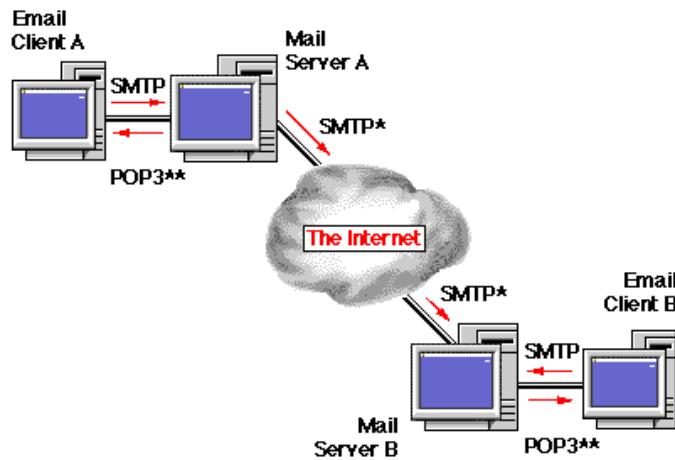
1. משתמש A שולח הודעה ל-mail server שלו שברצונו לשלוח מייל.
2. ה-mail server מכניס את ההודעה לתור לשילוח
3. ה-mail server של משתמש A פותח חיבור TCP עם ה-mail server של משתמש B –

זהו ה-SMTP

4. החיבור עוברת mailbox B שם את ההודעה ב-
 5. ה-mail server של משתמש B יರצה לקרוא אותה agent פונה ל-User agent שלו. לצורך כך יש
 6. כשהמשתמש ירצה לקרוא אותה הוא פונה ל-User agent שלו. לצורך כך יש
- IMAP = Internet Mail Access Protocol**



תרשים ה SMTP:



כiom, כיוון שהmaiL מבוסס web – ה"קצוות" (השלוח והמקבל) מתחברים לשרתים המail ב http, ורך החיבור בין שרתים mail הוא ב SMTP.

SMTP VS. HTTP

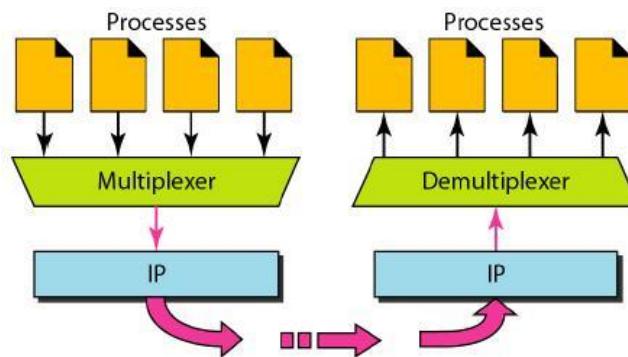
SMTP	HTTP
פעולות Push – שלוח MaiL	מבצע פעולה Pull – מבקש אליו מידע
moveber בפורמטים שונים	
קשר עקבי – persistent	לא מחיב עקבות, מאפשר יצירת קשר בכל פעם מחדש
מעביר מס' אובייקטים בהודעה	כל הודעה מכילה אובייקט אחד

---- סוף שכבת האפליקציה ---



שכבה התעבורה

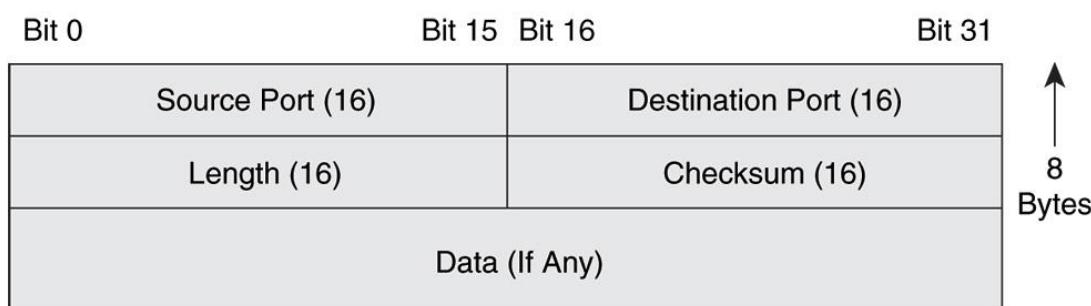
שכבה זו מבצעת ריבוב – multiplexing (de). אחראית על קבלת מידע מ ports שונים והפיכתם לחבילות לשילחה, ולהפר בצד מקבל – פירוק החבילות ומיון ל ports השונים. מהוּה תקשורת לוגית בין התחליכים.



UDP – User Datagram Protocol

פרוטוקול לא אמין, ד"א שיתכנו חבילות שאבדו בדרך, וכן החבילות שהגיעו ליעדן – יתכן והגיעו בסדר הלא נכון. איןנו מתחשב בעומסם בדרך.

از מה כן טוב בו? שהוא זול ומהיר – איןנו מציריך הקמת תקשורת בין לקוח ושרת ("לחיצת ד"ימ"), יש לו header קטן- 8 bytes בלבד :



No Sequence Or Acknowledgment Fields

ה checksum – מהוּה בקרה על תקינות המידע שהגיע. לחבר את כל bits של המידע ואז הופך את כל הספורות – ומשווה עם הערך שאצלו.



עקרונות העברת מידע

הגדרות חשובות:

RTT = Round Trip Time – הזמן שלוקח לחבילה לעבור הלוֹ-חזר מclient לserver (חזר – עברו התשובה).

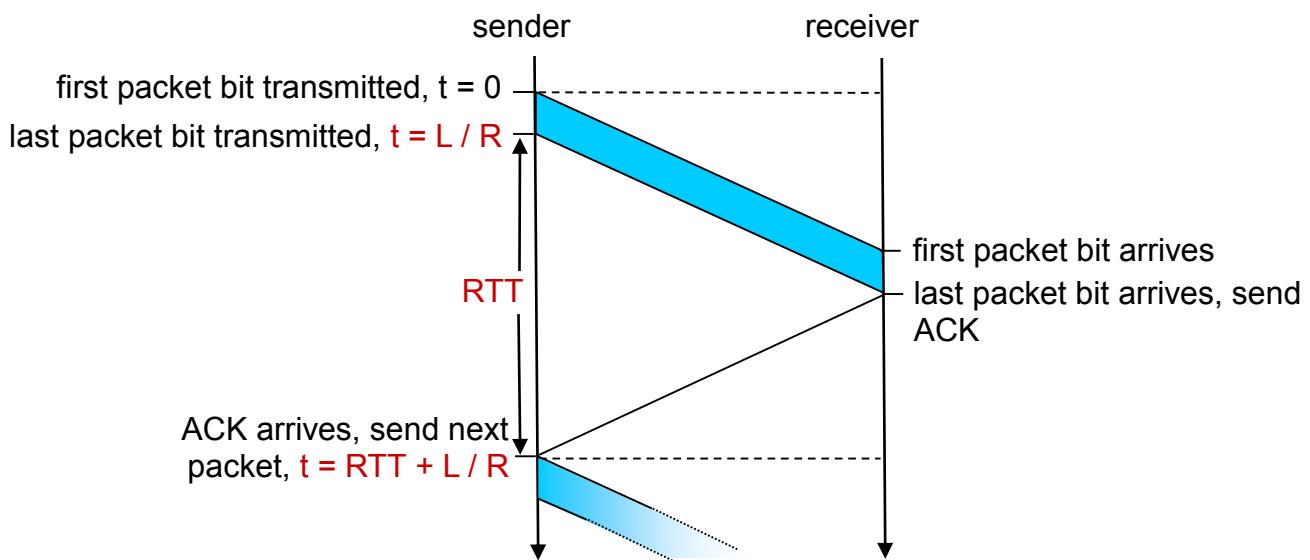
חיוי שחבילה הגיעה ליעדה. = Ack = Acknowledge

3 דרכי העברת המידע:

Stop & Wait . 1

אחרי כל חבילת ששלחת, השולח מchecker לחיוי מהמקבל שהחבילה הגיעה ליעדה,

ורק אכ"כ שולח את החבילת הבאה. ניצולת מאוד נמוכה של הקוו: $\frac{L}{RTT + \frac{L}{R}}$



יתכן כי חבילה תאבד בדרך ואז לא יגיע חיוי (או לחłówין ack אבד ולפניהם לא הגיע). לכן מוגדר זמן timeout שמאוזナル מחדש לאחר כל שליחת חבילה, ואם לא הגיע ack בתור זמן(timeout – אז חבילה תישלח מחדש).

בנוסף, כדי לדעת על איזו חבילה הגיע ack וליזודא שזו ה ack שהוא ממתינים לו ולא אחד קודם שהתעכבר בדרך – יש מס' סידורי לכל חבילה.



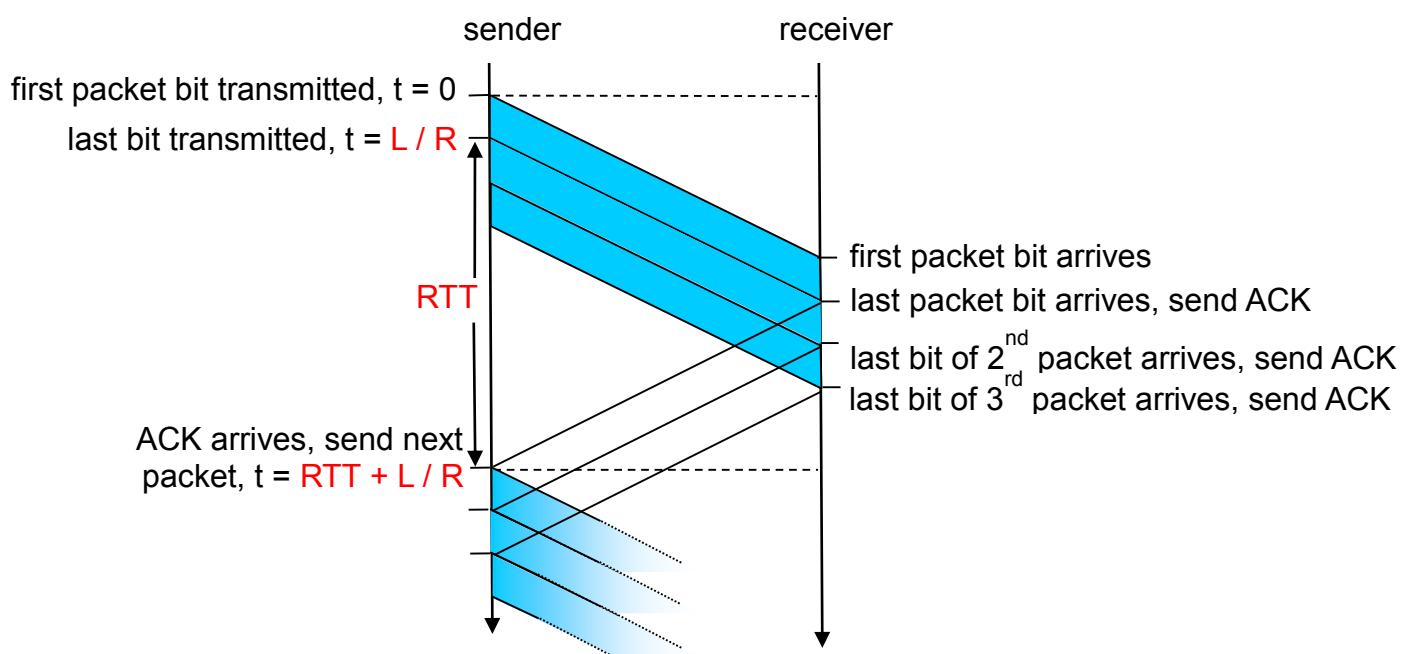
Go Back N .2

העברת מידע בשיטת pipeline – שליחה מקבילה על הקו. במקרה זה לשולח יש חלון בגודל N של חבילות שהוא יכול לשלח, ולן שולח N חבילות במקביל. לעומת זאת, הצד מקבל מצפה להגעת חבילות אחת-אחד עפ"י הסדר, ולכן בכל רגע נתון הוא מצפה לקבלת אחת מසימנת שתתקבל, וברגע שהתקבלת – הוא שולח ack עם המס' הסידורי שלה. שליחת ack בעצם מהו אישור גם על הגעת כל חבילות שלפני חבילה הנ"ל, כיון שחבילות שגויות הגיעו למקבל כאשרינו מצפה להן – נזקנות על ידו.

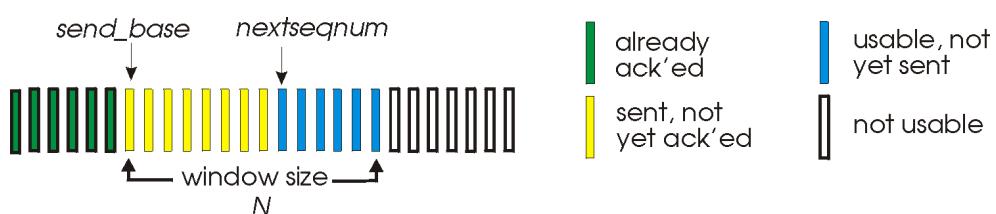
על כל ack מתקבל, הוא יכול להציג את החלון ולשלוח חבילות נוספות. גם פה מופעל timeout ומאותחל בכל פעם וחינוי מגיע. אם עבר timeout מבלי שהגיע חינוי על החבילה הici 'ישנה' בחולון (זו שנמצאת בחולון הici הרבה זמן) – כל N חבילות שבחולון יישלחו מחדש, כי במליל אם הין הגיעו לעד – הין נזרקו ע"י המתקבל שהמתין לאותה חבילה שאבדה.

עמ"נ שלא תהיה חפיפה בין מס' סידורים של חבילות שהתקבלו בעבר לבין חבילות שהשלוח מתין לACK מהן – עבור חולון בגודל N יש צורך בטוחה של N^2 מס' סידורים לחבילות.

חיסרון : על כל חבילה שנאבדה – יש צורך בשילחה מחדש של N חבילות...

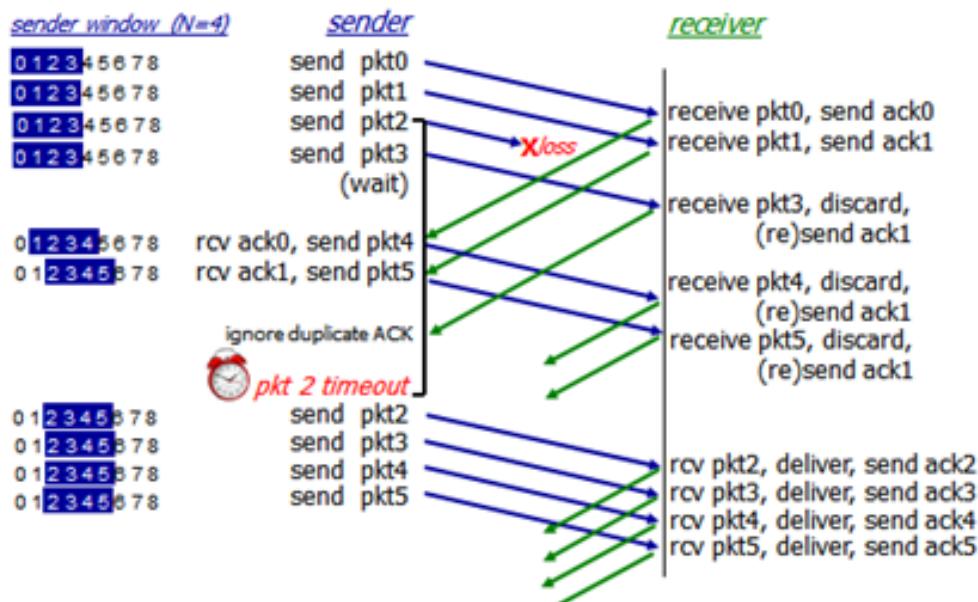


החולון נראה כך :



תרשים מפורט יותר, כולל את החלון וחבילות שאבדו:

GBN in action



הרצאה 5

Selective Repeat .3

גם כאן יש שימוש בpipeline. העיקרון דומה ל-N Go Back, אך במקרה זה יש חלון גם אצל ה-Receiver, וכל עוד הוא מקבל חבילות שבתווח החלון, גם אם לא לפ' הסדר – הוא שומר אותן ושולח Ack. כיוון שה-Ack לא נשלח עפ' סדר הגעת החבילות, והגעה היא לא בהכרח לפ' הסדר – אז הגעת Ack לשולח לא מחייבת שכל החבילות שלפני אותו Ack שנשלחו הגיעו ליעדן. כמו כן לא מוגדר טימר יחיד, אלא טימר לכל חבילה.

מצד השולח – יש חלון שליחה, ויתכן שחלק מהחbillות כבר קיבלו Ack וחלק לא. החלון יוזץ רק כאשר חבילה ה'ישנה' ביותר קיבלה חיוי. בפועל, חיוי שנשלחה כשותפה מתקבלת – מכיל את המס' הסידורי של חבילה שהמקבל מצפה לקבל, ז"א – חבילה שהוא ממחכה לה הכי הרבה זמן. لكن, אם יתקבלו מס' חבילות אחריו חבילה X מסויימת שלא הגיעה – כל Ack על חבילה יכול את המסר "מצפה לחבילה X". השולח יראה שקיבל Ackים על אותה חבילה – ויבין שהיא אבדה בדרך ויסלח אותה מחדש. שליחה חוזרת של מס' Acksm סימכילים אותו מס' סידורי נקראת "**duplicate Ack's**".
אפשרות נוספת, זה הגדרת טימר לכל חבילה שנשלחה.

ברגע שהשולח מקבל Ack על חבילה ה'ישנה' ביותר – הוא יכול להזיז קדיםמה את החלון. נבחין כי יתכן וכבר קיבל Ack על מס' חבילות שנשלחו אחריו אותה חבילה, ולכן יוכל להזיז את החלון מס' מקומות ולא רק מקום אחד.

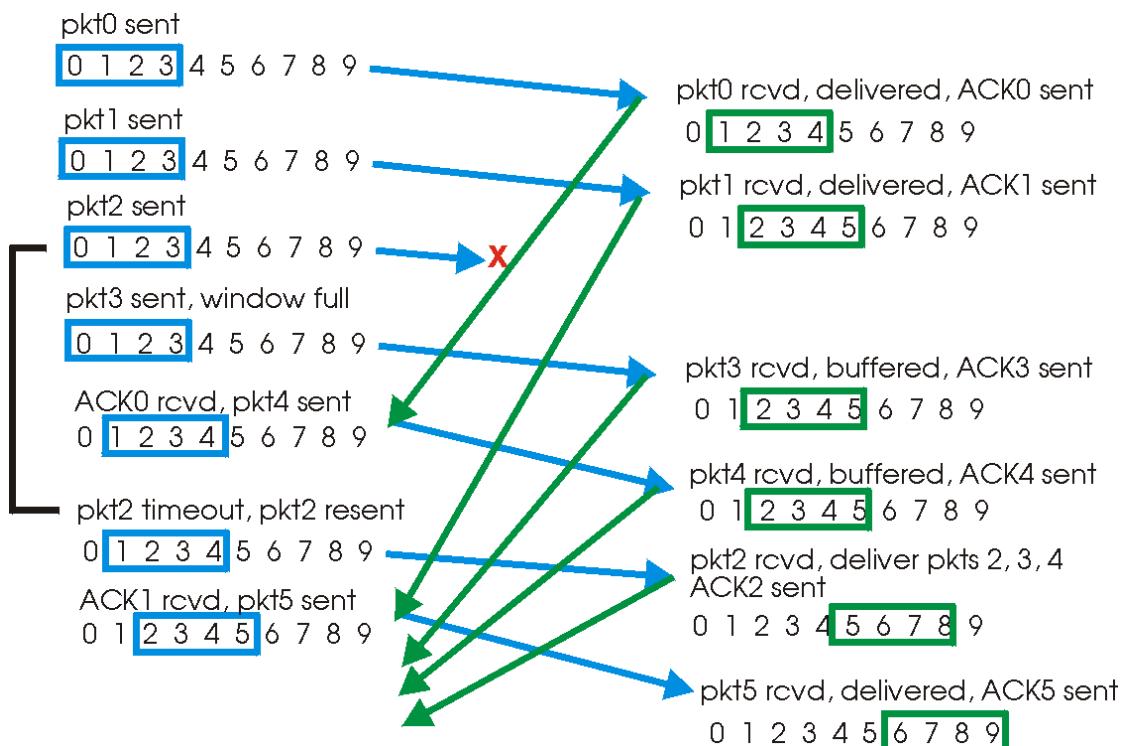
מצד מקבל – אחרי שבחbillות התקבלו הוא שולח Ack ומzie' את החלון לפ' הצורך (-אם זו חבילה ה'ישנה' ביותר). אך יתכן שהמקבל הזיז את החלון, ו-Ack מסויים שנשלחו אבד בדרך. השולח מבינהתו לא קיבל את Ack ולקן מסיק שהחbillה לא הגיעה לעדדה, ושולח אותה שוב! וזה מקבל יקבל חבילה שנמצאת כבר מוחוץ לחלון... במקרה זה, ה-Receiver ישלח שוב Ack על חבילה הנ"ל. כדי שלא יקרה מצב שישלו Ack על חבילות ישנות מדי שכבר אין רלוונטיות, או אולי שהמס' הסידורי שלhn כבר שוחרר והוגדר מחדש עבור חבילה אחרת ויוצר בלבול – מוגדר שהמקבל יוכל לשולח Ack רק על חבילות עד מרחק N מהחלון.

נבחין כי בכל פעם שהחלון של מקבל ZZ – זה מעיד על חבילה או רצף של מספר חבילות שהגיעו אליו והן מסודרות לפי הסדר, ולכן יכול להעביר אותן ללא אפליקציה.

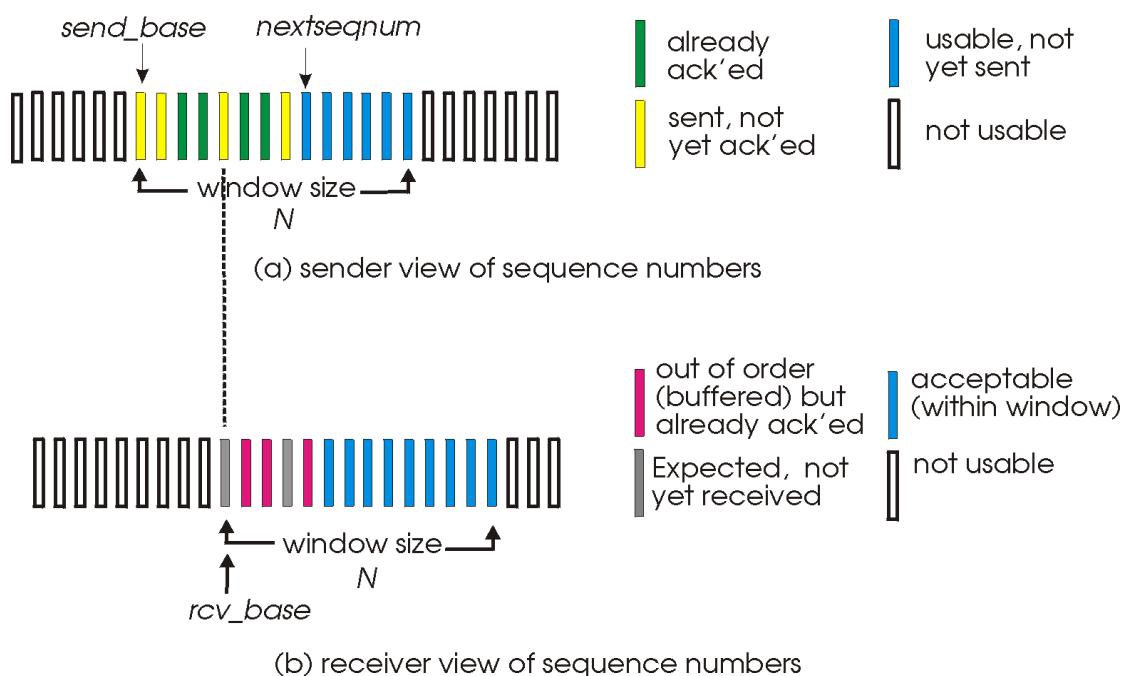
שיטת זו פותרת את בעיית הניצול הנמוכה של N Go Back, כיוון שב GBN תמיד ממתינים לחווים, וכך יש חלון מספק גדול לשולח מידע עד להגעת חיוי בודד.



תרשים Selective repeat (מעבר חלון בגודל 4):



והחלונות אצל השולח ואצל המקלט יהיו:



BDP

כדי שኒצולת של קוו תהיה 100%, נגידר את הchlון להיות:

$$\text{Bandwidth} \times \text{Delay} = \text{Product}$$

מדוע יש צורך במציאות ניצולות אופטימלית?

אם הניצולות נמוכה – יש בזבוז של משאבים שימושיים לשילוח מידע מועט.

אם הניצולות גבוהה – אז יתכן שהמידע "צפוף מאוד", וכל עיכוב קטן תוקע את כל החבילות.

מושגים נוספים (שלא הוזכרו בעבר):

קצב שידור – **Bandwidth**

W – גודל הchlון

תפוקה, נמדד ב Mbps – **TP = Throughput**

UTH = U – ניצולות (אחוז השימוש בקו)

סימון ל-Transmission – **Txmt**

TCP – Transmission Control Protocol

פרוטוקול תקשורת בשכבה הרביעית, מתאר קשר בין 2 רכיבי קצה בלבד. ב프וטוקול זה לא יהיה שימוש בעת שליחת מידע מרכיב אחד למספר רכיבים, אך יתכן כי שני הצדדים שולחים מידע. TCP הוא אמין, מקפיד על סדר הגעת החבילות ומוסות אותן כדי להימנע מעומסים. כדי ליצור תקשורת נפתח socket לאפליקציה, שמזויה ע"י כתובת IP + מושך port.

הפרוטוקול משתמש בשיטה משלבת של N-Go Back Selective Repeat – לוקח את הצד הטוב מכל עיקרון: יש לו חלון בגודל N בצד המקבל וכן טימר יחיד (נקרא מ Selective Repeat), אבל ack הוא accumulative (נקרא MNBG). בהמשך נסביר כיצד זה מתבצע.

בצד השולח- TCP מקבל מהשכבה מעליו רצף של Bytes ומעביר אותן לשכבה מתחתית בצורת חבילות.

בצד המקבל – TCP מקבל מהשכבה מתחתית חבילות אך מעביר לשכבה מעליו רצף של Bytes, לאחר שסדר אותן לפי הסדר.



TCP header

TCP Header																																								
Offsets	Octet	0								1								2								3														
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31							
0	0	Source port																Destination port																						
4	32	Sequence number																Acknowledgment number (if ACK set)																						
8	64																	Window Size																						
12	96	Data offset	Reserved	N 0 0 0	S R E G K H T N	C W C R E G K H C S S Y T N F I																	Window Size																	
16	128	Checksum																Urgent pointer (if URG set)																						
20	160	Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)																...																						
...	...																																							

פירוט רכיבי ה-TCP header

mos' port – מושך בצד של השולח – מגדר לאיזו אפליקציה במחשב מיועד传输 החיבור. במקרה שני הצדדים שולחים מידע, במקום זה יציין מושך port של הרכיב ממנה בשלחה החיבור.

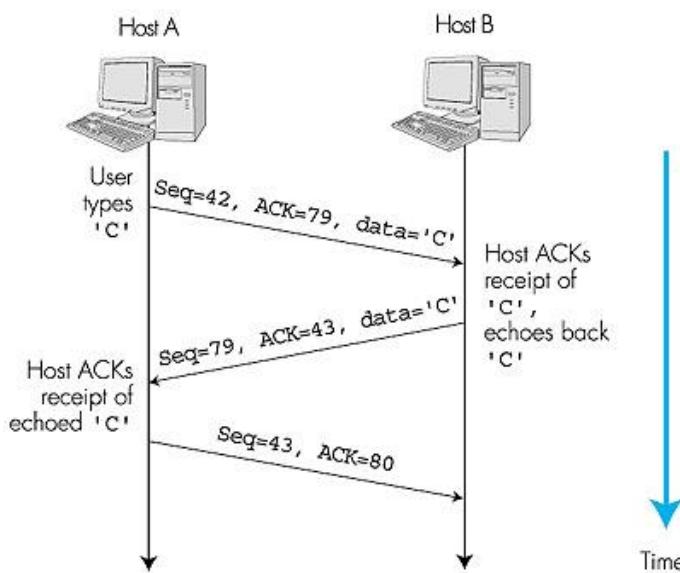
mos' port – מושך בצד מקבל. בקשר דו-כיווני, יציין מושך port של רכיב הקצה אליו בשלחה החיבור.

Sequence number – מהוות מושך ייחודי לחיבור, אך לא החיבור היא זו שממוספרת – אלא Bytes של המידע. ז"א, שכאשר אפליקציה רוצה לשЛОוח מידע – היא מגירה באופן אקראי מושך עברו Byte הראשון של המידע (ראינו בעבר – שהמשם מוגREL אקראיות לצורך אבטחת מידע), ושאר Bytes בחיבור ממוספרים בסדר עולה החל ממשם זה. בחיבור שנשלחת ייתכן ווילוח Byte בודד של מידע או כמה Bytes יחד, והוא Sequence number יכול את מושך Byte הראשון (=המשם הנמוך ביותר) שנמצא בחיבור זו. עפ"י המשם בשדה זה יישלווח Acks בהמשך.

Acknowledgment – אלו Acks שהוזכרו עד כה, מהווים אישור קבלת על החיבור. המשם sequence number**הבא** שהרכיב מקבל מצפה לקלות מהשלוח.

דוגמא – הצד מקבל קיבל חיבור שמשם sequence number הולא 72, וגודל המידע שהוא הוא 5 Bytes. אז הצד מקבל ישלח Ack שבו יציין המשם 77, כיוון שהוא קיבל כרגע את Bytes מושך 72-76 (כולל 2 הנקודות- זה יוצא 5, תספרו), וה-Byte הבא שאותו הוא מצפה לקבל הוא 77.





** בדוגמה הנ"ל, שני הצדדים שולחים ומקבלים מידע. A שלח לB חבילה עם sequence = 42 number, ו- Ack שמודיע ש"מצפה לקבל מ马克 את חבילה 79". במקרה זה שני הצדדים שולחים אותו מידע – 'C', אך זה לא חילק מהפרוטוקול. B, שקיבל את חבילה מס' 42 בגודל Byte 1 – שולח Ack "מצפה לקבל את 43", ובמקביל גם מכיל את מס' החבילה שהוא בעצם שולח – 79. A מוחזיר תשובה – חבילה מס' 43, שהוא גם Ack שאומרים "קיבלתី את 79 מ马克, מצפה לקבל את 80". במקרה זה לא נכלל מידע נוסף.

(המשר רכיבים בheader)

.Reserved – מס' כלשהו שתמיד מכיל 0.000. NS, CWR, ECE – הם חלק מהheader.

Flags – Control bits: על חלקם נפרט בהמשך

URG: Urgent Pointer field significant

Ack: Acknowledgment field significant

PSH: Push function

RST: Reset the connection

SYN: Synchronize sequence number (יפורט בהמשך בהקשר של יצירת קשר)

FIN: No more data from sender (יפורט בהמשך בהקשר של ניתוק קשר)



Window size – גודל החלון שפוני (בצד המקביל) ברגע שליחת החביליה. מצין כי שהשולח לא ישלח חבילות מעבר לגודל הbuffer הפוני ולמקבל לא יהיה איפה לאחסן אותן עד שיעביר מידע לאפליקציה, וכך חבילות אלו יזרקו.

– מהוות בקרה על טעויות במידע שנשלח (כפי שהסביר בעבר). נרחב כי גם אם מקבל checksum באופן תקין – יתכן כי קיימים מס' זוגי של טעויות, אם כי הסבירות לכך היא אפסית.

RTT Estimation

בכל מידע שנשלח/מגיע, ניתן איזשהו עיכוב קל חד פעמי, או לחולופין יועבר בצורה מהירה באופן מקרי. נרצה להגיר את `timeout` באופן אופטימלי – שלא יהיה ארוך מדי או קצר מדי, ולשם כך נרצה להגיע להערכתה מדוקיקת ככל הניתן של זמן RTT – שבעקבותיו נגידיר את `timeout`. לכן, ניקח כל פעם ממוצע של זמן RTT של מס' הabiliaות האחרונות.



הרצאה 6

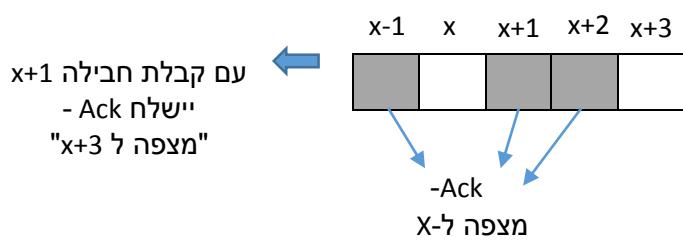
(המשך TCP header)

. – משמש עבור 'מידע דחוף'. בפועל כמעט ולא שימושו וכאן מיותר.
flag – שמצוין אם להתייחס או לא לשדה Ack בheader. מדוע? כי לעיתים שדה ack אינו רלוונטי, למשל בחבילה הראשונה שנשלחת מהשולח למקבל...
Ack

– (עוד הרחבה לגביו) – בTCP הוא Ack הוא **accumulative** – מאשר שהגינו כל החבילות (או באופן יותר מדויק – כל bytes) עד לחבילה המצוינת בהודעה. אם התקבלה חבילה לפניה זו שמוסיפה קודמת לה, למשל התקבלה חבילה $x+1$ לפני חבילה x – היא תישמר בbuffer כל עוד היא בטוחה החלון, אך Ack יודיע כי הוא "מצפה לחבילה x ". רק כאשר חבילה x תתקבל – הtcp יכול "להתקדם" לחבילות אחרות. כדי לחסוך בשילוח Acks, TCP ממתין מס' רגעים (~500 millisecond) לפני שמחזיר Ack על חבילה בודדת, כדי לראות אם אפשר לשלוח ack על חבילה שהגיעה אחרי ובעלת sequence number עוקב, וכיון שהוא accumulative – ה Ack על חבילה הקודמת נחסר.

דוגמא – כשהמקבל קיבל את חבילה מס' $x-1$, הוא ירצה להמתין רגע כדי לראות אם קיבל גם את חבילה x ולהסוך ב-Ack אחד. אם עבר הזמן או שהתקבלה חבילה לא עוקבת ($x+1$ למשל) אז הוא ישלח את ack "מצפה לחבילה x ". כמו כן, יתכן ובינתיים יקבל גם את חבילה $x+2$, אבל עדין יש לו מקום רק שמתינו לחבילה x , ולכן גם אחריו קיבלת חבילה $x+2$ ישלח ack "מצפה לחבילה x ". הזכרנו כבר שמצב זה נקרא **Duplicate ack**. כאשר השולח יראה שהוא מקבל duplicate ack – הוא יבין שהחבילה ננראת אבדה ולא סתם התעכבה מעט ולכן ישלח אותה שוב. ברגע שחבילה x תתקבל בצד המקבל – אז ack שהוא ישלח הוא יכתוב "מצפה לחבילה $x+1$ ", בהתאם למס' החבילות שכבר מארז ונמצאות אצלו בחילון (אלו שמספרן גדול מ- x אך הגינו לפני x).

להלן המקביל יראה כך:



Flow-control – בקרת זרימה

לפעמים, בשל מהירות התקשורת הפיזית, יתכן כי אפליקציה תקבל מידע רב מדי אך לא תספיק לעבד את המידע וה-buffer יתמלא ולא יהיה מה לעשות עם מידע נוסף שיתקבל מהשולח. לכן יש חיפוי של גודל buffer הנקרא הפנוי שיש למקבל.

חיפוי זה נשלח בheader של TCP תחת השדה **window size** (64 Bytes). אם זה לא מספיק (מציריך יותר סיביות) – תעבור הודעה ב**option** מסווג "תכפיל את המידע ב-2". מנגנון זה יוצר מצב שהחלון בצד של השולח הוא динמי בהתאם לחיפוי המתקבל.

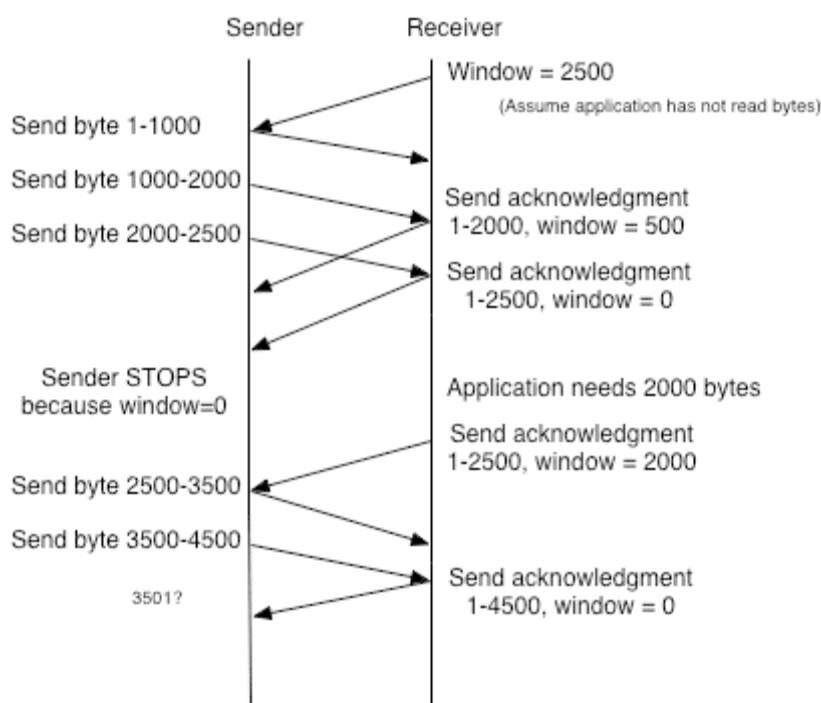
בפועל, המקלט שולח בהודעה גם את ה-Ack – מס' החביליה אותה הוא מצפה לקבל, וגם את **window size** – כמה עוד השולח יכול לשלוח. זה מהו חסם מינימלי ומקסימלי לחולן אצל השולח.

אם המקלט מודיע "אין לי יותר מקום". איך השולח ידע מתי שוב يتפנה מקום ויכול להמשיך לשלוח?

← כשיתפנה מקום אצל המקלט, הוא ישלח Ack!

אבל יתכן שה-Ack יאבז בדרך וזה יגרום ל-**deadlock** = מצב בו התקשורת נתעמת. הצד המקלט ממתין לחבילות, והצד השולח ממתין לאישור על כך שיתפנה מקום כדי שיוכל להמשיך לשלוח...

כדי לפתור זאת, השולח י قادر לעצמו **timer** ובהתאם לטימר ישאל בכל כמה זמן "האם התפנה מקום?"



** הערכה חשובה: **window size אינו MSS!** (MSS = Maximum Segment Size) ה-SMS מושפע מהתעבורה ברשת, כלומר מוקסימלי של סegment שਮותר להעביר. הסבר: כל קוו בין 2 נק' במסלול בין השולח והמקבל מוגבל בקבילות שלו לגודל מסוים לחבילה. ניקח את הגודל המינימלי מבין כל הקטעים במסלול, והוא יהווה חסם עליון לגודל החבילות שניתן להעביר. גודל זה הוא SMS.

תסמנונת החלון הטיפשי

אם כל פעם מתפנה רק מקום קטן בחולון של מקבל, כגון 1 Byte (נניח כי האפליקציה איטית), אז השולח יוכל לשולח כל פעם חבילה קטנה – של 1 Byte, אבל זה יזבוז כי התקורה מאד גבוהה בגין לחבילה (header של TCP הוא בגודל 20 Bytes).

פתרונות:

1. אלגוריתם **Nagle** – בצד השולח – מבקש מהמקבל לא לעדכן על מידע קטן שפנוי, אלא רק כשਬאותה יש צורך. למשל – כשהחולון בגודל מינימום SMS (זה גודל של חבילה מלאה), או – שעבר RTT שלם – אם בצד השולח נותרה רק חבילה קטנה לשולח.
2. אלגוריתם **David Clark** – בצד מקבל – יודיע רק כאשר:

$$\text{window size} \geq \min \left\{ \text{MSS}, \frac{\text{Buffer - size}}{2} \right\}$$

כלומר, כאשר יש גודל של MSS או כאשר לפחות חצי חלון פנוי. לררוב יתרונה מהר יותר מקום בגודל SMS, אלא אם כן ה buffer של מקבל קטן מאוד.

לעתים יש מצב שכן נרצה לשולח חבילות קטנות. למשל כשהתבצעה פעולה קטנה אבל נרצה שזמן התגובה שלה יהיה מהיר (לדוג' – הזרת עכבר במחשב, או הקלהת אות בודדת). במקרים כאלה ניתן לעשות "off" לאלגוריתם ע"י הדלקת דגל **push** ב-TCP header.

הקמה וסגירה של קשר TCP

גם בהקמה וגם בסגירה נדרשת "לחיצת ידיים" – הסכמה ואישור קבלה של 2 הצדדים. אבל ב"לחיצת ידיים פשוטה" יתכונו מקרי קצה רבים. כדי להפחית את הסיכויים של מקרי קיצון מתבצע "3-way handshake".

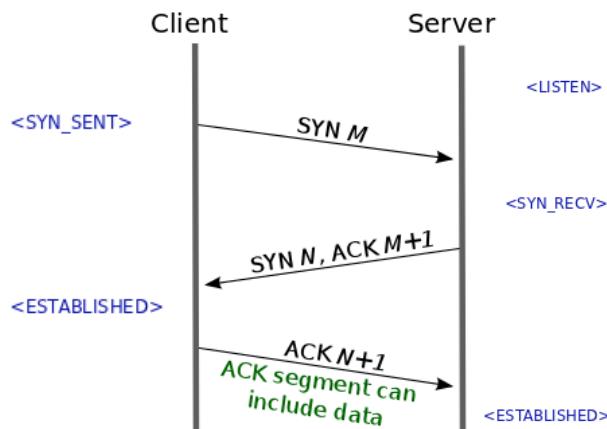


הקמת קשר

בתקנת קשר, כל צד מחזיק מידע רב. תיתכן שליחת מידע לשני הצדדים, וכך כל צד מחזיק $seq = 2$ – אחד של שליחה ואחד של קבלת. בנוסף, כל צד מחזיק את גודל ה buffer של הצד השני (כדי לדעת כמה ניתן לשלוח לו), ועוד אפשרויות שונות במידעה ויש (מצוינות) option.

שלבי ההקמה:

1. היוזם שולח בקשה ראשונה ליצירת קשר, בה הוא מדליק את ה flag **SYN**, ועובד למצב הנקרא **SYNSENT**. הבקשה לא כוללת העברת מידע אלא בקשה לחבר ו- sequence number (כדי שהצד השני ישלח Ack שייזוהה בהתאם).
2. השרת שמקבל את הבקשה (או לחופין 'עמית', זה שאין זם את הקשר) עובד למצב **SYNRCVD** (= SYN-received), ושולח חזרה בקשה הכוללת: דגל SYN דלק, Ack+=1 sequence number משלו.
3. היוזם מקבל את החיבור, עובר למצב **ESTAB** (= establish) ויכול כבר לשלוח מידע (אך לא חייב, יכול רק להזכיר ack כדי להודיע שהוא עומד לשלוח מידע).
4. רק כאשר השרת מקבל חב' מידע ראשונה – הוא עובר למצב **ESTAB**, שני הצדדים מחוברים.



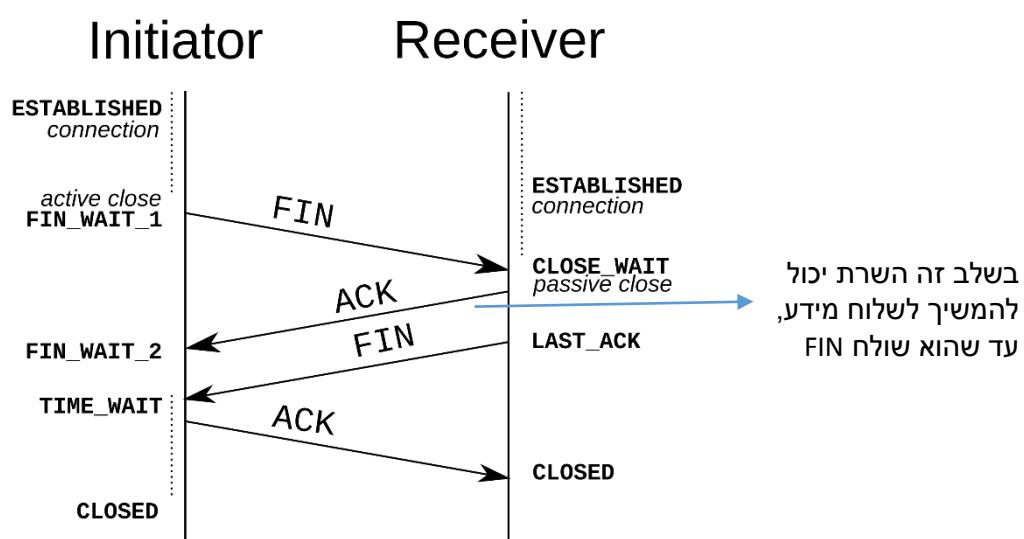
סגירת קשר

מדליקים את דגל **FIN**, ולא סגורים את הקשר באופן מיידי עד שהצד השני מודיע שגם הוא מתכוון לסיום, כי יש מצב שצד אחד מבוחינתו סיים לשלוח מידע ורוצה לנתק אך מבחינה הצד השני יש לו עוד מידע לשלוח.



שלבי הסגירה:

1. יוזם הסגירה שולח בקשה בה הוא מדליק את ה **FIN Flag** וועבר למצב **FIN_WAIT_1**.
2. הצד שמקבל בקשה FIN עובר למצב **close_wait** אך יכול להמשיך ולשלוח מידע. אם הוא שולח מידע – אז היוזם עובר למצב **FIN_WAIT_2**.
3. כשמקבל בקשה הסגירה יסימן לשלוח מידע, הוא גם ידליק FIN ויעבור למצב **Ack** – 'ממתין ל-Ack אחרון'.
4. כשיותם הניתוק מקבל FIN, הוא שולח Ack חוזרת ומפעיל timeout time שבסופה מתנתק – כדי שלא יהיה מצב שהוא מחדש קשר (במידה ויצטרך) לפני שהצד השני נסגר לחלווטין.
5. רק כאשר השירות קיבל חיווי על הגעת FIN ששלח – הוא יփוך ל **closed**.



TCB = Transmission Control Block

מחזיק מידע על קשר TCP:

1. מושם **port**
2. מיקום **buffer** בזיכרון
3. משתנים: Ack, sequence number, windows size, options



במקרה של בקשת סגירה בו-זמןית מ-2 היכיונים, יהיה דילוג על שלב **2 wait_FIN** ושנייהם יփכו ל "מוקן ל close" ויפעלו timeout.

במצב של "סיום פתאומי" – למשל ניתוק כבל/כיבוי המחשב וכו', ירידת המתח נעשית בהדרגה והקשר יכול להספיק להסתיים כמו שצריך. הצד שאצלו יש ירידת מתח מדליק את ה **RST Flag** ש牒בוקש מהצד השני לשיים את הקשר במהירות.

ה **Flag RST** משמש גם לנסיבות של תקיפות, או שהשרת ביטל את הבקשה שלנו להתחברות –ע"י firewall או AdBlock – תוסף בו המחשב שלנו חוסם בקשות עפ"י המידע שביהם.

TCP timers

– טימר פשוט שמודד זמן לשילוח מחדש של חבילת במידה ולא הגיע Ack Retransmission

– טימר למשך שהחולון הצד השני הוא 0 – מודד זמן לשילוח השאלה "האם התפנה מקום?" Persistence

– **למנועת מצב "זומבי"** = מצב בו הקשר פתוח אך אין העברת מידע בין הצדדים. בודק מדי פעם אם הצד השני נמצא בקשר, ואם לא קיבלנו תשובה – נסגור את הקשר.

– **לניתוק הקשר, מרגע קבלת FIN מהצד האחר עד לסגירה.** TimeWait

בקרת עומסים – Congestion Control

במצב של נתב בדרך שיש לו עומס רב, יכולות יכולות לiful בכניסה לתור של הנתב ולהיאבד, או להמתין זמן רב בתור. זה קורה כישיש יותר מדי מקורות שלוחים מידע ביחס לרשת, כמה שהוא מסוגל לקבל ולהעביר.

אבל – עומסים בנתבים זה מידע הקשור לשכבה ה-IP ולא לרמה של TCP, אז כיצד TCP יכול לדעת למתן את שליחת המידע?

- גישה עיקרית לטיפול: "**end to end**" – בקרה מ Każב ל Każב. TCP לא יודע מה קורה בדרך, אך יכול לשים לב שיש השהיות עד קבלת Ack, שינויים בתזמון, או duplicate Ack - שאומר שחביבה אבדה בדרך.
- גישה אחרת: בעזרת הרשות – **דיווח של הרשות על כך שהייתה בעיה.** נתב שעומד להתמלא ידוע על כך ע"י חיויים עוד לפני שייאלץ לזרוק חבילות. מנגנון זה נקרא ECN – וನשלח מהנתב לTCP (יפורט בהמשך).



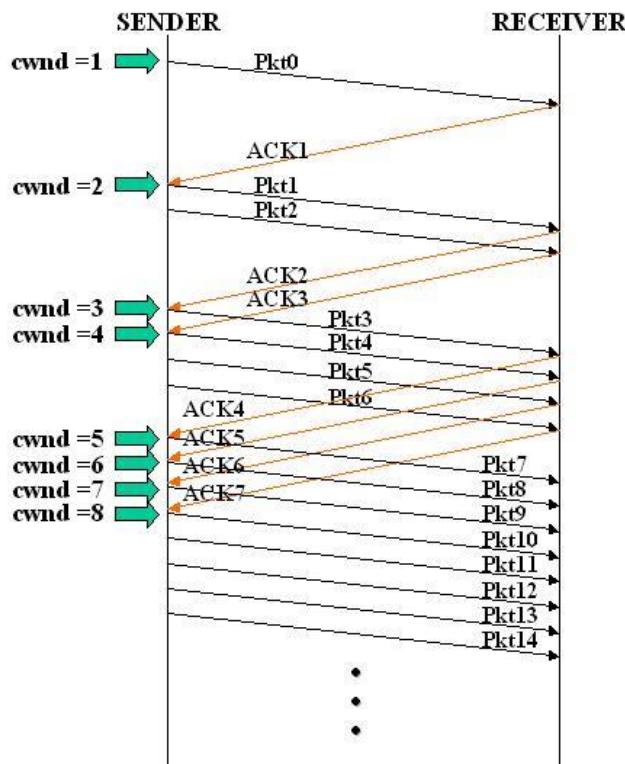
חלון TCP מוגדר כ **cwnd** (מוסמן ב *pdw*) – דינמי בהתאם לתעבורה ברשת. ה-TCP שולח מס' Bytes כגודל ה-*cwnd*, מחייב בurre RTT לקבלת Ack, ואז שולח עוד Bytes. מכאן שהתפקה (=קצב)
$$\text{Throughput} = \text{Rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ Bps}$$

Slow Start (SS)

קשר שנפתח מתחילה מגודל חלון קטן ושולח מידע מיידן מועט, אך מייצ' מהר מאוד עד ל'נפילה'. הרעיון: להתחיל מchlון בגודל חבילהבודדת – גודל של MSS 1, ועם כל הגעת Ack על חבילה – להגדיל את החלון ב- MSS 1 (בגודל חבילה). יצא שהchlון גדל פי 2 על כל מס' חבילות שנשלחו, וזהו קצב גדילה אקספוננציאלי.

נפרט בדוגמה: החלון מתחילה בגודל 1, ושולח חבילה 1. ברגע שהגיע Ack על חבילה זו – הוא גדל ב-1, וכעת הוא בגודל 2. נשלחות 2 חבילות. עם קבלת Ack ראשון על אחת החבילות – שב הוא יגדל ב-1 – ויהיה בגודל 3, ועם קבלת Ack על החבילה השנייה – הוא יגדל שוב ב-1, ויהיה בגודל 4.

cut נשלחות 4 חבילות. על כל Ack של חבילה הוא יגדל ב-1, ולכן אחרי 4 Ack's הוא יהיה בגודל 8....



מי שמאגדיל את מס' החבילות הנשלחות זהו ה **cwnd = congestion window**. נבחן כי אם החלון לא היה גדול – אז התקשרות הייתה עובדת בשיטת Stop & Wait



Congestion Avoid

لتואיצה ב Slow Start יש איזשהו **Threshold** – גודל מסוים שהחל ממנה הchlון גדל בצורה פחותת מהירה ולא אקספוננציאלית. בעצם מגודל זה – הchlון גדל ב- MSS 1 על כל chlון שלם שנשלח, ולא על כל חבילה. ז"א שעבור chlon בגודל N, החיל מרגע Threshold – עבר שיחת N החבילות הוא לא יגדל בנ והפוך ל-N2 כמו בגדייה ההתחלהית, אלא על כל N החבילות יחד הוא יגדל ב-1 בלבד יהיה N+1. קצב גדייה זה נקרא **Congestion Avoid**.

עדמתי הchlון ימשיך לגדול? עד שתתקה איזושה נפילה - ז"א עד שעבר timeout על חבילה בודדת או שהתקבלו 3 duplicate Ack's – אז הchlון יקטן בחצי, ושוב יעלה בקצב לינארי עד לנפילה הבאה (וחזר חלילה....)

בפועל, ב congestion avoid - הchlון לא מחייב ל – N ואז מגדיל את הchlון ב-1, אלא עם כל Ack בודד שMagnitude הוא גדול עפ"י הנוסחה $(\frac{MSS}{cwnd}) * MSS$, וויצא שאחריו כל N Ack's הוא גדול בערך MSS אחד. אבל בכל רגע נתון הcwnd משתנה ולכן בפועל ההגדלות טיפה יותר קטנות מ-MSS, אבל בקירוב טוב ל MSS 1.

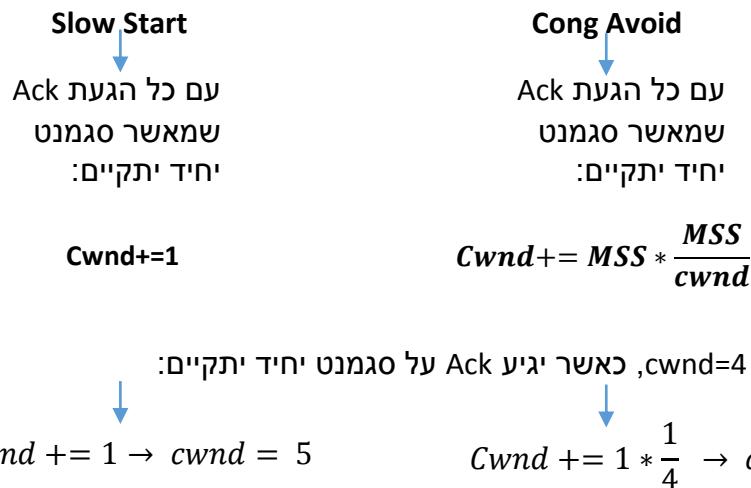


הרצאה 7

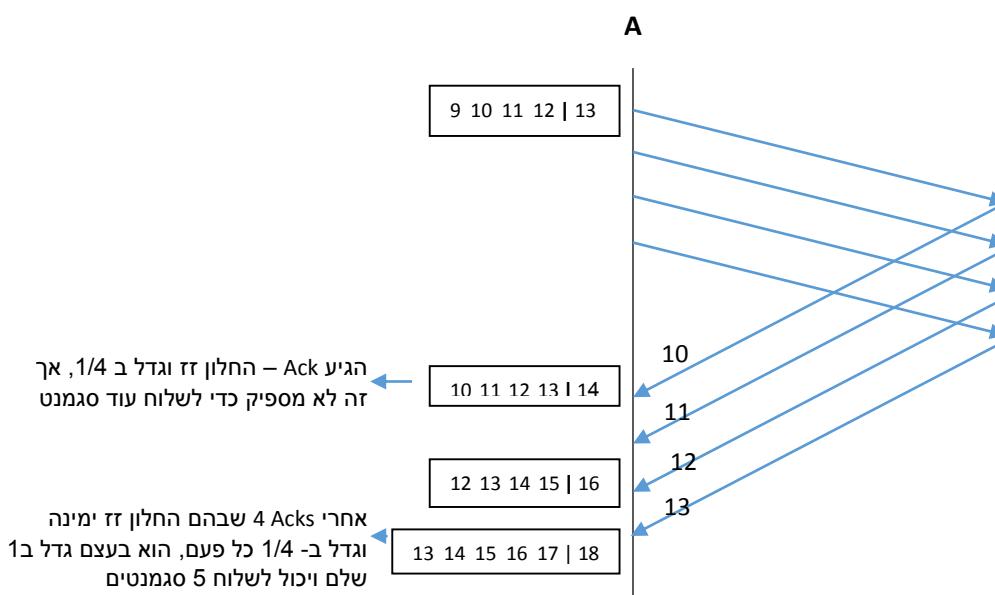
↖ נזכיר כי בקרת זרימה - זהו חסם עלין של גודל החלון השולח בהתאם לגודל החלון של המקלט, והוא נשלח בשדה window size ב header של כל הודעה מהמקבל.

בקרת עומס – זהו גודל החלון השולח בהתאם לקצב התעבורה ברשות, שגדל בתחילת בקצב אקספוננציאלי (Slow-Start) ולאחר מכן בקצב לינארי (Congestion Avoid) (Congestion Avoid), וקטן כשיש נפילה.

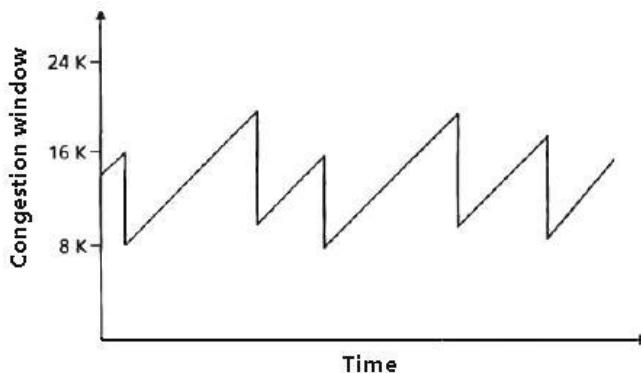
גם כאשר לפי בקרת העומס החלון גדול, הוא תמיד חסום מלמעלה ע"י גודל החלון בצד $\text{MIN} \{ \text{Congestion control}, \text{Flow control} \}$ (Congestion control, Flow control) המקלט. لكن גודל החלון השולח הוא תמיד



הגדלת החלון ב-Cong-Avoid:



החלון גדל עד שיש נפילה, וזה הוא קטן בביטחון אחת לחצי מגודלו וגדל שוב בקצב של cong avoid, עד לנפילה הבאה. השינויים בגודל החלון יראו פחות או יותר כך:



בגודל, אם ניקח ממוצע בכל קטע של עלייה וירידה, נקבל קצב גידול לינארי (פחות או יותר).

TCP Reno – גרסה עדכנית של TCP, בה יש התийחות לסיבת לנפילה והקטנת החלון בהתאם.

סיבת הנפילה מתחולקת ל-2 סוגים:

- **Timeout** – חביתה נשלחה וה-timeout שלה הסתיים לפני שהגיע עלייה Ack.
- **כיוון שצפו תרחיש גרוע**, החלון יקטן מיד לבודל 1, וכן ה **SSThreshold** (Slow-Start Threshold) יקטן :

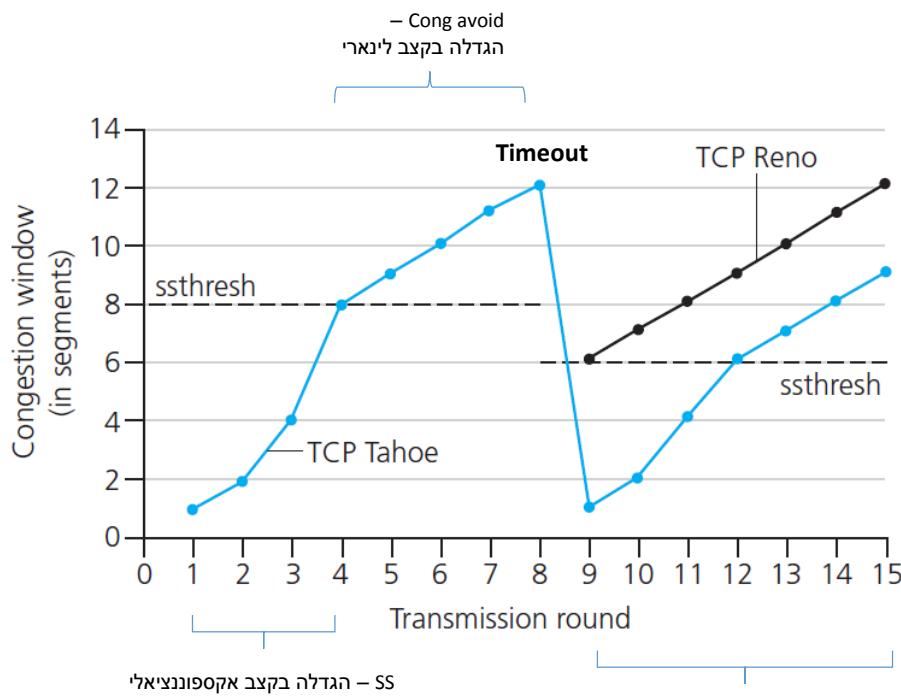
Cwnd = 1 MSS

SSThreshold = MAX { 2 MSS , FlightSize/2 }

כאשר FlightSize זהו מס' החבילות שנשלחו עד רגע הנפילה. במה זה שונה מגודל החלון? – ניתן כי החלון גדול אך לא נצל במלואו, ונשלחו פחות חבילות מגודל החלון. מיד לאחר הקטנה, החלון ישוב לגודל בקצב של SS (אקספוננציאלי).



דוגמה ל蹶ה של SSThreshold ו-timeout התחלתי עם ערך 8:



הקטנת החולון ל-1, הקטנתה-
הגדלה בקצב אקספוננציאלי
חbillot והגדלה מחדש לפ'
cong avoid SS ואח"כ לפ'

- Duplicate Ack – כאשר מגיעים מוש' Ack' זיהים על חbillה מסויימת, אנו מסיקים כי הגיעו החbillות שנשלחו אחריה זו אבדה. במקרה זה אין סיבה להקטין את החולון יישורות ל-1, ולכן קיימים מנגנון ה **Fast Recovery** – אחרי קבלת 3 dup-Acks החbillה האובדת תישלח שוב, וקצב גידילת החולון ישאר ב-cong avoid, אבל ה-SSThreshold וה-cwnd יקטנו:

$$\text{SSThreshold} = \text{cwnd}/2$$

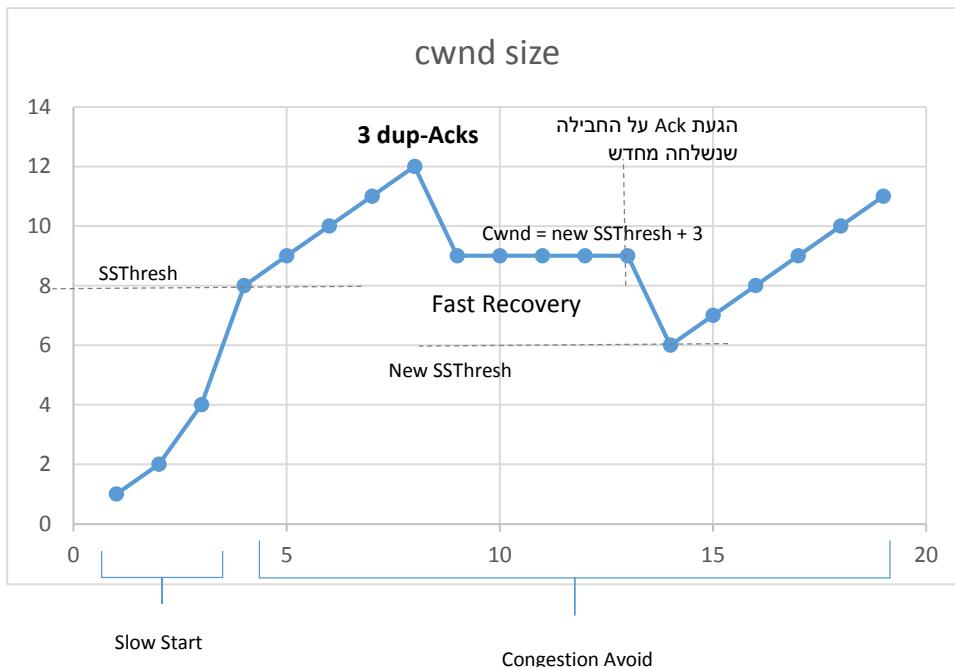
$$\text{Cwnd} = \text{SSThreshold} + 3$$

ולאחר שנקבל Ack על החbillה שנשלחה מחדש נוריד את החולון ל SSThreshold, ורק אז נתחיל להעלות אותו שוב. (בעצם קיבל Ack עם sequence number sequence number שובי' שווים' החbillות שנשלחו אחריה הגיעו גם גדול בהרבה מהחbillה שנשלחה שובי' כיוון שגם המקלט הגיעו גם והחולון של המקלט זו מוקומות בבת-אחת).

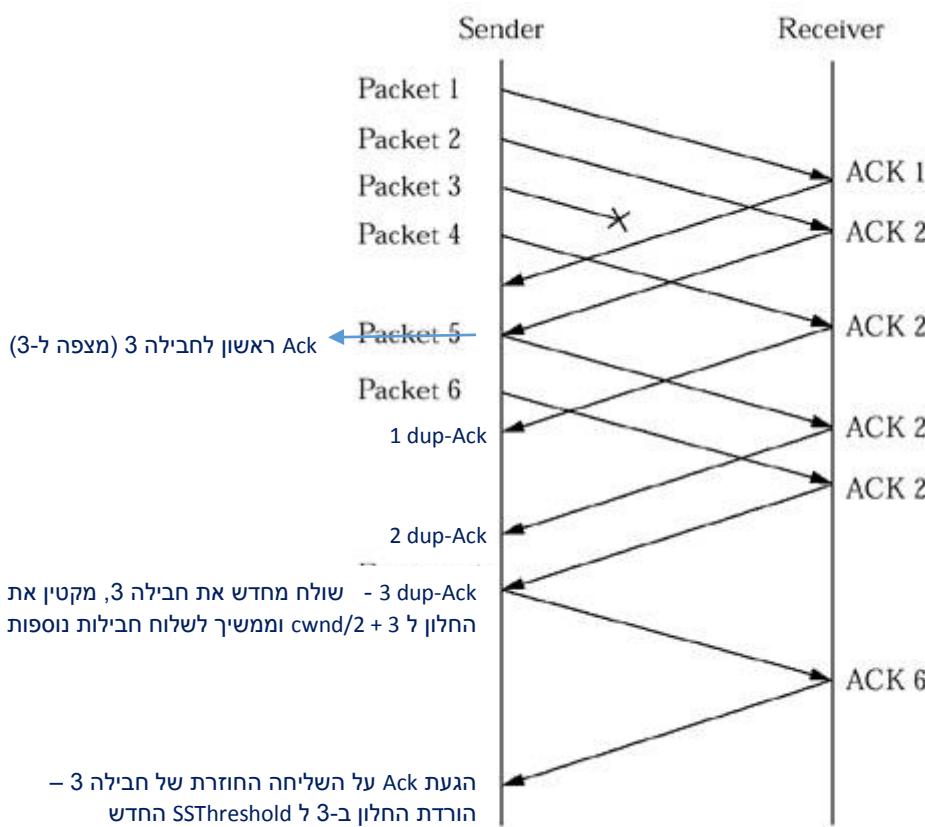
מנגנון זה מאפשר התגברות על הנפילה אף גם המשך ריצה בקצב סביר ללא התחלת מ-0.



דוגמא ל蹶ה של SSThreshold התחלתי עם ערך 8:



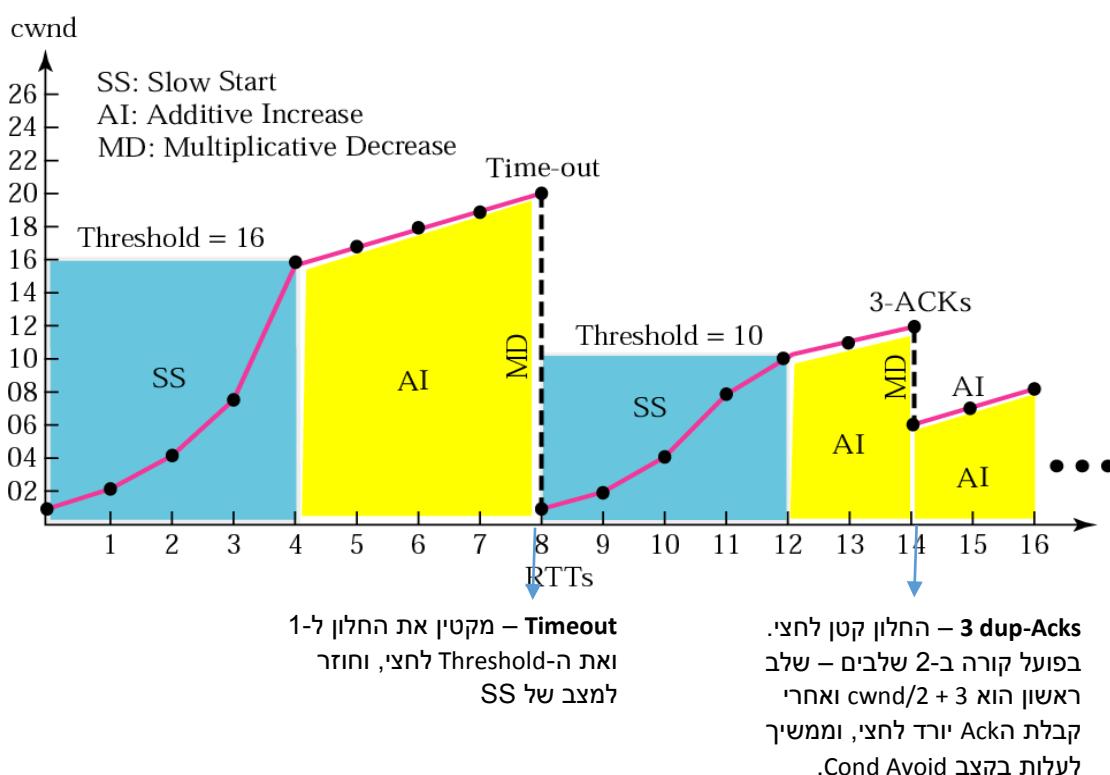
מתי בדוק זה **3 dup-Acks**? – כאשר מגיע **Ack שלישי משוכפל**. נבחן כי **Ack ראשון על חבילה הוא 0 num 0, ואין dup**:



לפני שחבריה מגיעה ל-timeout, החבילות שנשלחו אחריה יחזירו Acksl ולקן במלוא נגיעה במצב של 3-dup-Acks. בעצם הוא 'מקרה גורע' שנרצה להימנע מהגעה אליו, וכן מנגנון ה 3-Acksdup מגן עליו. متى כן נגיעה ל-timeout? כשים איבוד של יותר מחבילה אחת, אז כל ה-Acks יתיחסו לחבילה הראשונה שאבדה ולא נבחן בכך שאבודה חבלה נוספת, עד שיקרר timeout.

כדי להימנע גם ממקרה זה, בגרסת מחודשת של TCP בשם Reno New, המצב של fast-recovery "ישאר זמן-מה כדי לבדוק אם אבדו חבילות נוספות או לא, ורק אז התהיליך חוזר למצב רגיל של הגדלת החלון לפי avoid Cong."

נסתכל על תהליך שלם של החלון, שכולל מקרה של timeout וכן מקרה של dup-Acks:



TCP throughput

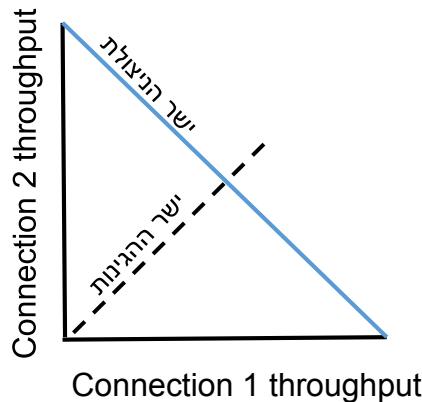
הTCP הוא 'עקשן' ומנסה תמיד לשפר את התפוקה, יוצא שפונק' גודל החלון היא קטנה או יותר:



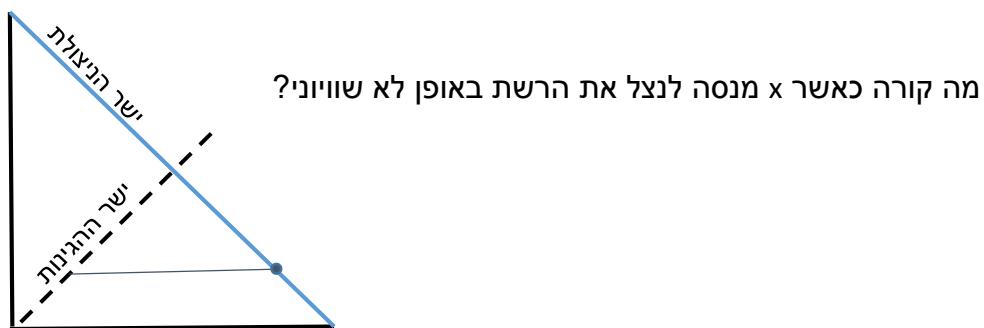
$$\text{הממוצע: } \frac{3}{4} \frac{w}{RTT}$$



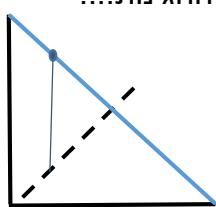
TCP יוצר מצב של הגינות בראשת, כך שאם משתמש מנסה לחרוג מהקצב הסביר ולהעmis את הרשת או להקנות את כל המשאים אליו – אז נופלת לו חבילה והוא נאלץ להפחית את הקצב באופן משמעותי. יצא שוגול החולון הנע סיבוב "ישר ההגינות":



2 היצרים אלו 2 קשרים שונים בראשת, הישר $y = x$ הוא ישר ההגינות והישר $b - x = y$ הוא ישר הניצולות, והוא השימוש המקורי האפשרי – כל נק' על ישר זה ומתחתיו היא אפשרות לניצול היקו, ולא קיימת אפשרות לניצול מעבר לקו זה. ישר ההגינות מסמן ניצול הוגן של הרשת, כאשר שני הקשרים משתמשים בראשת באופן שווה.



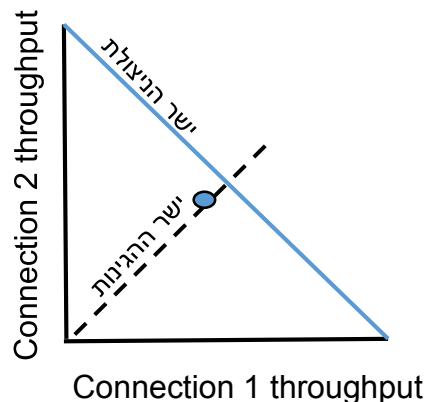
א מושך את הקשר אליו, ומישר ההגינות ערך ה x גדול יותר של היקו) ואילו y נשאר באותו הקצב. כאשר x יכול לישר הניצולות וינסה לעבור אותו – הוא יabd חבילה ויאלץ להקטין את החולון בחצי ולחזר חזרה לטוווח שה讚ב לישר ההגינות. כאשר הוא מקטין את החולון – אז y יכול להגדיל את החולון שלו ולנצל את היקו, וכך ימושך את הקשר אליו – ויגע גם הוא לישר הניצולות וינסה לעבור אותו, ויפול חזרה לישר ההגינות....



יוצא שיש התכונות סיבוב ישר ההגינות, כי בכל פעם שהוא מושך הוא קטן בחצי והקשר השני מושך אליו וחזר חלילה...



בנוסף להגינות טובה, נרצה שהיהה ניתן כמה שיותר גבה וקרוב למרוחב הפס, ולכך האзор האופטימלי הוא על ישר הגינות וקרוב לשער הניצול:



← TCP הגן יותר מUDP!

לעתים אפליקציות מנסות לעקוף את ההגינות ופותחות קשרי TCP רבים כדי לנצל את הרשות ולמשוך את המשאים אליהם. Browsers עושים זאת באופן כמעט קבוע, ניתן לראות זאת בWireshark שבקשות רבות נשלחות מהמחשב.

TCP Options

Header של TCP הוא בגודל מינימלי של 20 Bytes, אך יכול לגודל עד ל 60 Bytes בהתאם ל.options.

ECN=Explicit Congestion Notification

אפשרה לIP/TCP לבקש עמוסים, חלק מ.Active Queue Management בזמן עמוסים, נתב בדרך יכול להפעיל חבילת. היינו רצים שהנתב לפחותה שהחbillת הופלה כדי לצמצם את זמן חוסר הוודאות של השולח וכן שיידע שעליו להפחית את קצב השילחה, אך הנתב גם ככה עמוס וכן יוכל להוציא הודעות משלו.

ה- ECN מהו זה פתרון לכך: במצב שנתב רואה שהוא עומד להתמלא ועוד לפני שהוא מלא ממש – הוא 'מסמן' את החbillת, מצין עליה שבדרכו היא עברה בננתב שעומד להתמלא. כשהחbillת תגיע לשכבת הIP בקצת הקשר – IP יראה את הסימון וידיע לשכבת TCP שננתב בדרך עומד להתמלא. TCP יסמן זאת באck לשולח, והשולח שיקבל את הסימון יידע שעליו להפחית את קצב השילחה.

* קיימת גרסה TCP חדשה יחסית בשם DC-TCP (DC=Data Center) שבה השולח 'סופר' כמה חבילות הגיעו עם סימון לעמוס, ומפחית את הקצב בהתאם למסת העמוס.**



כדי להשתמש במנגנון זה, על הנתבים בדרך לדעת אם קשר TCP מעוניין בכך, וכן על הTCP לדעת אם הנתבים תומכים במנגנון זה או לא. הTCP מסמן זאת בחסption, ובשילובת ה^{רנו} קיימים הסימונים הבאים:

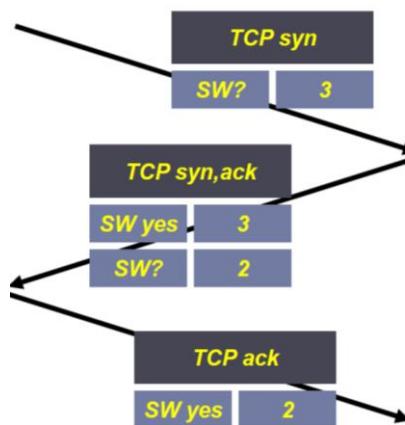
- 00 – נתב אינו תומך במנגנון.
- 01 / 10 – נתב תומך ומשתמש במנגנון.
- 11 – הנתב תומך, ועמום – זה בעצם סימון החבילה ע"י נתב שעומד להתמלא.

Window Scaling Option

אפשריה עבור שדה ה-size-window, למקורה של 12 הסיביות (64 Bytes) המוקצים לו בheader אינו גדול מספיק כדי להכיל את המספר.

הຽריעון: הודעה מהצורה "תוסיף ? אפסים מימן למס' המצוין ב header = הכפלת המס' ב 2".
דוגמא: עבור 2 SW (Scaling Window), וערך 11....100 בשדה size-window – הכוונה לחילון בגודל 1100....100.

כמובן שהצדדים צריכים לידע אחד את השני על שימוש במנגנון זה, וכן עם הבקשה ליצור הקשר, היוזם שואל את הצד השני – האם מעוניין להשתמש בWS? וכן מצין את מס' האפסים שיש להוסיף במידה וכן. הצד השני מאשר, ושובאל אם גם הוא יכול להשתמש במנגנון, עבור הוספה ? אפסים למשל. היוזם יצריך גם לאשר זאת. חילופי ההודעות ייראו כך:



התרשימים הנ"ל מחדד את ההבנה מדוע בקשרי TCP יש צורך ב- 3-way handshakes על מנת כל ההסכם על שימושים במנגנונים שונים.



SACK = Selective Ack

מנגן שמספרת את החבילות שהתקבלו, מעבר ל-Ack שמודיע "מצפה לחבילה x". ה-SACK שולח בoptions רצפים של מס' Bytes שהתקבלו ומספרם גבוה יותר מהחבילה אותה הוא מצפה לקבל. (בעצם מתכוון לשאר החבילות בחילון שהתקבלו אך ה-Ack שנשלח עליו מצין חבילת בעלת sequence number נמוך יותר).

דוגמא:>User החולון-



כל Ack שיגיע מספרו יהיה 13 (החבילה בעלת ה-seq הכי נמוך שלא התקבלה), אך בנוספ' יציין בoptions החבילות שהתקבלו במספרן גדול מ-13:

14-16

19-23

25-25

נעיר כי מנגנון זה אינו מבטל אף אלגוריתם של מקרי נפילה, אלא רק מוסיף מידע שיעזר לשולח בהמשך.

גם עלusr כרוך צריכה להיות הסכמה של 2 הצדדים בתחילת הקשר. מנגנון זה מרחיב את header בכמות רבה של Bytes, אך נדרש לב כי לא לצורך לציין את כל מס' החבילות בכל Ack שנשלח אלא רק במקרה של עיכוב או נפילת חבילת – מה שגורם לעקיפה של חבילות אחרות.

גרסאות TCP

לTCP יש גרסאות רבות שתומכות במנגנונים שונים:

TCP Tahoe – גרסה בסיסית שלא שימושה ביום.

TCP BIC – גרסה הכוללת דרך לחפש חילון אופטימלי בצד השולח ע"י חיפוש ביןари.

TCP CUBIC – שיפור של TCP BIC

ישנן גם גרסאות שמודדות RTT באופן מדויק, ומסיקות שיש עומס אם יש סטייה גדולה – גם אם לא התקבלו חיויים עלusr.



הרצאה 8

чисוב מס' חבילות בהתאם לגודל החלון (נדרש בעבודות הגשה)

سؤالה 1: נניח כי מגודל 1 החלון גדול כל פעמי ב-1. כמה חבילות נשלחו לאחר 100 הגדלות?

תשובה: נרצה לחשב את הסדרה החשבונית: $1 + 2 + 3 + \dots + 100 = ?$

$$\text{דרך 1} - \text{שימוש בנוסחה: } 1 + 2 + \dots + n = \frac{(n+1)n}{2}$$

דרך 2 – נפצל את הסדרה לשורה עולה מ-1 עד 50, ולסדרה יורדת מ-100 עד 51.

נחבר כל 2 איברים מקבילים ונקבל:

$$\begin{array}{r} & 1 + 2 + 3 + \dots + 49 + 50 \\ + & 100 + 99 + 98 + \dots + 2 + 1 \\ \hline & 101 + 101 + 101 + \dots + 101 + 101 = 101 * 50 \end{array}$$

سؤالה 2: חשב את מס' החבילות שנשלחו מרגע $SSThreshold=50$, ועד לגודל חלון = 100.

תשובה: מרגע $SSThreshold$ החלון גדול ב-1 על כל חלון שלם שנשלח. לכן כדי לחשב מס' חבילות שנשלחו בחולון שגדל מ-50 ועד ל-100 ב-1 כל פעם, יוכל לחשב את ההפרש:

$$50 + 51 + 52 + \dots + 100 = (1 + 2 + \dots + 100) - (1 + 2 + \dots + 49)$$

سؤالה 3: כיצד חשב גודל חלון שמתחליל ב-1 וגדל ב Slow-Start (פ' 2) עד לגודל 128?

תשובה: דרך 1 - במקום לחשב את $1 + 2 + 4 + 8 + \dots + 128$

ננפורט את סדר האיברים – $128 + 64 + \dots + 4 + 2 + 1$

ונמצא גורם משותף: $256 \left(\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{32} + \frac{1}{64} + \frac{1}{256} \right)$

נבחן כי סכום הסדרה שווה ל-1, ולכן סכום הסדרה שבstoiיגרים ≤ 1

ונקבל: $256 \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{256} \right) \leq 256 * 1$

דרך 2 – נבחן כי מדובר בסדרה של 2^x שזה בדיקת הדריך לייצוג בבסיס 2, ולכן ביצוג בינארי הסדרה תיראה כך:

$$1 + 10 + 100 + 1000 + \dots + 10000000 = 11111111$$

$$100000000_2 - 1_2 = 256_{10} - 1_{10} \quad \text{שזה בעצם}$$



שכבה הרשת – Network Layer

שכבה זו אחראית להעביר segments מה host השולח ל host המתקבל.

מציר כי segment זה מונח לחבילה בرمמות שכבת האפליקציה והתעבורה – ז"א המידע המועבר + headers של שכבות 4,5. חבילה לעומת זאת זה כשה מידע עובר לשכנת הרשת, וזה בעצם segment שכולל גם את header של שכבה זו.

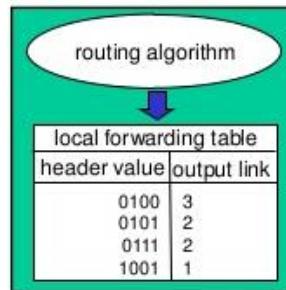
בצד השולח – שכנת הרשת מארגנת את המידע לdatagrams, ואילו מצד המתקבל השכבה מקבל חבילה ומעבירה segments לשכנת התעבורה.

בניגוד לשכבות 4, שנמצאות רק ברכיבי קצה – שכבה זו קיימת בכל רכיב בדרך, כולל ראותרים ונתבים שונים. בכל נתב זהה החבילה צריכה לטפס מהשכבה ה-1 ל-2 ואז ל-3, שם היא מעובדת כדי שהנתב יידע לאן לשלוח אותה, ואז יורדת חזרה עד לשכבה 1 וועברת לנtab הבא.

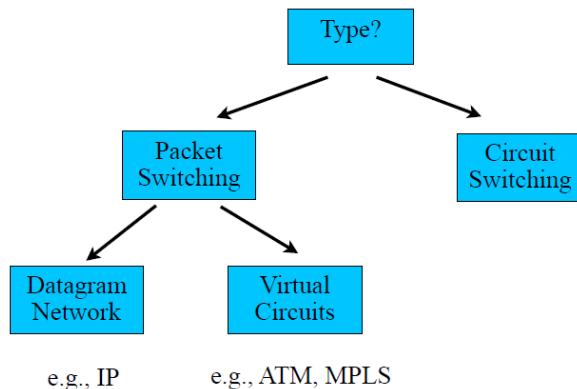
2 פונק' עיקריות בשכבה (קיימות בכל נתב):

– ניתוב, החלטה באיזה מסלול לשלוח את החבילה, לפי חישוב המסלול האופטימלי לייעד. האלגוריתם מלא טבלה בה הוא רושם לכל כתובת את היציאה המתאימה.

– העברת החבילות בנתב מה ל המתאים לפי מה שנקבע בטבלה routing.



אפשרויות ניתוב חבילות



נכיר את שיטת ה **circuit switching** – החלטה מלכתחילה על מעגל מסוים עם נתיב מסוים למיידע בין 2 hosts.

יתרונות: הנתבים הופכים לצינורות ולכן יש פחות זמן המתנה בנתבים, וכן אין headers ולקמן זמן המשלוח מהיר יותר.

חסרונות: זמן הקמת הרשת, וניתולת נמוכה של משאבי בריגעים בהם אין מידע מועבר אך הקוו תפוס ע"י הרשת. השימוש בשיטה זו פוחת בהדרגה, ומשמש בעיקר קווי טלפון.

ה **virtual circuits** מהו זה דרך ביןיהם בין שיטת ה **packet switching** לשיטת ה **circuit switching**: מהוגדר **packet switching** הוא לוקח את החלוקה לחבילות קטנות, ומה **circuit switching** – את המסלול הקבוע.

בעת העברת מידע ברשת, נדרש כמה דברים:

- משלוח מובטח – שנדע שחייב בוודאות הגיע ליעדה (טור זמן מוגבל כלשהו).
- רוחב פס מובטח – שיווקצתה לקשר רוחב פס מינימלי להעברת חבילות.
- הגעת חבילות עפ"י סדר.
- jitter – נרצה שהפרש הזמן בין קבלת 2 חבילות יהיה זהה להפרש הזמן בשליחתן.
- אבטחה – נרצה כמוון שהמידע יהיה מאובטח.

פרוטוקול ה-IP לא מבטיח לנו שום דבר מכך! ה-IP משתמש מאוד בכל זה יתקיים אך אין הבטחה לאף אחד מהדברים. לכן יש דרכי נספנות להעברה – בשיטת Virtual Circuit, ע"י הפרוטוקולים ATM או MPLS.

ATM- Asynchronous Transfer Mode

פרוטוקול שכמעט ולא שימושי היום (לא נרchieb עליו).
דרך המשמשת לרשותות לצורך שילוב מידע מסוגים שונים ובעל דרישות שונות (כגון אלו שצינו לעליהם). למשל – חבילה בה הזמן להגעה הוא קריטי, וחביבה שיכולה להתעכ卜 אף דורשת שהמידע יגיע במלואו.

פרוטוקול זה יכול לעבוד על סיבים גבוהים ועל ISDN – Integrated Services Digital Network.

גישה ה-ATM מהו זה תחליף ל-3 השכבות התתונות (השכבה הפיזית, הקו והרשת). הוא יוצר מעגל וירטואלי (מה שגדמה אותו ל **circuit switching**) בו הוא משתמש בתאים קטנים שבهم מעברות חבילות (בפרוטוקול זה הן נקראות Sales) ובגודל קבוע – מה שמקל על הנטבים גם בטיפול בתקורה וגם בזמן הטיפול. בכל כמות קבועה של תאים, יש תא שמוקדש להיות כ-header – מידע בקרה.

חסרון לגישה זו – צריך להקים את המעגל מראש.



לעומת ה-IP שלא מבטיח אף דרישת – ל-ATM יש סוגים שונים כשל אחד מבטיח דברים ברמה אחרת:

- CBR – Constant Bit Rate** – הרמה הטובה ביותר – מבטיחה קיום של כל הדרישות. רוחב הפס מיועד לקשר זה בלבד ולכן הוא קבוע וmbtih הגעה ללא השהיota או איבודים.
- VBR – Variable Bit Rate** – לא מבטיח רוחב פס קבוע ולכן זמן המשלוח משתנה, אך קיים לו קצב ממוצע מסוים ומוגדר זמן מקסימלי להריגת מקצב זה.
- ABR – Available Bit Rate** – מבטיח קצב מינימלי להעברה. כיוון שאין לו חסם עליון, הוא יכול לנשוט לדחוף חבילות ולשלוח יותר – אז להעמיס על נתב ולאבד את החבילה.
- UBR – Unspecified Bit Rate** – מבטיח רק הגעה לפי סדר השליחה.

Network Architecture	Service Model	מה מבטיח?						Congestion feedback
		רוחב פס קבוע	שלא יאבדו	הגעת עפ"י סדר	זמן הגעה קבוע			
IP	Best effort	X	X	X	X	X	X	
ATM	CBR	V	V	V	V	V	V	
ATM	VBR	ממוצע	V	V	V	V	V	
ATM	ABR	mbtih מינימליות	X	V	X	X	V	
ATM	UBR	X	X	V	X	X	X	

ההבדל בין ATM (פרוטוקול של circuit switching) לבין ATM זה שב ATM יש חלוקה של המידע לחבילות קטנות והעברת header ייחיד אחרי כל מס' חבילות שנשלחות, בנויגוד לcircuit switching שלא משנה את החבילות שעליו להעברה. כמו כן הקצאה של קו ב-ATM יכולה להיות רוחב פס שלם או רק חלק ממנו.

אבטחה ברשת

יכולת להישאות בכמה שכבות:

- שכבה האפליקציה – שימוש SSH – Secure Shell – SSH מאפשר אבטחה גם אם הרשת אינה מאובטחת.
- שכבה התעבורה – שימוש בSecure-Socket Layer – SSL – IPSec – אבטחה בתוך חבילת IPSec
- שכבה הרשת – עטיית חבילת בתוכה נספפת.

כמובן שאין צורך להשתמש בכל מנגנון האבטחה, נרצה להשיק אבטחה בשכבה אחת ובכך למנוע עסקך באבטחה בשכבה נוספת.



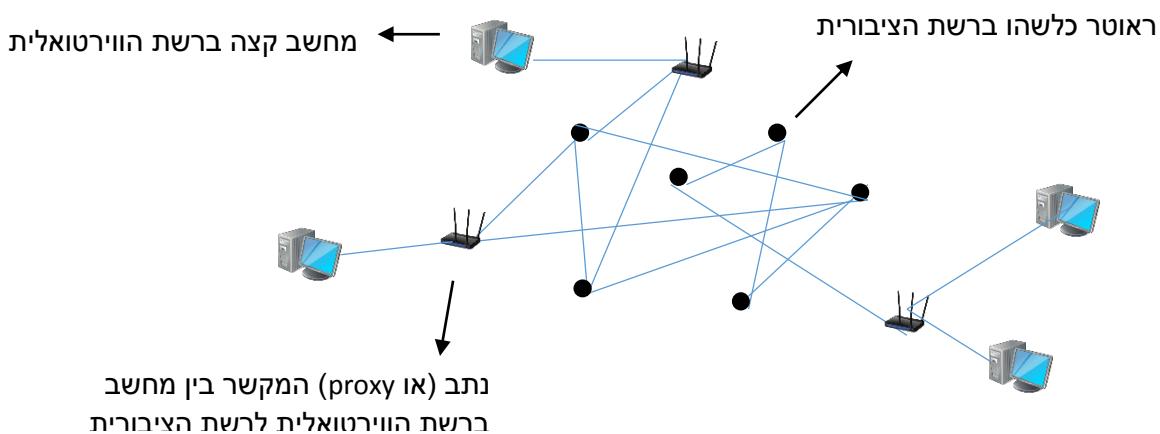
VPN = Virtual Private Network

גישה לפרטיות ובטחת מידע: מנגנון ברמת שכבה הרשות המאפשר לתקשר בمعنى 'רשת פרטית' (וירטואלית) בתוך הרשות הציבורית. יעיל מאוד לחברות גדולות שמצריכות בטחה גדולה ואין חברות תקשורת, ומשתמשות בתשתיות של חברות אחרות (בנק למשל).

ה VPN יכול לתקשר גם כsheduler במרחקים גדולים (חווי ישות), מאפשר שימוש ב wireless וכן משתמש ב proxy דואק אমצעי לאבטחה – הפרדה בין הרשות ה'פרטית' לעולם החיצון – משתמשים מחוץ לרשות הפרטית רואים רק את הכתובת של proxy ולא של המחשב בתוך הרשות ממנו נשלחה הודעה, כמו כן הוא יכול לסנן קבלת מידע מבוחר אל תוך הרשות.

צורת ההצפנה בVPN היא באמצעות 'מנהרה' – **tunneling**. חבילות שעוברות בתחום ה' הציבורי' (הרשות שאינה פרטית) יהיו מאובטחות ומוצפנות, והראוטרים לא יגעו במידע ולא יוכלו לקרוא אותו, אלא ישמשו כמעברים בלבד – כמו מנהרה.

איך מונעים מהראוטרים גישה למיידע? עוטפים את החבילה הרצויה שעליה פרטי ה- IP של host השולח והמקבל בחבילה חיצונית מאובטחת – בחבילה החיצונית יש את פרטי הנטבים (או proxies) שמקשרים בין הרשות החיצונית לבין מחשב הקצה השימוש לרשות הפנימית (וואירטואלית), והראוטרים שבדרכ מעבירים את החבילה עפ"י המיידע שבמעטפת החיצונית בלבד. רק כאשר החבילה מגיעה לנטים המקשרים בין הרשות החיצונית לפנימית – Tipeth המ�עתת והיא עובר לenza הרשות המתאים. לנטים המקשרים יש רוב מס' IP קצר יותר מאשר של מחשב הקצה, והם יכולים לשמש במקביל גם כנתבים רגילים ולהעביר חבילות שאין קשרות לרשות הוירטואלית.

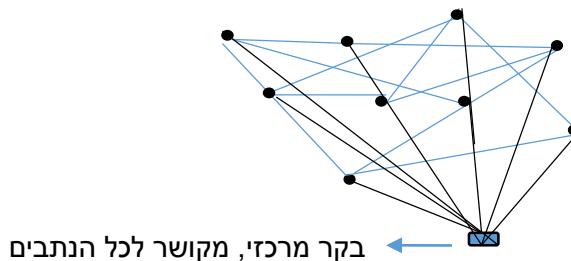


קיימים 2 פרוטוקולים לעטיפת החבילות בצורה מאובטחת – **MPLS** ו- **IPSec (security)**.



MPLS = Multi-Protocol Label Switching

פרוטוקול קרוב יותר לVPN, נמצא במעטפת השכבה. מספק אבטחה ע"י **אופן הניתוב**, פועל תחת VPN (משמעותו פרטני שלו). ב- MPLS הוראות הניתוב ניתנות מראש, והוא מקצה בקר מרכז' שיהי אחראי לכל המידע ולניתוב החבילות על סמך עומסם בראשת. אותו בקר יכול לטפל במס' שיחות שונות, ומנתב את ההודעות עפ"י תוויות האבטחה שעליהן. אין מבטיח הגעה עפ"י סדר. משתמש ב **Out of Band** – בקרה **פרדת מהחbillה**.



יתרונות MPLS –

- מ��ר את הפרטיות והאבטחה
- CIION שהבקר הוא זה שאחראי על הניתוב יש פחות תקורה
- אין צורך אלגוריתמים מסובכים ויקרים או טבלאות
- קל יותר להנדסת תנועה'

חסרונות –

- חספota של נתבי הקצה
- הבקר המרכז' מסובך, אין תמיד שרת יחיד אלא קב' של שרתים
- הרשת פגעה יותר – פגעה בבקר מפילה את כל הרשת.
- דרוש ייצור קשר – מצריך אישור ומעבר חבילות דרך הבקר, דבר שדורש זמן.

מגמות בעולם התקשורת

בעבר רכיבי התקשורת היו קטנים מספיק ויכלו לנוהל אותם בצורה ריכוזית – ניתן היה להציג מישחו שיודע הכל. כשה.harשות התפתחו והתרחבו – היה מעבר לגישה מבודדת (כל רשת אחראית למאה שקורה בתוכה ולא מבוחץ לה), CIION שהיא מדע רב מיד לניהול. מה שגורם לפוטוקולים רבים, והחומרה לא הייתה סקליבלית.

בשנים האחרונות, האפקציות של מרכז' נתונים חזירות להיות ריכוזיות, מה שמתאים לניהול ב奏ת MPLS. ריכוזיות הרשת מאפשרת גם התאמה דינמית של הפעלה וכיבוי שרתים מסוימים בהתאם לצורכי.

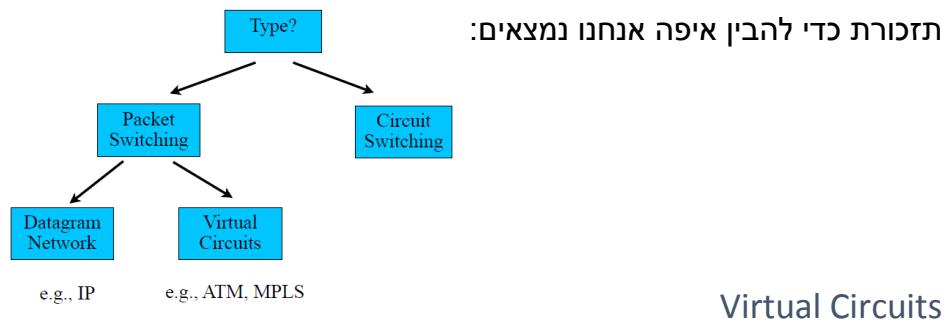
באופן דומה, היה גם מעבר מתוכנה לחומרה כדי ליעיל את מהירות העברת הנתונים בין נתבים, והיום שוב יש מגמה של חזרה לתוכנה בגל הריכוזיות – קל לנוהל בתוכנה שרת אחד שמקבל החלטות וכל התוכנה נמצאת אצלו, ואילו שאר הנתבים פשוט מעבירים את המידע.



DCN = Data Center Network

DCN הוא מרכז נתונים גדול של חברת מסוימת, שספק אחיזת נתונים של חברות שונות. התהילכים בו לא גלויים לחברות המשתמשות בו, ובד"כ הוא מעביר מידע בין עצמו עצמו.

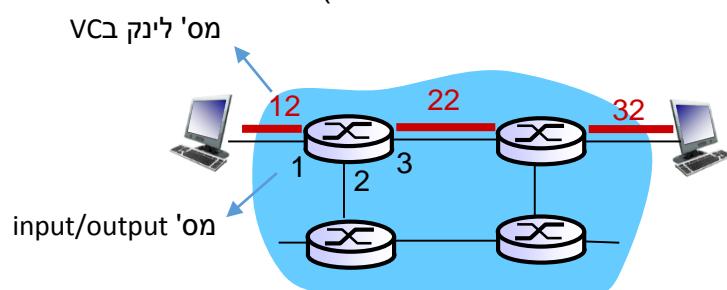
במרכז נתונים יש הפרדה בין קשר שמעביר כמות גדולה של נתונים (גיבוי שרתיים לדוג') לבין קשרים קטנים (בקשת דף אינטרנט). אלו מכונים בשמות ה'פיל' וה'עכבר'. אם משתמש ב MPLS להעברת 'עכבר' זו תהיה תקורה גדולה מדי והרבה מאמץ להעברת מידע מועט, لكن לעיתים משתמשים ב MPLS רק עבור קשרים שמעבירים מידע רב.



קשר בין 2 רכיבי קצה. לצורך המימוש נדרשים:

- מסלול דרכו ניתן יהיה להעביר את המידע
- מס' זיהוי (תוויות) לכל קשר, שכל החבילות המועברות בקשר זה יישאו אותה, ל'זיהוי הקשר ע"י הנטים.
- מס' זיהוי לכל-link בדרך. יכול לשינוי במהלך התקשרות.
- כניסה של כל הנטים ל-forward table.

(MPLS לדוג' מ眞ש את virtual circuits בצורה זו).



לכל נתב בדרך יש ב-forwarding table פירוט لأن לשירות כל חבילה, בהתאם למס' ה input ממנו היא נכנסת ובהתאם למס' הלינק ממנו הגיעה. לדוגמה – עבור הנתב השמאלי העליון בציור שלמעלה, יכולה להתאים טבלת הניתוב הבאה:

input on	Incoming VC#	output on	Outgoing VC#
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87



Datagram Networks – Protocol IP

(העליה השמאלי בעץ ה types types בעמוד הקודם – שיטת packet switching)

נכיר כי שיטת ה circuit switching קובעת מסלול מוגדר מראש עבור קשר מסוים, בעוד שיטת ה packet switching מנטבבת כל חיבורה בדרכה.

פרוטוקול IP נמצא תחת packet switching ולכן אין מסלול או קשרים מוגדרים מראש וחיבורה שעוברת דרכו יכולה להיות מיועדת לכל נתב אפשרי שיש. לכן לצורך הנি�זוב היינו צריכים שלכל נתב יהיה בforwarding table מידע על כל הנטים שיש – וכך כל כתובות הIP האפשריות הוא כ-4 ביליאן!

לכן כדי לחסוך זאת, הנתבים מזהים את הרשות הפנימית של הנתב אליו החיבור מיועדת – ומקדמים את החיבור בכיוון המתאים, ורק כאשר היא תגיע לנットו הראשי – החיבור תנותב ישירות לו אליה מיועדת.

שיטת הזיהוי נעשית באמצעות LPN –

LPN = Longest Prefix Match

השוואת כתובות היעד של החיבור לכתובות שבforwarding table, והכתובות בעלת התאמה הגבוהה ביותר של מס' הביטים הראשונים – היא הכתובה (נתב) אליה תעבור החיבור. בעצם כל רשומה מסמנת טווח כלשהו של כתובות IP בעלי אותה תת-רשות, ולכן יונטו דרכו אותו הoutputו בנתב.

משמעותו של האלגוריתם ניתן להבין כי אם יש 2 רשומות שמתאימות לכתובה יעד של חיבור – החיבור תעבור ליציאה לפ' הרשומה שמתאימה למס' הביטים הרוב ביותר (Longest).

בנוסף קיימת רשותת default עבור כתובות יעד שלא נמצא להן אף התאמה – והן מופנות ליציאה לעולם החיצוני. כדי להגדיר כתובות IP לנэт הכתובה צריכה להתאים לתת-הרשאות, אך להיות שונה מכל הgatesways כדי שלא תנותב לאיזה 다른 interface.

דוגמא:

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

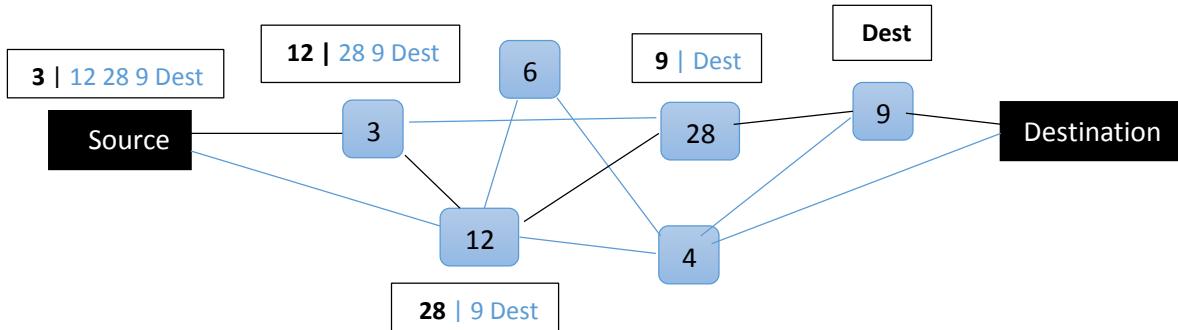
על ידי הטבלה, את כל כתובות IP שמתחלים ב- **00010** 11001000 00010111 - נוציא מיציאה מס' 0.

לעומת זאת, הכתובה 10101010 1011000 00010111 11001000 יש התאמה של יותר ביטים מרשימה 2.



Source Routing

שיטת ניתוב בה שולח החבילה מחייב מראש באיזו מסלול היא תעבור עד ליעד. כדי לממש את המסלול מוצמדות תוויות ב header אחת אחרי השניה כאשר הן מודבקות בסדר הפוך – מהסוף להתחלה, וההתווית החיצונית היא מס' הנתוב הראשון אליו היא תעבור. כל נתוב אליו מגיעה החבילה – מסיר את התווית החיצונית שמצינית את מספרו, ו מעביר את החבילה לפוי מס' התוויות הבאה שנגלית.



יתרונות: מהיר ופשוט!

חסרונות:

- יתכן ויש פטאות עומס באחד הנתבים – ואז החבילה יכולה להיתקע זמן רב בהמתנה לא יכולה לעבור במסלול אחר, או שהיא טיפול ותאבד.
- אורך header גדול מאוד ככל שהמරחק גדול, כי הוא מכיל את כל כתובות ה-IP שבדרך.
- השולח חייב להכיר את כל הנתבים במסלול כדי לציין את מספרם ב-header.

שיטה זו שמישה בד"כ ברשותות ספציפיות מקומיות, ולא בפרוטוקול ה-IP.

נושא בין השיטות השונות לנитוב:

	Source Routing	Global Addresses (IP)	Virtual Circuits
Header Size	Worst	OK – Large Address	Best
Router Table Size	None	Number of hosts (prefixes)	Number of circuits
Forward Overhead	Best	Prefix matching	Pretty Good
Setup Over Head	None	None	Connection Setup
Error Recovery	Tell all hosts	Tell all routers	Tell all routers and Tear down circuit and re-route



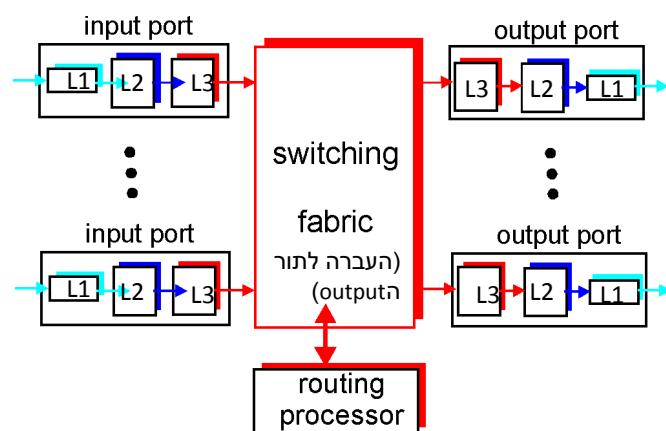
מבנה הנטבים

הבהרה: router – מתקיך עד רמת שכבה הרשות, switch – עד שכבת הקן.

כל נטב מבצע 2 פעולות עיקריות:

- הרצת אלגוריתם ניתוב (או פרוטוקול)
- העברת החבילה מ לoutput

לכל נטב יש תורי כניסה, תורי יציאה ומרכז בקרה:

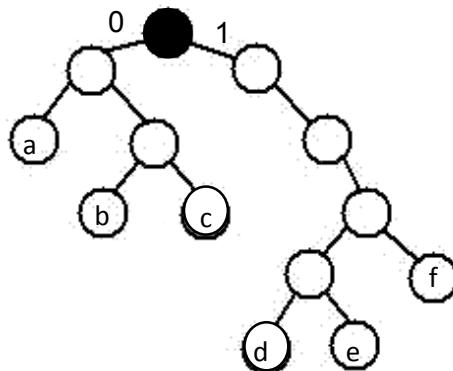


כשחביבה מגיעה, היא בא מהסוף הcano הפיזי – עוברת לשכבת הcano – ומשם נכנסת לתור ה. יש כאן מעין 'טייפוס' במעלה השכבות – Layer1, Layer2 ועוד 2 Layer 3 ובסוף הגיעו ל3. ביציאה מהנטב החביבה עוברת תהילך מקביל ארף הפור – יורדת ברמות. מעבר החביבה בשכבות אלו מתואר בציור בריבועים בתורי הנטב.

התורים וחלק הניתוב (הרכיבע המרכזי) שבנטב – ממומשים בד"כ בחומרה, בעוד שחלק הבקרה הכלול אלגוריתמים ווחילוטות בקרה שווה – ממומש בתוכנה. בין היתר במרכז הבקרה יש מנגנון שקבע עבור 2 חביבות שמאגדות במקביל מים שונים וצריכות לצאת לאורכו הים – מ' מבניהו תיכנס ראשונה לתור הים.



נחזיר לMPM – התאמת יצאה בנתב עפ"י כתובת ה-PI בעלת מס' רב ביותר של ביטים תואמים לאלו של כתובת היעד. חיפוש אחר הכתובות המתאימה ביותר דומה לחיפוש בעץ ביןארי, בו אנו יורדים רמה-רמה לפי הסיבית הנקראת. נניח כי כל ירידה ימינה בעץ היא קיראה של '1', וכל ירידה שמאליה – '0':



בעץ הנ"ל קיבל את prefixes:

a – 00 b – 010 c – 011 d – 11100 e – 11101 f – 1111

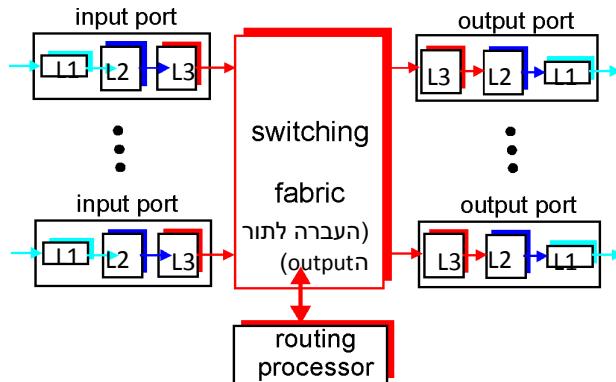
חיפוש אחר ה-ANPMPM בנתב עפ"י עז אורך זמן רב עד לקבלת תשובה, וכן צורך הרבה מאוד חומרה. לכן משתמשים היום בשיטת "זיכרון אסוציאטיבי" – בו הנתב מבצע אלגוריתםיעיל שבזמן קצר סורק את כל הרשומות ומצביע את ההתאמה הגבוהה ביותר בצורה מהירה. זה אמן דרוש המון משאבי חומרה אך יעיל מאוד.



הרצאה 9

(המשך מבנה הנットבים)

מציר את צורת הנטב:

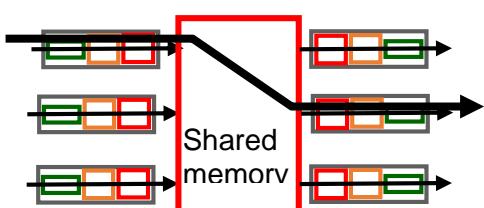


הנטב כולו ממומש בחומרה, מלבד *routing processor* שבד"כ ממומש בתוכנה. כפי שהזכרנו, מבחינת החומרה לא קל לבצע את אלגוריתם התאמות הקידומות בזיכרון צוז שכל ההשוואות נעשות במקביל, כמו כן נecessיות קרייאות רבות מהזיכרון, ולכן הזכיר אין זיכרון רגיל.

מציר את בעיית ה **HOL - Head Of Line breaking** – חבילות שנמצאות בראש תור אחת הכניסות ומתעכבות שם בשל המתנה ליציאה מסוימת שתתפנה, 'תיקעות' מאחריהן חבילות אחרות שמזווידות ליציאות אחרות שאול פניות.

Switching Fabrics

הניתוב ומעבר החבילות מתורי הדוטקסו לتوجيه הדוטקסו נעשה באמצעות שלוש הדריכים:



1. Memory – זיכרון משותף.

כל אחד מתורי הכניסה כותב לזכרון המשותף במקום המתאים לנק' היציאה של החבילה בראש התור, וכל יציאה תקרא את החבילות הממتنויות לה בזמן החופשי. מזכיר צורה של 'טא-דואר'.

חרונות: * תלוי בהגדרת הזיכרון המשותף, ומזכיר חלוקת זיכרון באופן דינמי כדי

להימנע מצב שיציאה מסוימת עומסה מאוד ומבדת חבילות בעוד

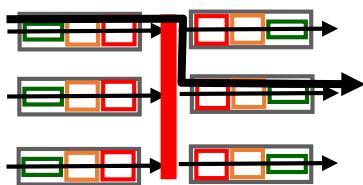
שיציאה אחרת פניה לחЛОוטין.

* מתבצעים 2 שלבים עבור כל מעבר חבילה:

1. כתיבה לזכרון 2. קראת מהזיכרון והעברה ליציאה.



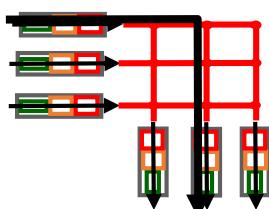
2. Bus – עורק ראשי.



העורק הראשי מעביר את החבילות במהירות, אך בכל רגע נתון יכול להעביר חבילה אחת בלבד.

חסרען: זמן! כיון שלא יכול להעביר כמה במקביל. שימושי בערך כשייש מס' רב של כניסה.

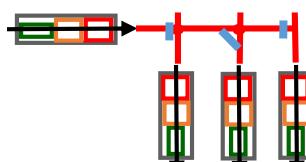
3. Crossbar – רשת



בכל חיבור בין שורה לעמודה יש מתג שבמצב דלוק מחבר בין השורה לעמודה מסוימת ומאפשר מעבר חבילה לעמודה זו, שבוסף מהוברת ליציאה. הדרך הנפוצה ביותר כינום.

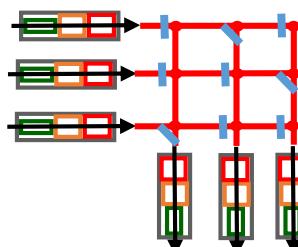
דוגמא 1: נרצה שהחבילה מכניםה 3 תצא מיציאה 2 (בתמונה רק יציאה 3).

אז, עבור יציאות 1, 3 המtag כבוי ולא מחבר את שורה 3 עם עמודות אלו, בעוד שעבור יציאה לעמודה 2 המtag דלוק וחבילה המגיעת מכניםה 3 תונטו בעמודה מס' 2 ברגע ההגעה למtag.



דוגמא 2: מכניםה 1 החבילה מנותבת ליציאה 2, מכניםה 2

החבורה מנותבת ליציאה 3, ומכניםה 3 החבילה מנותבת ליציאה 1.



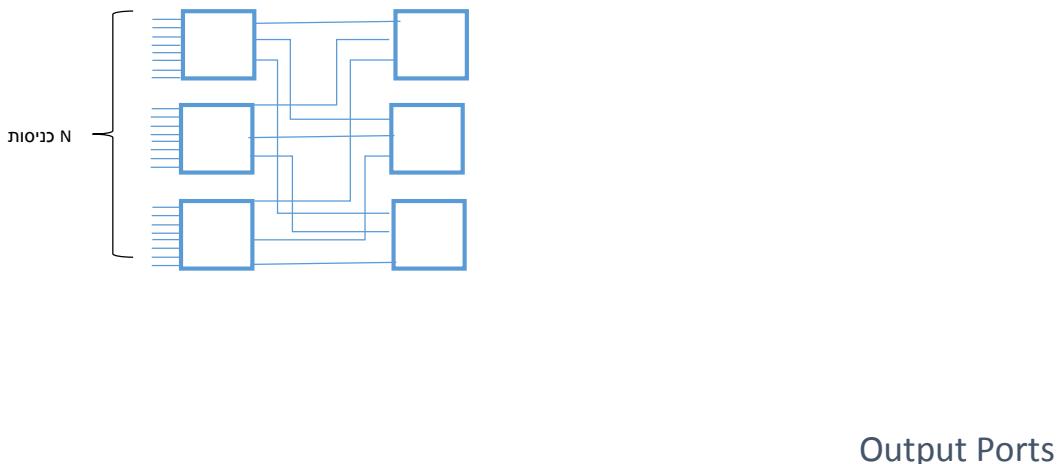
נבחן כי בכל שורה ובכל עמודה יכול להיות רק מתג אחד דלוק, אחרת יש 'קצר'. לכן לא ניתן לנוטב במקביל 2 חבילות מאותו הנסקו שרכוזות לצאת לoutputs שונים, וכן 2 חבילות מודולריות שונות לא יכולים לעבור במקביל לאותו הנסקו.

כדי לנצל את המעברים באופן אופטימלי נרצה למצאו "שידור מקסימלי" בין כל הכניסות והיציאות, שכן נבצע אלגוריתם שיבדק עבור כל תורי הכניסה – איזו חבילה מallow שנמצאות בתור כדי להעביר במקביל לחבילות אחרות מתחום אחר. האלגוריתם יכול להיות חמדני או לתת עדיפות לתור עמוס יותר.



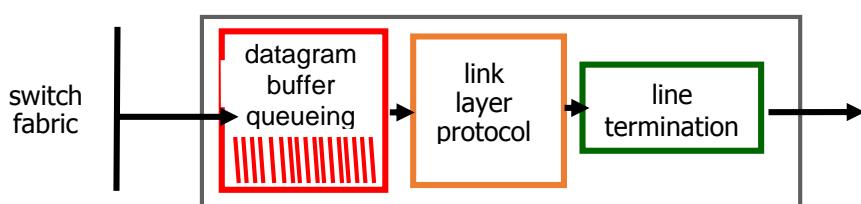
הקטנת המtag למתגים קטנים

בכל נתב, ללא קשר לדרכו מותבצע ניתוב החבילה `input` ל-`output` – ישנה היררכיה של מתחים (switches) – עם נ' להתמודד עם מס' גובה של ports. זה נעשה ע"י חלוקת המיתוג לרשת של מתחים: מכל כניסה בנתב, החבילה תעבור לשלב אמצע' המתאים ליציאה אליה היא מיועדת. בעצם נעשית חלוקת עבודה, במקום שמחtag אחד יבצע את כל הנזקונים – מס' מתחים קטנים יעבדו במקביל ויזרו את התהילה.



בapon אידיאלי הימן רצים רק תורים ביציאות מנותב, אך תורי הכניסות ואפשרויות הניתוב הפנימי מכילים אותם להשתמש בתורים נוספים.

נזכיר כי מבנה תורי היציאה הוא סימטרי לטור הכניסה, בסדר הפוך: ירידת משכבות הרשת לשכבות הקו וממנה לשכבה הפיזית:



בכל יציאה יש תור לשילחה, אבל הכניסה לתור מורכבת מוסף כניסה מכל הדעתינו של הנtab, וצריך להחליט מבין כל הכניסות הללו – איזו חביבה תיכנס קודם קודם לתור השילחה, בעcas איזו חביבה תצא קודם מהיציאה החז בנתב.

שיטת מס' 1: Round Robin – RR – מעבר בלולאה כניסה-אחר-כניסה והכנסת החבילה לתור לפי סדר הכניסות.

חומר: יכול להיות שם-`input` מסוים יהיה עומס, אבל בכל סיבוב תישלח רק חבילה אחת והעומס לא יתממש.



פתרון-שיטת מס' 2:

Weighted Round Robin – שיפור של שיטת RR, כאשר יש משקל לכל חבילה/ כניסה, וטיפול (ה כניסה לתור) לפי RR אך בעבר על כל skuton נטפל במס' חבילות בהתאם למשקל.

כיצד נקבעת עדיפות?

עדיפות לחבילה – אם חבילה מסומנת כ'דוחפה' בheader – נרצה להכניסה לתור לפני חבילות אחרות, וכן לקבל משקל גבוה.

עדיפות לכניסה – אם כניסה עמוסה, נרצה לתת לה משקל רב יותר מהאחרות, כך שבמעבר RR נכנס לתור מס' חבילות כניסה זו ולא רק אחת.

דוגמא –

עבור 4 כניסה לנתח עם העדיפויות הבאות:

1

3

1

2

במעבר RR, כניסה לתור חבילה אחת מה כניסה הראשונה, אח"כ 3 חבילות מה כניסה השנייה, חבילה אחת מה כניסה השלישי ו-2 חבילות מה כניסה הרביעית, ושוב חבילה אחת מה כניסה הראשונה וכו'....

יתכן גם כי בכל כניסה לתור יש flows שונים לחבילות שבה, ולכל flow משקל próprio. (ההגדרה המקובלת ל "flow" – חבילה עם אותן כתובות IP למקור וליעד).

מסך כל ההגדרות אפשריות למשקל – אנו מבינים כי העדיפות משתנה באופן דינמי.

חסרונות RR (כולל בתוכו את WRR):

- 'פספוס תור' – כאשר חבילה מגיעה לכניסה בדיק אחריו שעבר התור של אותה כניסה, והיא צריכה להמתין 'סיבוב שלם' של האלגוריתם על כל שאר ה כניסות, עד שהיא תישלח. וככל שהוא skuton גדול יותר – כך זמן המתנה של החבילה גדול יותר.
- גודל החבילות אינו זהה, כלומר skuton שמקבל עדיפות גבוהה וכל חבילה שלו היא גדולה יחסית – יכול יותר זמן טיפול מאשר החבילות שמתכונות להיכנס לתור.

פתרון-שיטת מס' 3:

General Processor Sharing – אלגוריתם זהה, מלבד שבמקום שכל עדיפות תהיה מס' החבילות שיועברו כשייגיע תורו של אותו skuton – העדיפויות תסמן את מס' הסיביות שיועברו לתור.

חסרון: יוצר מצב של פיצול החבילות לסיביות בודדות, והחבילה לא תישלח כחבילה אלא כסיביות.



פתרונות-שיטת מס' 4:

Weighted Fair Queuing – "טור הגון" – מבצע את המעברים של החבילות במלואן, אך מחשב את סדר העברת החבילות בהתנהלה שהעדיפות היא פר סיבית. ז"א, מחשב מה הינה סדר שליחת החבילות בתנעה שהאלגוריתם עובד לפי סיביות, ושולח את החבילות עפ"י סדר של איזו חבילה הייתה מסתimated להישלח ראשונה מבין כולן. אלגוריתם זה מהווים מעין מימוש אופטימלי של RR.

מסרנו: יקר ומורכב!

פתרונות-שיטת מס' 5:

Deficit Round Robin – אחרי כל טיפול בחבילה (העברתה מהזקוקו לטור היציאה מהנתב), אותו הזקוק מסומן ב'גירעון' של גודל החבילה ומוסר מהסבב, וכך רק אחרי ששאר הזקוקים עברו גם את הגודל הזה – הזקוק יחזור לשוב השליחה.

air נקבע גודל של תור בכל output? נרצה שגודל התור יהיה סביר וימנע נפילת חבילות. במשך זמן רב היה שימוש בהקצאת גודל עפ"י הנוסחה ($RTT * BW = Buffer_Window$) שגדירה את הניצול המקסימלית של קוו – משולח חבילות ללא הפסקה. חישוב זה מההערכה על קצב המשלוח בקשר TCP יחיד. לאחרונה הבינו שבפועל אין ניצול מלאה של גודל התור כי יש חלוקה של הזמן בין כל הקשרים הקיימים, וכן מספיקים תורים הרבה יותר קטנים. כתע גודל תור מוקצה ע"י $\frac{RTT * BW}{\sqrt{n}}$ (לא נלמד על air בבדיקה הגיעה לכך).

הקטנת התור תורמת לביצועים טובים יותר, לזמן טיפול קצרים יותר וזמן התואשנות מנפילות קטניות יותר.

** הערתת קטנה: כשמדובר על נתב אז עולה בראש מחשבה על קופסה קטנה כמו בבית, צריך לזכור שככל נתב זה משחו עצום שישיר לרשת גדולה...

Router Design

דברים שצראים להיליך בחשבון בעיצוב ראותה:

- שיקולים של חיסכון באנרגיה, תפוקה, גמישות, אמינות
- צואර בקבוק – בבדיקה ה longest Prefix Match, ומעבר ב crossbar
- טרנדים – מעבר לחשיבה מבוזרת וזרה לסדרן של ניהול מרכזי



לממנו על חישובים רבים שנעשים עבור מסלולים (בפרוטוקול TCP למשל), אבל בעת שינוי מסלול השולח לא ידע שהמסלול שונה וועלוי לחשב מחדש כל מה שנוגע במסלול. כנש בקר מרכזי, הAKER יודע על שינויים כאלה וכך יוכל להודיע לשולח בעת הצורך.

Network Processors

טיפול בחבילות יכול להיעשות ע"י תוכנה אך יעיל יותר שימוש בחומרה.
Time To Live – מוגדר ברגע השיליחה, וכל נתב בדרך מקטיין אותו ב-1. אם חבילה הגיעה ל-0=TTL לפני שהגיעה ליעדה – זה מעיד על משהו לא תקין במסלול.
ה-TTL ניתן למימוש פשוט בתוכנה, אך מצרי בכל פעם קריאה מהזיכרון, חישוב כתיבה בחזרה וכו', וזה דבר שאינו יעיל. בחומרה לעומת זאת זה קל מאוד, ופשוט יותר להחסיר 1 בכל נתב. נבחן גם כי כל שינוי ב-TTL גורם לשינוי ה checksum וה-ecn ויש לבדוק גם אותן – אך גם זה קל יותר למימוש בחומרה.
از למה בכלל זאת לא לעשות זאת בחומרה? כי היא יעודית. בبنיתה היא מיועדת לביצוע משאנו מסוימים ואין להקליבליות, לא גמישה לשינויים. כל שינוי קטן בפרוטוקול גורם לחומרה להיתקע.

פתרונות: **Network Processors** – מעבדי רשת – צירוף של תוכנה וחומרה, עם מעבדים ספציפיים מאוד לטיפול בחבילות. EZCHIP לדוגמה היא חברה מובילה בתחום זה.

בעיות:

- יש הרבה מטלות שונות ומגוונות לטיפול בכל חבילה
- זמן לא אחיד לטיפול בחבילות
- לחבילות יש עדיפות שונות ורמות שירות שונות אותן הן דוחשות

תזכורת איפה אנחנו נמצאים - שכבת הרשת

铭记 כי פעולות שכבה זו הן אלגוריתם חישוב המסלול והעברת חבילות עפ"י טבלת הניתוב. פרוטוקולי השכבה הם IP ו-ICMP.



4. IP של IPv4 (גרסה 4 – העיקריות עליה נלמד)

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																												
0	Version	IHL	DSCP										ECN										Total Length																																					
32	Identification										Flags		Fragment Offset										Header Checksum																																					
64	Time To Live					Protocol					Header Checksum																																																	
96	Source IP Address										Destination IP Address																																																	
128	Options (if IHL > 5)																																																											
160																																																												

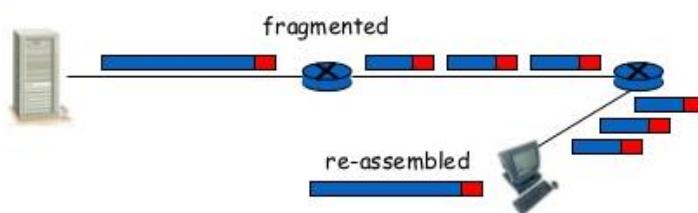
- גרסת ה-IP – גרסה 4 במקורה זה.
- אורך header - Internet Header Length - **IHL** – אורך header ב- Bytes. איןנו קבוע בגל ה-options.
- **DSCP** = Type Of Service – זוהי ה-Service Code Point – משפיע על הממשק בהתאם ל-service.
- **ECN** – Error Congestion Notification – דיווח הנتابים שבמסלול על עומסים בדרך (למדנו על כך במסגרת TCP) – מופיע בעמוד 46).
- **Options** – כולל דברים רבים, ביניהם יכולם להופיע חתימות זמן או כתובות IP של כל נתב שהחביבה עברה אליו בדרך לעד.
- גודל כולל של כל החביבה, נמדד ב- Bytes. **Total Length** – סימון האם החביבה פוצלה ע"י הנתבים בדרך לקטנות יותר.
- **Flags** – זיהוי החביבה המקורי שנשלחה. ז"א שאם היה פיצול – כל תתי-הביבות יכולים אותו IP של החביבה המקורי ממנה פוצלו.
- **Identification** – זיהוי החביבה המקורי שנשלחה. במילים אחרות אומר: 'תשים את החביבה הזה בתיאריה של המקורי שפוצלה.'
- **Fragment Offset** – offset בהסתמך לפיצול, מעיד על מקום תתי-הביבה בין כל בתיאריה של המקורי שפוצלה. במילים אחרות אומר: 'תשים את החביבה הזה במקום מהחביבה המקורי'.
- **TTL** – Time To Live – הזכרנו קודם, שכל נתב בדרך מפחית את TTL-1, ואם TTL הגיע ל-0 לפני הגעת החביבה לעד – זה מעיד על תקלת כלשהי בדרך.
- **Protocol** – הפרטוקול בו נשלחה החביבה בשכבה הרביעית מצד השולח. חשוב רק למקבל הסופי בקצת הקשר ולא לנتابים שבדרכ, עם"נ שידע כיצד להעביר את החביבה לשכבה מעלי. בד"כ הערך מצביע על TCP או UDP.
- **Header Checksum, Source/Destination IP Address** – הרחבענו רבות, אין מה להסביר...



פיזול חבילות = Fragmentation

קורה כאשר גודל החבילה גדול מקיבולת הליינק- הגודל המרבי להעברה. גודל זה נקרא **Max Transfer Unit – MTU**. הפיזול יכול לkeroot ע"י כל נתב בדרך, אך איחוד מחדש של חבילות שפוצלו יעשה רק בהגעתן לעד הסופי.

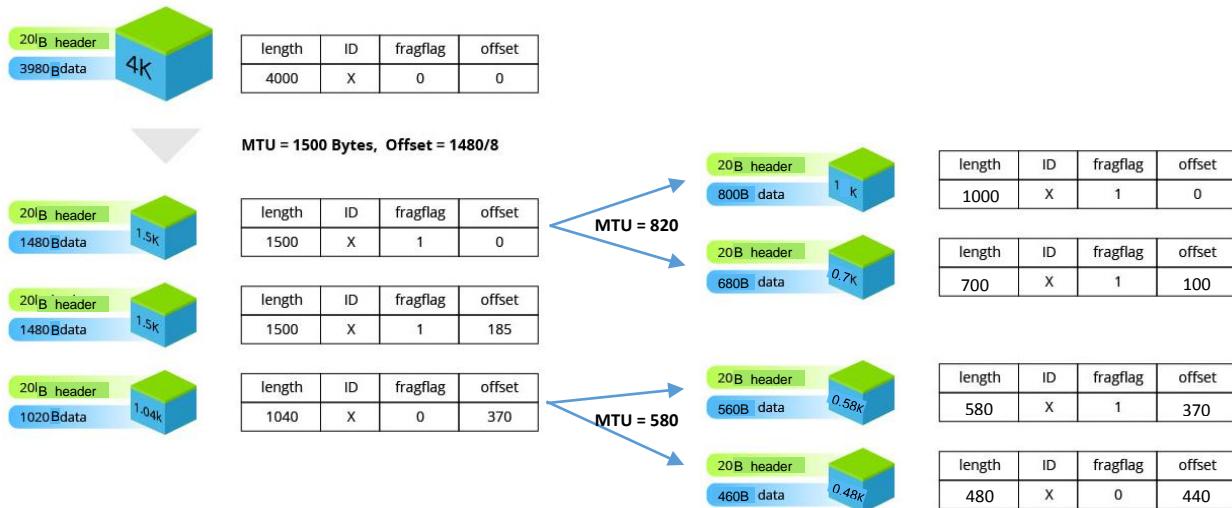
בבקורת הזרימה של TCP למדמו כי הגדלת החלון בצד השולח נעשית בגדיי MMS – שהוא גודל מקסימלי של חבילה שנוגדר מראש בצד השולח, ונקבע על סמך הליינק שיכל להעביר גודל חבילה קטן ביותר במסלול מהמקור לעד. אך אם גודל חבילה נקבע מראש על סמך לינק זה – מדוע ניאלץ לפצל חבילות? כי לעיתים ניתוב החבילה משתנה בדרך.



בעת פיזול של חבילה בגודל X, הפיזול נעשה על המידע שבה ולא על header. נסמן את גודל header ב-H, ונקבל ש H-X של המידע של חבילה יפוצלו ל- $\left\lceil \frac{X-H}{MTU-H} \right\rceil$ חבילות. המכנה בנוסחה מגיע מכך של מידע כל תת-חבילה מצורף header בגודל header – H, ויחד זה גודל חבילה מקסימלי בlienik.

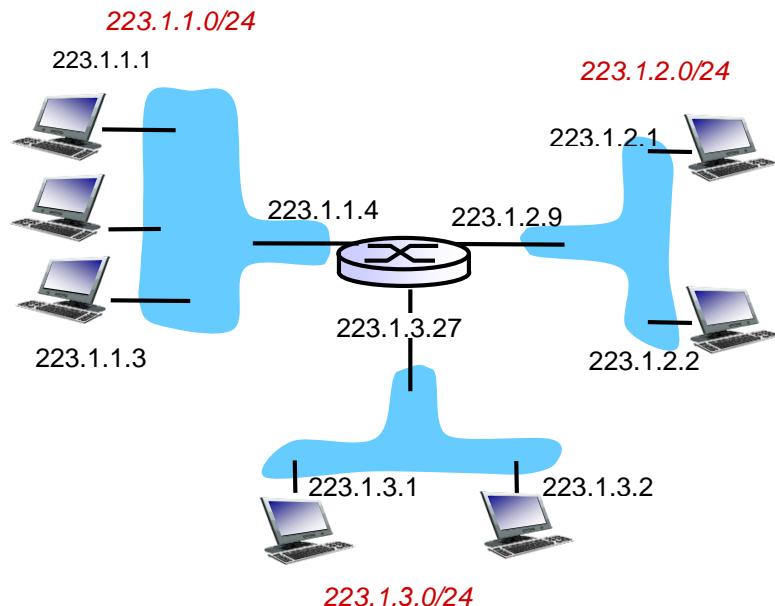
Flag שערכו 0 מסמן שי זו חבילה האחרונה בפיזול. במקרה של פיזול חוזר של חבילה, ז"א שתת-חבילה התפצלה שוב – ב-HID ואשר שדות header לא השתנו, מלבד offset שידיros ע"י הפיזול הנוסף. בנוסף, יתכן מצב שהflag של חבילה האחרונה ישנה, כיוון שאם תת-חבילה הראשונה לא הייתה האחרונה בפיזול, וה-flag שלה הוא 1 – אז כל תת-ה חבילות שלה גם יכולו 1.

אם תת-החבילה הייתה האחרונה בפיזול, וה-flag שלה הוא 0 – אז תת-ה חבילות שלה יכולו 1, ורק האחרונה מבניהן תכיל 0.flag.



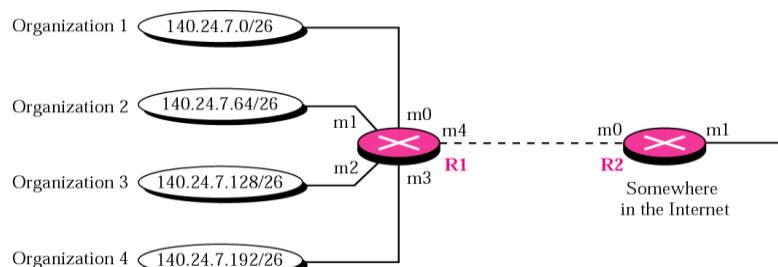
כתובות IP

כל כתובת IP של נטב קשורה באופן ישיר לרשות שאליהן הנטב מחובר. לנטב יחיד יכולות להיות מס' כתובות IP:



Mask – התחלת הארוכה ביותר המשותפת לכל hosts באותו תת-רשת. (longest prefix match). מסומן בתמונה באדום.

נביט בדוגמא לניטוב:



בהעברת מידע מהנטב R2 – מספיק להסתכל על **24 סיביות**, כדי לדעת האם להעביר את החבילה לרשת ע"י interface m0, או אם להוציא את החבילה לאיזור אחר ברשת ע"י interface m1, כיוון שהMask של 4 hosts שבחלק השמאלי של התמונה הינו 140.24.7.***.***** וכולל את כל האפשרויות Byte האחרון, ולכן מספיק לבדוק את 3 Bytes הראשונים כדי לשלווח חבילה לכיוון זה. לעומת זאת, חבילה שייצאת מאחד hosts יכולה להישלח או לנטב מחוץ לרשת הפנימית, או לאחד hosts שברשת, ואלו נבדלים ביניהם רק ע"י 2 הסיביות האחרונות וכאן יש צורך לבדוק 26 סיביות ולא רק **24**.



טבלאות הניתוב של הנתבים ייראו כך:

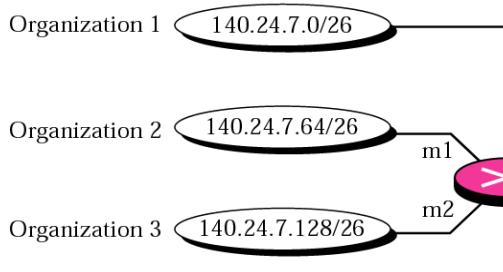
Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	-----	m0
/26	140.24.7.64	-----	m1
/26	140.24.7.128	-----	m2
/26	140.24.7.192	-----	m3
/0	0.0.0.0	default router	m4

Routing table for R1

Mask	Network address	Next-hop address	Interface
/24	140.24.7.0	-----	m0
/0	0.0.0.0	default router	m1

Routing table for R2

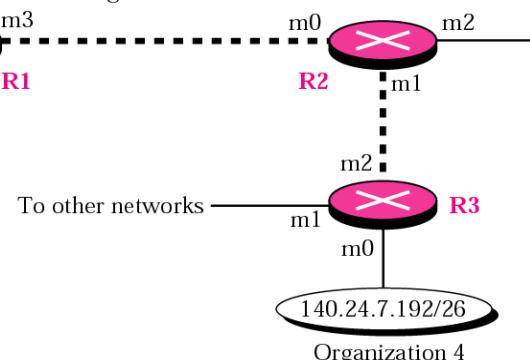
אם נסרק את העניין קצת יותר, אז עבר הראותים הבאים יהיו הטבלאות הבאות:



Routing table for R1

Mask	Network address	Next-hop address	Interface
/26	140.24.7.192	-----	m1
/24	140.24.7.0	-----	m0
/??	???????	?????????	m1
/0	0.0.0.0	default router	m2

Routing table for R2



To other networks

Organization 4

Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	-----	m0
/26	140.24.7.64	-----	m1
/26	140.24.7.128	-----	m2
/0	0.0.0.0	default router	m3

Routing table for R3



הרצאה 10

חלוקת כתובות IP

בעבר, כתובות ה-IP היו מוחולקות למחצית כאשר תחילית של כל מחלוקת היא באורך Byte שלם – ז"א 16, 8 או 24 סיביות, וכל מחלוקת הייתה אחראית למספר המחשבים שתחתייה.

כתובות רבות בוצעו בצורה כזו, למשל עבור מחלוקת שלא ניצלה את כל כתובות ה-IP שיכולות להיכלל בה.

כמו כן, לעיתים כשתחילית המחלוקת הייתה באורך 24 – ז"א שהיה לה רק 8 סיביות לכל המחשבים במחלוקת זהה ולא תמיד היה מספיק.

לכן עברו לשיטת ה- **CIDR – Classless Inter Domain Routing** – קידומות באורכים שונים (כמו היום), כאשר מצינים את אורך הקידומת (-הmask), לדוגמה: 1.23.056.2\23.

אפשריה 1 לחלוקת IP:

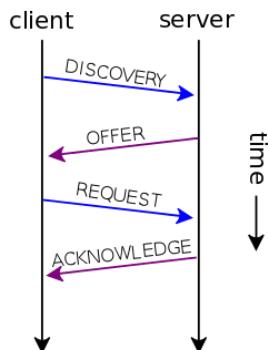
DHCP – Dynamic Host Configuration Protocol

מגדיר כיצד מחשב שמצטרף לרשת מקבל כתובות IP, עובד בשיטת 'Plug-and-play':

כיוון שמחשבים שאינם שרתים מתחברים לרשת כל פעם בצורה זמנית, הם לא מקבלים כתובות IP קבועה, אחרת כתובות IP יגמרו מהר מאוד. לכן כתובות ה-IP 'מומוחזרות'. כל ארגון מחזיק אוסף כתובות IP אותן 'MSCIR' לזמן מסוים ל-hosts שמתמחבים, ובסיום הזמן המוקצב המחשב צריך לבקש הארוכה אותה הכתובת.

ה- DHCP מיידע את המחשב המctrף על:

- כתובות IP
- מהו ה mask
- מהו ה Default Gateway
- כתובות ה DNS server

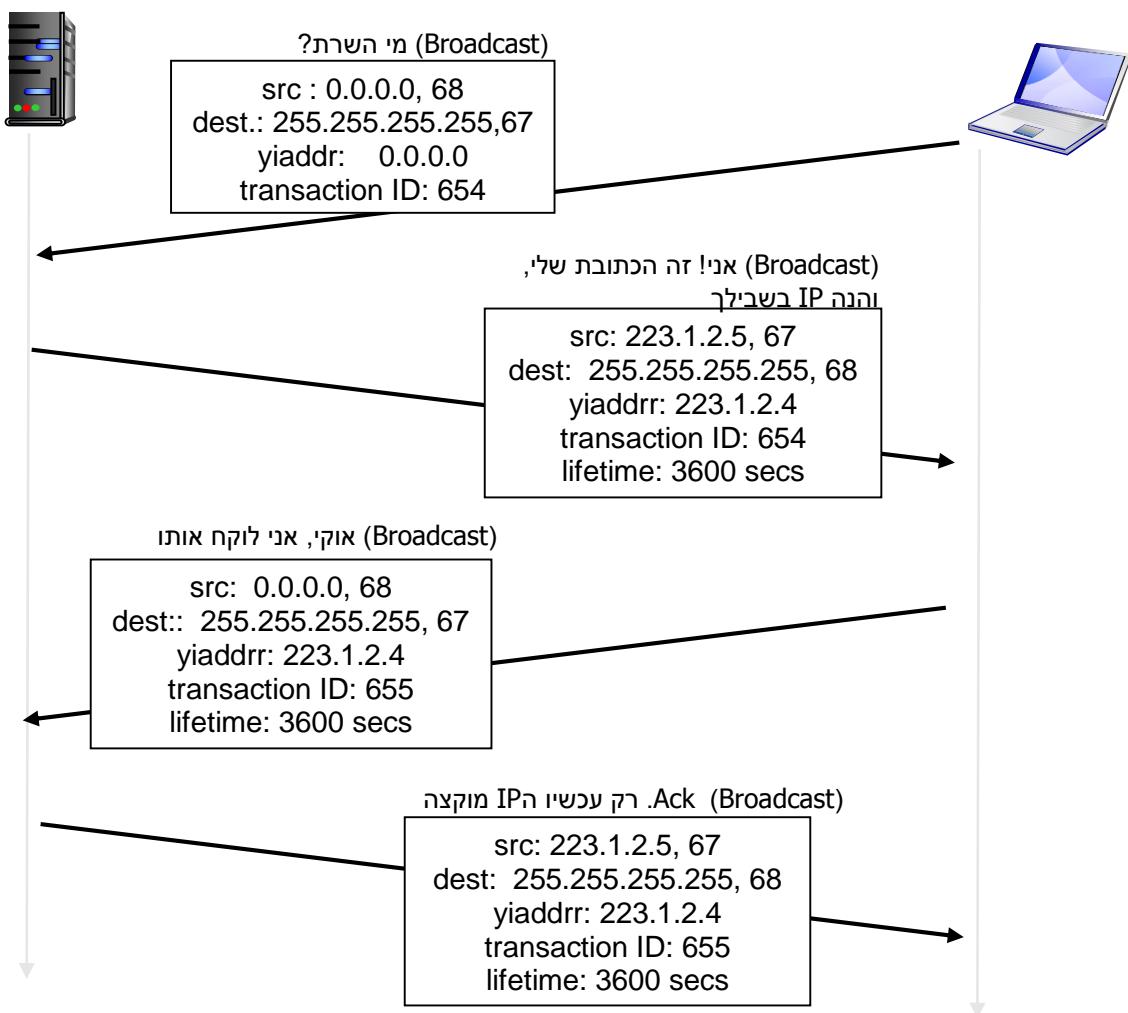


ה策רפות המחשב לרשת וקבלת כתובות IP נעשית למען לחיצת-ידיים 'מרובעת' (4 הודעות), כאשר חלק מההודעות נשלחות ב- **Broadcast** – הודעות שמופצות לכל המחשבים ברשת (כולל השרת).



DHCP server: 223.1.2.5

Arriving client



Discovery : בעת הבקשה הראשונה, המחבר מחשף את השרת ולכון מפי' לכולם 'מי', yiaddr = Your Internet Address, כוונת המקור היא 0. מי שעובד אין לו IP, כתובת המקור היא 0. מועד לתשובה השרת. הבקשה מקבלת גם IP למקורה שיש מס' מctrפים בו-זמןית.

Offer: השרת מוסר למצטרף את כתובת IP בשדה `iyaddr`. גם זה נעשה בBroadcast, כיוון שהכתובת טרם הוגדרה ולמצטרף החדש אין שום כתובת מדוייקת כדי שיוכלו לשולח לו הודעה באופן פרטי.

Request : על המctrarף לעדכן את השרת כשהוא לוקח את כתובות הPI כיון שייתכן ומוס' שרתים סבירו שלחו לו כתובות והוא לוקח רק אחת מכלן, וכן השרתים שלא קיבלו הודעה שמאשרת את קבלת הPI יידעו שהכתובות עדין פנויה ויכולו להינתק למסטרפים אחרים. עדכן זה לא תמיד נעשה broadcast אך פעמים רבות כן, גם בשביל שאר השרתים וגם כדי לעדכן את שאר המחשבים ברשת על הctrarfto.

Ack : רק ברגע זה כתובת ה-IP מוגדרת.



אופציה 2 לחלוקת IP:

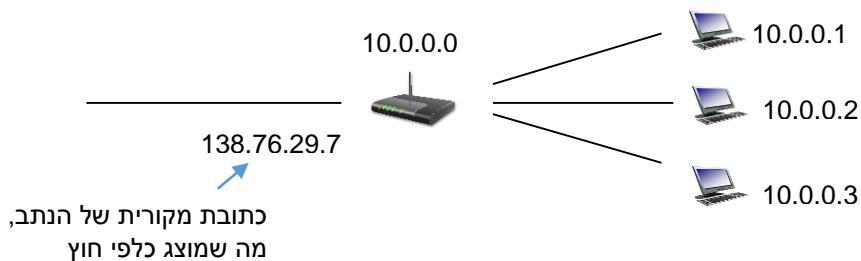
ICANN – Internet Corporation for Assigned Names and Numbers

אחראי על חלוקת הכתובות. מקצת כתובות ומשם Domains וכן מנהל את ה DNS. מארגן באופן היררכי – מתן טווח כתובות ('בלוק', רצף) עבור מחוז, בתוך מחוז מתן טווח כתובות עבור רשויות מקומיות וכו'.

למרות אפשרות החלוקת, אנו עדין מוגבלים כיון שכטובת IP מוגדרת מ-32 סיביות ולכן ניתן להגעה רק עד 2^{32} כתובות, לצורך כך הוגדר ה NAT:

NAT – Network Address Translation

נתב מקבל כתובת IP מסוימת, אך מוגדר עבור הרשת הפנימית (כל המחשבים שמחוברים אליו) כתובת IP : 10.0.0.0 : ומקצה למחשבים ברשת כתובות וירטואליות מהצורה *.*.*.*. כל חבילה שמגיעה מחוץ לרשת הפנימית, מיועדת למחשב ברשת – לא משנה איזה מהם – כתובת היעד שמצוינת על החבילה היא הכתובת המקורית של הנטב, וכאשר החבילה מגיעה אליו – הוא אחראי על העברתה למחשב המתאים, ע"י port אותו הגדר הנטב עבור אותו מחשב.

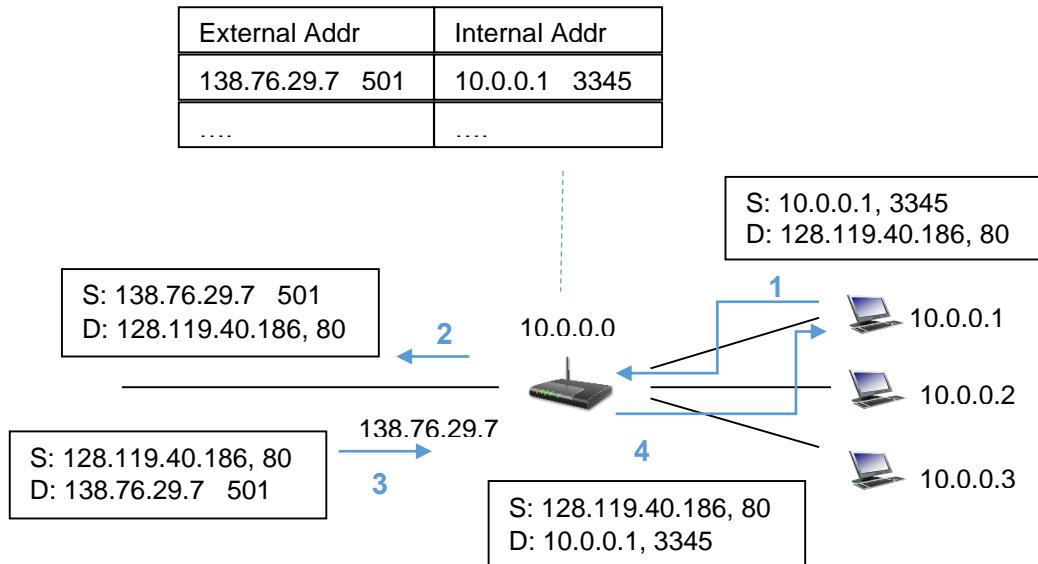


כדי שהנטב ידע להבחין בין החבילות ולנתבן למחשבים ברשת, הוא שומר לעצמו טבלה. כל מחשב ברשת ששולח הודעה אל מחשב חיצוני, הנטב מוסיף רשומה חדשה בטבלה בה הוא שומר את הכתובת הפנימית של השולח ואת הזוק שצוינו על החבילה (כדי לזהות חבילות כניסה באופן חד-ערכי), מגדר במקומן על החבילה את הכתובת IP שלו (הכתובת החיצונית לכל הרשת) ו-port מסויים שבחר, ושומר את הרשימה בטבלה:

External Addr	Internal Addr
138.76.29.7 501	10.0.0.1 3345
....

כאשר תגיע חבילה בחזרה מבוחר אל השולח, היא תוכל את כתובת היעד של הנטב, כי זו הכתובת שנראתה על חבילה שנשלחה, ולכן כשהנטב קיבל את החבילה הוא יסתכל בטבלה שאצלו כדי לזהות למי היא מיועדת, יחליף בחזרה את IP והport ויעביר את החבילה למחשב שברשת.





השימוש ב-NAT יוצר מחלוקת כיון שהוא דורס את החלוקת המוכרת שנtab מתעסק בחבילות עד השכבה השלישית, וכך הוא נוגע בערכים של port ששוויכים לשכבה רביעית ומשנה אותם.

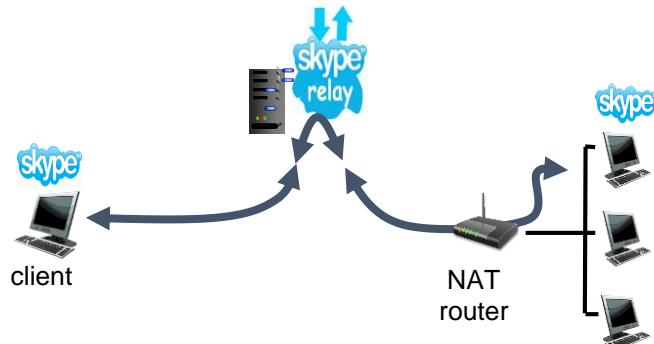
כדי לא לבלבל בין חבילות ולשלוח אותן לאפליקציות שונות לא מיעדות אליהן (החלוקת לאפליקציות נעשית עפ"י *port*) – הנtab קובע ערכי *port* שאינם בשימוש. לדוג', הוא לא יקבע 80 *port*. ***קיימת רשימה ידועה של ports שמדוברים כיחידים.**

מה קורה כאשר מחשב חיצוני ירצה לשולח הודעה למחשב פנימי בלי שנשלח לו הודעה קודמת ממחשב ברשות? – ישנה בעיה כאשר יוזם הקשר (הצד ששולח ראשון הודעה) הוא מחוץ לרשta, כי לא קיימת רשומה מתאימה בטבלה שתוכל לנtab חבילה שmagעה.

נחקק את הבעיה ל-2 מקרים:

- עבור פניות לשרת: כדי למנוע בעיות, ה-NAT אחסן אצל רשותה ובה הכתובת הפנימית של השירות -*port* מסוים שיידוע לכלום שזהו *port* ספציפי לשרתים (סטנדרט כלשהו), וכל חבילה שתגיע עם *port* זה – תופנה לשרת.
- בעצם חיבור P2P נעשים בפועל דרך לרשת ולא ישרות בין מחשבים.

דוגמה: Skype



ICMP – Internet Control Message Protocol

פרוטוקול המאפשר משЛОח הודעות בקרה ודיוח על מפעים.

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

כל הודעה מכילה מס' type (קטgorיה) ומס' code.
 כל נתב בדרך יכול לשלוח הודעה בקרה, בהתאם לסוג ההודעה אותה רצה לשלוח:
 הودעת TTL (time to live) למשל. יכולה להישלח ע"י כל נתב בדרך, בעוד שהודעת ping יכולה להישלח רק ע"י המקלט.
 לעומת זאת, הודעה dest host unreachable – יכולה להישלח רק ע"י הנתב שלפני המחשב המקלט, וכך שמחפש את המחשב אך לא מוצא התאמה למה שרשם על החבילה.

ICMP הוא פרוטוקול מעל ה-IP ששולח חבילות בצורת IP. הודעה תכלול את error, ואת header ואת Bytes הראשונים של החבילה שגרמה לביאה.

ה-TTL מאותחל ע"י המקור כרצונו. כפי שאמרנו בעבר, כל נתב בדרך מפחית את ערך TTL ב-1, ואם נתב רואה TTL=0 – זה סימן שהחבילה לא הגיעה לעדה לפני הזמן שהוקצב לה, ולכן נדרש לשלוח הודעה בקרה מסוג TTL.

השלוח יכול לזהות את המסלול דרכו עוברות החבילות עד לעד ע"י שליחת חבילות עם ערך TTL שמתחליל מ-1 ומעלה עם כל חבילה שנשלחת. הנתב הראשון שהשליח יפחית את ערך TTL, וכיון שקיבל חבילה עם TTL=1 – לאחר ההפחתה יהיה TTL=0 ולכן יאלץ להחזיר הודעה שגיאיה. כמו"ל לגבי הנתב השני – כאשר מקבל חבילה שאותחה מראש TTL=2, אז הנתב הראשון יפחית ב-1, והנתב השני יפחית ל-0 וגם הוא יצטרך לשלוח שגיאיה. כך כל נתב בדרך ישלח הודעה, והשלוח ידע מי הנתבים שדריכם עוברות החבילות.

כיצד ידע השלוח שהמסלול נגמר? הוא ישלח חבילה עם כתובת IP תקינה אך עם port שאינו קיימ. אז הנתב האחרון ירצה להעביר את החבילה לפ' הорт אך יתקע ולכן ישלח שגיאיה.



** ניתן לראות זאת מה@aoleo ע"י הפקודה tracert וה-link המבוקש. בכל ttl המחשב שולח 3 ניסיונות, ומחזיר מי הנטב, כאשר * מסמן לא נמצא. עבור כל נטב בדרך ליעד נקלט שורת מידע:

```
caj C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\Reut>tracert bgu.ac.il
Tracing route to bgu.ac.il [132.72.138.38]
over a maximum of 30 hops:
  1      3 ms      19 ms      17 ms  132.73.207.254
  2      18 ms      1 ms       3 ms  132.72.123.131
  3      1 ms       1 ms       3 ms  tzin.bgu.ac.il [132.72.138.38]

Trace complete.

C:\Users\Reut>
```

IPv6

קם בגל החוסר בכתובות IPv4, שהו מוגבלים לאורך של 4 Bytes. IPv6 מאפשר כתובות באורך 128 סיביות. לעומת זאת, header שלו בגודל קבוע – 40 Bytes. IPv6 מנסה להיות יוטר מהIPv4, כי המורכבות של IPv6 מקשה על הנטבים, בעיקר בגל headers שבאים בגודלים שונים וריאציות שונות.IPv6 אין פיצולibilities, במקרה של חבילת גודלה מדי – תישלח הודעה בקירה מסוג 'packet too big'.

IPv6 Header

Version (4 bit)	Traffic Class (4 bit)	Flow Label (24 bit)	
Payload Length (16 bit)		Next Header (8 bit)	Hop Limit (8 bit)
Source Address (128 bit)			
Destination Address (128 bit)			



:header

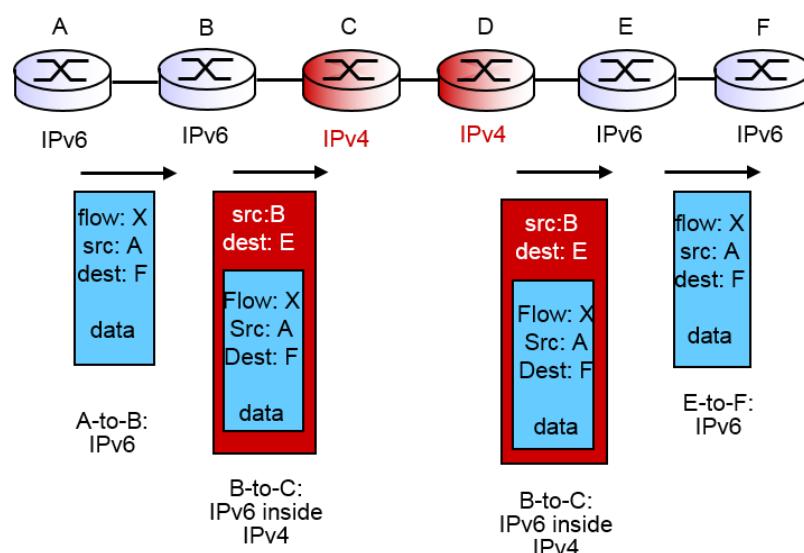
- **Ver** – מס' גרסה. במקורה זה – 6.
- **Pri/Traffic Class** – עדיפות לחבילה. מקביל לdscp של גרסה 4.
- **Flow label** – תופס 24 סיביות ולא מוגדר היטברגע, מיועד לשימוש עתידי.
- דוגמת שימוש: במקורה שפותחים browser מס' tabs לאוטו שרת – אז header של שכבה שלישית יהיה זהה בכל הבקשות, וההבדל בין החבילות יהיה מס' port header של שכבה רביעית. لكن יתכן וזה יצביע בחבילות ע"י שדה זה.
- **Payload len** – אורך החבילה (תליי בגודל המידע)
- **Next hdr** – כתובת header הבא בחבילה. נזכיר כי כל חבילה שיוצאת משכבה האפליקציה- מתווסף לה header בכל שכבה עד שמועברת בדרך. לכן במקורה זה header של שכבת הPA מצביע לheader של שכבה רביעית.
- **Hop limit** – זה TTL – זה הזמן.

נבחן כי שדה checksum ירד, כיון שהIPv6 מסתמך על הבדיקות שנעשות בשכבות אחרות – וזאת מכיוון שהheader לא משתנה במהלך המשלוח מלבד ערך 'יחיד' – nexthop, ולכן חבילתו יתבצע כל פעם חישובים בגלל שינוי אחד קטן. مكان שיש פחות זמן עיבוד בנתבים! וזה מטרתו העיקרית.

המעבר מIPv4 לIPv6

קורה במהלך 20 השנים האחרונות. כיון שהמעבר לא יכול להיות פטאומי אלא קורה בתהליך ארוך, יש צורך בתמיכה דו-גיטריסטית בנתבים. לכן משתמשים במנטור – העברת חבילות לצורה של מעיטה בתור מעיטה.

למשל: אם הקצאות תומכי IPv6 אך בדרך ישנו נתב שלא תומך, אז הנתב שלפני אותו אחד שתומך בגרסה 4 בלבד – יכניס את החבילה של IPv6 במעטפה עם header עם IPv4, וכשיש חיבור לנットב תומך IPv6 – החבילה תצא מהמעטפה.



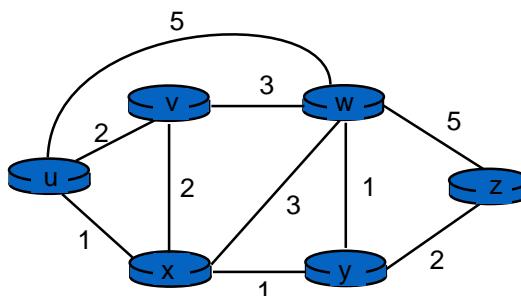
נשים לב שיש בעיה של כתובות IP – חלקם באורך 32 סיביות וחלקם באורך 128! לכן יש כפליות כתובות בתחום המעבר. המחשבים הנוכחיים שלנו למשל – מחזיקים 2 כתובות IP באורכים שונים.

כמו כן התחילה איטי מאד, וכן גם השפעת המהירות למשתמש אינה משמעותית כי גם אם הנתב אצל המשתמש יוחלף לנtab תומך IPv4 – בדרך עדין החבילה נתקעת בנתבים שתומכים בIPv6 בלבד.

לכן לחברות אין אינטראס איטי בהתקדמות וההתקדמות היא איטית. המטרה במעבר היא רק כללית יותר.

Routing algorithm

רשת תקשורת היא בעצם גרפ – אוסף של צמתים (=נתבים) וקשתות (لينקים). לכל קשת יש משקל, כאשר לעיתים הוא סטטי (קבוע) אך לרבות דינמי, יכול להיקבע עפ"י: אורך, עומס, רוחב פס וכו'.



אלג' למציאת מסלול מינימלי

ברשת ריכזית – כל נתב מכיר את כל הרשת, לכן משתמש ב **link state**.
ברשת מבוזרת – כל נתב מכיר את הדרכ היכי קצרה עד לנק' מסוימת, ולא מעבר לה.
השימוש הוא באlg' **distance vector**.



הרצאה 11

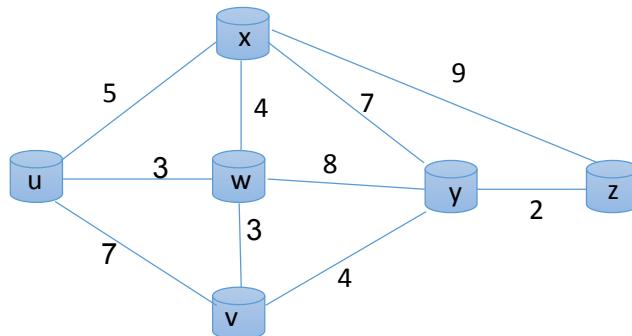
אלגוריתם Link-State

אלגוריתם בו כל נתב מכיר את כל הרשת: כל נתב מפיץ את המרחק שלו משכניו, וההודעות העוברות בין כל הנתבים עד שכל הנתבים ברשות מכירם את כל המרחקים (הקשרות). כדי לחשב לכל נתב את המסלולים הći קצרים ממנו לשאר הנתבים, משתמשים באלגוריתם של דijkסטרה.

Dijkstra's Algorithm

העיהון האלגוריתם: נרצה לבנות עץ פורש מצומת מסוים – u. נעדכן בטבלה את המרחק מ-u לכל אחד מהשכנים, ואת המרחק ממנו לכל שאר הצמתים – ∞ . בכל שלב נבחר את הצומת הבא דרכו יעברו המסלולים – על סמך המסלול הקצר ביותר, ועבור כל קשת שיוצאה ממנו צומת: אם הגיעו לצומת חדש – נוסיף את הקשת לטבלה, אחרת – נבדוק אם שווה לנו להשתמש בקשת זו ולעדכן את המסלול לאוטו הצומת כך שהיא יהיה קצר יותר.

דוגמא: עברו הגרף הבא –



נרצה למצוא את העץ הפורש, שמכיל את המרחקים הכי קצרים מקדקדוד u לכל שאר הקדקדדים. לצורך כך, נמלא את טבלת המרחקים הבאה:

step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0						
1						
2						
3						
4						
5						

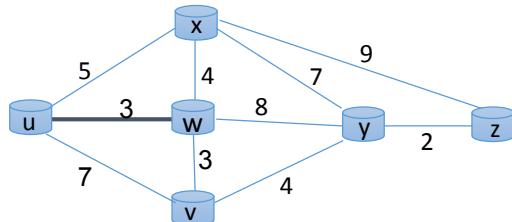
כאשר י'א אלו הקדקדדים שכבר נמצאים בעץ הפורש, ו- k הוא המרחק מ-u לקדקדוד.



בשלב 0, נעדכן את המרחקים מ-u לשכניו. כל קדקוד שאיינו שכן – נסמן את המרחק ב- ∞ . נבחן כי כל מסלול צזה עובר דרך קדקוד u בלבד, ולכן מסומן u במשבצות המתאימות. כמו כן, בשלב זה רק u שייר לעצט:

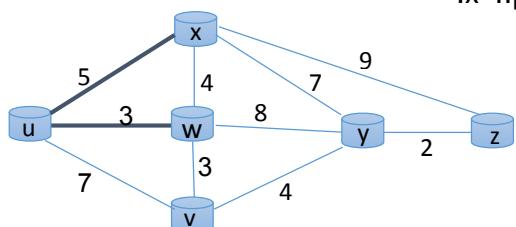
step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	u	7, u	3, u	5, u	∞	∞

כעת מבין כל המרחקים שמצאמנו, נבחר את הצעמת שהמրחק אליו הוא המינימלי מבין כלום (במקרה זה – w) ונוסיף אותו ל-'N', ונמלא את המרחק מ-u לשאר הקדקודים בגרף, כאשר לפנינו כל עדכון בטבלה אנו בודקים אם כדאי להשאיר את המסלול מהשלב הקודם, או שיש מסלול קצר יותר דרך הקדקוד שבחרנו. נבחן כי לא כבר מצאנו את המרחק המינימלי ולכן לא נציגר למלאשוב את המשבצת שלו:



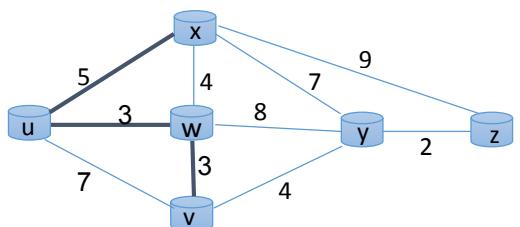
step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞

שלב 2: מבין כל המרחקים שערכנו בשלב 1 – המרחק המינימלי הוא לא-w, לכל נוסיף אותו ל-'N' ונועדכן בשלב 2 את המרחקים מ-u לשאר הקדקודים שטרם שייכים לגרף, תוך התחשבות בקשות החדשות שהתווסףו עם בחירת קדקוד x:



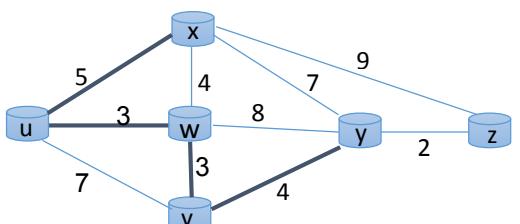
step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	U	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwx	6, w			11, w	14, x

בשלב 3, בחרנו את v להתווסף לעצט ועדכנו את הטבלה בהתאם:



step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwx	6, w			11, w	14, x
3	uwxv				10, v	14, x

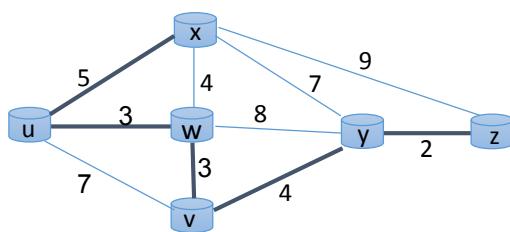
שלב 4 – בחירת y:



step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwx	6, w			11, w	14, x
3	uwxv				10, v	14, x
4	uwxvy					12, y



שלב 5 – בחירת z (האחרון שנשאר):



step	N'	P(v)	P(w)	P(x)	P(y)	P(z)
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwx	6, w			11, w	14, x
3	uwxv				10, v	14, x
4	uwxvy					12, y
5	uwxvyz					

כעת לאחר שגילינו את המרחקים לכל הקדקודים בגרף – נוכל לשמר בטבלה של u את המרחק הסופי לכל נtab וונtab:

D(v)	D(w)	D(x)	D(y)	D(z)
6, w	3, u	5, u	10, v	12, y

אבל מה ש-u באמת צריך לדעת, זה באיזו יציאה (לאיזה ציון) עליו להוציא חבילה עבור כל יעד. נבחין כי חוץ מלנתב x, כל המסלולים בעץ הפורש עוברים דרך w, ולכן שyi שימר בטבלה של u בסופה של דבר זו הטענה:

Destination	v	w	x	y	z
Link	(u, w)	(u, w)	(u, x)	(u, w)	(u, w)

את השלבים הנ"ל של בניית הטבלה נעשה מכל קדקוד בגרף, שכן מקדקודים אחרים יתכו עזים פורשים שונים.

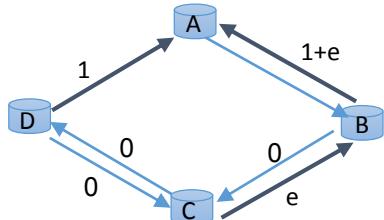
לצורך בניית העצים, כל נתב יצרך לשלוח הודעות לכל שאר הנתבים על המרחקים מהם לשכנים, עם"נ שכולם יכירו את כל המשקלים בראשת. מידע על כל קשת עובר לכל הנתבים בראשת, מכאן שסך כל ההודעות המועברות: $(E * N) * O$.

חסרונות שימוש באלגוריתם:

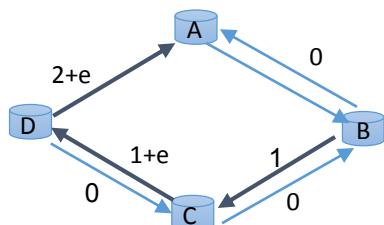
- עומס רב של הודעות (לפני בניית העצים)
- חישובים רבים בכל נתב צריך לעשoten: במקרה הגרוע, בכל שלב בחישוב נוצרך לעبور על כל הקשתות (המסלולים לכל אחד ואחד מהנתבים), ולכן סך החישובים יהיה: $O(N^2) = O(N * (N - 1))$.
- אם נמזרע את החישובים, נוכל להגיע ל- $O(N * \log N)$.



אם משקל הLINKים הם דינמיים בהתאם לעומס בכל LINK, נקבל מצב של FING-FONG.
לדוגמא במקרה הבא:



B ו-D רצים לשולח חבילה בגודל 1 לנット A, וכן C רצאה לשולח ל-A חבילה בגודל e ובוחר להעביר אותה דרך B.
אז המשקל בין B ל-A (גודלו סך החבילות שעוברות) הוא $e+1$.



B רואה שמשקל המסלול ממנו ל-A הוא $e+1$ בעוד שהמשקל של המסלול דרך C הגיע המורבה יותר, ולכן רצאה לשולח את המידע שלו דרך C!

ושוב, המשקל של LINK (A,D) גבוה מאוד ביחס לשאר המסלולים, וכל הנטים יבחרו לחזור למסלול דרך B....
וכך העניין חוזר חלילה, ונוצר FING-FONG בבחירה המסלולים.

נבחן כי זה יוצר גם קפיצות (שינויים תכופים) בעומס של כל LINK, והרי על כל שינוי ציריך לעדכן את כל הנטים שעוברים דרך אותו LINK וכן מציריך הודעות רבות שייעברו ברשת!

פתרונות:

1. משקל כל LINK לא יהיה תלוי בעומס בלבד, אלא יכלול בתוכו פרמטרים נוספים.
2. פיזול חבילות, שחלק מהBitte יעבור במסלול אחד וחלק במסלול אחר (-- יוצר בעיה של הגעת חבילות לא עפ"י הסדר).

Algorithm Distance-Vector

אלגוריתם בו כל קಡוק מכיר רק את שכניו, ומסתמך על הרעיון שכל מסלול אופטימלי בנוי מתתי-מסלולים אופטימליים.

עבור משקל מסלול מ-x ל-y, נסמן:

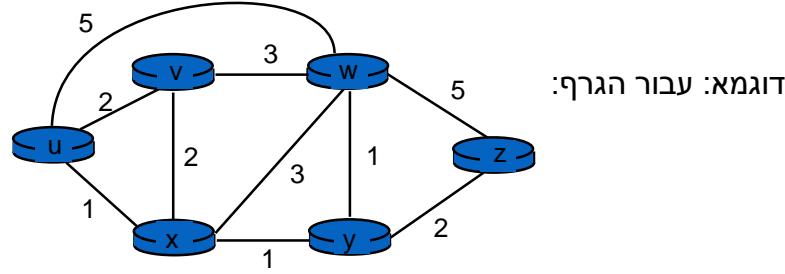
(y)_x = המשקל הנמוך ביותר מ-x ל-y.

(x)_y = המשקל שבין x לשכן y

אץ מתקיים:

$$d_x(y) = \min \{ c(x,v) + d_v(y) \}$$





חישוב המסלול המינימלי מ-u ל-z יבוצע באופן הבא:

$$d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \} = \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4$$

כדי לחשב לכל נתב את המסלולים הכי קצרים ממנו לשאר הנתבים, משתמשים באלגוריתם של בלמן-פורד.

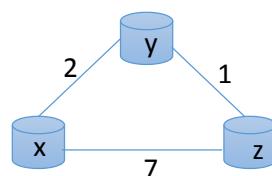
Bellman-Ford's Algorithm

כל צומת מחזק אצלו מטריצה ובה מידע על המסלולים בין כל הצמתים:

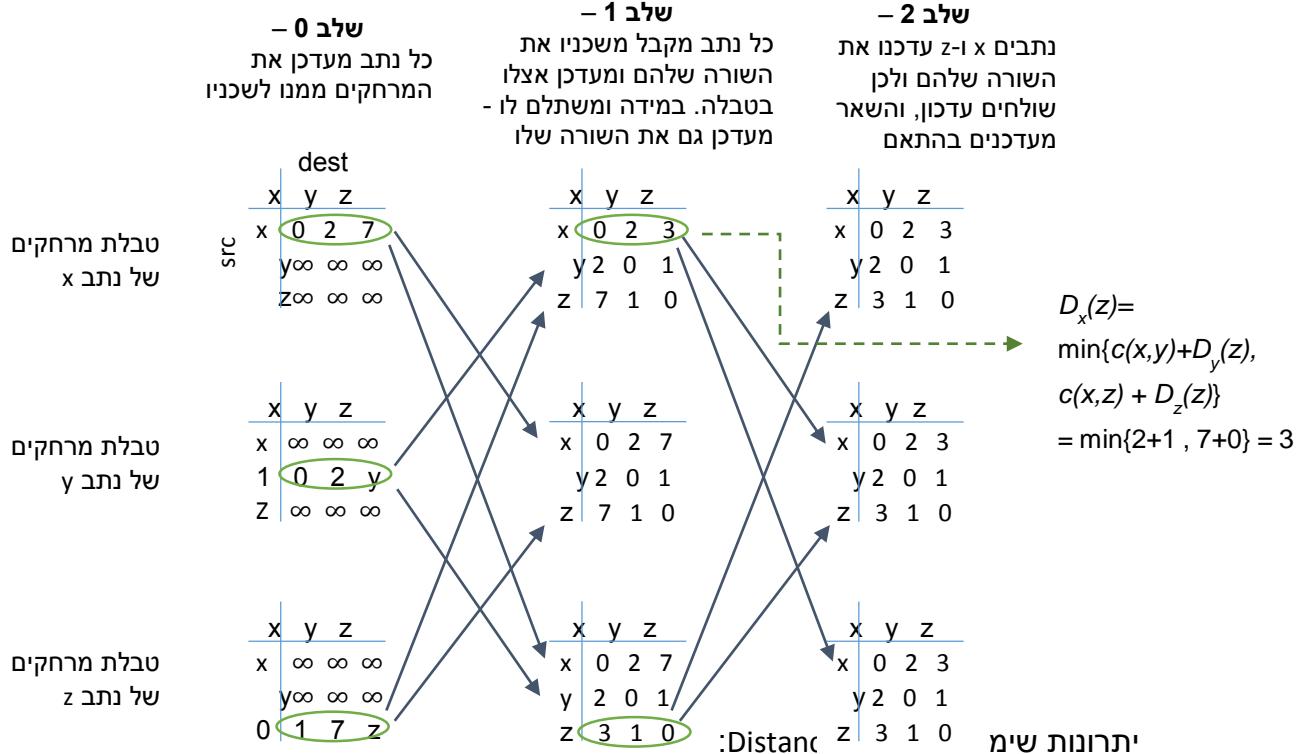
בתחילת - כל צומת מלא בטבלה את המרחק ממנו לשכנים שלו.

בכל שלב - מעדק בטבלה את הנתונים שמקבל מן השכנים, עובר על כל שכן ובודק דרכי הגעה לצמתים אחרים דרך אותו שכן, ובוחר את המינימלי.

* גם באלגוריתם זה כל נתב שומר אצלו בטבלה מי ה-hop next לאו אותו מסלול, למרות שבדוגמה הבאה זה לא מופיע.



דוגמא: נרצה לחשב את המסלולים מ-x עברו הגרף:

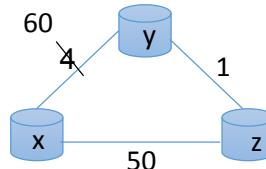


- כל נתב מכיר רק את שכניו ולא צריך להכיר את הרשת כולה, כמו כן אינם מתעניינים כיצד השכנים מגיעים ליעד המבוקש. מכך נובע שאין צורך במבנה עץ פורש.
- כמות המידע שעובר ברשת בתחילת, וכן כמות החישובים קטנים בהרבה- (³tz) לכלות ייחד.
- (כל נתב מחזיק אצלו טבלה בגודל ²tz אך כמות ההודעות קטנה יחסית והחישובים פשוטים יותר).
- סך כמות ההודעות: מוצע של מס' השכנים * tz.

חומר:

- פרוץ יותר. לא בדיק ברור מודיע, אבל ניתן למשוך הودעות בקלות יותר מאשר ב-link state, ע"י עדכון המרחוקים ל-0.
- **"בעיית הספירה לאינסופף"** – כאשר לינק מסוים מעדכן את משקלו לגובה יותר מאשר מהיה, יתכן והנתב שבקרה הלינק יראה כי לשכנו קיימת דרך כלשהי בעלת משקל נמוך יותר, למחרות שאינו יודע שמשקל זה מtabסס על הקשת שכרגע התעדכנה. הנתב יבחר לעبور דרך אותו שכן ויסתרך על משקלו, והשכן גם יעדכן בהתאם, וכך יעדכו בכל רגע על סמך הודעות אחד של השני ויוצר פינג-פונג...

למשל בדוגמה הבאה:



במצב ההתחלתי, עלות לינק (ע, x) היא 4. אז נתב z יודע שהמסלול הקצר מ-ע-ל-x הוא 4, ולכן המסלול ממנו ל-x הוא 5 דרך ע ומסמן: (ע,5). (z אינו יודע כיצד ע מגיע ל-x, אלא יודע רק שקיימות דרך כלשהי במשקל זה).

נניח ולינק (ע, x) משנה ערכו ל-60. אז ע ישלח עדכון מתחאים, וכן יראה כי בפעם الأخيرة עודכן מ-z שיש לו מסלול ל-x באורך 5, ולכן יבחר הפעם להשתמש במסלול דרך z, יעדכן אצלו: (z, 6) וישלח עדכון. (שוב, ע יודע רק כי קיימים מסלולים מ-z ל-x במשקל 5, מה שאינו יודע – זה שהמסלול הזה תוכנן לעبور דרכו).

נתב z מקבל את העדכון, והוא יופיע בטבלה שלו שהדרך שבחר – עוברת דרך ע. ואם ע הרגע עודכן שהדרך דרכו ל-x היא במשקל 6 – אז z יבין שמננו ל-x זה 7! ומסמן: (z,7). ושוב – z מקבל את העדכון וראה שהוא עצמה בחר את המסלול דרך ע – ולכן מעדכן אצלו (ע,8) וכן הלאה... וכך נוצר פינג-פונג בין השניים.

מתי זה יעצר? כאשר z יקבל עדכון מ-ע שהמסלול מ-ע-ל-x הוא במשקל 49, ו-z יוכל לבחור את המסלול ישיר במקום לעبور דרך ע באותה המשקל!

ומה הבעה בפינג-פונג? שנשלחות הודעות רבות ומיותרות, וככל שהפער גדול יותר – כן מס' ההודעות עלי כר.



הפתרון לכך: כשנתב z בוחר את המסלול ל-x דרך נתב y – הוא יפרנס לכולם את עלות מסלולו ל-x, מלבד לנtab y- שעבורו הוא יפרנס ∞ .vrענ ש Ichוב של-z אין מסלול ל-x ולכן לאחר שינוי הLINK הוא לא יבחר לעבר דרך z.

פתרון זה פותר את בעיית הספירה לאינסוף, אבל אינו פוטר מעגליים, וכן מה קורה אם הودעה על עדכון מסלולים הולכת לאיבוד.

2 אפשרויות לשЛОח הودעות עדכוניים ל פרוטוקוליים:

1. ע"י טימרים – עדכון בכל מחזור זמן מסוים.
2. עדכון רק במקרה של שינוי משמעותי.

← העדכן קורה עפ"י המוקדם מבין השניים!

Hierarchical Routing

האינטרנט הוא רשות של רשותות, כאשר כל רשות נקראת **AS – Autonomous System** ומקבלת מס' ID ייחודי בעל 32 סיביות (AS number) מהIANA או ICANN (איגודים להסדרת הכתובות ברשות).

החלוקת מאפשרת לכל רשות להיות אוטונומית ולקבוע בעצמה כיצד היא פועלת בתוכה, ובין היתר גם מחליטה איזה אלגוריתם ניתוב יופעל על הנתבים שלה.

בעצם הניתוב באינטרנט מחלק ל-2:

- eBGP – ניתוב בין רשותות ASs, מתבצע באמצעות פרוטוקול **Inter domain routing**
- Intra domain routing – ניתוב בתוך רשות AS. נק' היציאה מהרשת מוכנות ע"י **OSPF/RIP**.

לכל AS יש נתבי יציאה מהרשת הנקראים **gateways**, שמחברים לנתבים בסיסיים אחרים, עם"ג שהרשת כוללה תהיה מוקורת.

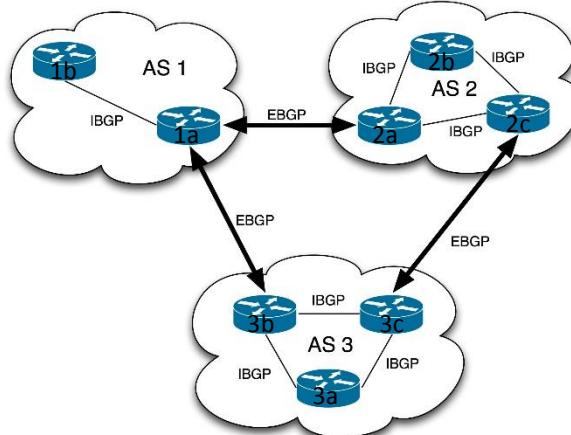
באמצעות פרוטוקול **eBGP** – נתבי gateway יודיעים לאיזו רשות שכנה עליהם להעביר הודעות המיועדות לרשות אחרת. הרשת BGP בעצם מגדר את הקשרים שבין ASs השונים ומהם מהסלים מ-AS ל-AS אחר.

כדי שכל נתב באס יידע מי הם gateways ואיזה ASs ניתן להגיע דרכם – מגדר פרוטוקול **BGP**. את המידע הזה הנתבים שומרים בטבלת הניתוב, בנוסף למידע שנשמר שם על שאר הנתבים שבתוך AS.

(שאר פרטי פרוטוקול BGP יוסברו בהמשך).



לדוגמא: ברשת הבאה, ראותר 3a רוצה לשלוח הודעה לראותר 1b:



از 3a יצטרך לדעת מי הוא gateway שבתת-רשת שלו (AS3) שמחובר לרשותתו נמצא 3b (או – אם לא היה מחובר לשירות AS1 – היה מחובר לרשות אחרת שבסוף איכשהו הייתה מתחברת לרשות AS1). ע"ז IBGP הוא יודע שעליו לשלוח את ההודעה ל-3b, ואוותר 3b מעביר את ההודעה מהרשת שבו הוא נמצא – לראותר 1a שבתת-רשת AS1, וזהו הוא כמובן יודע עפ"י BGP. כיוון שא-1b נמצאים באותו AS – לא-1a אין בעיה להעביר את ההודעה לייעדה.

נניח כעת כי 3a רוצה לשלוח הודעה לראותר שנמצא בתת-רשת AS4, שמחוברת לרשותות AS1, AS2. אז ע"ז פרוטוקול IBGP, הראותר 3a מתעדכן להעביר את ההודעה גם דרך gateway 3b ו.gateway 3c וגם דרך gateway 3b, ועליו להחליט מי מבניםם עדיף!

בבחירה זו נעשה בשיטת **Hot-Potato**: הרשות AS3 רוצה להוציא ממנה את ההודעה כמו שיוטר מהר (כמו hot potato שגם רוצים להחזיק אצלם כמה שפחות זמן כדי שלא נקבל כו"ה...), ולכן 3a יבחר את gateway הnearest ביחס אליו (עפ"י משקלים), ואם הם שווים – יבחר באופן אקראי.

Intra AS Routing

פרוטוקולי ניתוב בתחום AS. הידועים שבהם: RIP, OSPF, IGRP

RIP = Routing Information Protocol

ממשש את distance vector, ומקטין גם את בעיית הספירה לאינסוף. מגדר את המרחק המקסימלי בין נתבים ל-15, כך שסיום הספירה יגיע במהירות. הגבלה ל-15 מתאימה את הפרוטוקול לרשותות קטנות.

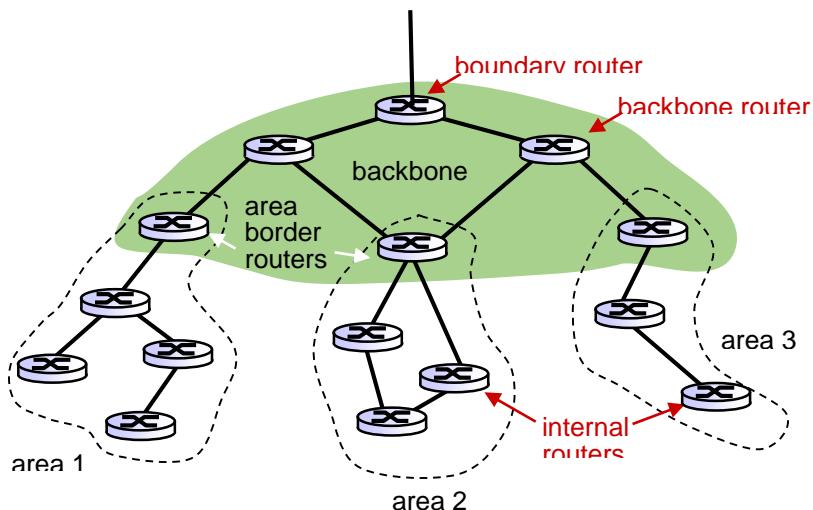
במקרה זה, ינתן מראש משקל קבוע של 1 לכלリンク, וכך שבחירת המסלול היא בעצם בחירת הדרכ שעובדת דרך כמות נמוכה ביותר של נתבים.



OSPF = Open Shortest Path First

ممמש את link state, ובניגוד RIP- תומך במשקלים שונים. כמו כן האבטחה שלו משופרת ביחס RIP. **תומך ב-BGP Path = Equal Cost Multi Path = ECMP** – מאפשר ניתוב במסלולים שונים בעלי מחיר זהה.

הweeney: חלוקת הרשות לאזוריים, והגדלת backbone – אזור שמחבר בין כלום. כך נוצר למדוד את המרחקים הקצרים ביותר שיש בתוך כל אזור, ויציאה מהאזור תיעשה דרך נק' אחת בלבד. כל שינוי של קשת יופץ בתוך האזור בלבד.



ע"י החלוקת לאזוריים, ניתן להגדיר לכל אזור את אלגוריתם הניתוב שלו, וכך מכל אזור יש רק יציאה אחת וכן נתבים בתוך אזור שאיןם area border routers לא צריכים להזכיר את כל אלו שמחוץ לאזור, אלא רק את נק' היציאה מהאזור.

ה-boundary routers מחוברים לאס אס אחרים, אלו הם בעצם ה-gateways.

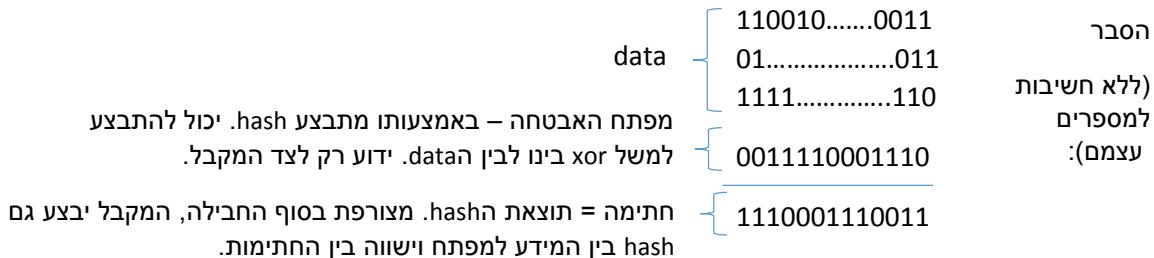
ה-OSPF שימוש יותר ברשתות גדולות – ספק אינטרנט. במקרה זה צומת הוא צומת גדול ומשמעותי, ונbnb יכול לסמן רשת שלמה (רשת של עסק/מודול).

למרות שמדובר בצמתים גדולים – אין בעיה של הגעת חבילות לא עפ"י הסדר, כי בצוות יש תעבורה עצומה ומדובר בנitinob של שיחות ולא חבילות בודדות. צמתים אלו יוצרים איזון בתעבורת ברשת.

ה-OSPF מתגבר גם על בעיית האבטחה בה תוקף מנסה להחדר חבילות לתוך הרשות, וזאת ע"י מפתח אבטחה שמוגדר לכל קשר:

ה-OSPF מגדר מפתח שידוע רק לצד המקלט, ומבצע הצד השולח hash על החבילה באמצעות אותו מפתח. תוצאת hash מצורפת כחתימה לכל הודעה, והצד מקבל יוכל לבדוק עפ"י המפתח שבידיו עם ישנה התאמה בין המידע לבין החתימה עפ"י אותו מפתח. בנוסף, כדי למנוע פרצה בה האקר משכפל את ההודעות – מתווסף לכל הודעה גם מס' סידורי, כך שאם שכפלה חבילה נקלט פעמיים אותו מס' ונדע לזרוק את החבילה.





Inter AS Routing

.**BGP = Border Gateway Protocol** מוגדר ע"י הפרטוקול

.iBGP = Internal BGP eBGP = External BGP

נתבי החיבור שבין ה-AS מעבירים ביניהם הודעות eBGP שמודיעות לאיזה רשתות הם מחוברים כדי שהודעות יעברו מ-AS אחד לעד, וכן כדי שהנתבים שבתוך ה-AS שלהם ידעו להעביר אליהם הודעות לרשתות חיצונית, הם מעבירים לנတבים ברשות הודעות PGP.iBGP.

אוסיף כל שמות ה-AS במסלול מנתב כלשהו לנットב אחר נקרא **AS-PATH**, וכהתובת הנットב הבא במסלול מוגדר להיות הק-hop-Next.

לנתבים יכולים להיות מס' דרכי להגיע לנットב אחר, ולכן עליהם לבחור מבין המסלולים. סדר העדיפות הינו:

1. **Local preference** – עדיפות שנובעת בעיקר משיקולים כלכליים (מסחריים). AS יעדיף להוציא הודעה ממנו החוצה דרך AS שכן במקרה שלא יהיה צורך לשלים לו על החבילה, ז"א יעדיף להוציא את החבילה דרך לקוח או עמית, ולא דרך הספק שלו.
2. **AS-Path length** – מס' ה-ASים שההודעה תעבור בדרך, ולא מס' הנתבים.
3. **Closest next hop** – אם 2 התנאים שמעלה לא קבעו מסלול באופן חד-משמעי, ההודעה תצא דרך gateway הקרוב ביותר לנットב ממנו ההודעה יוצאת. ז"א – הוצאת החבילה מה-AS בצורה המהירה ביותר (hot potato) – המסלול הכי קצר בתוך הרשת הפנימית.

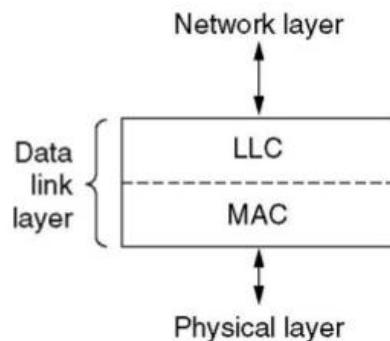
נבחן כי גם בין ה-ASים יכולות להיות אפשרויות ניתוב, ונעשה מעין distance-vector בין ה-ASים כדי למנוע מעגלים בין ASים, הנקרא Path Vector: הודעות על מסלולים לא כוללות את המשקל בלבד, אלא גם את המסלול כולו (כל ה-ASים דרכם עובר המסלול). כאשר AS יקבל מידע על מסלול מנתב אחר – הוא יבדוק שהוא עצמו לא נמצא באותו מסלול, ורק אם לא – ישקו להשתמש במסלול זה.



הרצאה 12

Data Link Layer

שכבה ה-2 – מחלקת ל-2 תתי-שכבות:

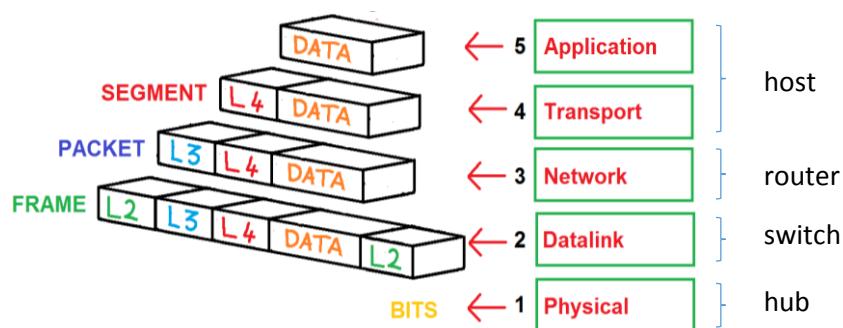


ה- **LLC** = **Logic Layer Control** אחראי על בדיקת שגיאות במידע שהועבר, **acks**, **requests**, **header** ועוד,

וה- **MAC** = **Media Access Control** אחראי על הוצאת המידע לינק = הגישה לווקו עצמו.

שכבה זו מוסיפה לחבריה את התקורה שלה שבנוספף לheader מօסיפה גם trailer, והחבריה הופכת להיקרא **Frame**. רכיבים שתומכים עד שכבה זו נקראים **switches**. השכבה מזזה (ומתקנת במידת האפשר) שגיאות שקרו במהלך העברת המסגרת ומעבירה את המסגרת דרך ערוץ התקשרות לנק' הבאה בדרך. השגיאות יכולות להיגרם ע"י היכלשות אותן הנגרם מאוחר הcabל או רעש במהלך השידור. ככל שזמן השידור ארוך יותר – כך יש סבירות גבוהה יותר לשגיאה.

נזכיר את מודל השכבות, השמות הנחוצים לחבריה שעוברת והרכיבים שמתפללים לחבריה:

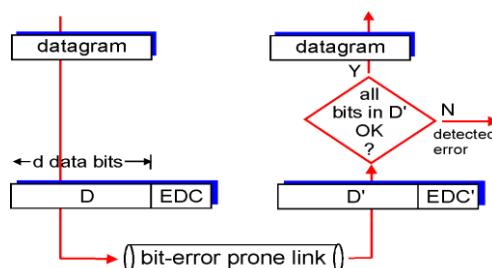


בכל רכיב בדרך שמעביר חיבור (כולל hosts) מותקן NIC = Network Interface Card (שאוחראי על שכבת הקו והשכבה הפיזית. הNIC ממומש בחומרה כאשר ה-sources (היציאות) לרוב ממומשות בתוכנה.



Error detection

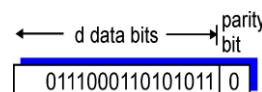
מנגנון בקרת השגיאות נקרא EDC = Error Detection and Correction bits. מדובר בבייט אחד של סימולריים (bits) שנוסף לאחר data header. כאשר החיבור מגיעה לקצה השני – בודד או מס' בייטים שמתוווספים לסופם של data header. כאשר החיבור מגיעה לקצה השני – הוא יבדוק עפ"י EDC האם כל הביטים שהתקבלו ב data header תקין. המנגנון אינו אמין بنسبة 100%, אך ככל ששדה header ארוך יותר – כך יכול לאזהות ולתקן שגיאות באופן טוב יותר.



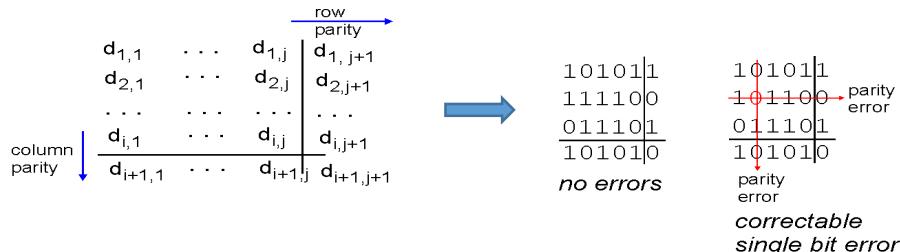
רעיון ה-EDC נעשה לרוב ע"י הוספת סיבית בכל בלוק מידע מוגדר, שתסתמן אם מס' ה-'1' בבלוק הוא זוגי או אי-זוגי:

- סיבית זוגיות זוגית – תוגדר להיות 1 אם מס' ה-'1' בבלוק הוא אי-זוגי
- סיבית זוגיות אי-זוגית – תוגדר להיות 1 אם מס' ה-'1' בבלוק הוא זוגי.

השיטה הבסיסית לזיהוי שגיאות היא Single bit parity – הוספת בית אחד לכל בלוק של מידע. מקרה זה בעיתוי ביעילות שלו כאשר ניתן יותר מביט אחד בדרך. כמו כן, הוא מאפשר זיהוי של שגיאה אף לא יכול לתקן כיון שלא ידוע איזה מהbeitים ניזוקו.



שיטה מתקדמת יותר זו היא – סידור הdata בצורת מטריצה וסימון parity לכל שורה ולכל עמודה. אם נזוק ביט בודד – ניתן לדעת את מיקומו המדויק ע"י הצלבות המידע שבשורה והעמודה ייחד, וכך ניתן לתקןו. כמו כן אפשר לזהות שגיאה נוספת מביט בודד.



שיטה זו גוררת תקווה רבה יותר לdata, שכן יש צורך בשילוח שורה ועמודה של ביטים לבדיקה, ומס' הסיביות תלוי בגודל המידע שנשלח ואינו קבוע.

שיטה עוד יותר טובה זו שיטת ה- **CRC = Cyclic Redundancy Check**, ש כוללת יותר חישובים שנעשים על המידע המתתקבל. CRC משתמש בחילוק ארוך. לפני העברת המידע מחושב CRC ומתווסף למידע המועבר, כאשר על הצד מקבל לבצע גם כן את החישוב ולהשוות בין החישוב שקיבל עם המידע. נגדיר:

D = data – רצף בינהרי.

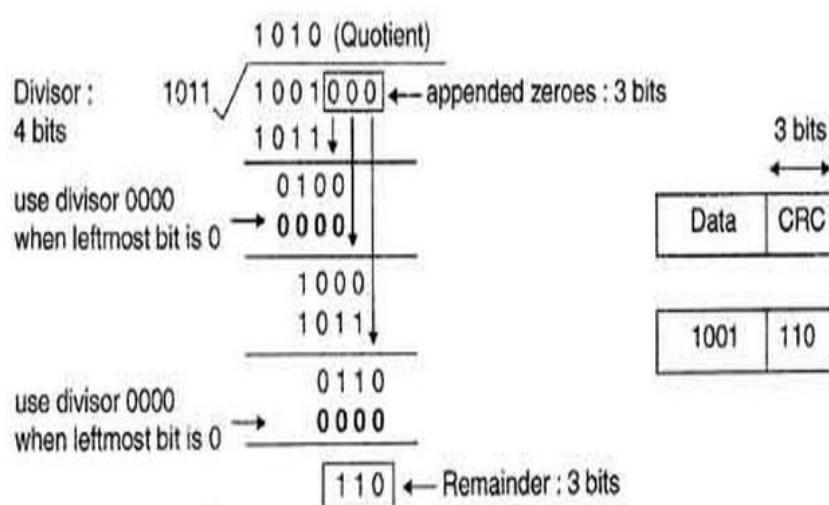
G = פולינום יוצר – מוגדר מראש ע"י הפרוטוקול בו משתמשים

r = אורך הפולינום היוצר פחות 1.

לבлок המידע הנבדק נוסיף r אפסים בסופו. נחלק את התוצאה בפולינום היוצר, ונחסר את השארית ע"י שימוש בORX.

הנוסחה המתקבלת היא: $CRC = remainder \frac{D * 2^r}{G}$

דוגמא: עבור $G = 1011$, $D = 1001$



כפי שהוזכר, ה-EDC יכול להתווסף כערכ בקירה יחיד ולכך מיקומו יהיה בסוף ה-data בלבד, אך יכול גם להיות מוגדר לכל בלוק של מידע, וכך יכנס בתוך המידע עצמו מס' פעמים. במקרה זה הצד המקביל יידע שבקריאת המידע יהיה עליו לדלג כל מס' ביטים על הביטים שמשמעותם את ה-EDC. כמו כן, עבור כל שכבה – המידע הוא כל החבילה שהגיעה מהשכבה מעל. לשכבות הקו למשל, המידע ה-packet כולל שהתקבל משכבה ה-IP – כולל headers של שכבות 3, 4 ו-5.

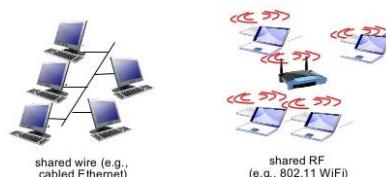
Multiple Access Control

LINKS בין switches או כל רכיב אחר, יכולים להיות באחת מ-2 השיטות:

1. **Point-to-point**- כל LINK מקשר בין 2 רכיבים בלבד, והם היחידים שיש להם גישה לLINK. פוטר בעיה של 'מרחבי התנגשות'.



2. **Broadcast**- כאשר השידור משותף ליותר מאשר 2 רכיבים. כולל אפשרות של כבל פיזי משותף או למשל מס' רכיבים בראשת אלחוטית.



כאשר הLINK משותף ליותר מ-2 רכיבים, יכול לקרות מצב בו יותר מרכיב אחד מתחילה לשדר על הLINK ויש התנגשות בין המסגרות המועברות – **collision**.

נראה 3 דרכים להתגבר על Multiple access:

1. חלוקת הערז לחלקים
2. אפשרות שידור לכולם, והגדלה כיצד יש להתגבר על התנגשות
3. קביעת תורות מי יצא לשידור

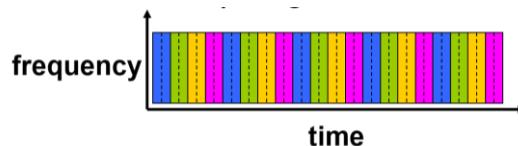


Channel partitioning

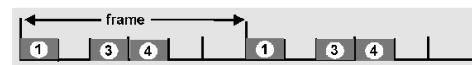
נראה 2 גישות לחלוקת הערוץ- לפי זמן ולפי תדר:

TDM = Time Division Multiple Access

חלוקת זמן מוקצב לשידור על הוקן לכל אחת מהתחנות. מתבצע סיבוב (Round Robin) וכל תחנה בתורה מקבלת את רוחב הפס המלא ומנצלת את זמן השידור שלה כרצונה.

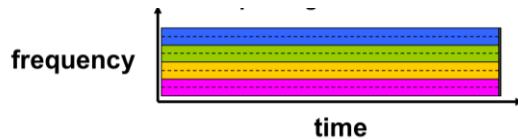


בגישה זו אין ניצול מלא של הזמן, שכן יתכן ולתמונה מסוימת אין חבילות לשדר, ולכן הזמן שМОקצב עברוה מיותר.



FDM = Frequency Division Multiple Access

חלוקת קבועה של התדרים בפס לתחנות השונות, וכל תחנה משתמשת בתדר שלה כרצונה. במקרה זה אין ניצול מלא של רוחב הפס, אם לתמונה אחת יש הרבה חבילות לשדר ולתמונה אחרת אין כלל – אז ישנו תדר מבודד שהתחנה הראשונה יכולה להשתמש בו ולשדר את החבילות שלה בזמן קצר יותר.



Random Access

גישה המאפשרת התנגשות, אך מגדירה איך מגלים זאת ומה לעשות במקרה והתגלתה אחת. נראה 2 פרוטוקולים לכך.

ALOHA

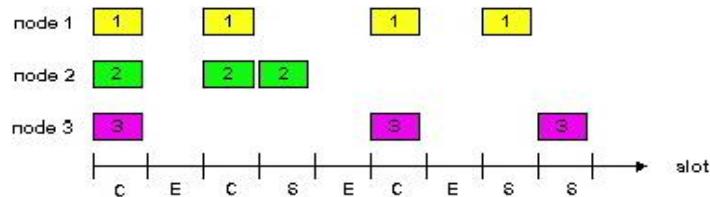
בפרוטוקול זה כל המסגרות מוגדרות להיות בגודל קבוע, והזמן מחולק לפרקי זמן קבועים ליציאה לשידור, המוגדרים 'חריצים' – 'slots'. כל חרץ הוא זמן שלוחה מסורת אחת.

תחנה יכולה לצאת לשידור בחרץ זמן. התחנות מאזינות לערוץ ולכן יכולות לזהות התנגשות. אם המסגרת נפגעה, התחנה המשדרת ממתינה פרק זמן אקראי להמתנה לחרץ הזמן הבא לשידור, ואז שולחת את המסגרת שוב. מדוע אקראי? כיון שאחרת 2 התחנות שSSIDן התנגש – יצאו לשידור שוב באותו חרץ זמן, ויגרמו להתנגשות נוספת!



בדוגמה הבאה, חרץ זמן מסומן באותיות :

S = successful E = empty, C = collision,



בדוגמה, 3 תחנות יצאו לשידור בחרץ זהה. כל אחת מהן מגירה זמן כדי לדעת מתי עליה לשדרשוב את המסגרת שהתגשה. לתחנות 1,2 יצא להמתין 2 חריצים, ואילו לתחנה 3 יצא להמתין 5. תחנות 1,2 שוב יצאו לשידור והתגשו, ולכן מגירות זמן חדש. הזמן שיצא לתחנה 1 בהगלה זו – זהה לזמן של תחנה 3 מההగלה הקודמת! ולכן תחנות אלו מגירות שוב זמן, עד שמצוות לשדר מסגרת ללא התגשות.

יתרונות:

1. פרוטוקול פשוט
2. התגשות מתגללה מיד עם רגע היציאה לשידור, כיוון שזמן היציאה מוגדרים באופן חד משמעי ע"י החריצים

חסרונות:

1. כישיש לתחנות שידורים רבים – ישנו סיכוי גבוה יותר להתגשות
2. בשל הגדרת גודל החרץ – הוא דרוש שככל המסגרות יהיו באוטו הגדל, או שהוא מוגדר להיות כגודלה המסגרת המקסימלית – אז כאשר מועברת מסגרת קטנה יותר חלק מהזמן מתbezבז.
3. בכל התגשות חרץ הזמן מתbezבז, כי היה מנוצל אם לא הייתה התגשות ואילו CUT יש להמתין ליציאה עד החרץ הבא
4. דרוש סינכרון שעוני בין כל התחנות

CSMA = Carrier Sense Multiple Access

פרוטוקול נוסף לגישת random access. כאן יש האזנה לפניהם יציאה לשידור. אם הקו פנוי – המסגרת נשלחת, אחרת – השליחה נדחתה.

עדין, יכולות להיות התגשויות. למשל כאשר תחנה יוצאה לשידור אך הסינגל טרם נשלח לתחנה אחרת שמאזינה, וכן גם היא משדרת. במקרה זה, זמן ההתפשטות הוא הזמן שלוקח עד שתחנות מגילות שהמסגרות שליהן התגשו, וכך ככל זמן ההתפשטות ארוך יותר – יהיה יותר התגשויות וביצוע הקו ירדן. כמו כן, התגשות יכולת לקרוות גם כאשר 2 תחנות מאזינות בו-זמנית ורואות שהקו פנוי ומיד יוצאות לשידור. במקרה וקרתה התגשות, התחנות המתגשות מודיעות לשאר התחנות כדי שימתינן זמן-מה עד שתיפתר הבעיה.



CSMA/CS (Collision Detection)

SHIPOR בו ההתנגשות מתגללה בתוך זמן קצר יותר, והתקנות מפסיקות להמשיך לשדר את המסגרת וכך הזמן המבוזבז מופחת. השיפור קל ליישום בכבלים פיזיים, וקשה יותר במקרים של wireless.

איך נמנעים כמה שניות מההתנגשיות? כאשר תחנה מאזינה לקו והוא פנוי – היא מטילה מטבע אם לשדר באותו רגע או לחכות לחריץ הבא. מכאן שגם כאשר הערז פנוי – ישנה הסתברות של חצי שהתחנה תצא לשידור. אם היא לא יצאה לשידור – היא מגדילה את טווח ההסתברות ליציאה בחזקת של 2.2", שברגע הראשון שתרצה לשדר – ההסתברות לשידור היא חצי, ואם לא שודרה מסגרת – ההסתברות לשידור בזמן הבא הוא רבע, ואח"כ שמינית וכו'....

סיכום התהילה:

```
A: sense channel, if idle
    then {
        transmit and monitor the channel;
        If detect another transmission
            then {
                abort and send jam signal;
                update # collisions;
                delay as required by exponential backoff algorithm;
                goto A
            }
        else {done with the frame; set collisions to zero}
    }
else {wait until ongoing transmission is over and goto A}
```

הבדלים העיקריים בין ALOHA לCSMA הם זמן ההאזנה לפני או אחרי יציאה לשידור, וכן אופן הטיפול בההתנגשות – ALOHA מגറיל זמן חדש ליציאה, ו-CSMPY מגറיל בכל פעם המ יצא או לא.

Taking Turn

נקרא גם שיטת *the line*. אחד מכל התחנות שלhnן 'אזר משותף' מוגדר להיות master. הוא 'מزمין' את התחנות האחרות לשדר – שואל כל פעם תחנה אחרת האם ברצונה לשדר, ואם לא – עובר לתחנה הבאה. הפרוטוקול מונע התנגשיות אך יוצר תקורה רבה וזמן מבוזבז – לכל שאלה של master ותשובה של התחנה הנשאלת. כמו כן, אם master נתקל – כל השידור נפסיק.

** פרוטוקול זה שונה מTDMA, כיון שגם תחנה לא מעוניינת לשדר – 'מדלגים' עליה וזמן השידור ניתן לתחנה הבאה שמעוניינת.



נסכם את שלוש הגישות:

– פרוטוקולי חלוקת הערז (channel partitioning)

- שימוש גבורה בעומסים גבוהים

- לא יעיל בעומסים נמוכים (עומסים של חלק מהתחנות)

– פרוטוקולי גישה אקראית (Random access)

- יעיל לעומסים נמוכים – צומת בודד יכול לנצל את הערז כולו

- לא יעיל בעומסים גבוהים – ריבוי התנגשויות

– taking turns

הכי יעילים – לעומסים גבוהים וنمוכים!

Link Layer Addressing

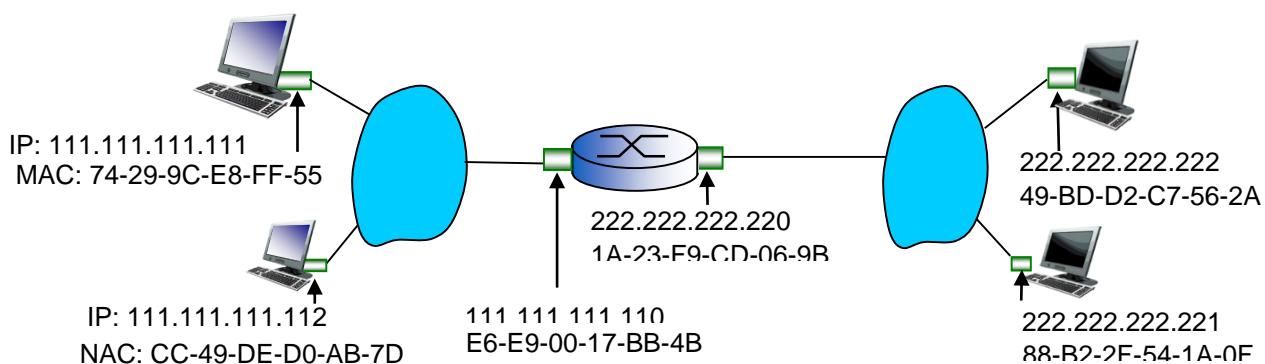
מציר כי שכבה ה-2 מתחולקת ל-2 תת-שכבות:

ה-L2A שאחרראית על גילוי השגיאות ועוד, וה-MAC – הגישה לקו.

כל רכיב תקשורת מכיל כתובות MAC ייחודית שגדירה את כתובתו הפיזית. כתובות זו אינה כתובות IP, שכן כתובות IP מגדירה את הרשות אליה שיר הנטב ויכול להיות משותפת למוש' רכיבים. ה-MAC לעומת זאת ייחודי ואין קשר בין בין הרשות. כתובת MAC ניתנת לרכיב עם היוצרתו (בנויות הפיזית) ומוגדרת בCAN, ולא עם הטרופתו לרשות. (רכיבים שתומכים עד שכבה שנייה) מכירם רק את הכתובת זו, והוא מסמנת עבורה את 'היעד הבא' של החיבור. הם אינם מתעניינים לאן החיבור מיועד בסוף המסלול, אלא רק באיזה interface עליהם להוציא אותה. יתכן ול-switch בודד יהיה מס' כתובות MAC – כתובות לכלי .interface

כמשמעות מועברת בlienק, בheader של שכבה שנייה מצוינות כתובות MAC של ה-switch השולח ושל מקבל (שני קצאות הלינק הנוכחי), וכך במקרה של שגיאה ניתן לצמצם את הבעה לlienק הספציפי והswitch האחרון יכול לשלוח שוב את המסגרת שנפגמה.

מצור כי לעיתים ישנן הודעות Broadcast למס' רכיבים. במקרה זה כתובות MAC של היעד תוגדר להיות FF-FF-FF-....-FF (לפי מס' הביטים)



ARP = Address Resolution Protocol

פרוטוקול שלל ידו switch מגלה לאיזה שיכון עליו להוציא את המסרוגת, ובאיזה介面 להשתמש.

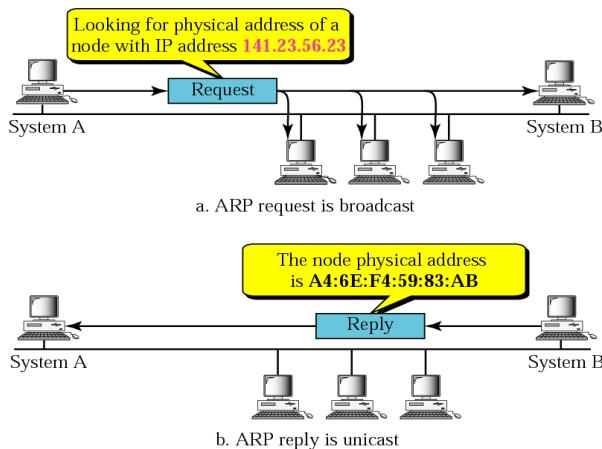
כל switch מחזיק אצלו טבלת ניתוב, כאשר כל רשומה בה מכילה את כתובת ה-IP, כתובת MAC ו-TTL. במקרה זה אינו מוגדר על המסרוגת אלא סופר את הזמן לשימרת הרשומה בטבלה, וכך אשר עבר הזמן – הרשומה נמחקת.

הרשומות בטבלה שייכות לתחנות שכנות שהעבירות דרך אותו switch מסגורות, ועל יד שמירת המידע עליו ניתן לזהותם ולהעביר אליהם מסגורות בהמשך.

סביר: נאמר שבשלב כלשהו הטבלה של A ריקה, והוא מקבל מ-B מסגרת להעברה ל-C. A ישמור אצלו בטבלה את הפרטים של B שכרגע התגלו לו, יגלה את C ע"י בקשת ARP (שתcanf נרחב עליה) וישלח את המסרוגת. כאשר A קיבל מסגרת עבור B – הוא לא יצטרך לגלות שוב מי זה ואיך ניתן להגיע אליו, אלא שולח את המסרוגת ל-B עפ"י הנתונים ששמר אצלו בטבלה.

כיצד מתבצעת בקשת ARP?

רכיב שרצה לגלוות את כתובת MAC של רכיב אחר, שולח ב广播 Broadcast שאלת ARP שמכילה את כתובת IP של הרכיב אותו הוא מchipש. את כתובת ה-IP יכול לזרום כל כיוון שהוא תומך גם בשכבה שלישית. כדי לשלוח הודעת Broadcast, כתובת MAC של יעד המסרוגת תוגדר להיות FF-FF-FF-FF-FF-FF (לפי מס' הביטים), וכל הרכיבים השכנים יקבלו את המסרוגת. הרכיב שכתובת IP המצוינת שייכת אליו – מחזיר הודעה לשלוח בה הוא מצין מה כתובת MAC שלו. הרכיב שchipש את MAC שומר אצלו בטבלה את כתובת MAC המתאימה לאותה כתובת IP עם TTL.



LAN = Local Area Network

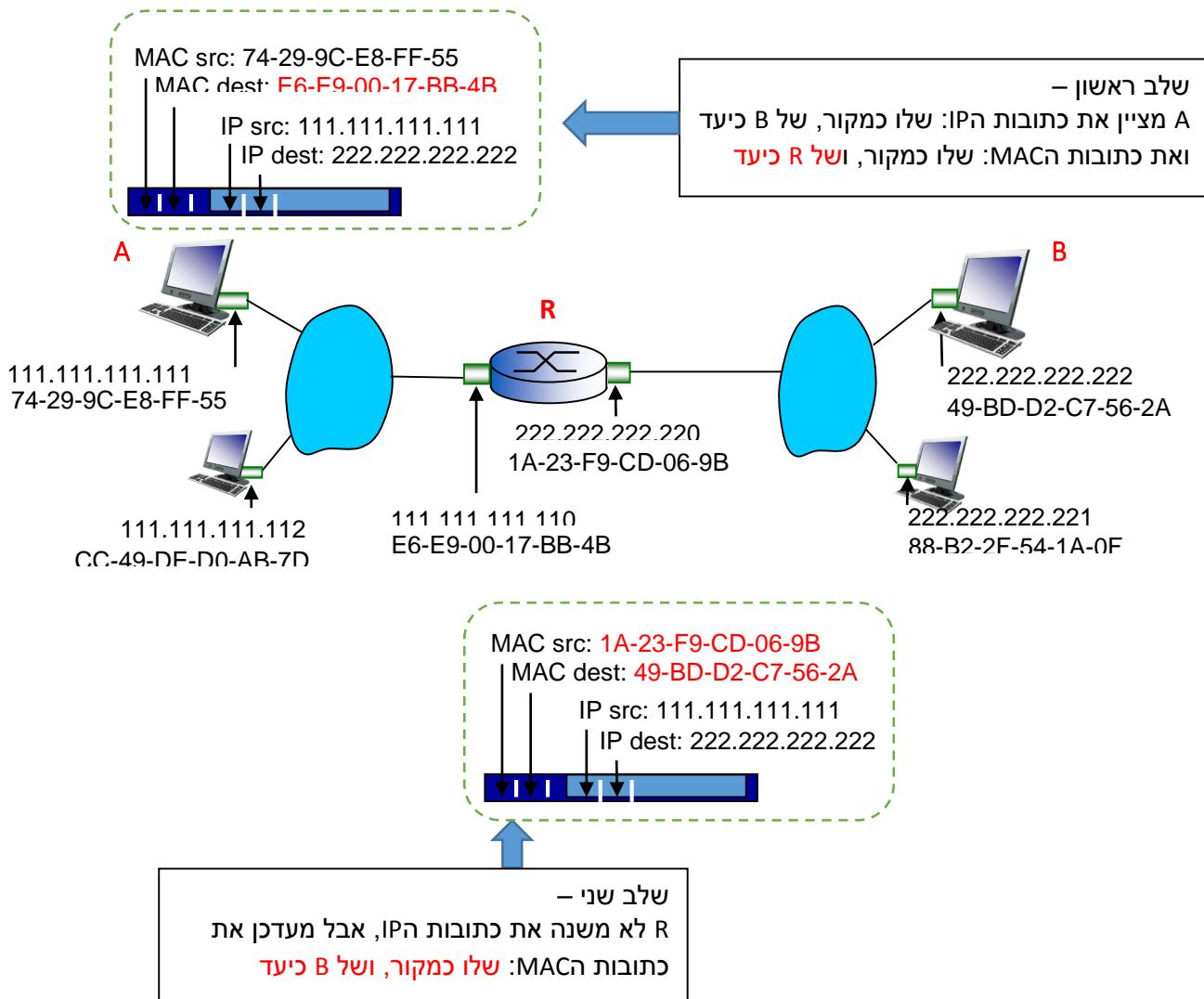
רשת של נק' קטנה על שטח גיאוגרפי קטן יחסית, כגון שכונה/marshדים. בד"כ תחנות באוטו LAN מכירות מראש את כתובות MAC של השכנים, אך כיצד ניתן להוציא חבילת שמיועדת לכתובת IP מוחוץ ל-LAN של השולח? כמובן שאין יכול להוציא בקשה ARP לכל הרכיבים ברשת האינטרנט!



נניח כי מחשב A רוצה לשלוח חבילה למחשב B ברשות אחרת. ע"י פרוטוקול שכבת 3, A יוכל למצוא את הראטור האחראי על העברת חבילות מתוך הרשות אל הרשות המזענדת. لكن, A ירצה לשЛОח את החבילה אל אותו ראטור. החבילה המיועדת תיכל של שכבת 3 את כתובות ה-IP של A המקורי ושל B הנוכחי, אבל – כדי להגיע לראטור המקשר צריך ציריך כתובות MAC! לכן, A מוסיף לframe את כתובת MAC שלו המקורי ושל הראטור המקשר הנוכחי. כשהוא יגיע לאותו ראטור, הוא יקלף את ה-frame וויסוף frame משלו שיכיל את כתובות MAC שלו המקורי ושל B הנוכחי!

החלפת כתובות MAC במאפשרת העובדה שתובות אלו נועדו כדי להעביר את הframe לתחנה הקרובה בלבד, ואילו כתובות ה-IP מגדרה את כיוונו של המשך המסלול.

נראה זאת בדוגמה הבאה:



"יתכן כי גם הראטור שבאמצע לא יכיר את B או נתב אחר בדרך ויצטרך גם לבצע ARP, וכן הנתב הבא, וכך כל אחד בדרך ..."



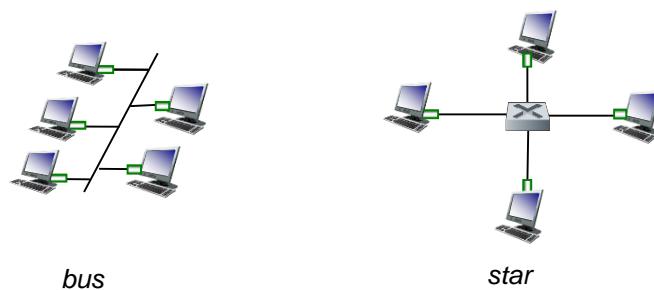
Ethernet

טכנולוגיה לתקשורת נתונים בתוך LAN, שבעצם מגדרה את פרוטוקול שכבה ה-2.

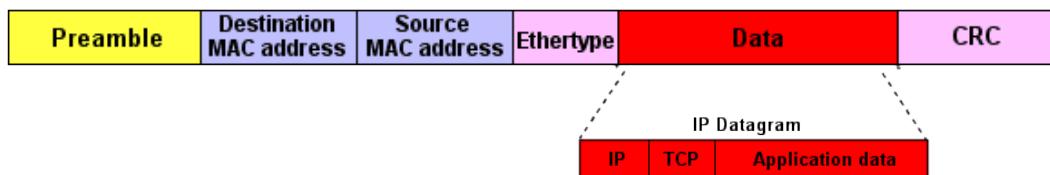
הטופולוגיה של Ethernet מתחילה ב-2:

Bus – ציר משותף למשתמשים מחשבים

Star – הגדרת switch שימנע מצב של ציר משותף ליותר מ-2 תחנות



Ethernet frame structure



- **Preamble** – 7 Bytes – מהצורה 10101010 ... 1 Byte – מהצורה 10101011.
- נועד כדי לזהות תחילת frame חדש שמועבר בlienik.
- **Src & Dest MAC Address** – 6 Bytes – כל אחד. בכל פעם שמאגר מגיע לתחנה – הNIC בודק אם frame מיועד אליו (או שהוא broadcast). אם לא – זורק אותו.
- **Ethernet** – מוגדר גם תחת השם **Type** – מכיל את סוג הפרוטוקול של השכבה מעל. ב"כ מצין את פרוטוקול ה-IP, אך יכול להיות גם סימון לARP ועוד כמה אחרים. סוג הפרוטוקול הוא קבוע וערכו מעל 1500. אם בשדה זה מצינו ערך שקטן מ-1500 – אז הערך מצין את גודל המסגרת. סוג הפרוטוקול מוכר לrouter כיוון שהוא תומך גם בשכבה שלישית.
- **CRC** – מכיל את הבדיקה לתקינות המידע המועבר. אם אינו תקין – המסגרת נזרקת.

הEthernet אינו אמין. אין יצרות קשר או לחיצות ידיים בין תחנות, אין גם acks. דומה בהתנהגותו UDP.

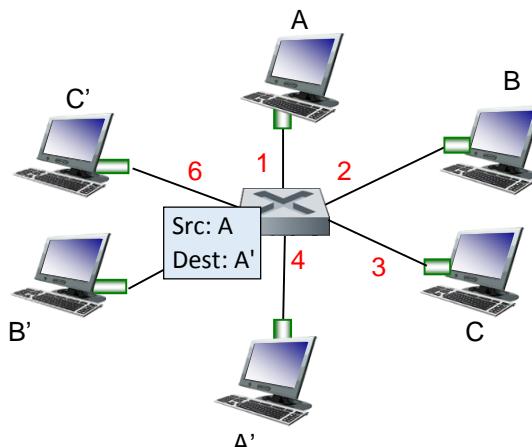


Switch

אוגר ומעביר מסגרות של ethernet. אינו ידוע לhosts ששלוחים את ההודעות, ולא מוגדר מראש איזה frames יעברו דרכו, אלא עובד בשיטת plug-and-play – מזזה חביבה, מփש לאן להעביר אותה, ושולח להלאה.

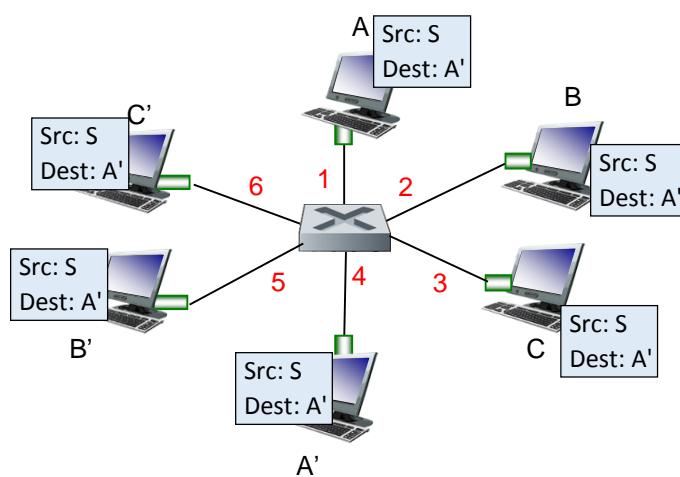
נראה באופן מפורט כיצד עובדת טבלת הניתוב של switch (דוגמא עבור הhost שברmatrix):

A רוצה לשולח הודעה ל-A'. ההודעה מגיעה לswitch מ-1.switch.interface מוסף לטבלת הניתוב שלו את הפרטים של A עם"ג שיוכל להזותו בעתיד:

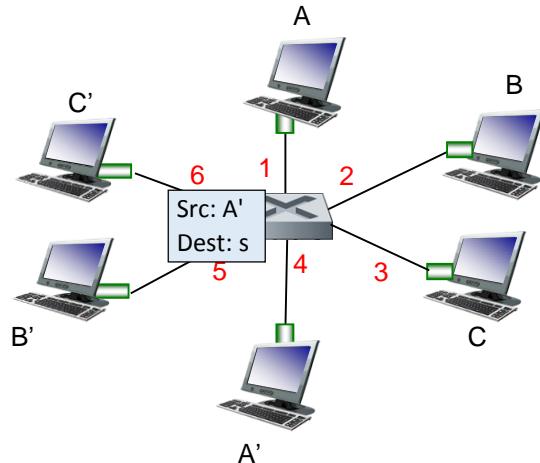


MAC addr	Interface	TTL
A	1	60

הchains switch אינו ידוע היכן יושב 'A' ומאייה interface עליו להוציא את הframe, ולכן שולח בקשה ARP broadcast לכל interfaces שלו, כאשר בבקשת מצוינת כתובת MAC של החains (כדי שיידעו למי לענות) וככתובת IP של 'A' (כי אינו ידוע עדין את כתובת MAC שלו):



כל המחשבים מתעלמים מהבקשה מלבד 'A', שמחזיר הודעה לswitch ובה כתובת MAC שלו:



הswitch מקבל את הודעה ומוסיף את הרשומה על 'A' לטבלה שלו, ושולח לו את frame המיעוד. כאשר בהמשך 'A' ירצה לשЛОו הודעה ל-A ולהיפך, הswitch יידע מאיזה interface עליו להוציא את הודעה עפ"י הטבלה. אם עברו זמן TTL – הוא יבצע ARP מחדש.

MAC addr	Interface	TTL
A	1	60
A'	4	60

** אם מתג מחליף את כתובת MAC שלו ושולח הודעה – הswitch יוסיף רשומה מתאימה לפ'י MAC שקיבל כתעט, והרשומה החדשה תימחק כאשר עברו זמן TTL.



הרצאה 13

נזכיר כיצד מתבצע ניתוב בכל שכבה:

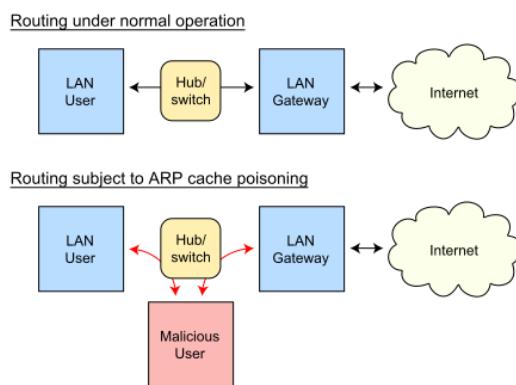
שכבה 2 – לפ' MAC, שכבה 3 – לפ' IP, שכבה 4 – לפ' port.

שכבה הילינק מאפשרת לראות מי התחנה האחורונה שלחה לנו את החביליה. המקבל בודק אם החביליה מיועדת אליו, ורק אם כן – היא עולה להלאה לשכבה IP.

נזכיר גם כי לצורך הכרת כתובת MAC של תחנה – יש לבצע שאלחת ARP.

ARP spoofing

תקיפה בה המחשב התוקף מגיב לבקשת ARP שלא מיועדת אליו ושולח את כתובת MAC שלו למי שchipsh, וכך גורם לשולח להעביר אליו את החבילות המיועדות למשהו אחר. לרבות התקיפה היא בצורת "man in the middle" – התוקף יקבל אליו את ההודעות ואז יעביר אותן למי שאליהן היו מיועדות, וכן חבילות תגובה מהצד מקבל לצד השולח – גם יעברו דרכו.



פרוטוקול ARP הוא פרוטוקול פשוט. אין לו זיהוי, וה'מחפש' זוכר רק את התשובה האחורה. כך שאם תוקף שלח תגובה אחרי המענה המקורי – הרשמה של התוקף תישאר בטבלת הניתוב של המחפש והרשומה של הרכיב המקורי תימחק.

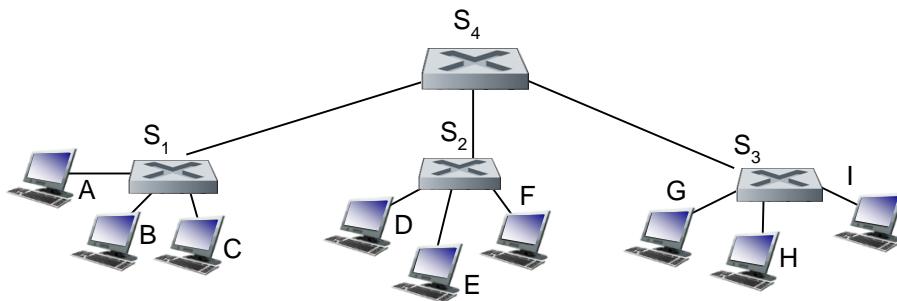
דרכים למניעת ARP spoofing:

1. גישה סטטית - הגדרת טבלת הניתוב של שכבה שנייה בזיכרון כך שאינה משתנה. כМОבן זהה יוצר בעיה של סקלריליות
2. הוספת תוכנה שתציג להודעות ARP ותבדוק תשובות לבקשתות (תווודא אם מי שעננה הוא אכן היעד)
3. תוכנה שתעדקן כאשר יש כפילות של תשובות לARP (כיוון שהיעד המקורי גם יגיב)



פרישת switches

(=מתג) הוא רכיב שלא מכיר את דרכי הניתוב אלא רק את הנק' הבאה, ולכן בניתוח מסגרות יכולה להיווצר לולה. כדי למנוע זאת, משתמשים בפרישה של switches בצורת עץ.



ברשתות ארגוניות, המתג בשורש העץ לרבות יהיה מחובר ל-servers, וכן ל-router שיחבר את הרשת לרשתות החיצונית.

צורת העץ יוצרת מס' בעיות:

1. פרטיות וابتחה – יש צורך ביותר הודעות broadcast שכן תת-עץ אחד לא מכיר את המתגים בתת-עץ אחר.
2. חוסר יעילות – יותר RECEIVES, דרכיהם ארוכות יותר להעברת מסגרות.
3. חוסר יעילות פיזית – רכישת מתגים נעשית לרבות עם מס' קבוע של פורטים לכל רכיב, ובכך צוז סיכוי גבוה שרק חלק מה포רטים יהיו בשימוש.

VLAN = Virtual Local Area Network

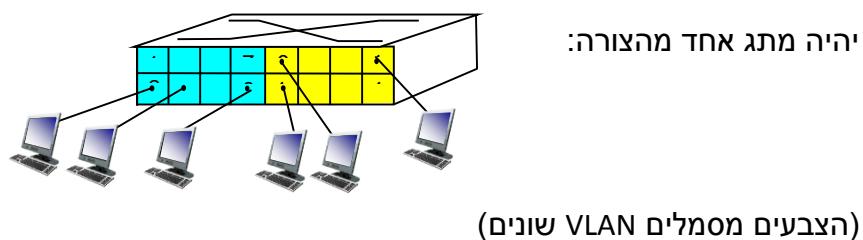
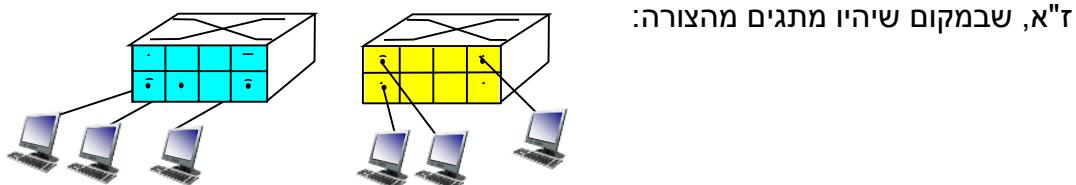
מעבר מ-LAN פיזי לוירטואלי. עד כה, LAN מוגדר (בין היתר) ע"י שיקות המחשב למתגים, אשר מתג שירך ל-LAN אחד בלבד. כאשר מחשב ברשת המקומית עבר לשימוש של תחת-רשת אחרת או עבר מיקום פיזי – המעביר נדרש עבודה פיזית של העברת הcabל מהמתג אליו היה מחובר – למstag אחר, וכן חיבור מחשב אחר במקומו – דורש שוב עיסוק פיזי בכבלים. כמו כן, ניתן שברשותנו מתג עם מס' פורטים גדול ממה שנוצרה כתה, וכך חלקם יהיו מבזבזים.

הפתרון: חלוקה של הפורטים במstag ל-LANS שונים.

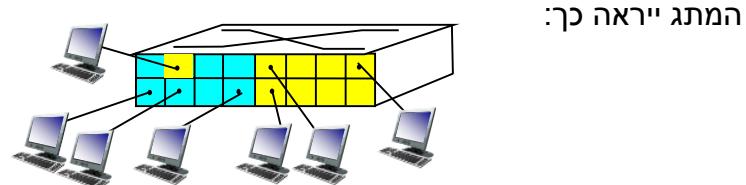
הlayer של שכבת הקו יכול לתגית של מס' LAN אלו שיכת המסגרת, ובמקום חלוקה של מתגים לתתי-מתגים עפ"י LAN – החלוקת תיעשה בתוכנה.



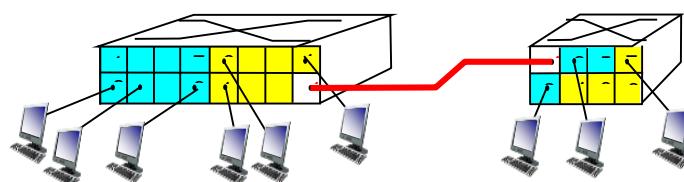
למשל, בדוגמה מהעמוד הקודם – במקומות שמתג S_4 יהיה מחובר פיזית ל-3 מתגים שונים שכל אחד מהם הוא LAN שונה, נגידיר מתג S אחד בעל מס' פורטים גדול יותר, וכל קבוצת פורטים תהיה שייכת ל-LAN אחר כפי שהוגדר מראש. כך נחשוך את השימוש במתגים S_1, S_2, S_3 , S_4 !



השינוי הנ"ל מאפשר סקלビיליות: כאשר מחשב יעבור מחת רשת אחת לשניה – המוגנות שייצאו ממנו יכולות שונות, ולא נדרש להעביר פיזית את הcabלים המוחברים למתג!

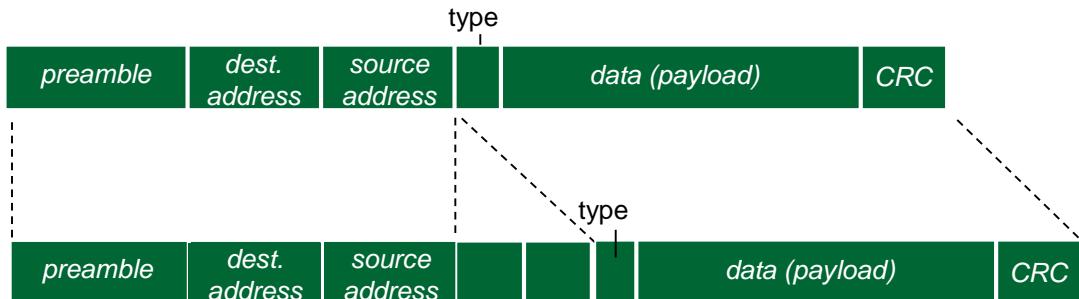


כמו כן, אם נדרש פורטים נוספים ל-VLAN מסוים – במקומות לדאוג להביא מתג גדול יותר ולהעביר מחדש את כל הcabלים, נוכל לחבר מתג נוסף ולהגדיר את הפורטים שלו עבור VLAN המבוקש:



כיצד התווית תוגדר בheader?

נרחיב את שדה header: במקום שיכיל רק את סוג ה프וטוקול/אורך ההודעה, נוסיף שדות נוספים לפני הערך המקורי:



השדה הראשון שהתווסף יהיה שדה לזיהוי שמדובר בheader שמכיל תווית VLAN, וכן ערכו יהיה מס' type קבוע שאינו בשימוש עבור types אחרים שהוגדרו בעבר.

השדה השני – יהיה התווית עצמה, זיהוי VLAN אליו שייכת המסגרת.

← נזכר כי כיוון ששינויו את המסגרת, נדרש לחשב מחדש את CRC.

Data Center Network

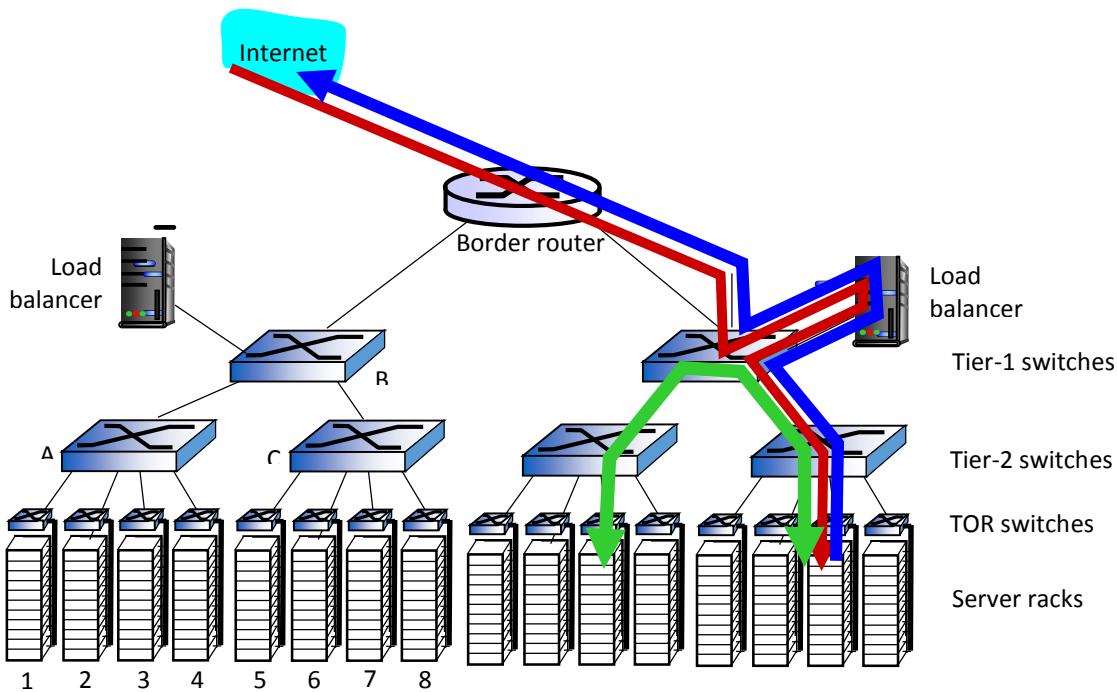
דיברנו בעבר על מרכזי נתונים, כתע נרחיב זאת לגבי שכבות הקו במרכזי אלו.

נזכיר כי על מרכזי נתונים:

- לתמוך במס' רב של לקוחות
- לתמוך בכך חישוב חזק, בעיקר אם מדובר על מנועי חיפוש
- להיות מסונכרנים כל הזמן
- לתמוך בחלוקת עבודה הגיונית כדי להימנע מצוארי בקבוק.

המבנה הבסיסי המקובל למרכזי נתונים הוא מבנה עץ: ראטור לעולם החיצון, שמחובר למוגדים שמייקושרים לload balancer ולמוגדי השירותים.



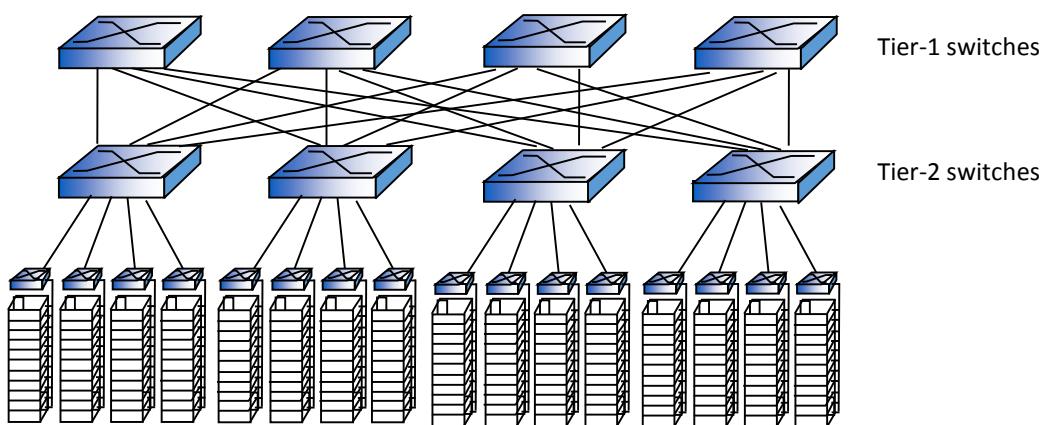


כל Rack הוא קב' שרתים, I-TOR = Top Of Racks – אחראי על קב' השרתים שתתחתיו. בchipos למשל, שרת אחד מ-Rack יוגדר להיות המaster. כל שרת יבצע את החלק שלו בחיפוש, והmaster יקבל את התשובה מכלום, יבנה תשובה וישלח אותה למעלה.

על פיזור העומסים ממונה load balancer. כאשר ישנה בקשה חדשה להקצתה שרת – הוא מחליט איפה להקצות אותה.

חסרון במבנה: מספיק שייהי ניתוק אחד ונבד תקשורת עם שרתים רבים. לכן נרצה לבנות קשרים רבים בין המתגים – והם יכולים לסייע באיזון העומסים. אבל אם zusätzlich עוד קשרים נוכל ליצור מעגלים וחבלים יתקעו במעגל ולא יגיעו לעדן!

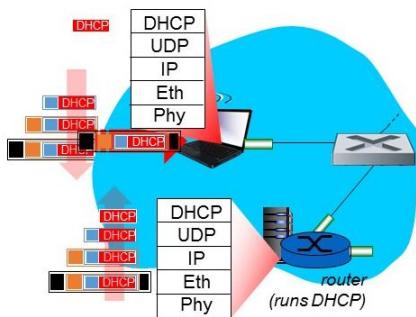
לכן הקשרים ישמרו על צורת העץ ההיררכית: מתגים ברמה העליונה יהיו מחוברים רק למתגים ברמה התחתונה, כך שאף חבלה לא תעלה ותרד בין הרמות ותר מפעם אחת. כמו כן, כדי להקל על המתגיםعلיאים במציאות interface המתאים – נוכל לנפג את כתובות MAC בצורה היררכית. זה יעזור גם למקורה של העברת שרתים ממיקום-למקום.



סיכום הקורס

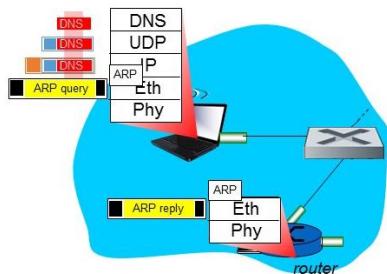
نبיט על תהליך שלם של הצגת דף אינטרנט מהמחשב הביתי שלנו:

שלב 1: חיבור לאינטרנט. המיקום וה כתובות דינמיים.



נשתמש בפרוטוקול DHCP עם"נ לקבל מהראוטר הביתי כתובת IP. הפרוטוקול עצמו רץ כאפליקציה – נוצרת חבילה ומתווסף לה .headers. החבילה נשלחת מהמחשב שלנו ב广播 broadcast ובין היתר מגיעה גם לראוטר. הראורט הביתי מפרק את החבילה עד לשכבות העליות ונותן לנו כתובות IP בלחיצת ידיים משולשת'. (הראוטר קצט מפרק את מודל השכבות שכאן ראוטרים מטפלים רק עד שכבה 3 אבל בפרוטוקול DHCP הוא נטפל גם בשכבה עליונה יותר). מלבד כתובות IP, הפרוטוקול מסור לנו גם את כתובות DNS והMAC של הראורט הביתי, mask, כתובות שרת DNS ועוד.

שלב 2: מציאת כתובת IP של שרת האינטרנט המבוקש.



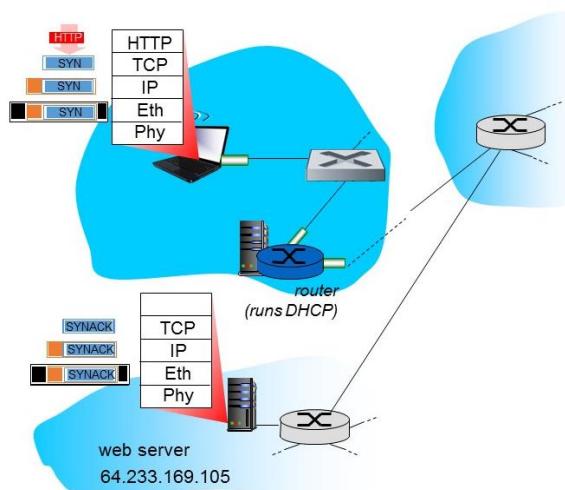
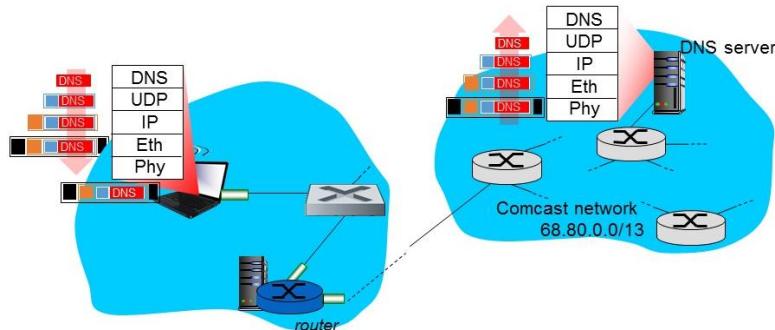
כדי להריץ בקשת DNS צריך למצוא קודם קודם את כתובת MAC של התחנה הבאה בדרך לשרת, ולכן מושכת שאלילת ARP שנשלחת broadcast ע"י המחשב שלנו, והראוטר הביתי יחזיר לנו תשובה עם כתובת MAC שלו.

לאחר קבלת MAC, המחשב שלוח את בקשת DNS לשרת DNS.

שלב 3: החבילה עוברת את מסלול הנitinob עד לשרת DNS ע"י אחד ה프וטוקולים (RIP/OSPF) ככה"נ שהוא גם תעבור מס' Ass בדרך ויהיה שימוש ב-BGP.



שלב 4: שרת DNS (local) צריך להחזיר כתובת IP של השרת המבוקש ע"י URL ששלחנו לו. אם זה לא שמור אצלו בcache – הוא יצטרך לפנות לroot שיבצע שאילתא דרך TLD עד שתוחזר לנו כתובת הIP.



שלב 5: כשבידינו כתובת IP של השרת המבוקש – ניצור TCP socket ונסלח לשרת בבקשת Syn ליצירת קשר. הבקשת Tagיע לשרת (גמ, עפ"י פרוטוקול ניטוב) ותעלה עד לשכבה 4, והוא יענה לנו SYN_Ack ויקבעו בינינו תכונות שונות linked.

שלב 6: לאחר שנוצר הקשר – נוכל סופסוף לבקש מהשרת את הדף המבוקש! נשלח בקשה HTTP (סוג GET), והשרת ישלח לנו את המידע (=הדף), שלרוב נשלח בתווך.html. אחרי שהדף יוצג אצלנו – נוכל עפ"י ה策ור לבקש גם את האובייקטים שמתווספים לו.

