

# Lecture 1 - Introduction to Networking

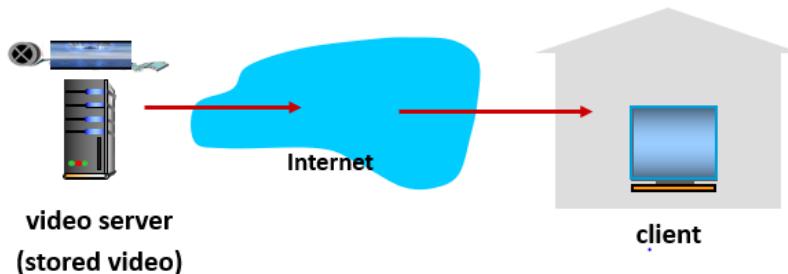
## Introduction

### מה נלמד?

- עקרונות מפתח ברשתות תקשורת – מה זה רשותת תקשורת? מה הם האטגרים המרכזיים של מי שבונה רשותת תקשורת?
- באופן יותר ספציפי, איך האינטרנט עובד – ארכיטקטורות ופרוטוקולים, IP וכו'.

### Dynamic Streaming over HTTP (DASH)

האפליקציהutz מושתמשת ברטה ומביבה דרכה מידע. לדוגמה, streaming video. מצד אחד אנחנו משתמשים באפליקציה כדי לצפות בוידאו, מהצד השני יש את הרשת שם מאוחסן הוידאו, והתוור בינויהם הוא רשותת התקשורת, האינטרנט, דרכיה עברו המידע.



כדי לאפשר צפייה בחולziega גבואה משתמשים ב-

ה-client מבקש chunk של וידאו (מספר שניות מהוידאו) בחולziega מסויימת מה-server, והוודה של chunks מאחסנתו באיזשהו ב퍼 (הקו האודם בשרטוט מייצג chunk). המשתמש מבקש מהרשת לקבל את chunk באיזשהו קצב, bitrate. הקו האודם השרות מייצג bitrate – chunk.

האורך הוא bitrate, כלומר כמה ביטים מעבירים בשנייה. ככל שהאורך יותר גבוה בר החולziega יותר טוב.

הוודה מתבצעת בקצב שונה בהתאם הרשות וב-bitrate, אבל הצפייה בוידאו (התנקודות של chunks מהבפר) מתבצעת בזמן אמת ולכן מתבצע בזמן קבוע.

אם chunk שהקלינט מבקש הם עם bitrate גבוה יותר מההרשת מסוגלת לספק זה בעיתוי, chunk יתנתק מהברפר מהר יותר מהקצב שבו הם מגיעים, וזה יגרום לבפר ריק ולבufferring period. لكن יש אלגוריתמים שמתאימים את bitrate לרשף ולמצב הבפר, ומחליטים מתי ובאיזה chunk להורד.

צפה שה-video server יספק ללקוח צפייה בוידאו בחולziega שבחר בה כאשר רשותת תקשורת מתוווכת. איך תספק הרשות ליזר (QoE) Quality of Experience?

### האינטרנט עובד, אבלעובד בקורס!

הכוונה היא שהאינטרנט הוא סבואופטימלי, לא בטוח ולא צפוי בעוד שיש בנות תעבורת ודרישות גבואה. נחלק את האינטרנט לשש שלוש שבבות:

1. **שכבות האפליקציות**, מזרימות מידע לרשות לצורך המשתמש (facebook, youtube).
2. **פרוטוקולים** דואגים לבצע את המשימות הבסיסיות של רשותת התקשורת.
3. **טכנולוגיות**.

ברגע יש חדשות מבחןת האפליקציות, אבל הפרוטוקולים קצת "תקועים מאחוריה" ופחות מותאמים להתקדמות של האפליקציות. החדשנות הטובות הן שכרגע יש חדשנות בתחום זהה.

במה זמן לוקח לפקטה להגיע מירשלים לניו יורק?

תלוי בהרבה פרמטרים:

- המסלול שהפקטה עוברת (יכול להיות מעגלי).

- מהירות הلينק

- קצב השידור (וחוב פס) של הリンק (ביטים לשניה).

- מספר הקודוקדים (routers) בדרך.

- תחרות בין פקודות. פקטה יכולה לחכות בתור בראותר.

בזמן זהה ה-CPU עושה מחזור של הרבה סיבובים (420 מיליון בערך), נצח במנוחים של זמן חישוב. לכן הפידבק שנתקבל מרשת תקשורת יהיה לא מעודכן והתקשרות תהיה אסינכרונית.

## Computer Network

**מה זאת רשת תקשורת?**

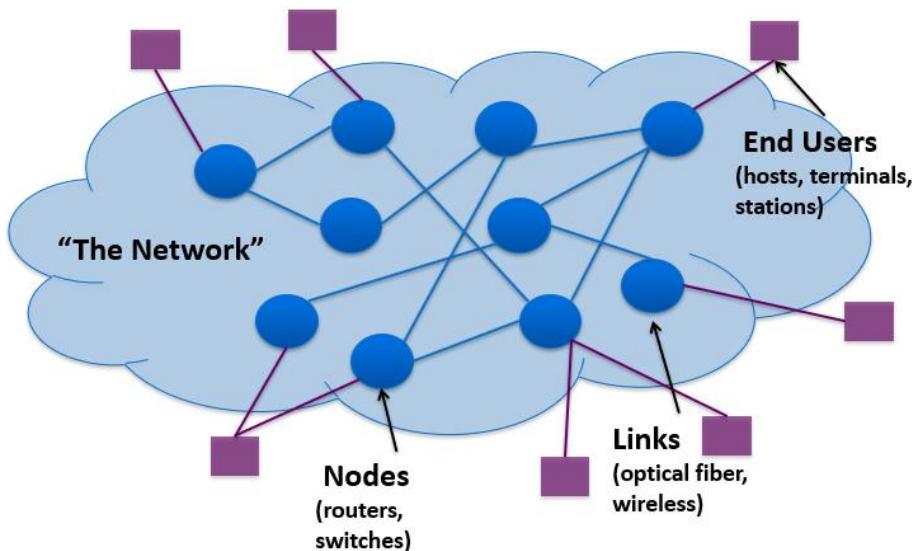
המטרה: העברת מידע בין משתמשי קצה.

לרשת יש שלושה מרכיבים:

1. **משתמשי הקצה** – משתמשי הקצה מזרמים מידע לרשת ומקבלים ממנו מידע.

2. **קודקי הרשת** – מקבלים את המידע ומעבירים אותו להלאה.

3. **אמצעי התקשרות** (لينקים – סיב אופטי, אלחוטי..).



רשתות תקשורת נבדלות אחת מהשנייה בגודלים, במשתמשי הקצה, קודקדים ו שימושים ברשת:

- שליחת הרבה מידע לעומת מעט.
- תקשורת זו ביוניות לעומת חד ביוניות.
- גישות שונה לעיבוביים.
- תקשורת יציבה או פחתה.
- גישות והתייחסות לשילוח פקודות.
- ועוד.

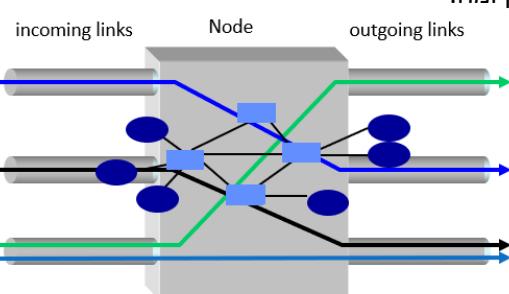
רשת היא תשתית להעברת מידע בין משתמשי הקצה, לעומת **מערכת מבוזרת** שהיא מערכת של משתמשי קצה המשתמשים ברשת (כמו פיסבוק למשל), לא מייצרת מידע. אלגוריתמים מבוזרים גורמים לרשת לתפקיד.

## Circuit switching

### רשת תקשורת לעומת רשת טלפונייה

בעת שיחת טלפון, רשת הטלפונייה יוצרת קשר לאחר החיבור ובעצם שומרת משאבים הדורשים לקיום השיחה עד לסיוםה. כאשר המשאבים לא קיימים לא יוכל לקיים את שיחת הטלפון. רשת הטלפונייה משתמשת במה שנקרא **Circuit Switching** - מיפוי כניסה ויציאה (switching) והקצתה משאבים להן:

**Establish:** נוצר מעגל בין המקור לבין היעד.



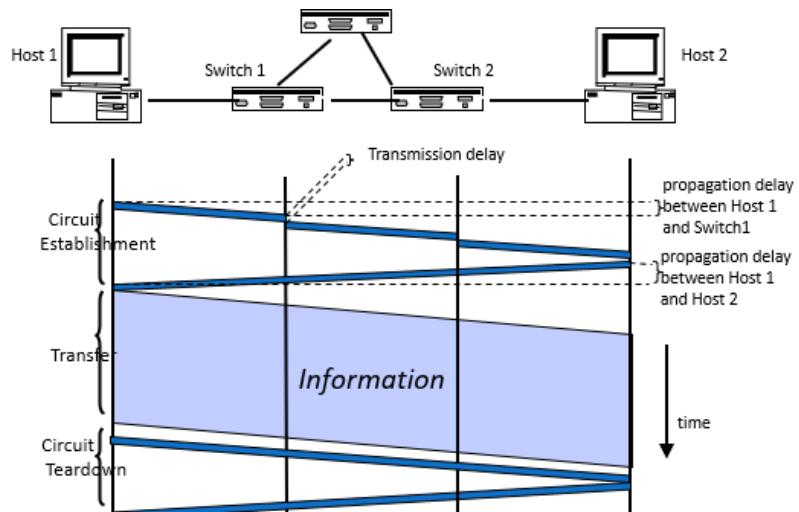
- קודקודים לאורך המרחק את המידע על התקשרות ומשאבים לקיומה.  
- אם המרجل לא זמין מועבר סיגナル שמצין זאת.

**Transfer:** המקור שולח את המידע דרך המרجل שבו שמורות משאבי (קיבולת) ובכך המידע יכול לעבור במסלול.

- אין בתובית יעד, הקודקודים יודעים לאן להגיע.
- זרם רציף של מידע – המשאבים בבר מוקצים והמסלול ידוע.

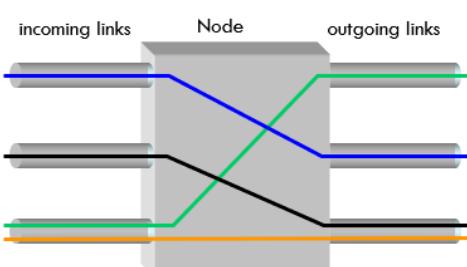
**Teardown:** המקור מבבה את המרجل בסיום ומשחרר את המשאבים.

ככה זה נראה:



### חלוקת זמן

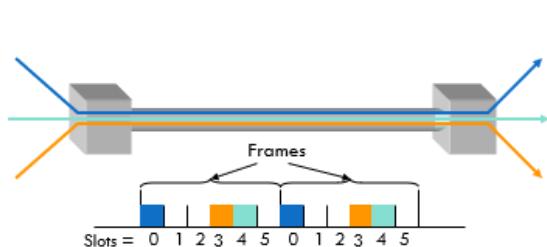
נשים לב בשרטוט משמאל שני ה-circuits הכתום והשחור חולקים את הلينק היוצא. איך זה מתאפשר?



**time division:** חלוקת הזמן לאינטראולים שמחולקים בין ה-circuits.

**frequency division:** שימושpter מדרים שונים לリンק וחלוקת של מרחב התדרים.

(אפשר גם לשלב את שתי האפשרויות)



ב-time-division-time הזמן מחולק לפחות לשאים שמחולקים ל-slots, כל פרויים מוקצה לשיחת מסויימת (למשל בשרטוט 0 שייך לשיחת הכתובת) ודוחש סינכרון בין השולח לבין המקלט. הבעה היא שאין ניתן מקסימלי של הリンק באופן זהה, יש slots שלא מוקצים לאף אחד. צריך לחבר דינמיות slots לשיחת, ואם השיחת לא תנצל את הקיבולת אז היא אבודה.

יתרונות של רשת טלפון ני

1. שמירה והקצת משאבי למודרים בשיחה. אין עיכובים בקודקודים באמצע, מה שנכנס זה מה שיוצא ואין עיבוד של הנתונים.
2. מערכת צפואה.
3. אין נפילות ברשת (למעט נפילת-link או קודקוד).
4. קל לעקב אחריו שיחה בשביל reverse engineering.

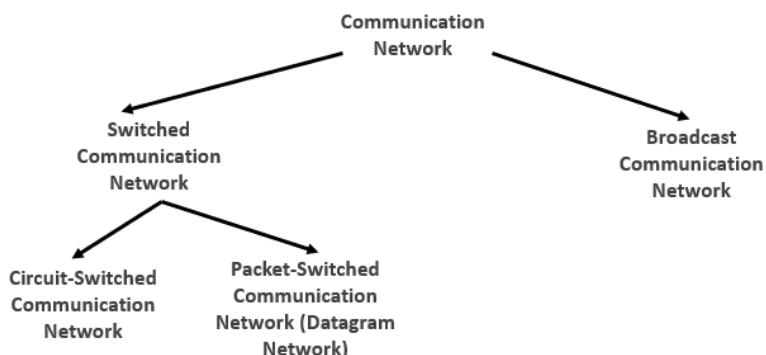
 חסרונות של רשת טלפון ני

1. "הכל או כלום", אין **חסינות לתקלות** וכל תקלה מונעת את השידור.
  2. **בזבוז משאביים**.
- עבור אפליקציות רשת עם רוחב פס מקסימלי של  $P$  (מידע מקסימלי שעובר בשיחה) אבל ממוצע שליחת של  $A$ , הרשת תקצת מראש קיבולת של  $P$  אבל איז הניתול הממוצע (average throughput) יהיה רק  $\frac{A}{P}$ .  
 לחلك מהאפליקציות יש יחס קטן של  $\frac{P}{A}$  (smooth applications, בערך 3:1) אבל לרוב אפליקציות הן יותר bursty, יחס יותר גדול (1:100 ומעלה). עבור אותן האפליקציות שימוש ב-circuit switching יהיה לא יעיל בגלל הקצת המשאביים מראש.
3. **רשת הטלפונייה** נבנתה **על-****אפליקציה-****ספקטיבית**, מותאמת לדרישות של שירותי טלפון ולא למגוון אפליקציות כלליות ולא צפויות כמו באינטרנט.
  4. **Setup Time** – בזבוז משאביים רק כדי לבסס את ה-circuits במקורה בו התקשרות יותר קצרה מהזמן שה비스 לוקח לחוב בטלפונייה התקשרות יותר ארוכה מהזמן הזה, אבל באינטרנט לא.

כדי להתגבר על הבעיה האלו באינטרנט אין circuit switching, אלא **packet switching** – חלוקת המידע לסגמנטים קטנים - pakcets, כל סegment נשלח לרשת והוא אחראי להעברתו הלאה.

Taxonomy of Communication Network

ניתן לחלק את רשתות התקשרות באופן הבא, לפי הדרך בה הקודקודים מעבירים מידע:



הבעיות: **Broadcast Communication Network** – אין switching, במשמעותו מדובר כולם שומעים אותו (כמו מרצה בשיעור). קורה בד"כ ברשתות מקומיות. המידע מועבר ע"י כל קודקוד ומתקבל ע"י כל קודקוד. בד"כ בשימוש ב-LANs כמו Wi-Fi, ethernet.

1. טווח מוגבל.
2. צורך במנגנון שקובע מי מדובר ומה (Multiple Access Problem).
3. פרטיות.

**-Switched Communication**

1. Circuit Switching, כמו בטלפונייה. משאבי נשמרים כאשר שיחה מותקימת.
2. Packet-Switched שזה בעצם האינטרנט.

## Datagram Packet Switching

### פקטה Datagram

כל פקטה בוללת:

1. **הדאטה** שציריך לשלוח.
  2. **header / תחילה / מטה-Dאטא.** הוראות לרשת לשילוח הפקטה.
- לא קיים בטלפוניה.

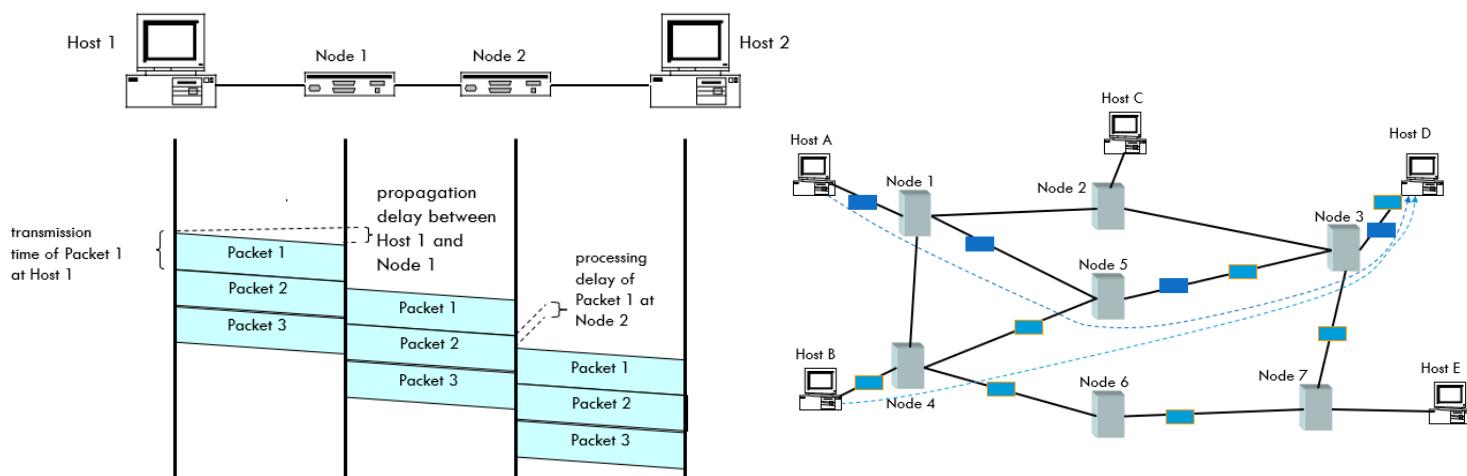
### Datagram Packet Switching

בדרכ ניתוב זו כל פקטה מנוטבת באופן עצמאי, נכנסת לרשת בהתאם למשאים שברשת, אין משאים מוקצים מראש ולכן תחכום בין הפקטות. כדי לאפשר את השילוח הפקטה מכילה את המטה-Dאטא ובין היתר את כתובות היעד המלאה.

בעת העברת הפקטה מ-*host* אחד לשני, הפקטה עוברת דרך הקודקודים של הרשת, הם קוראים את המידע שנמצא ב-*header* ומעבירים את הפקטה.

בעת שליחת פקטה משתמש קצה לקודקוד או מקודקוד אחד לאחר יהיה *propagation delay*, הזמן שהוקלט לביט הראשון בפקטה הגיע מהמקור לעד.

המחשה:



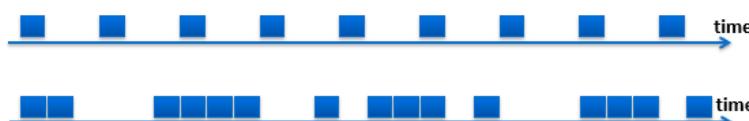
### Bursts

פקטוות יכולות להגיע לקודקודים בקצבים שונים.

נסתבל על שני היצרים שלמטה – היצור העליון מתאר שליחת פקטוות במשך זמן קבוע (smooth), אבל לרוב אפליקציות באינטרנט מתנהלות כמו התיכון, ב-*bursts*, בפיטרנים שונים של שליחת ולא בקצב אחיד. ה-*bursts* מאפשרים לנו (נדבר על בר בקרוב).

בעקבות ההתרכזיות, מתישחו יקרה מצב בו קצב הגעה היה גבוה גובה מקצב העברה. הבעיה היא שלא שמננו יכולות עבורי הפקטוות ולפעמים המשאים לא יספיקו למידע שעובר. פתרונות אפשריים הם:

- להפיל פקטוות ולגרום לאיבוד המידע
  - להוסיף בפרטים בהם נשמר פקטוות עודפת, ונשלח רק כשיאפשר (בשאן התרכזיות ופחות עמוק).
- אבל גם איז פקטוות יכולות ליפול (גם אם הבפרים היו אינסופיים פקטוות יכולות ליפול).



## Lecture 2 – Layers + Broadcast

### Statistical Multiplexing

איך מתחממים עם זרים של תעבורה שמנעים לרשף? – **Queuing Theory**

טרמינולוגיה:

1. **Arrival Process** – איך פקודות מגיעות.

- קצב ממוצע של העברת מידע.

- $P$  השיא "peak".

2. **Service Process** – זמן שידור ממוצע לרשות בפונקציה של גודל הפקטה.

3.  $W$  זמן המתנה ממוצע של פקטה בתור/בפר.

4.  $L$  מספר הפקות בממוצע הממתינות בתור.

טלפונייה צריכה לשמר משאבי שימוש תקשורת ב-peak, ההבדל בין A ל-P הוא לא גדול ולן זה לא משנהו על עצמה אינטרנט.

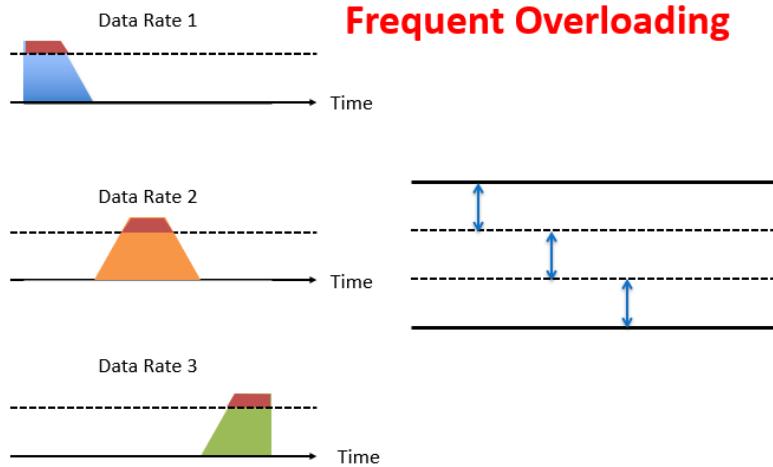
התושים הבא מתאר שיחה "מתפרצת" של שלושה flows, דרך לינק ייחד עם קיבולת מסוימת (מיון) לאורך זמן.

בצד שמאל מתוארת כמות המידע שבכל גורם רוצה לשולח.

קצב השילחה המקורי של כל גורם, ה-P, הוא קטן יותר משליש מהקיבולת של הלינק. במקרה זה בטלפונייה נשמרים משאבי לשיחה מראש (בහנכה שכולם מדברים במקביל) לפי ה-peak, אבל אין מספיק משאבי לשם כך. הראשון והשני היו כננסים, אבל השלישי לא יוכל להיכנס.

לעתם זאת, בכל זמן לאורך הזמן אין נצליח של יותר מקיבולת הלינק. אם הגורמים היו מדברים מתי שהם רצים, הקיבולת של הלינק הייתה מספיקה. הפרצים עליהם דיברנו לא קוראים באותו זמן – גם אם האפליקציות הן bursts, הן לא דואו חוץ לדבר בפרצים באותו זמן, ולכן זאת לא חיית להיות בהכרח בעיה ונרצה לנצל את זה.

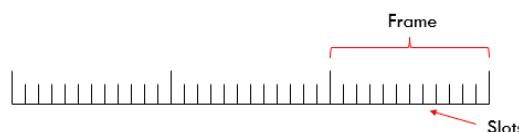
### Frequent Overloading



### Statistical Multiplexing

נשענת על ההנחה שהתעבורה ברשת היא burst, לא כל השיחות "מתפרצות" באותו זמן.

זמן מחולק למסגרות, frames, וכל מסגרת מחולקת ל-slots. בכל frame כל flow מייצר פקטה.  $P$  יהיה השיא ביצירת פקטה למסגרת, ו-A הממוצע.



בעקרון נרצה לקבוע את זמני השילחה של הפקות ולהקציב ל-flow יחיד מספר slots לפי  $P$  כדי למנוע אבוד מידע.

יתכן כי ב-flow ישיחס גדול בין  $P$  לבין  $A$  וזה יגרום לבזבוז (למשל בדוגמה למעלה,  $P = \frac{\text{capacity}}{3}$  ו-  $A = \frac{\text{capacity}}{9}$ ) – אם יש הרבה ניסויים בלתי תלויים ( $N$ ) עם תוחלת  $A$ , אז בהסתברות מאוד גבוהה בסופו נצפה לקבל ערך קרוב לתוחלת של כל הניסויים –  $A \cdot N$ .

נסתכל על שתי הנחות לא נכונות:

1. באינטרנט כולם היו מדברים באותו זמן, bursts קורים באותו זמן.
2. ה-*flows* מחולקים באופן יפה על הזמן, שימושו רצhaft לדבר האחרים שותקים.

לעומתנו, Statistical Multiplexing עושים שימוש בחוק המספרים הגדולים ומניחים שיש  $N$  - flows בלתי תלויים שרצחים להשתמש בלבד בリンク מסוים בתפלגות מסוימת. במקרה, נדרש להקצות  $A \cdot N$  ולא  $P \cdot N$ .

המודל הזה לא מדויק אבל מדויק יותר מאשר ההנחה. לא תמיד יוכל להניח אי תלות בין ה-*flows* אבל זה יהיה יותר מדויק.

## Circuit Switching vs. Packet Switching

### Circuit Switching

אם ידוע מראש שהפער בין  $P$  ל-  $A$  הוא לא גבוה, כמו בטלפוןיה, כדאי להשתמש ב-*circuit switching*:

יתרונות:

1. קיבולת מובטחת, התנהלות צפופה.
2. אבסטרקציה פשוטה, קל "לדבר" על זה ולזהות תקלות.
3. קל למשוך את העברת המידע. מבוסס על סמך חלוקת הזמן או התדר.
4. אין בזבוז משאים על מידע של העברה, תקורה נמוכה פרמידע.

חרוגונות:

באינטרנט אין מספיק משאים לספק למשתמשים בגלל הקצתה משאים מראש.

1. חוסר יעילות בכל שהיחס בין ה- $P$  ל-  $A$  מאד גדול.
2. מניעת תקשורת, אחד הצדדים לא יכול לדבר בעוד אין מספיק משאים.
3. שמיירת משאים מראש לכל שיחה ע"י הנتب דרכו השיחות עבותות. אם קורה משאו והשיחה "געלמת", הנتب לא ידע על זה ועודין יחזיק את המשאים. נדרש מנגנים כדי לחלק את המשאים שהוקצו.
4. במקרה של שיחה בה עבר מעט מידע, הקצתה המשאית בזבוני לוקחת יותר זמן מהשיחה עצמה.

### Packet Switching

יתרונות:

1. אמינות: כל פקטה מטופלת בראשת באופן עצמאי וקל להתמודד עם תקלות בזמן שיחה.
2. Statistical Multiplexing: ניתן לנבוע של הרשות במקורה בו ה-*bursts* לא מסוכרים, שיפור הייעילות.
3. האינטרנט מחבר רשותות שלא תמיד משתמשות באותה טכנולוגיה ומאפשר להן תקשורת. צריך שכל צד ידע לתקשר עם הצד השני וזה יותר קל בשמהדר בפקותות ולא הקצתה משאים.

חרוגונות:

1. התמודדות עם עומסים: יפלן פקודות יהיה צריך להתמודד עם עומסים כי לא שמרנו משאים מראש. נדרש לייצר כלים להתחמיזות עם הבעה.
2. חוסר צפיפות: מכיוון שלא נשמרים משאים, אין הבטחה להגעה של פקודות ובזמן מסוים (יפלו, יחכו בתור וכו').

**מודולריות**

חלוקת מערכת לモודולים שונים. המימוש של פונקציה הוא בקורסא שחורה, והקשרים שונים מקבלים את הגישה אליה. המודולריות מאפשרת גמישות ושינוי הקוד בצורה נקייה. הבעה היא חסר וודאות שיכל לפגוע ביעילות, אך שודוקן אם כן הייתה חשיפה לפרמטרים או בדרך חישוב היה אפשר להפוך דברים לעילום יותר.

**Network system modularity**

- בأنternet יש מודולריות מבוצרת, מימוש המודולריות במכונות שונות. פונקציונליות שרק משתמשי הkaza שותפים אליה.
- ישן החלטות קritisיות לגבי המודולריות:
1. שליחה מייד אל יעד לא מוגדרת היטב – מהי חולצתו של השילוח? איך שוברים את המערכת למודולים? ע"י **שכבות**.
  2. איפה המודולים ממומשים? **עקרון הקaza לקaza** – לדבר שהוא מעבר לפונקציונליות הבסיסית של הרשת צריך לקרה בקצבות (הקצבות למשל יתעסקו עם נפילת פקודות ולא הרשת).
  3. איפה המידע לגבי קיומה של שיחה נשמר? **עקרון Fate Sharing**.

**חולצויות, משימות ושליחת מידע**

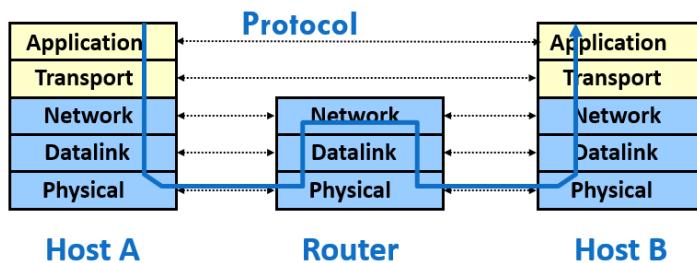
- ← **פיזיקלית**: אלקטرونים שעוברים על סיבים.
  - ← **ביטים**: מעבר של 0 ו-1 בסיבים.
  - ← **מעבר פקודות**, רצפים של 0 ו-1.
  - ← העברת פקודות ברשתות מקומיות (LAN) ע"י בתובות מקומיות ורשתות שמרשות רשתות מקומיות ע"י בתובות גלובליות.
  - ← שימושות נוספת כמו שליחת מחדש החדש של מידע שאבד למשל.
  - ← בסוף נצטרך לעשות משהו עם המידע שנשלח.
- הأنternet מתמודד עם המשימות האלה באמצעות שכבות, סוג של מודולריות שבה כל המודולים נמצאים אחד מתחת לשני ולכל מודול גישה למי שמעליו ולמי מתחתיו בלבד. ניתן להמחיש את זה בעזרת הדוגמה הבאה – מכ"ל כתוב מכתב למנכ"ל אחר, מקפל את המכתב ומעביר לមזכיר. המזכיר שם את זה במעטפה יחד עם הפרטים של המנכ"ל השני ולקח לדואר שם דברים נוספים מהתוכן שלפניו שמאפשרים הגעה לעד.
- ← גורמים באויה שכבה מבינים זה את זה.
  - ← לשכבה התחתונה יש הכى הרבה עטיות.

**מודול שבע השכבות**

1. **השכבה הפיזית**: העברת ביטים על לינק בודד (אבטרנטיקה להעברת אלקטرونים וייצוגם כביטים).
2. **שכבת הלינק**: העברת פקודות (רצף של ביטים) על פני לינק.
3. **שכבת ה-network**: העברת פקטה דרך דרכו כמה רשתות מקומיות.
4. **שכבת ה-port:transport**: דואגת להעברה אמינה של פקודות.
5. **שכבת האפליקציה**: עושה מייצרת את המידע או מקבלת אותו ועשה אליו משהו.

**מימוש השכבות**

הकצוות, hosts ממשים את כל חמש השכבות, לעומת זאת הקודקודים בראש רק את השכבות שרלוונטיות להעברת המידע. הפקטה נוצרת בשכבת האפליקציה, געטפת בפרטים נוספים כדי שאויה שכבה בkaza השci תדע מה לעשות עם המידע. בrama הפיזית היא נשלחת לנתח שmapsית את השכבות ברמות שרלוונטיות לו (התחתונות). הוא מעביר את זה הלאה עד לkaza השני שם הוא מעלה את זה בשכבות עד להגעת הפקטה לשכבת האפליקציה.



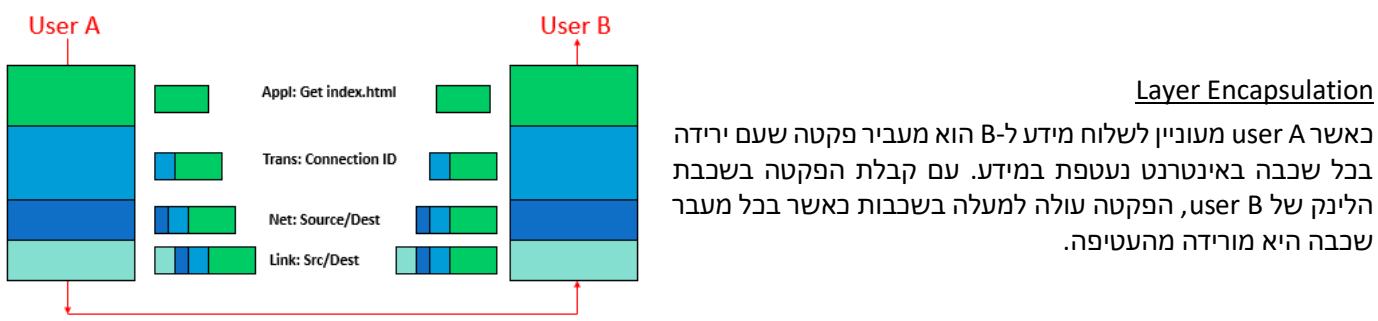
### פרוטוקול

**פרוטוקול:** הסכם על דרך תקשורת, איך להעביר>Data ו-air לתחום הקצאת משאים משותפים.  
חייב להיות מוגדר באופן מלא ומפורט (לעומת אלגוריתם).

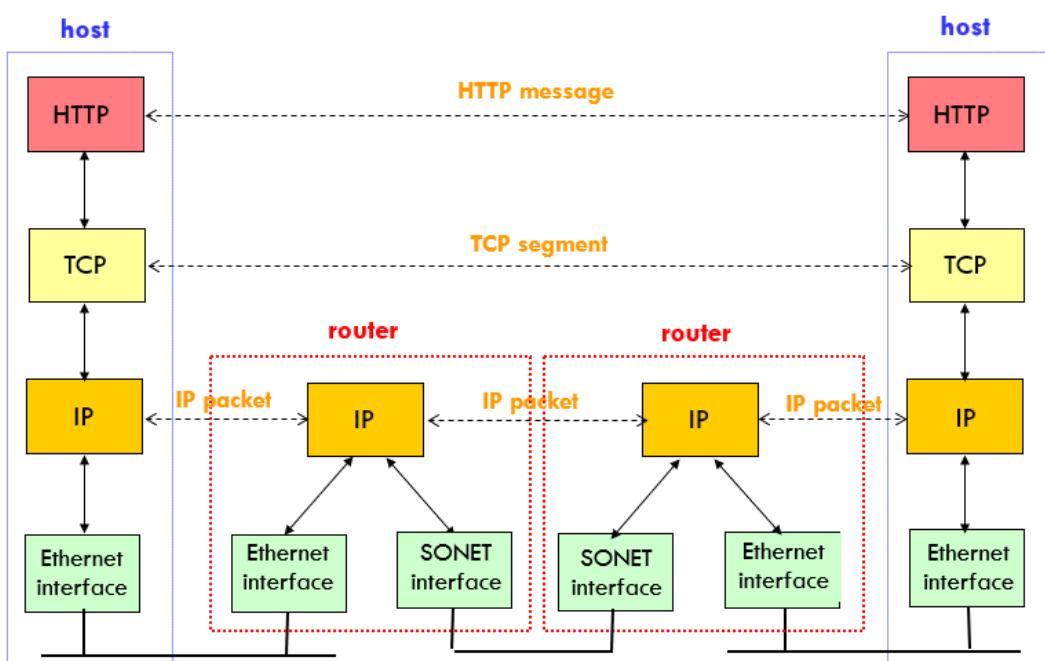
מגדיר סמנטיקה וסינטקס:

- ← סמנטיקה – מבנה הפרוטוקול, פורמלט, סדר ההודעות וכו'.
- ← סינטקס – איך להגביל להודעות, לאירועים וכו'.

כל-hosts חייבים לעבוד עם אותו פרוטוקול, שימושים בפרוטוקול יכולם לגרום לביעות גדולות וכן לגרום ל-host להתנתק מהאינטרנט כי הקודקודים לא יוכלו לשלוח אליו ולקלם ממנו. לכן יש **סטנדרטיזציה של פרוטוקולים**: שינוי של פרוטוקול צריך לעבור אישור של גוף בשם IETF.



פרוטוקולים ברמות שונות:



## Characterizing the Layers (OSI)

כל שכבה מאופיינת ע"י מספר פרמטרים:

1. **Service**: השירות שהשכבה מספקת, מה היא עשויה.
2. **Interface**: כיצד ניתן לגשת, להשתמש דרך השכבה שמעליה.
3. **Protocol**: איך מתרת השכבה ממומשת? ע"י איזה פרוטוקול?

### Physical Layer

– העברת נתונים על תווך פיזיקלי בין שתי מערכות.  
**Service** ←  
**Interface** ← – מצין איך לשלוח נתונים ולקבל נתונים.  
**Protocol** ← – פרוטוקול שמודע לקחת את הסיגנלים ולפרש אותם בהתאם בית ווש שכבה שמודעת להעביר אותם.  
הפעם האחרונה שנדבר על השכבה, מעתה נניח שיש בית ווש שכבה שמודעת להעביר אותו.

### (data) Link layer

– מאפשרת העברת הودעות בצורת פקודות בין משתמשי קצה בסוג מסוים של רשות. בין השאר, רשותה בהן לוגרמים השונים יש בתובות מקומיות.  
**Service** ←  
**Interface** ← – שליחה וקבלת הודעות למשתמשי קצה אחרים.  
**Protocol** ← .Routing, MAC

### Network layer

– רישوت רשותות מקומיות, שליחת פקודות לעודים ספציפיים דרך בתובות גלובליות.  
**Service** ←  
**Interface** ← – שליחת פקודות לעוד internetwork ספציפיים, קבלת פקודות ממשתמשי קצה.  
**Protocol** ← – בניית טבלאות ניתוב, IP למשל.

### Transport layer

– תקשורת בין תהליכיים, פילוג תקשורת בין hosts (Demultiplexing). אופציה לאミニות בהעברת המידע וההתאמה הקצבית.  
**Service** ←  
**Interface** ← – שליחת הودעה לתהליך ספציפי ביעד נתון, תהליכיים מקומיים מקבלים הודעות.  
**Protocol** ← – אミニות, framing, flow control, UDP ו-TCP. לדוגמה TCP

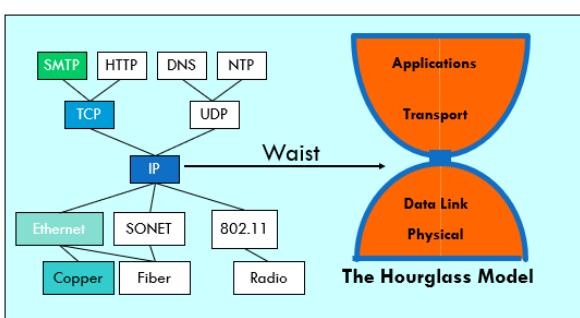
### Application Layer

השכבה שהכי פחות מוגדרת, לא חלק מהפונקציה של הרשות. מספקת שירות למשתמש הקצה. מזרימה מידע לרשות ועשויה בכךו בסוף.

### מודל שעון החול

מודל הרוחב של השכבות.

Narrow waist – ככל שמתקרבים לשכבה ה-IP כך מספר הפרוטוקולים שימושתיים כל שכבה פוחת, עד כדי כך שהוא שיש בשכבה הרשות הוא ה-IP בלבד. לא מדובר לגבע שאר השכבות אבל כן נכוון לגבי שכבת הרשות.



**בנייה האינטראנט – Communicating in a LAN**

Application
Datalink
Physical

דבר בעת על شبكة 2, ונסתכל ברגע על המצב הבא: האפליקציה מדברת שירות אל רשות מקומית, אל שכבת-Datalink שמתהכלה השכבה הפיזיקלית. הרמה הבסיסית שהשכבה אמורה לספק היא גישה של הרבה גורמים (Multiple Access) לlienk בווד (Broadcast Domain). כאשר מישחו מדבר במרחב זה, בולם שומעים אותו.

טרמינולוגיה

1. בשכבה זו נכתבה פקטה בתור **frame**.
2. End host נקרא **nodes**.
3. **lienks** מרכיבים את המסלולי תקשורת שבין nodes השונים.

השכבה השניה אחראית על שליחת המידע node אחד לאחר על גבי הלינקים, דרך סגמנטים שונים. בטור המחברה, נרצה לנסוע מהונולו לירושלים ונעשה זאת זה ע"י מוניות לשדה התעופה, טיסה לישראל ורכבת לירושלים. אפשר להתייחס אל הנושא באל המידע, ה-**datagram**. כל סגמנט הוא link שונה וכל דרך לנסעה היא הפרוטוקול על גבי הלינק. סוקן הנטיות ישתמש באלגוריתם הנכתב.

השכבה מספקת את יכולת לקבל גישה לטור מדויים מקומיים שמתחלק בין גורמים (lienks פיזי, ווIFI) לפי כתובות MAC שנמצאות ב-frame ומכוילות את המידע על היעד ועל המקוור. בנוסף השכבה מספקת גם אמינות למעבר המידע: זיהוי ותיקון טויות, שידור מחדש וכו'.

השימוש של השכבה הוא בכרטיס רשת (NIC) במכשור של המשתמש.

**Multiple Access Protocols**

גורמים שונים חולקים את אותו broadcast domain (אזור ווIFI, link בווד). בעצם יהיה מדבר וכולם שומעים, ונרצה לאפשר את זה תוך מניעת התנגשויות וניתול עיל של הלינק. זה מתאפשר באמצעות **Multiple Access Protocol** – אלגוריתם מבוזר שמחלית מתי node מסוים ידבר, כדי לאפשר ניתול טוב של התווך. אבל תקשורת על שיתוף urzots מחיבת להשתמש בעוז עצמו (האלגוריתם יצטרך גם להיות חלק מה-link), וכן נרצה להגדיר את הפרוטוקול הראשי.

Multiple Access Protocol אידיאלי:

- עבור broadcast channel שיש לו קיבולת (רוחב פס) של  $zkb R$  (קצב שידור) -
1. אם קודקוד אחד משדר נתן לו את כל הקיבולת, ניתן לו לשלוח בקצב  $R$ .
  2. אם יש  $M$  קודקודים שמשדרים נרצה שבמוצעם כל אחד יקבל  $\frac{R}{M}$  כדי לקיים הוגנות וחוסר הרעבה.
  3. נרצה שהשידור יהיה כמה שיטור מבוזר, ללא סינכרונייזציה ולא קודקוד מרכזי שמנטב את השידור.
  4. פשוטות.

שתי מחלקות של פרוטוקולים לשם כך:

1. **פרוטוקול ריבוצי:** אפשר להמחייב ע"י גיר שMOVEDER בכיתה כך שמי שמחזיק את הגיר הוא מי שմדבר או ע"י מרצה בכיתה שמחלית מי מדבר. יתכן שתנאי 3 לא יתקיים ולא תהיה הוגנות.
- **Channel Partitioning** – חלוקת העזרץ ל-slots והקצתם לקודקודים.
- **Taking Turns** – חלוקת תורם לקודקודים כך שכל קודקוד ידבר בתורו, קודקודים עם יותר מידע לשלוח ירצו לקחת יותר slots.
2. **Random Access**: גורמים מדברים בצורה רנדומלית, אין סנכרון. יתכן שייהיו התנגשויות וצריך ללהות אותן ו"להחלים" מהן. מה שקרה בפועל באינטראנט.

## Lecture 3 – Multiple Access to a Single Link

Review

דיברנו עד עכשיו על רשת הטלפוניה, שקדמה לרשת התקשרות, שפעלה ע"כ circuit switching ומאפשרת ברכות חיבור לudsonים וולוקה בזמן כדי לספק קיבולות לגורמים שחולקים את הרשות. התחילו "לבנות את האינטרנט" והתייחסנו לשילוח שכבות בלבד (במקום חמיש), והתעסקנו בשכבות ה-layer link בהנחה שהשכבה הפיזיקלית מאפשרת וממשתתפת(interface) לשילוח פקודות. הבעה הראשונה שנפתחה נוגעת לרישות של לינק בודד – broadcast domain, בו כאשר מישוה מושדר כולם שומעים, המידע נשלח לכלם.

### טרמינולוגיה

1. End hosts וקודקי הרשות נקראים **Nodes**.
2. **לינק** הוא התווך בו הקודקים מדברים.
3. **פקטה** = frame.
4. **דאטא**, המידע שהפקטה עטפת.

בתוך כל קופיצה יש פרוטוקול שפותר בעיות. בסוף יש מהשו שקובע באיזה דרך הלכנו. שכבת הلينק מעבירה את המידע בין קודקים על גבי התווך.

### שכבת הリンק תומכת בשני סוגי שירותים:

1. קישוריות
2. זיהוי שדים עוברים כמו שציריך – אם מידע נפל או לא הגיע לעד הוא ישלח שוב.

שכבת הリンק ממומנת בין שתי רמות – רמה משתמשת במידע ורמה שעבירה את המידע בתוך הפיזי. לעומת השכבה ממומנת בין ה-CPU לבין ה-physical interfaces שאחראים על לשילת המידע החוצה. (Network Interface Controller, כרטיס הרשות)

נתעסק ברגע בתקשרות מרחב אחד (לינק פיזי או wifi) שמאופיין ע"כ **Broadcast / Collision Domain**, בו אם מישוה מושדר כולם שומעים או במקרה של התנגשות אף אחד לא ידבר. נרצה אלגוריתם שיקבע מתי כל קודקן מדבר, ובו כל אחד ידבר ע"י ניצול מרבי של המרחב.

- כל תיאום בתקשרות עבר בתוך המשותף, כן יכולה להיות התנגשות.

תווך קווי אליו גורמים שוכנים שלא מאפשר תקשורת מנוקודה לנוקודה בלבד שהנקודות בדרך יישמעו. פעם ה-**bus topology** – תווך קווי היה ממומש בכבה, אבל היום broadcast domain יהיה לינק בודד בין שני גורמים (שני מחשבים / מחשב ו-switch) כמו בטופולוגיה כוכב – כל הקודקים כוכב – כל הקודקים דרך switch יחיד.

נזכר בתוכנות אידיאלית של פרוטוקול **Multiple Access**:

- עבור broadcast channel שיש לו קיבולת (רווח פס) של  $R$  bps (קצב שידור) –
- 5. אם קודקן אחד מושדר ניתן לו את כל הקיבולת, נאפשר לו לשולח בקצב  $R$ .
- 6. אם יש  $M$  קודקים שמשדרים נרצה שבמוצע כל אחד יקבל  $\frac{R}{M}$  כדי לקיים הוגנות וחוסר הרעבה.
- 7. נרצה שהשידור יהיה כמה שייותר מבודה, ללא סינכרונייזציה ולא קודקן מרכזי שמנטב את השידור.
- 8. פשוטות.

הגישה **Random Access** היא גישה אחת למימוש הפרוטוקול ואומרת שבгинון פקטה לשילחה, היא תשליח. נראה בעת פרוטוקולים שמממשים את הגישה.

## Slotted & Unslotted ALOHA

### Slotted ALOHA

הפרוטוקול שיתואר הוא לא אידיאלי ולא נמצא בשימוש, אבל מובא לשם המלצה.

#### הנחות:

1. כל הפקטות בגודל שווה.
  2. הזמן מוחלט – **slots** בגודלים שווים כך שכל אחד מהם הוא בגודל הזמן לשידור פקטה אחת.
- מהתרגול – רוחב הפס הוא  $B$ , גודל פקטה הוא  $L$  אז גודל כל-slots יהיה  $L/B$  כך שכל-slots יהיה אפשר לשולח פקטה.

3. שידור פקטה יהיה בתחילת slots.
4. **הקודקודים מסונכרים** (ה-slots אצלם מתחילה באותו זמן).
5. (לא בהכרח מתקיים) אם שני גורמים מדברים באותו זמן, מזהים שהייתו התנגשות.

#### האלגוריתם:

- קודקוד ינסה לשלוח פקטה חדשה ב-slots הבא.
- אם לא הצליח (יש התנגשות), בכל slot אחריו הוא ינסה בהסתברות נקבע לשולח. (זרק מטבע)

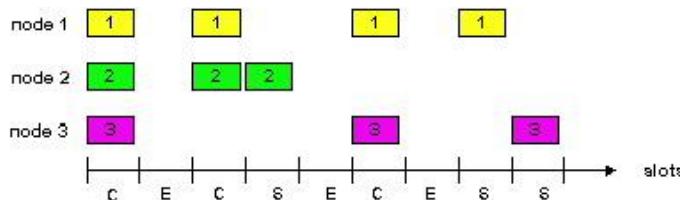
#### דוגמה:

← שלושה קודקודים חולקים אותו תוקן ומנסים לשולח פקטה ב-slots הראשון, אבל הייתה התנגשות ולבן אף אחד לא הצליח לשדר.

← ב-slots הבא אף אחד לא מנסה לשולח.

← לאחר כר 1 ו-2 מנסים לשולח, אבל שוב הייתה התנגשות ולבן אף אחד לא שלח.

← רק 2 מנסה לשדר, אין התנגשות ולבן הוא מצליח.



#### יתרונות:

- קדקוד יחיד היה מנצל את הلينק באופן אופטימלי (לא היו התנגשויות ולכן היה שולח מתי שרצה).
- מבוזר, אף גורם לא משדר בזמן שגורם אחר משדר.
- פשוטות.

#### חסרונות:

- בוודאות יהיו התנגשויות.
- יהיו מקרים בהם לא יהיה ניצול מרבי של הリンק, הרבה slots מבוזבים.
- סבךון שעוכבים.

#### Goodput vs. Throughput

**Throughput** – בכמה זמן (אחווד-slot) הリンק נוכל, כמה זמן מישחו דבר בו.

**Goodput** – בכמה זמן הリンק נצל בהצלחה. נחשב אותו ע"י חלוקה של הזמן לשילוח מידע בזמן הכלול (שליחת מידע + זמן שבזובץ).

$$\text{Goodput} = \frac{\text{time taken to send data}}{\text{time taken to send data} + \text{overhead}}$$

(שווין יהיה באשר כל פעם שימושו שידר הוא הצליח).

מלהוגמה הקודמת: ניסינו לשדר ב-6 slots והצלחנו לשדר ב-3.

$$\text{throughput} = \frac{6}{9} = \frac{2}{3} \quad \text{goodput} = \frac{3}{9} = \frac{1}{3}$$

#### Slotted ALOHA goodput

ישנם  $N$  קודקודים שחולקים את ה-channels broadcast domain ומשתמשים ב프וטוקול שהגדנו.

בכל רגע נתון הקודקודים שולחים בהסתברות  $\alpha$  פקטה.

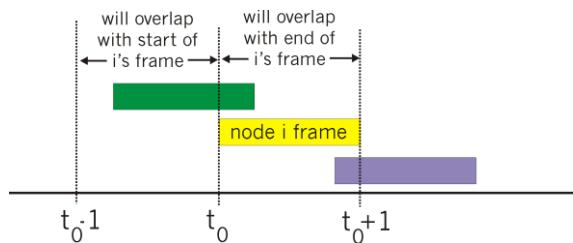
**הסתברות שב-slots מסוים קודקוד ספציפי יצליח** (הקודקוד שולח וכולם שומעים):  $p^{-(N-1)}(1-p)$

הסתברות שאיזשה קודקוד צליח היא  $(1 - p)^{N-1} \cdot p \cdot N$  והיינו רצים למצוא  $k$  שמקסם את ההסתברות הזאת. מהגדירה של goodput – הזמן שהשדרה מוצלחת חלקי סה"כ הזמן, יוצא שהוא הסתברות slot מוצלח היא בעצם  $\text{goodput}$ .

אם  $N$  שואף לאינסוף נגלה שה- $k$  האופטימלי נתון ערך בו ה-goodput הוא  $\frac{1}{e}$ , בערך 37% יש הצלחה – בולגרו הפרוטוקול שלו לא מספיק ויעיל כמו שהיה רצוי.

### Pure (Unslotted) ALOHA

נסיר את ההנחהשה-slots מסונכרנים. אם קודקוד ירצה לשלוח פקטה התנגשות יכולה לקרות עם שני slots של קודקוד אחר. הסרת ההנחה פוגעת ביעילות של האלגוריתם. במקרה של התנגשות בפרוטוקול יותר זמן יהיה מבוזב.



### Pure ALOHA Goodput

הסתברות שקידוך ספציאלי יצליח תהיה ההסתברות שאותו קידוך שיידר ואך קידוך אחר לא שיידר באותו הזמן שלפניו ואחריו. בולגרו אם אותו קידוך שיידר בזמן  $t_0$ , אף קידוך אחר לא ישדר בזמן של  $[t_0 - 1, t_0 + 1]$ .

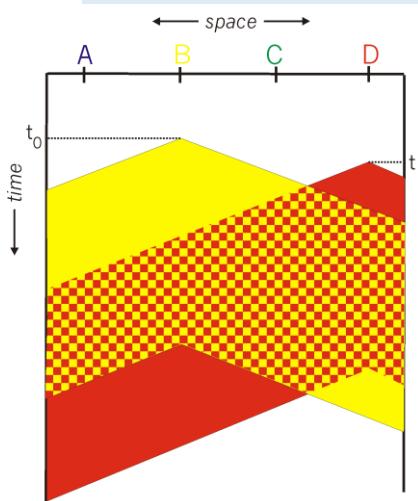
$$P(\text{success by given node}) = P(\text{node transmits}) \cdot P(\text{no other node transmits in } [t_0 - 1, t_0 + 1]) =$$

$$P(\text{node transmits}) \cdot P(\text{no other node transmits in } [t_0 - 1, t_0]) \cdot P(\text{no other node transmits in } [t_0, t_0 + 1]) =$$

$$p \cdot (1 - p)^{N-1} \cdot (1 - p)^{N-1} = p \cdot (1 - p)^{2(N-1)}$$

בחירה שוב  $k$  שיביא לביטוי אופטימי כאשר  $k$  שואף לאינסוף ונקבל שה-goodput המירבי הוא  $\frac{1}{2e}$ , בערך 18%, פחות אפילו מאשר slotted ALOHA (עדין יותר יעיל מ-).

### CSMA (Carrier Sense Multiple Access)



**CSMA** – הקשبة לפני השידור. לפני קידוך מנסה לשדר פקטות הוא צריך לוודא שאין קידוך אחר שරצה לשדר. במודל זהה נרצה להניח שההתנגשויות נמנעות (למעט מקרה נדיר בו הגורמים מסונכרנים לחולטיון).

נסתכל על הדוגמה הבאה: ציר  $x$ - $y$  מתרח布 למרחב של גורמים שרוצים לשדר. ציר  $z$ - $u$  מתרח布 בזמןן שעובר בשידור.

התחיל לשדר בזמן  $t_0$  ו- $D$  בזמן  $t_1$ . עד ש- $D$  מתחילה לשדר השידור של  $B$  עדין לא מגיע אליו, אך גם אם  $D$  היה מקשיב ושותק לזמן מסוים (עד להתנגשות, האזר בתמונה בו האדום והצהוב חופפים) הוא לא היה מזהה את ההתנגשות.

יש בעיה נוספת – זמן הפעוף של ההודעה.

לא יוכל להניח שאחננו מזהים ההתנגשויות – האם עדין יוכל לעשות את הפרוטוקול? כן, לפי הפרוטוקול קידוך נדרש להקשיב לפני שהוא משדר. אבל גם קידוך דיה התנגשויות בפרוטוקול הוא ממשיך לדבר

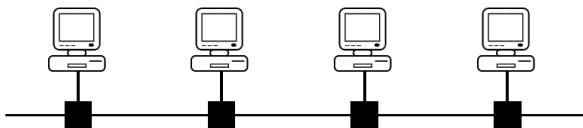
והתנגשות לא נפתרת. נצפה ל프וטוקול בו אם נניח שאנו יכולים לזהות התנגשויות (ההנחה מתקפה לקוים חוטיים) אז נפסיק לדבר.

## CSMA/CD

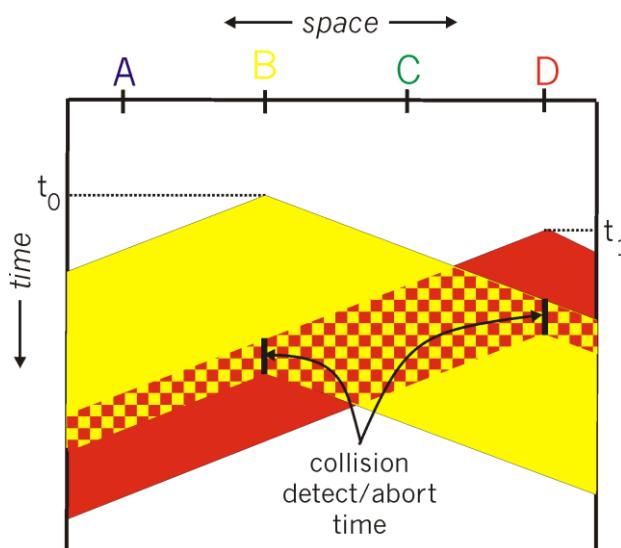
וננה על הדרישות ממקודם. בפרוטוקול זה, באשר קודקוד רצה לשדר:

- הקודקוד ממחכה עד זמן מסוים לפני שהוא מתחילה לשדר כדי לוודא שהקו פנוי ושללא יהיו התנגשויות (**carrier sense**).
- תוך כדי השידור הוא ינסה לזהות התנגשויות. אם אין, ישדר. אם קיימת התנגשות, יפסיק לשדר וימתן למשך פרק זמן מסוים שנקבע רנדומלית ויחזר לשלב הקודום (**collision detection**).

ריצה לחוכות זמן רנדומלי כדי למנוע התנגשות נוספת בין שני הקודקודות שהתנגשו.



בצד שמאל, בדוגמה הקודמת ברגע שהייתה התנגשות בין *B* ל-*D*, *D* היה מזהה אותה ומפסיק לשדר בזמן מה. הגרף היה נראה 다음과 הבא:



כמה זמן צריך לשדר כדי להבין בזמן השידור שיש בעיה?

ריצה לדעת שהפקטה ששודרה עברה בהצלחה, לזהות התנגשויות תוקן **תוקן שידור הפקטה** כדי להבטיח שידור ללא התנגשויות.

נסתכל על הדוגמה הבאה: **Propagation** (מסומן כ-*PROP*) – פרק הזמן שלוקח *A* ו-*B*, שני hosts שנמצאים רחוק אחד מהשני (מרחק מקסימלי בראשת). **Collision** (מסומן כ-*COLL*) – פרק הזמן שלוקח *A* ל-*B* (מיידע להגעה מ-*A* ל-*B*).

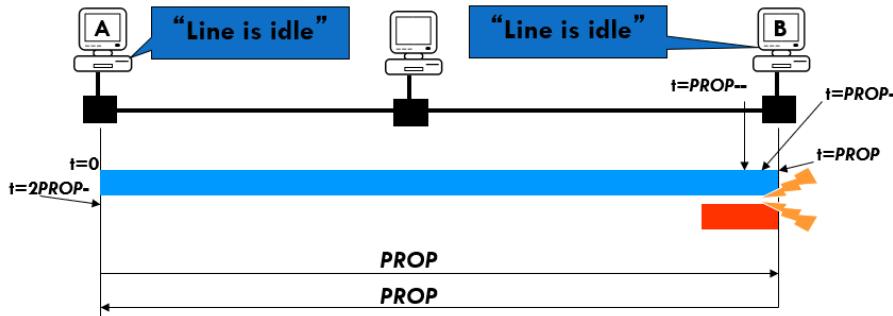
← בזמן  $t = 0$  קודקוד *A* מתחילה להעביר פקטה אחרי שהקשיב.

← רגע לפני שהבית הראשון של המידע מגיע ל-*B*, *B* מ Ksiיב להתקנשות (עד לזמן קרוב ל-*PROP* במעט) ומתחילה לשЛОח גם הוא פקטה בעצמו.

← ההתקנשות קורית ליד *B*.

← בזמן  $t = PROP$  קודקוד *B* מבין שיש התקנשות ועצר את שליחת המידע.

← ריצה גם מ-*A* והיה מסוגל לזהות את ההתקנשות - בזמן  $t = 2PROP$  גם *A* מקבל את הבית הראשון של>Data מ-*B*, מזהה את ההתקנשות ועצר את שליחת המידע. אם *A* היא משדר בפחות מפרק הזמן זהה, לא יהיה מקבל את המידע על ההתקנשות.



מהדוגמה ראיינו שכדי שקודקוד ידע ליהוות התנגשות לפני שסיים לשדר פקעה בהצלחה, נדרש שזמן העברה יהיה **TRANSIT > 2PROP**. לעומת זאת, חסם תחתון על הזמן שצריך לדבר כדי לדעת בזמן שליחת הפקעה ליהוות התנגשות. החסם התחתון הוא בעצם על גודל הפקעה כי קצב שליחתה תלוי בו, הפקעה צריכה להיות בגודל של **לפחות 2PROP**.

#### CSMA/CD Goodput

נכש לנתח את ה-goodput על שימוש של CSMA/CD עם ההנחות של ALOHA. נניח שוב שכל הפקעות באותו אורך, הזמן-slotted, כל-slots מסווגרים ובגודל 2PROP. בכל רגע נתון בהסתברות  $a$  קודקוד מנסה להיכנס אל תוך הרשות. נרצה למצוא את ה-goodput =  $\alpha_{max}$  (א) הסתברות שב- $a$ -slot אחד משדר ב-slots נתון (משתנה בינומי):

$$\begin{aligned}\alpha(p) &= \binom{N}{1} p(1-p)^{N-1} \\ \frac{d\alpha}{dp} &= N(1-p)^{N-1} - pN(N-1)(1-p)^{N-2}\end{aligned}$$

נקבל מכך שעבור ההסתברות  $\alpha_{max} = \frac{1}{N}$  קיבל את ה-goodput המקסימלי, שזה בערך 37%.

נסתכל על כך באופן הבא - נגיד אר את  $A$  להיות מספר-slots שבזבוזו לפני שפקעה שודרה באופן מוצלח. בהסתברות  $\alpha$  (הסתברות שב-slot זהה מישוה הצלח לשולוח פקעה) בזבוזו 0-slots ( $A = 0$ ) בהסתברות  $\alpha - 1$  בזבוז 1. נפתרו את נוסחת הנסיגת לחישוב  $A$ :

$$A = (\alpha \times 0) + (1 - \alpha)(1 + A)$$

ונקבל שעבור  $\alpha_{max} = \alpha$  בזבוזו  $A = 1.5$ .

באוטו אוף אפשר להטיל מטבע כך שהסתברות ל-head, להצלחה, היא 0.4. מהתכונות של משתנה גיאומטרי, תוחלת הריקות עד שיפול head, עד שנצליח, לראשונה היא 1.5 - 2.5 כישלונות (כמה-slots בזבוז בשיחה מוצלחת) וניצחון אחד.

מבחינת חישוב ה-goodput, בשרצה לחשב את הזמן שבזבוז (מספר-slots שבזבוזו  $A$  כפול גודל הפקעה). בדוגמא:

$$\begin{aligned}\eta_{CSMA/CD}(\text{Goodput}) &= \frac{\text{time taken to send data}}{\text{time taken to send data} + \text{overhead}} = \frac{\text{TRANSP}}{\text{TRANSP} + \mathbb{E}[\#\text{of wasted slots per packets}]} \\ &= \frac{\text{TRANSP}}{\text{TRANSP} + A(2 \cdot \text{PROP})} = \frac{\text{TRANSP}}{\text{TRANSP} + 3 \cdot \text{PROP}} \stackrel{(1)}{=} \frac{1}{1 + \frac{3\text{PROP}}{\text{TRANSP}}} \stackrel{(2)}{=} \frac{1}{1 + 3a}\end{aligned}$$

(1) נחלק את המונח ואת המכנה ב- $TRANSP$

$$(2) \text{ נסמן } a = \frac{\text{PROP}}{\text{TRANSP}} < \frac{1}{2}$$

ונשים לב שככל שנקטין את  $a$  בכלה נוכל לקרב את ה-goodput ל-1, וכך גם לאר את הלינק בצורה יותר טובה.

ההקטינה של  $a$  יכולה להיות ע"י הקטנת הרשות, כך שברשת שהיא קטנה זמן הפעוע קטן, או ע"י הגדלת הפקעות ועוד זמן השידור גדול. הדרך השנייה בעייתית כי היא יכולה לגרום להרעה של קודוקדים, אך יש הגבלה על גודל הרשותות האפשריות.

## Lecture 4

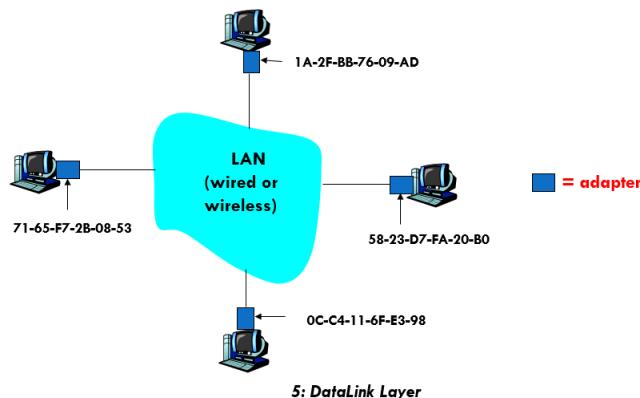
### Single (Logical) Link - Practice

#### בתובות MAC

בסוף של דבר, אף פרוטוקול מהפרוטוקולים שדיברנו עליהם לא ממושך במציאות. הסברנו עליהם ורק כדי לפשט את הבעיה שشبכת הילינק פותרת את הנזוחים ההסתברותיים הדורשים לשם כך. אחד הדברים הקרייטיים בכל רמה הוא הכתובות בה מורות את הקודקודים שמדוברים, את ה-host end. בתוך שכבת הילינק הכתובות הזרת נקראת **MAC address**.

הכתובת היא 48 ביטים אבל הם צחבים לתוך ה-ROM (Network Interface Card/ Adapter) NIC.

- אם לשניים יש MAC address זה יכול להיות שמדובר באותו יצן או באותו תקופה.



- סה"כ יש  $2^{48}$  כתובות שונות, בלומר NIC שונים.

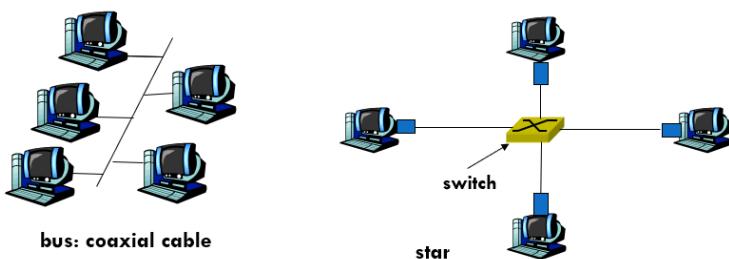
משתמשי קצה שונים חולקים (LAN) Local Area Network (LAN) קבוצה זהה (רשת מקומית, כמו בשרטוט). מדובר על כך שהמשך אבל רשות שדיברנו עליה בינהיים היא broadcast domain שהוא סוג של LAN. לאותם משתמשים יש NIC ולו MAC address זהה.

- גוף בשם IEEE מחלק את הכתובות.

#### רישות broadcast domain

ב-hub broadcast domain מחוברים ע"י לינק, כך שקוודוד אחד מעביר והודעת broadcast domain לשאר וכולם שומעים.

שכבה 2 היא לא יכולה broadcast domain אחד גדול, היא מורכבת מכמה באלה וצריך לרשות אותם. דיברנו על כך שרשויות יכול להתבצע ע"י טופולוגיה bus שכל משתמשי הקצה השתמשו בו (לא נפוץ ברגע), אבל ביום הרישות מתבצע בצורה "כוכב": יש switch במרכז שמחובר לשירות כל המשתמשים או אל switches אחרים והוא עשו את ה-switch בשכבה 2 ובעצם מרשת את ה-main broadcast domain ומונע התנגשויות. בטופולוגיה כוכב-hub הוא בין שני משתמשים שאחד מהם יהיה ה-switch ובדרך זו נוצרות רשתות גדולות יותר מאשר broadcast domain אחד.



## Ethernet

הפרוטוקול העיקרי הפעיל ב-hub domain называется .broadcast domain

בניגוד להרבה פרוטוקולים שתוכננו פעם לא מותאים לדרישות האינטרנט. בימינו, ה-Ethernet הצליח להיות מותאם עד לשימושים היום.

#### Ethernet CSMA/CD algorithm

הפרוטוקול קווי ולכן נובל להניח שאנחנו יודעים לפחות התנגשויות. זמן ביטים – מוגדר בפרוטוקול בשילה של 10 Mb/s ב-0.1 מיקרו שנייה.

שליחת מידע דרך הפרוטוקול:

1. ה-NIC מקבל מידע מהאפליקציה וועטף אותו בפקטה – מוסף לו למפלטת MAC, ובכללי מידע שהשכבה צריכה כדי לדעת לדבר.

- .2 הוא מחקיב קצר (CSMA) ואם אף אחד לא מדבר – מדבר.
- .3 אם לא זיהתה התנגשות, הפקטה נשלחת בהצלחה.
- .4 אם זיהתה התנגשות, פורשים ומפסיקים לשולח את הפקטה. אבל בנוסף – **שולחים Jam signal** (מוגדר ב-48 זמן ביטים) היסיגナル מיידע את הקודקודים ברשת לכך שהיתה התנגשות ע"י שליחת אוסף אחר של ביתים.

**Exponential Backoff:**

- **Goal:** adapt retransmission attempts to estimated current load
    - heavy load: random wait will be longer
  - first collision: choose K from {0,1}; delay is  $K \cdot 512$  bit transmission times
  - after second collision: choose K from {0,1,2,3}...
  - after ten collisions, choose K from {0,1,2,3,4,...,1023}
- Exponential backoff** 5. בהסתברות קבועה אלא במספר ההתנגשויותשה-NIC נתקל בהן בעת הניסיון לשידור הפקטה –  $m$  התנגשויות. לאחר ההתנגשויות, המשמש ייקבע רנדומלית  $K$  מטור  $\{1, 2, \dots, 2^m\}$ , וימתון  $K \cdot 512$  זמן ביטים. ככל שהעומס ברשת יותר גדול, כך "מרוחים" את השולחה של NIC (זמן ההמתנה קופץ אקספוננציאלית) וימנעו התנגשויות. אם העומס נמוך, הוא לא יוכנה הרבה זמן ובכך לא יפגע ביעילות. בחירת זמן ההמתנה היא רנדומלית כי במרקבה בו-NIC היה פשוט ממתי בזמן קבוע התנגשות הייתה ממשיכה ל��ות.

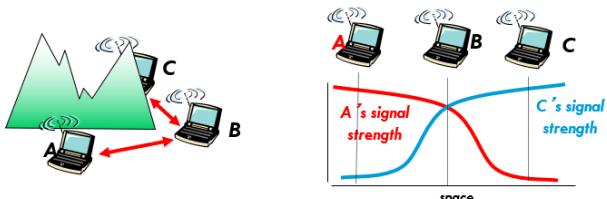
- $t_{prop}$  = max prop delay between 2 nodes in LAN
- $t_{trans}$  = time to transmit max-size frame

$$goodput \approx \frac{1}{1 + 5t_{prop} / t_{trans}}$$

- Goodput goes to 1
  - as  $t_{prop}$  goes to 0
  - as  $t_{trans}$  goes to infinity

**Ethernet Goodput**

כיתוח ה-PROP Goodput של הпрוטוקול נותן תוצאות דומות לשאר הпрוטוקולים שראינו. בסופו של דבר, כל עוד משמרם יחס טוב בין ה-PROP (שואף ל-0) ל-TRANS (שואף לאינסוף) הערך של ה-Goodput יתקרב ל-1 ויהי הרובה יותר טוב מ-ALOHA ודומו. ה-PROP נפוץ על לינק בודד ואכן ניתן לצפות שה-PROP יהיה מאוד נמוך.

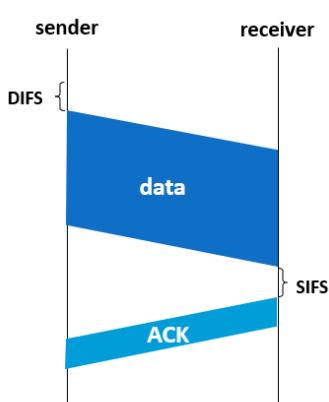
**Wireless and Mobile Networks IEEE 802.11**

ב广播 זה אין זיהוי התנגשויות, ולא יוכל למש את הпротוקולים שראינו קודם.

- CSMA: למרות שאין זיהוי התנגשויות עדין צריך להקשיב, ויש בпротокול שימוש בהקשיבה לפני דיבור.
- (AP) Access Point: קಡוק מוחד מוחד ברשת שמסביבן את השילוח. לחבר את הקודקודים לאינטרנט, יודע מה קורה ברשת (כפי אנחנו ב广播) ואחראי על הדיבור.

**CSMA/CA**

הפתרון לרשותות אלה הוא **collision avoidance**.



– ננסה להימנע מהתנגשויות מתוך הידעשה שלא ניתן לזהות אותן. כדי למש collision avoidance הרעיון הוא "לשמר את הארץ".

← קודוק שורצוה לשדר ישלח בהתאם RTS, פקטה קטנה, לקודוק המוחס AP access point לפי פרוטוקול CSMA.

ה-RTS זו בקשה לקבל את הארץ, כמה קודוקדים יכולים לשולח אותה ביחד ויכול להיות התנגשויות.

← הקודוק מוחכה לקבל תשובה מה-AP האם מותר לדבר – CTS clear to send. גם פקטה זו היא קטנה.

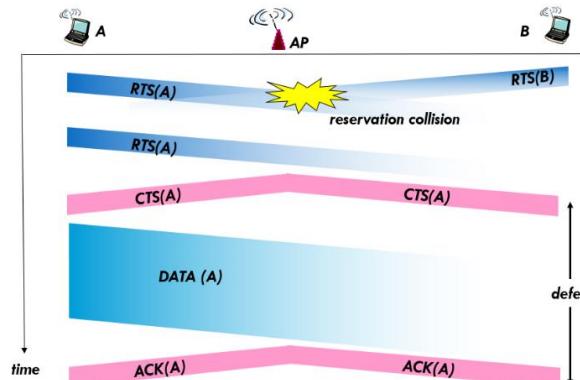
• מכיוון שה-RTS וה-CTS הן פקודות קטנות הסיבוכו להתנגשות הוא קטן.

:CSMA לימוש

← השולח ימתין DIFS זמן ולאחריו ישלח פקטה במידת הארץ פניו.

← אם הפקטה הגעה לעד, ישלח אליו סיג널 של ACK (acknowledgment).

← אם הערוצ תפוס, השולח יאותל טוימר backoff בתחילת רנדומלי ושלח בשיטתיים. אם הוא לא קיבל ACK על הפקטה ששלח, הוא יעלה את הטווח של ה-backoff. מביחסת הנמען, אם קיבל פקטה ישלח בחזרה ACK אחרי SIFS זמן שקיבל את הפקטה.



במהשנה משמאל - A ו-B חולקים אותו דומיין וביניהם יש את ה-AP.

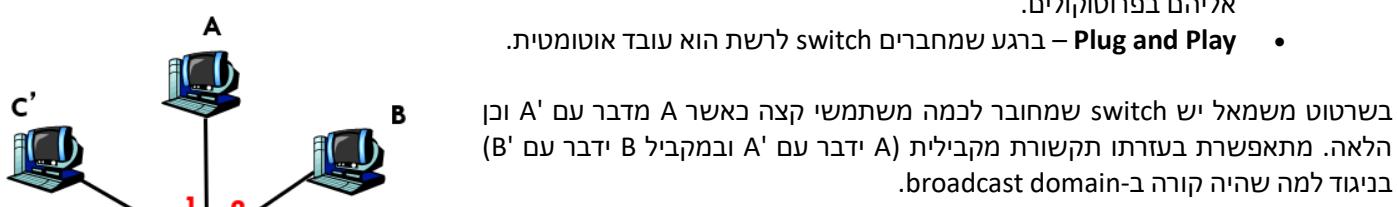
- A שולח RTS, וקצת אחריו גם B שולח – התנגשות.
- גם A וגם B אמורים לדעת על ההתנגשות כי לא קיבלו CTS.
- אחר כך A שלח שוב מידע ובגלל שלא נראה ש-B גם ניסה לשדר באותו הזמן, A קיבלCTS והצליח לשדר.
- גם קיבל את ה-CTS מ-AP, וידעו שעכשיו הוא לא יכול לשדר.
- המידע הגיע ל-B, וגם A קיבל מה-AP את ה-ACK, בלומן המידע עברה.

## LANs: Interconnecting Broadcast Domain

### Switch

רישות broadcast domain נועשה ע"י switch שבעצם עשו switching, מה שמאפשר לתקשורת להישמע ע"י מי שרצים שונריצה שישמעו, ה-*switch* מעביר פקודות לעיד הרצוי ביל' שבל מי שמחובר אליו ישמעו. כאשר מתקבלת פקטה (מסוג Ethernet), הוא מסתכל על בתובת ה-MAC של היעד ושולח לנמען הרצוי ומחליט אם לשלוח וככה בעצם שובר את ה-.broadcast domain

- משתמשי הקצה לא מודעים לקיום של סוויצ'ים בסביבה 2 ולא מתייחס אליהם בפרוטוקולים.
- Plug and Play – ברגע שמחברים switch לרשת הוא עובד אוטומטית.



בشرطוט משמאל יש switch שמחובר לבמה משתמשי קצה כאשר A מדבר עם 'A' וכן הלאה. מתאפשרת בערתו תקשורת מקבילתית (A ידבר עם 'A' ובמקביל B ידבר עם 'B') בנגדוד למה שהוא קורה ב-chain. broadcast domain

מהתרגול – **Hub**: לעומת switch, מה שעובר דרך ה-hub נשלח אל כל הLINKים שמחברים אליו ב-.broadcast domain.

**Switch Table**  
איך ה-switch יודע להעביר את המידע באופן נכון (מה ש-A שלח יכנס מנוקודה 1 וכיוצא לנוקודה 4 למשל)?

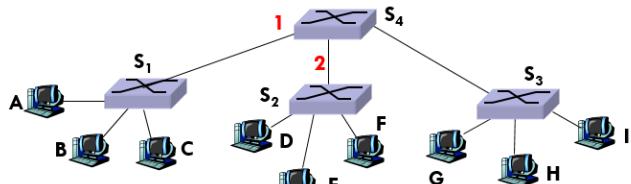
הטבלה ממחפה בין בתובת MAC של היעד לאינטראפיס דרכו השילוח נעשית. כל רשומה בטבלה מכילה את בתובת MAC, ה포רט דרכו צריך לשלוח אותה בתובת, ו-TTL שזה זמן התפוגה של אותה.

הswitch בעצם **לומד את הטבלה** דרך פקודות שנשלחו. ברגע שmagieha פקטה, ה-switch ישמור את בתובת MAC של השולח ימפה אותה לפורט דרכו הפקטה הגיעה. אם אין לו בטבלה את המידע על הנמען הוא ישלח לכלום – flooding, וברגע שימושה עוננה יהיה לו את המידע הנחוץ והוא ישמר אותו בטבלה.

### האלגוריתם של ה-switch

1. עבר פקטה שנשלחה, תזכור מאיפה היא הגיעה (אם אין רשומה בטבלה כבר).
2. אם כבר יש רשומה בטבלה, נדע לשלוח לאינטראפיס הנ龂ן.

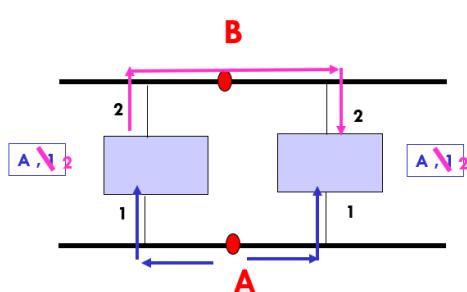
3. אם לא, ה-switch ישלח אותה לכל הכוונים בהנחה שתישלחו ידבר. באינטראנס אכן התקשרות לרוב היא זו כיוונית ולכן ההנחה זו היא סבירה.  
(המחשה לדוגמה בשקופית 8)



switch יכול להיות מחובר גם לSwitchים אחרים ולא רק למשתמשים קצה. במקרה של העברת בין switches האלגוריתם יפעל אותו דבר ועדיין יעבד נכון. רצחה להעביר פקטה ל-G. למשתמש נניח ש-A יرسل את הפקטה דרך S<sub>1</sub> שישמר בטבלה את ה-MAC של A והוא ישלח את הפקטה דרך S<sub>4</sub> שיגיע ל-G. הוא ידע לשולח אותה ואת הפורט דרכו שלח.

לאחר מכן S<sub>1</sub> ישלח ל-S<sub>4</sub> את הפקטה שישמר בטבלה את המידיע על A ובאשר התשובה תגיע ל-A דרך S<sub>4</sub> הוא ידע לשולח אותה דרך הפורט אליו מחובר S<sub>1</sub>. הפקטה תגיע ל-S<sub>1</sub> שישלח אותה ל-A.

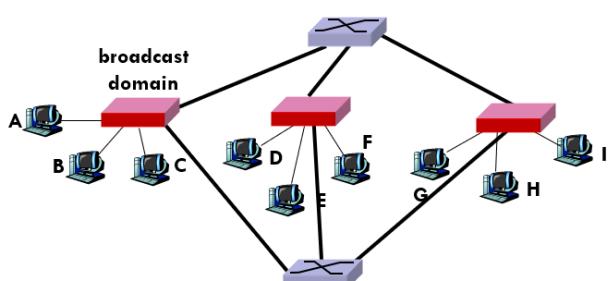
### Spanning Tree Protocol



נסתבל על A ו-B, שני משתמשי קצה שנמצאים על broadcast domain של A ו-B. הריבועים הם switches שמחברים ביןbroadcast domain של A ו-B. דרך אינטראנס 1 ומשני דרך 2. A רוצחה לדבר. הסוציאים לומדים שהאינטראנס של A הוא 1. בגלל שזו השיטה הראשונה ברשות, שני הסוציאים חיפשו את הפורט של B, יビאו לא-flooding. שני הסוציאים יקבלו את אותה הודעה אבל כל אחד מהשני, כך ששני הסוציאים "ילמדו" שהאינטראנס של A הוא 2. אבל מכיוון שהם עדין לא Learned עם מי A דבר, ימשיכו לעשות flooding מה שבוא לולאה.

ביבול הרשת האידיאלית תהיה בצורה עז וככה הלמידה תמיד תעבור נכון, אז למה שלא כל הרשות ברשתה בשבבה 2 יהיה עז?

1. אם לינק אחד נפל, יש שני חלקים ברשת שלא יכולים לדבר.
2. צואור בקבוק מעצם העובדה שניתן לשדר רק דרך מסלול אחד.



הפתרון – בכל רגע נתון נשלח דרך עז פורש של ה-switch, broadcast domain, למחרות שהרשת לא תהיה עז. ה-switch ילמד מאייה אינטראנס להתעלם כדי לקבל עז. אם קישור נפל, ה-switch יחשב עז פורש אחר. Broadcast שהוא לא חלק מהעז יונטרול ולא ישלווח דרכו הוודאות.

### Spanning Tree Protocol

הפרוטוקול אחראי על כך שלא יהיו מעגלים של Broadcast Domains (מעגלים של סוציאים זה בסדר).

נתאר את האלגוריתם בשלושה שלבים:

#### 1. מציאת שורש לעז הפורש –

- בכל switch יש מספר מזהה.
- בל אחד זוכר בכל רגע נתון את המספר הכי נמוך שראה עד עכשיו (כולל את עצמו) בתור השורש.
- ככל זמן switch שולח את המספר הכי נמוך שראה לשכנים (flooding).
- אם ראה מספר נמוך יותר, יעדכן אותו להיות המספר הכי נמוך שראה.

בסוף נבחר את השורש להיות המספר הכי נמוך שנראה. בעיה שיבולה להתקיים: ה-switchים המינימלי שמשמש בשורש יוצא מהמערכת. עדין בכל ה-switch האחרים מצביעים עליו.

2. חישוב העז הפורש מה-switches – נזכר מה המסלול דרכו המרחק הכט קצר לשורש (Bellman Ford). בכל רגע נתון כל switch מעדכן את השכנים מה המרחק שלו מהשורש.

3. דרך העז הפורש של הסוציאים בניית העז הפורש של broadcast domains.

## Lecture 5

### LANs – Spanning Tree Protocol

בשיעור הנקוט למדנו על רישות של LAN מלינק בודד, ודיברנו על אלגוריתם הלימוד של switches שמצירק הימנעות מלולאות כדי למנוע דרישת וטינה לא נכונה. כדי למנוע את הבעה אפשר היה לרשת בצורה עצ, הבעה היא שnitok שלリンク יביא לפיצוק חלקים מהרשת. הפתרון הוא שהswitches דרכם התקשרות מתבצעת יהו בצורה עצ, פורש, ורק יש שלושה שלביים כאשר את השלב הראשון – מציאת קדקוד השורש, תיארנו בהרצאה הקודמת. אנחנו מניחים לשם הפשטות שה-STP הוא בשלבים, אבל במצבים הוא לא.

#### (שלב 2) אלגוריתם בלמן-פודד מבוזר

נרצה לחשב עצ פורש מינימלי.

**הנחה:** נתנו לנו קדקוד השורש  $s$  וכרגע הוא קבוע.

**היעיון:** מה שירכיב את העץ יהיה המסלול הקצר ביותר של כל קדקוד אל השורש.

באלגוריתם, ככל וגע נתנו כל קדקוד ישמר את המרחק שלו מהשורש וידוח על בר לשכנים שלו, כאשר:

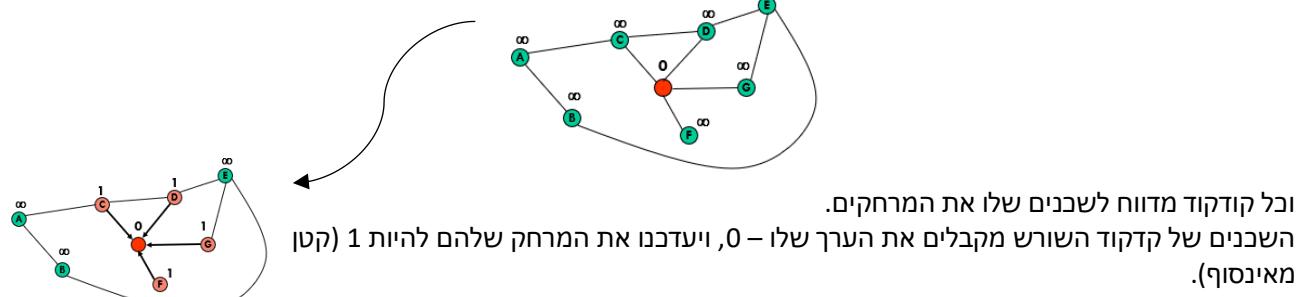
$$dist_s = 0$$

$$dist_v = \min\{dist_u + 1 \mid u \text{ is a neighbor of } v\}$$

- כל קדקוד שהוא לא השורש מאותחל למרחק אינסוף.

ההוראה של הקודקוד יהיה הקודקוד עם המרחק הכי קצר ממנו.

**המחשה:** המצב ההתחלתי הוא כמוتوا:



נסיר בטהיליך גם השכנים של שבני הקודקוד יתעדכנו, וכן הלאה. בסוף נקבל עצ פורש מינימלי.

- כל עוד המשקלים לא שליליים האלגוריתם פועל נכונה בצורה שתוארה קודם. משקלים שליליים יגרמו לולאות.

- גם אם האלגוריתם לא מסונכרן הוא יעבד. בסופו של דבר הקודקודים יקבלו את הערכים שהם אמרורים לקבל.

- האלגוריתם יעבד גם אם המרחקים הראשוניים יאותחלו אחרת כי בסוף העדכון תלוי בשכנים.

#### (שלב 3) עצ פורש של LAN segments

.בסוף דבר מהעץ הפורש של switches שיחסבנו מקודם נבנה עצ פורש של switches. **הנחה:** נתנו לנו עצ פורש של switches.

**המטרה:** main broadcast domain יהיה מחובר לרשת דרך switch אחד ורק דרכו ישלח, ויעדי "בחירה" את switch עם המרחק הכי קטן מהשורש.

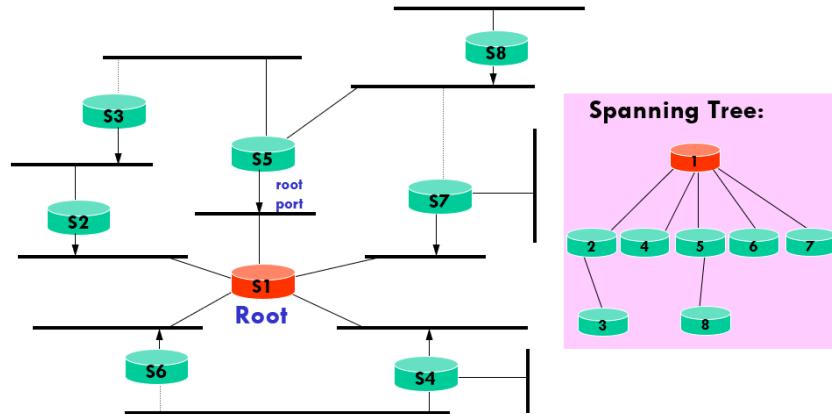
**היעיון:** ה-*broadcast domain* "בחירה" את switch האופטימלי, אבל בעצם switches בוחרים בשביילו. כל switch מחשיב ומחשב עבור ה-*bd* אם הוא ה-*switch* המתאים. אם כן, יעביר את המידע דרכו.

#### קונספטים מרכזיים בעץ פורש

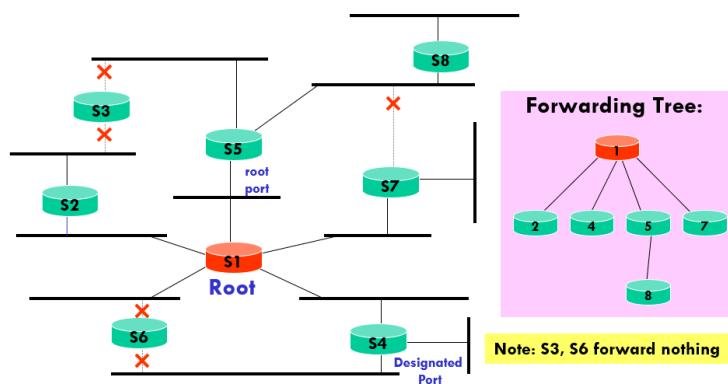
**אורך מסלול Path Cost** – איזשהו מחיר שמחושב לכל port, לכל יציאה של switch. הערכים הדיפולטיבים הם 1. אפשר להשתמש בו כדי לשנות את המסלולים הרצויים.

- לבל switch מלבד השורש יש port שפונה לשורש, דרכו הוא לומד את המסלול הביא קוצר (חושב בשלב 2). דרך ה-port עבר המסלול הביא קוצר לשורש, הוא בעצם ההתחלת של המסלול הביא קוצר. אם יש בינה, ה-port root יהיה זה עם ה-ID הבי נמוך.

במהышה הבאה, נבחר את 1 להיות השורש ונקבל את העץ הפורש:



של ה-chain broadcast domain – של ה-chain broadcast domain **Designated Port** יהיה הפורט עברו אותו switch. בעל ה-cost path המינימלי. מודגמה, העץ הבא יהיה זה שדרכו התעבורה באמת עוברת. נעדיף להעביר דרך 5 כי הוא במרחק 1 ו-3 מרחק 2, 3 "וודע" שהוא לא ה-port עברו אותו broadcast. כמו גם לגבי ה-port broadcast למטה, וכך לא ישדר בכלל.



- רק פורטים שהם root port או designated port יהיו פעילים.

#### מימוש האלגוריתם

**דרישות:**

- בכל switch יש ID ייחודי.
- בכל הפורטים של ה-switch יש זהה ייחודי: switch ID + port number.

**BPDUs: Bridge Protocol Data Unit** - כל הזמן כל switch שולח לשכנים הודעה הבאה - הכלולת את המידע הדרוש לפוטוקולום:

(my root ID, my cost to root, my ID)

טור כדי מתרחשים העדכנים של השורש וה-cost:

- My\_root\_ID: smallest seen so far
- My\_cost\_to\_root: smallest received to my\_root + link cost

- רק פורטים שהם root port או designated port יהיו פעילים, כל השאר חסומים ולא יעבירו מידע.  
- אם למשהו אין root port designated אז גם root port חסום כי לא מעביר מידע, אבל עדין הוא ימשיך להקשיב כל הזמן.

\* מהышה בעמ' 42 במצגת

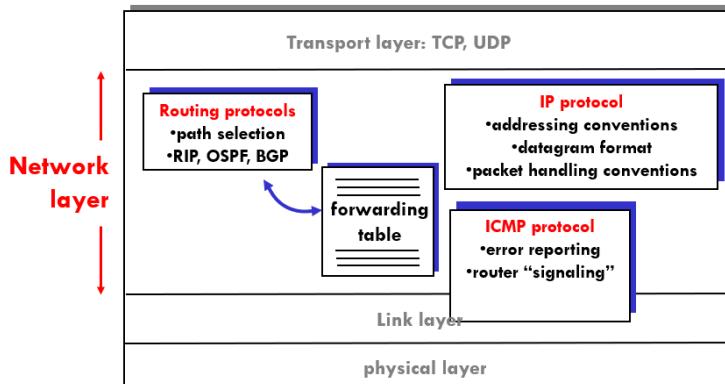
## IP Networks: Interconnecting LANs

אחריו שירשתנו broadcast domains ייחדים לרשתות LANs באמצעות switches, נרשת בעת בשכבה 3 – Network/IP layer broadcast domains, switches, נרשת בעת בשכבה 3 – Network/IP layer broadcast domains, switches, רשתות של שכבה 2, כך שנוכל להעביר הודעות דרך רשתות שונות.

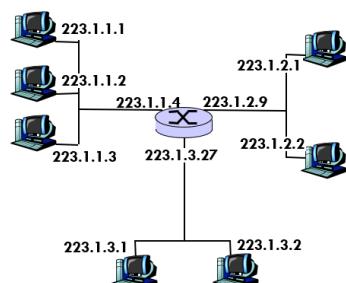
### למה האינטראנס הוא לא LAN אחד גדול?

- לא יוכל לתפקד עם כתובות MAC בלבד. יש הרבה כתובות MAC וככל נתב היה צריך לשמר במנות גדולות של מידע.
- היררכיה תהיה לא ידועה ולא יהיה לנו מידע על מיקום פיזי של קודקוד ברשת.
- עוז פורש על כל האינטראנס יהיה לא יעיל.

**מבנה שכבת האינטראנס**  
הקודקודים הם נットבים Routers, שמחברים בין ה-LANs.



## בתובות IP



בתובת IP היא רצף של 32 ביטים, המחלקה ארבע קבוצות שבכל קבוצה שמונה ביטים. בחלק זה יוצג בינאריו של איזשהו מספר, וכל מספר מופדק ע"י נקודה.

Interface: חיבור בין host או נתב לבין-link פיזי, שיש בתובת IP ייחידה לכל interface. לנットב יש מספר interfaces (יכול להתחבר למספר רשתות) ולכל host יש אחד.

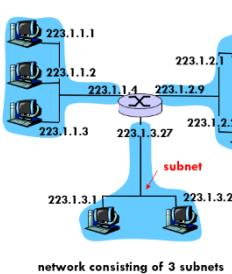
$223.1.1.1 = 11011111 \underline{00000001} \underline{00000001} \underline{00000001}$   
 223      1      1      1

### CIDR: Classless InterDomain Routing

ניתן לקבץ בתובות IP ולהתייחס אליהן כאל בלוק יחיד, באופן הבא: חילקו מוקדם את כתובות ארבעה חלקים, שלושה מהם ימשו לזהות הבלוק - subnet part (החלק הכהול), הרשות אליה הכתובה שייכת. החלק הנותר מהו איבר בבלוק שמצויה את host part ונקבע שירויית עם 32 פוחות מספר הביטים ב-subnet.  
 • 23/23 מציין בכתובה את מספר הביטים שב-subnet.  
 • כל הכתובה בעלות אותו subnet י��בו ביחד.

← subnet part → host part →  
 $11001000 \underline{00010111} \underline{00010000} \underline{00000000}$   
 200.23.16.0/23

### Subnet

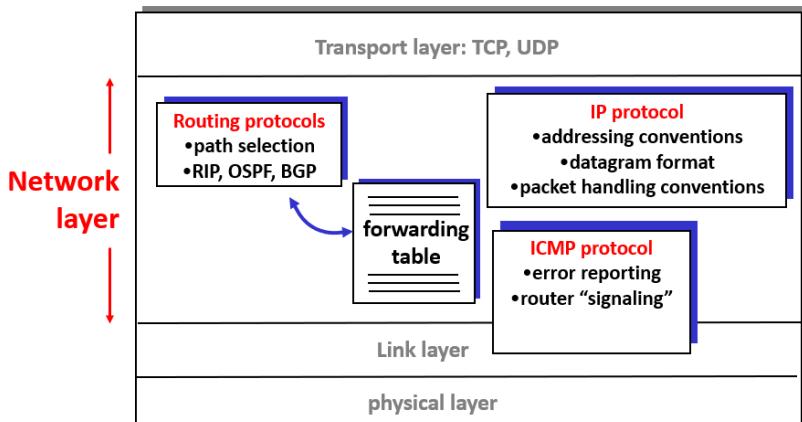


נתב מחבר מספר LANs, והוא בעצם שובר את הרשות ל-LANs. רצחה שהמספרים ב-LANs יאפשרו לדבר כמו קבוצה, ובכתובה ה-IP בה יהיה באיזשהו רצף. זה מאפשר באמצעות subnet – תת רשת, קבוצה של מספר קודקודים שחלק ה-subnet בכתובה ה-IP שלהם זהה. כדי לקבוע את subnet, נפריד כל המנתב כך שיוציאו קבוצות של רשתות שונות. לכל קבוצה יהיה זהה.

## Lecture 6

בתובות IP

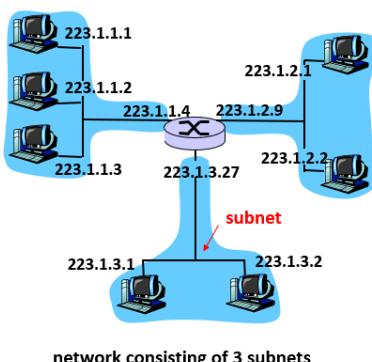
شبב ה-LAN היא מעל שבב ה-LAN ומקשר בין שבבות LAN שונות, ובולת שלושה דברים:



1. **פרוטוקול ה-IP:**
2. **פרוטוקול ה-Routing:** איך יוצרים את טבלת ה-forwarding (נגזר בהמשך)? איך יוצרים את המסלולים משכבות ה-LAN השונות.
3. **פרוטוקול ה-ICMP:** פרוטוקולי בקרה. הממשק בין ה프וטוקולים האלה נקרא **forwarding table**. פרוטוקול ה-IP מקבל את הטבלה בקלט שם בתובות הפקודות לשילוח פקודות. פרוטוקולי ה-routing וה-ICMP אחראים על עדכון הטבלה.

### זיכרון מייעור שעבר:

בתובות IP מורכבת מ-32 ביטים, seh"כ יש  $2^{32} = 4$  מיליאוד בתובות IP, וכל ברטיס רשות בתובות IP משלו. הרואוטר יש כמה פורטים וכל פורט תהיה בתובות IP משלו, וכי שרהואוטר לא יצטרך לזכור כל בתובות IP לכל אינטראפיס, לתחי רשותות (subnets) יש תחילת בתובות IP מסווגת כך שהרואוטר ידע לאיזה בינוין את המידע המיועד ובעצם געשה בר ארגזיה (והרואוטר יזכור רק את הבתוות של תתי הרשותות). למשל בדוגמה, ברגע שנקלב בתובות שמתחליה ב-223.1.3 הרואוטר ידע לשלוח את המידע לביוון מטה.

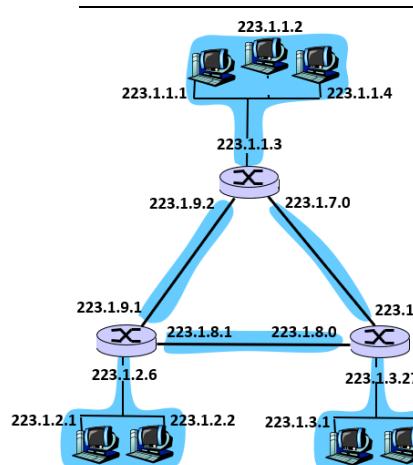


בתובות IP מוגדרות ע"י **CIDR**: הכתובת מחולקת לשני חלקים כאשר החלק השמאלי (בכלול) הוא הכתובת של תת הרשת (**subnet part**). החלק הימני (בשחור) מייצג לאיזה **host** בתת הרשת הכתובת שייכת. בתובות IP עצמה, המספר שאחרי ה-/ (host part) יציג בינה ביטים בכתובת מייצגים את תת הרשת, השאר מייצגים את ה-.host.



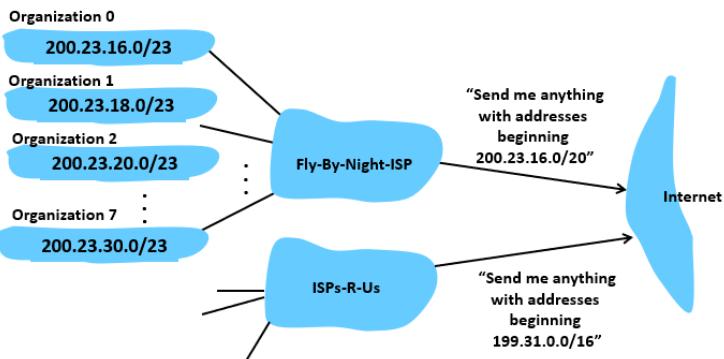
אם נסתכל על תת הרשת התחתונה, הכתובת שמייצגת את subnet היא 223.1.3, seh"כ 24 ביטים למשל. במקרה זה נותרו 8 ביטים (32-24) לייצוג ה-host, ולכן seh"כ יוכל לחבר 2<sup>8</sup> מחשבים לתשת.

### תתי רשותות (subnets)



איך נדע מהו subnet? ננתק את כל הרואוטרים ברשת ונישאר עם גרפ בעל כמה רכיבי קשרות מבודדים (LINKS), כמו מר כמה מחשבים/hosts שיוכלו לתקשר אחד עם השני ללא רואוטר. כל רכיב קשרות יהיה תת רשת, כי מחשבים/hosts באוטה subnet לא צריכים רואוטר כדי לתקשר אחד עם השני. עולות השאלות איך הרואוטר יודע מהו subnet mask? ואיך הקודקוד יודע לאיזה subnet שייך לו?

**דוגמא:** כמה תתי רשותות יש ברשת משMAIL? משתמש במתכון שלנו וננתק את כל הרואוטרים. נשים לב שבכל LINK שמחבר בין שני רואוטרים הוא רשות בפני עצמה. seh"כ קיבלנו שישה רכיבי קשרות – 6 תתי רשותות.

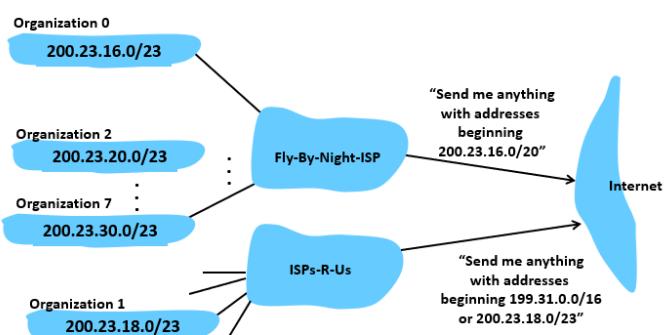


נניח שՁווחה רואוטר באינטראנט רוצה לשלוח מידע לארגון תחת רשות, מספיק לו לדעת שכל הקודקודים שהם תחת אותו-Fly-By-Night ISP, נמצאים תחת הכתובותשה-20 ביטים הראשונים שלן הם 200.23.16, ולכן ישלח הכתובות שמתחלות ב-20.23.16.0/20 לאותר בכתובת זו שייעד לנוכח את המידע לארגון המתאים. ה-ISP יזהה אם המידע שמגיע אליו מודיע להישלח אל כתובת פנימית שלו או מחוץ. אם ארגון בתוך ה-ISP ירצה לשלוח מידע לאוטר ISP, ה-ISP יctrיך להזיהה שהשליחה היא לכתובת מחוץ לו ולנתב את המידע החוצה. אם אותו ארגון ירצה לשלוח לארגון אחר, המידע יגיע ל-ISP שיזהה שמדובר בכתובת פנימית ושלח אותו לארגון הרלוונטי.

ראוטר שומר מידע על subnets בהיררכיה הći קרובה אליו ולא על hosts, וזה הכוון של הכתובות IP לעומת הכתובות MAC שם נדרש לשמור את המידע על כל הרשות.

### Hierarchical Addressing

חלוקת הכתובות נעשית בצורה היררכית, כך שכל רואוטר אחראי לניטוב המיידע לכתובות בתתי הרשותות שמתחלתו. לספק אינטרנט, ISP (למשל בדק), ינתן מרחב בתיבות גודל מטור  $2^{23}$  בתיבות האינטראנט, והוא מחלק קבוצות של כתובות לארגונים שימושיים בו. בדוגמה – ISP Fly-By-Night מקבל את כל הכתובות שמתחלות ב-200.23.16.0/23. כל ארגון שמקבל אינטרנט דרך אותוISP ייקח ממנו גם תת מרחב כתובות. (למשל, ארגון 0 יקבל את הכתובות (200.23.16.0/23



לחוב הגרפף לא יהיה עץ מוחלט – ארגון מסוים יכול לעבור מ-ISP אחד לאחר אבל להחזיק באותו IP שמתאימה לספק הקודם. האינטראנט יctrיך לדעת לא העביר את הכתובות של אותו ארגון ל-ISP ממנו עבר וכן לשלוח את הכתובות ל-ISP אליו עבר. בדוגמה, ארגון 1 עבר מ-Fly-By-Night-R, והאינטראנט יctrיך לדעת שכאשר הוא שולח אותה בתובות הוא עבר אותו דרך R-Us (Fly-By-Night-R) ישלח לו את כל מרחב הכתובות שלו + הכתובת החדשה). כיצד? – **Longest prefix match** – ברגע שנקלבל כתובות שיכולה להתאים לשני ISP, האינטראנט יעביר את הכתובות לכל היותר ספציפי (מתאים יותר ביטים ב-prefix של הכתובת). ככל שהרישא יותר ארוכה כך הרשת יותר קטנה ויתור ספציפית.

בדוגמה, בשנרצאה לשלוח לכתובת שמתחליה ב-200.23.18.0/23, הכתובת מתאימה באופן כללי לבליים של שני הספקים. אבל הספק השני מתאים יותר ביטים בכתובת (23 ביטים מתאימים לרשא של הכתובת לעומת 16 אצל Fly-By-Night ISP), לכן יהיה יותר ספציפי והכתובת תישלח אליו.

### כיצד ISP מקבל מרחב כתובות?

בעבר גוף בשם ICANN אחראי על חלוקת הכתובות וממן שמורות. היום כל הכתובות מוחולקות וכיום לקבות הכתובת חדשה צריך לקנות מגוף פרטי כי כל הכתובות נגמרו.

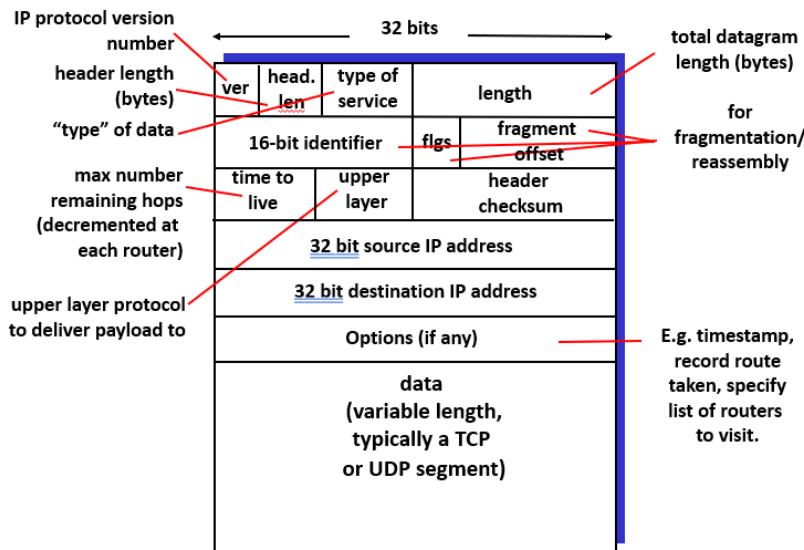
### אין רשות תקשורת subnet של כתובות IP?

דבר על כך בהמשך ההרצאה.

### IP datagram format

1. **IP Protocol Version Number** – גודל header-המוצג כל הביטים. חשוב כי חלקו של options הוא בגודל משתנה ונרצה לדעת מה גודלו כדי להבין מתי מתחילה הדטה האמיתית.
2. **Header Length** – דוחיפות לחבילות, לא רלוונטי כיוון.
3. **Type of Service** – גודל החבילה כולל הדטה.
4. **Length** – גודל החבילה כולל הדטה.
5. **16-Bit Identifier, Flags, Fragment Offset** – מועד למקרים בהם נרצה לחלק את החבילה לתתי חבילות (בשחbillah לא יוכל לעבור בתוך מסויים בגלל שהגודל שלו גדול מדי לתווים).

- .6. **Time to Live** – כמו זמן נשאר לחבילה לחיות. כל פעם שחבילה עוברת בראטור הערך יורד ב-1, ובאשר החבילה מגיעה ל-0 היא מפרקת. יכולות יכולות בעקבות טעויות לנוכח בין ראותרים ולא להגיע ליעדן וכדי שלא ישלחו לנצח במעגלים הן ימתו באיזשהו שלב. לחוב יאותחל ל-16 או ל-32.
- .7. **Upper Layer** – הפהוטוקול שימוש את השכבהعلילונה (שכבה 4, ה-*transport*).
- .8. **Header Checksum** – קוד ליהוי שגיאות על גבי header, במקרה בו header נפל אין טעם להמשיך עם השילחה.
- .9. **Source IP** – כתובת ה-IP של השולח.
- .10. **Destination IP** – כתובת ה-IP של היעד. רוב הראותר ישתמש בכתובת זו, אלא אם כן קרתה תקלת בעת השילחה ויהי צריך להודיע על כך לשולח.



ברגע שMagnitude חביבה ליעד מקלפים אותה בשכבות – מערכת הפעלה מקלטת את כתובת ה-MAC ודרך מגעה לחבילת ה-IP שאויה צריכה לפענה. לשם כך יש את הפהוטוקולים בשכבה 4 – ICP, TCP, UDP וכו'.

## Naming and Addressing

- דבר על שלושה פרטוקולים לשם קרייה לבתובות:
1. **DHCP** – באשר קודקוד מתחבר לראשוונה לרשות.
  2. **DMS** – פרטוקול שמקשר שמנות לבתובות.
  3. **ARP** – תרגום כתובת IP לכתובת MAC ולהפר.

## DHCP

### אין קודקוד מקבל בכתובת IP?

- הקודקוד שוחצה לקבל בכתובת IP מבקש אחת מה-DHCP server. DHCP - יש איזשהו מחשב (יכול להיות בראטור) שאחראי על הקצאת כתובות, יודע איזה כתובת פנינה כך שניתן למסור אותה, וכך לא קורה מצב שיש כתובות אחת לשני קודקודים זרים. דרך נוספת למילוט היא דרך מערכת ההפעלה. יש סיכון להתנגשות כי אין לנו ידע על כתובות בשימוש. לחוב DHCP נתון כתובות למן מוגבל, ובהתום הזמן הזה הקודקוד צריך לבקש אחת מחדש (אחרות יכול לקרות מצב שימושו לא קיבל בכתובת).

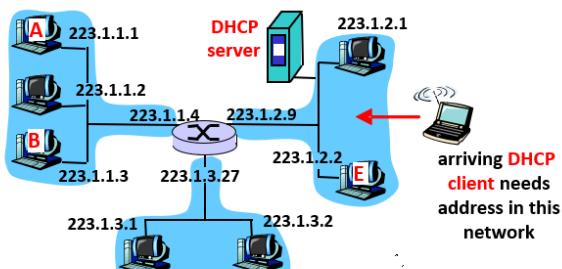
### אין הפהוטוקול עובד?

- המטרה:** לאפשר hosts, קודקודים, להתחבר לכתובת IP באופן דינامي כאשר הוא מצטרף לרשות.
- חידוש כתובות IP כאשר הן לא בשימוש או פג תוקפן.
  - הצטרפות משתמשים חדשים לרשות.

← המחשב שורצוה להתחבר לרשת שלוח הודעת broadcast, בעצם מבקש בתובת משרת DHCP פניו שנמצא ברשת.

← **DHCP offer**: שירות אחד או יותר יקבלו את ההודעה וימשו לשולח שהוא יכול להתחבר אליהם ויצעו בתובת אפשרית. ← **DHCP request**: הקודקוד מקבל הצעה אחת או יותר, ומאשר את התחברות לאחד מהשרתים וմבקש ממנו את תובת IP המוצעת.

← **DHCP ack**: ה-IP מסר את DHCP מאשר את קבלת הכתובות לקודקוד.



**דוגמא**  
המחשב נכנס לרשת ושולח DHCP discover. מי שב-subnet יקבל (כולל השירות DHCP) בഗל (DHCP-broadcasting).

- ה-src הוא 0 כי למחשב החדש אין בתובת.

המשתמש החדש שלוח לבתת הרשת ע"י שליחה לבתובת שכולה 1ום, זאת בתובת מיוחדת שיאומורה שהשליחה התבצעה בכל הקודקודים ברשת.

מספר הפורט הוא ספציפי – 67. כל DHCP servers מזמנים לפורט זהה כך שברגע שחביבה נשלחת דרכו הם יודעים שמדובר ב-DHCP discover, ולאחר מכן שאר המחשבים ברשת מתעלמים מההודעה.

:DHCP offer

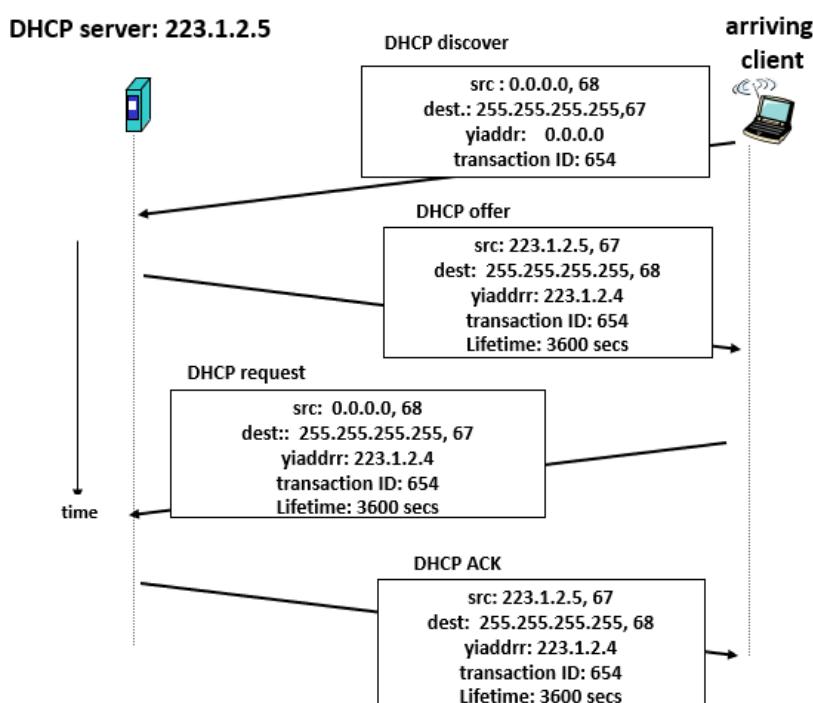
ה-dest יהיה גם בתובת של 1ם כי DHCP לא יודע לאיזו בתובת לשולח את ההודעה, למשתמש עדין אין בתובת. הפורט הוא 68 כי המשתמש מАЗון ומחייב לרשובה דרכו, וכן התקשרות מתאפשרת למרות שאין למשתמש בתובת.

:DHCP request & ack

המשתמש מאשר את קבלת בתובת ה-IP ושולח את ההצעה בחזרה לשרת דרך השדה yiaddr.

- השרת מאשר את בתובת ה-IP ושולח את הודעת ack.

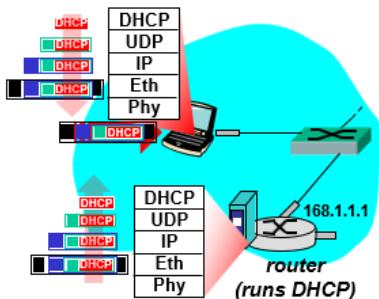
יכול להיות שכמה משתמשים שורצים להתחבר לרשת ישלחו את הודעת request ויקבלו את אותה בתובת. כדי למנוע זאת נשלחת הודעת ack כראשון שמקבל אותה זוכה בתובת.



## Lecture 7

### DHCP

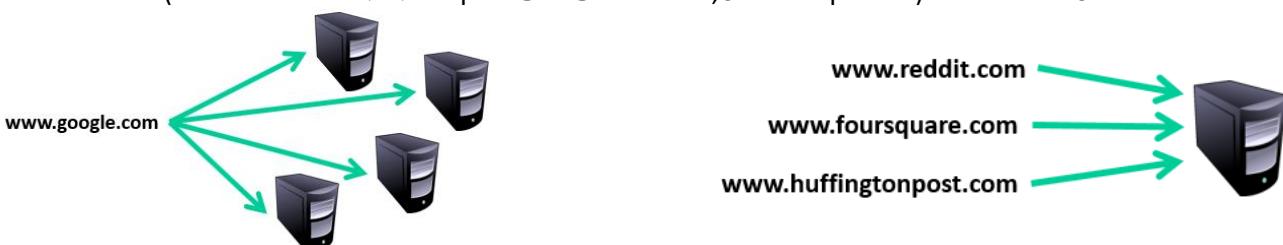
DHCP הוא פרוטוקול קיים ברמת האפליקציה ורץ ע"י השירות, ה-*host*.  
הפרוטוקול מייצר הודעה בצורה שראינו בסוף בהרצאה הקודמת עם המידע הרלוונטי. הודעה זו צריכה לעבור בין השכבות, וכל שכבה אצל השולח צריכה להוסיף header כדי שהפרוטוקול אצל הנמען ידע למלא את מטרתו.  
לדוגמה, לפטוף צריך כתובת IP ושולח בקשה ל-DHCP.



בקשה עוברת דרך כל השכבות (עם הפרוטוקולים UDP, IP, UDP, IP, Eth, Phy) וכל שכבה נוספת מוסיפה מידע ל-header כך שהיא תוכל לפעול בהתאם להזונה. הודעה זו עוברת שם היא על מעלה באמצעות המנגנונים עליהם דיברנו בהרצאות קודמות, מגיעה לנットב שם היא על מעלה השכבות עד לשרת האפליקציה. כתובות הרציה נשלחת שוב (בהודעת ACK) באותו אופן – משכבה האפליקציה של הנットב למטה, מגיעה לשכבה ה-2 אצל המחשב ומשם שוב עלה למעלה עד לשכבת האפליקציה.

אם נרצה להציגו לאתר נצטרך את כתובת ה-IP שלו אותה הנットב מקבל לפי תחביר ברור. הבעיה היא שלנו קשה לזכור כתובות IP, ולכן יש שימוש בשמות לכתובות ה-IP.

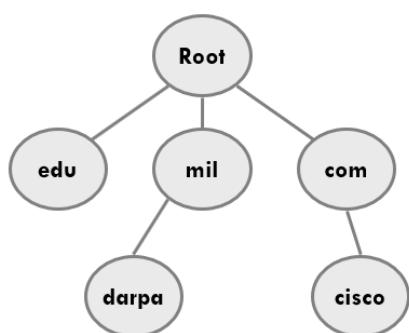
נרצה לתרגם דומיין לכתובת IP. הדומין קיים כדי שלא נצטרך לזכור כתובת IP כאשר נרצה להציגו לאתר, וזה מאפשר שינויים בעיצולותיחסית (אם הכתובת השתנתה לא נצטרך לעדכן את כל המשתמשים באתר). המיפוי הוא לא חד"ע ועל, לכתובת IP אחת יכול להיות מספר שמות (למשל בגוגל יש כתובות ל-gmail gmail.google.com וגם google.google.com אבל הם על אותו שרת), ולשם אחד יכולות להיות מספר כתובות IP (כדי להקל על עצם, ולהפנות משתמש למיקום הגאוגרפי המתאים עבורו).



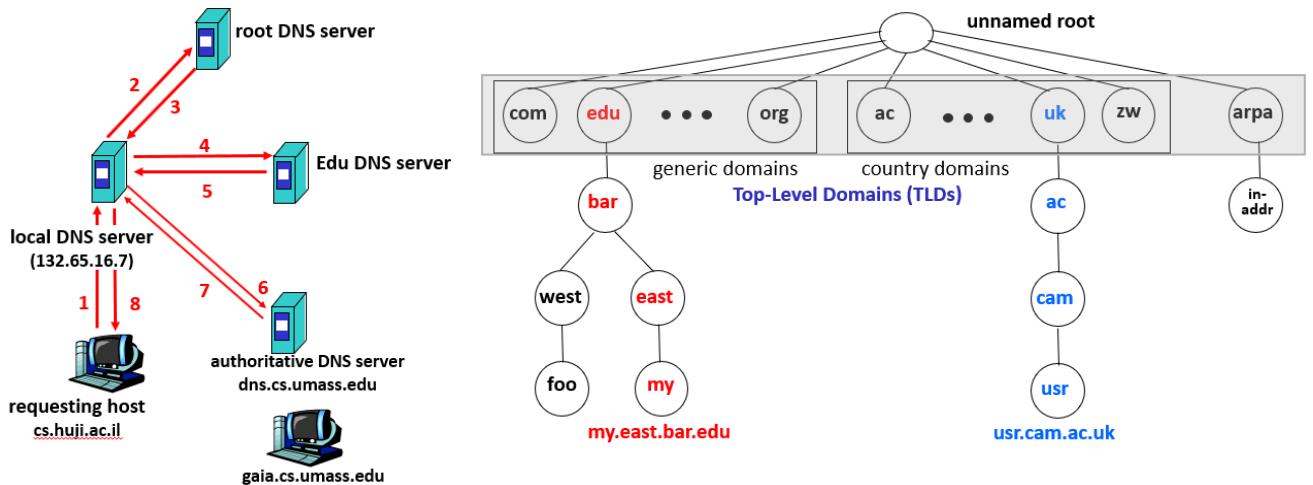
### DNS

- ספר הטלפונים של האינטרנט.
- מפה בין דומיין לבין כתובת IP.
- מסד נתונים שהוא מבוצר והיררכי שמייצג את ההיררכיה שמופיעה בדומיין (למשל → com → google.com → mail.google.com).
- המידע של DNS מארגן בעץ. בכיה נמצא את מה שאנו צריכים במסמך הנתונים.

לשורת אין שם (נקודה התחלתי), הבנים של השורש מייצגים את הסימולט של הכתובת (.gov, .io, וכו').  
ובו' **top levels domains**, הבנים מתחתיהם מייצגים את המשך ההיררכיה.



נסתכל על הדוגמה למטה – נניח שאנו רוצים לפנות לאיזשהו דומיין (gaia.cs.umass.edu), אנחנו מפנים לשרת DNS מקומי בשם resolver בשם localhost IP (שלב 1). אם ה-resolver לא מכיר את הבקשה הוא יודע בעץ ההיררכיה כדי לענות על השאלה. ה-resolver פונה ראשית לשורש, שיפנה את השרת לשירותים שמכירם את edu ומשם הלאה עד ל-authoritative DNS server, השרת שמוסמך להחזיר כתובת ה-IP הרציה לשאלתה. ה-resolver מחזיר את התשובה הסופית.



### DNS caching

בפעם הראשונה שנכנסים לאתר האינטרנט ה-DNS לא מכיר אותו, אין caching והוא DNS פונה לשורש כדי לראות אם יש חומר זהה. אם יש זהה אחר, צריך לחזור לפועל של DNS לפני שנקבל אותו. אבל משתנים לעתים רחוקות מאוד, ישנים אתרים פופולריים שנכנסים אליהם לעיתים קרובות, ונרצה לחסוך את זמן החיפוש במקרים באלה.

בשביל זה יש caching של כל מה שראים, בלומר שומר בזיכרון בעז. למשל, נבנסו במקרה הראשונה ל-com.cnn, פנימה לשורש וזה-com.cnn. במקרה הבא שנחפש facebook.com נדע מה זה-com כי הוא שמור. יש בנוסף caching של תוצאות סופיות, כמו למשל במקרה הבא שנחפש facebook.com יכיר אותו ויחזר תשובה סופית.

- ← שומר גם שגיאות ושאלות שנכשלו.
- ← ה-DNS שומר את התשובות עד שערכן time to live (זמן בו הן נשמרות ב-DNS) פג.
- ← יש סה"כ 13 שורשים שבגדול מכילים את אותו המידע, בשלטים ע"י ארגון אחד. נתיחס בקורס אליו ישנו שורש אחד.

### מבנה רשותה ב-DNS

ה-DNS מכיל רשומות מהצורה הבאה: {name, value, type, ttl}

- ← Name - הדומין
- ← Value - כתובת ה-IP
- ← time to live - TTL
- ← Type הוא סוג הרשותה. נדבר על שני סוגי רשומות:
  1. A היא רשומה שהיא authoritative, התשובה הסופית – name באותה רשומה הוא הדומין וה-value הוא כתובת ה-IP הרצiosa.
  2. NS יחויר את מי אחראי על קירוב לתשובה לגבי הדומיין.

סוגים נוספים:

- **Type=A**
  - name is hostname
  - value is IP address
- **Type=NS**
  - name is domain (e.g. foo.com)
  - value is hostname of authoritative name server for this domain
- **Type=PTR**
  - name is reversed IP quads
    - e.g. 78.56.34.12.in-addr.arpa
  - value is corresponding hostname
- **Type=MX**
  - value is name of mailserver associated with name
  - also includes a weight/preference

נזכיר לדוגמה מוקדם, רצינו לדעת מה כתובת ה-IP של google.com

← נפנה לשרת ה-DNS (Local DNS resolve) שפנה לשורש.  
 ← השורש החזיר תשובה NS:  
 m **answer:** downward delegation...  
 m **com NS a.gtld-servers.net**  
 m **a.gtld-servers.net A 74.292.124.59**

בולם השם המצוין באותה רשומת NS הוא שרת אחר שמכיל את המיעוד (authority) על com ויכול לענות על השאלתה.

ברשותה A תהיה כתובת ה-IP של אותו שרת שמצוין מוקדם.

← נפנה לשרת החדש והוא יתן לנו בהודעת NS את השרת הבא שאחראי על החזרת google.com וכן את כתובת ה-IP של אותו שרת בהודעת A.  
 m **answer:** downward delegation...  
 m **google.com NS ns1.google.com**  
 m **ns1.google.com A 122.45.212.57**

← הגענו לשרת ה-authoritative להחזרת הכתובת IP הרשמית של www.google.com כל קודקוד שעבר בדרך.

#### הכנת רקווד ל-DNS

נניח שאנו רוצים לרשום את האתר שלנוfoobar.com כדי שיוכלו להגיע אליו. נקבל מספק האינטרנט מרוחב בתובות שיש לנו. נרצה שמי שכותב את השם שלנו יגיע אלינו, בולמר מישחו צריך להפנות אלינו, ולמשל נרצה שהשרת שאחראי על מיפוי foobar.com יחזיר את השרת שלנו ובצム הוא יctrurk להחזק רשותה עם הדומיין וכתוות ה-IP כך שהוא יחזיר בשאלתה com. וכך צורר את השרת dns1.foobar.com אשר יחזיר להחזיק רשותה עם הדומייןfoobar.com.

את התשובה הבאה:

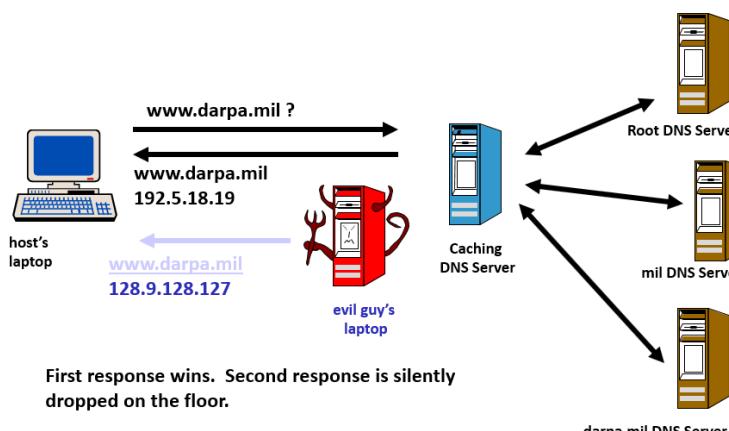
- (foobar.com, dns1.foobar.com, NS)
- (dns1.foobar.com, 212.44.9.129, A)

וכך צריך לעשות עם כל שרת לוונטי במעלה ההיררכיה.

#### אבטחת DNS

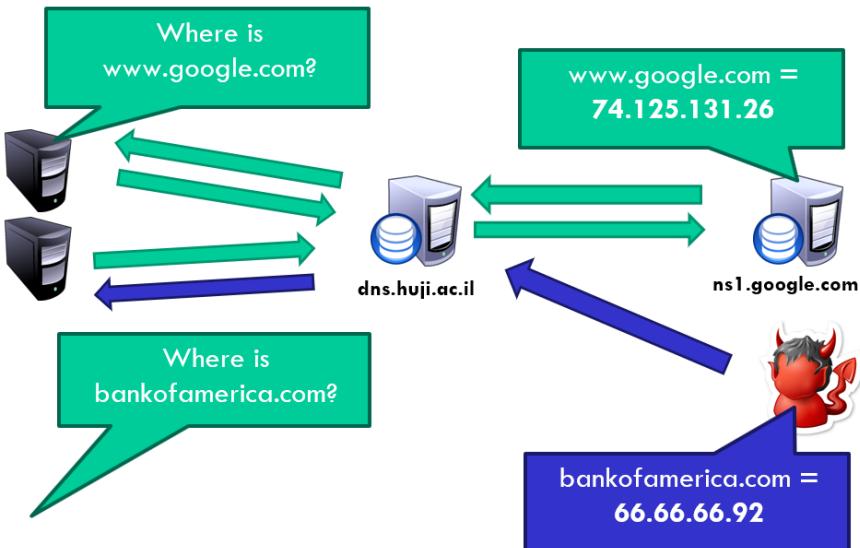
ה-DNS הוא ספר הכתובות של האינטרנט, חסיבותו ברורה. כמה דוגמאות לביעות אבטחת מידע ב-DNS:  
**דוגמה 1 – DoS:** הצפת שרת DNS בשאלות. ב-2002 שרתי root DNS הופצטו בימיroot כדי לא לאפשר להם לתקוף. לא ממש שמו לב מה כי יש ב-cache את הרקווד הרלוונטי, ל-root פונים רק כאשר פג התקוף או בשיש מישחו חדש. ההתקפה הייתה יותר גבוהה אם הייתה מוצבצת על local DNS או על authoritative.

**דוגמה 2:** ההתאחדות ל-local DNS resolver ומשמעותה. בשביל לבצע זאת נדרש מהחומר לרשות המקומית, בולמר להיות עם קרביה פיזית למחשב הנפגע, שכן חוות יפגעו מאשר במקורה הקודם אבל יפגעו יותר חזק. מצפים מה-resolver להחזיר תשובה, אבל קיימ גורם אחר שיכל לענות לו תשובה אחרת ושגויה. יתכן שענה יותר מה-resolver או שנהייה מוסמך לפניות במקומו.



**דוגמה 3 – DNS cache poisoning**

בתקופה זו מנסים להבננו דוגנית ל-cache. אם הצליחו, הרשומה זו תהיה ב-cache עד להतpagות TTL. אם למשל מישחו מבקש מה-DNS את כתובת ה-IP של איזשהו דומיין, ה-DNS יחפש את התשובה אצל שרתים אחרים ויקבל תשובה דזונית כלשהי מהשרת של התקוף עם כתובת ה-IP שלו. אם התשובה לא מופיעה ב-cache, ה-DNS יכנס אותה עם TTL אחר כך שפעם הבאה שימושו יעלה שאלתה מתאימה ה-DNS ימצא את התשובה ב-cache ויחזר את כתובת ה-IP של התקוף.



ל-DNS קיימים מנגנונים שאמורים להגן על מתקפות כאלה:

1. ה-resolver קיבל מענה רק על שאלתה פתוחה, כלומר שאלתה שהתשובה שלא מופיעה בבר-cache.
2. **Transactions ID:** לשאלות יש מספרים מזהים – 16 ביטים רנדומליים, ובאשר מישחו עונה הוא יציין על איזו שאלה הוא עונה. בולמרלתו הינו<sup>2</sup> אופציית לנחש כדי לאתגר לתקוף, והוא י策ר לעשות אתזה לפני ה-DNS. איך התקוף יכול לענות לפני ה-DNS? לשולח הרבה תשובות ולנחש את ה-ID transaction בכל אחת, מעורן יום ההולמת מתיישחו הוא יצליח.

אבל בעיה נוספת שהיא תוקפים נתקלים בה היא שהדומינום יכולים להשמר ב-cache, וכך לשלוח את התשובה בבדיקה בזמן בו TTL פג והשרות מחפש את התשובה בשרתים האחרים ולא פונה ישירות-cache. בכל הפסד התקופים יצטרבו לחכotta TTL.

**:The Kaminsky Attack**

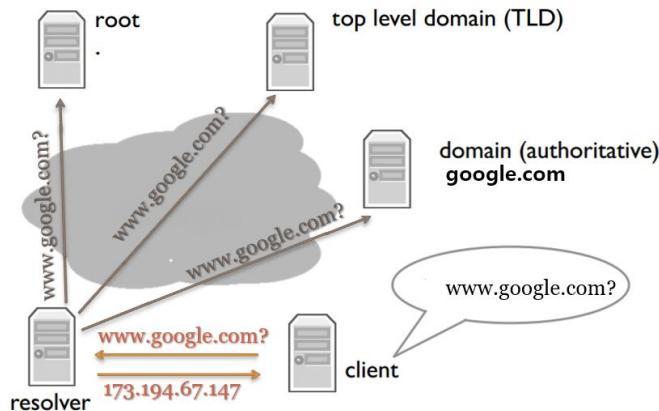
← במקור ID transaction לא היה בامت רנדומלי. התקוף יוכל לנסוט לזרות איזושהי חוקיות במתן ה-ID וכך לצמצם את מרחב ההתקפה. הוא עדין עונה הרבה תשובות ומוחש, אבל זה לא מטפל בבעיית ה-cache.

← קמינסקי תקף בצורה בה אין צורך לחפות לחילון הדומנויות. כלומר במקורה שציגו מוקדם בו התקוף מחדיר תשובה שלא נמצאת ב-cache, מזהה נכון את ה-ID אבל התשובה האמיתית הגיעו לפניו, יהיה צורך לחפות. אצל קמינסקי לא מחכים TTL במקורה של הפסד.

← התקוף שואל על google.com ועוד 1.google.com ועוד 2.google.com וכן הלאה. נניח שהתקוף הצליח ב-183.google.com גיעת IP של התקוף. למה זה עובד? בהערכתה הבאה.

**Lecture 8**

## DNS



הרעלה קמיןסקי  
נחזיר לתקיפה של קמיןסקי את פרוטוקול DNS.  
כדי לתקן DNS כך שנחזיר לשאלתה תשובה נכונה (הפניה לכתובת IP שגוייה) צריכים לkerות מספר דברים:

1. תקיפה כאשר פג תוקף הרשומה ב-cache transaction
2. ניחוש נכון של ה-ID transaction
3. הגעה לעד לפני התשובה הנכונה

הסיכוי שבל הדברים האלה יקרו הוא לא גדול, ובגלל זה עד קמיןסקי לא רוא חשיבות לאבטחת המידע בפרוטוקול.  
הנחה של קמיןסקי: התוקף הוא בעל יכולת ליצור שאלות.  
הידוש: התוקף לא יצטרך להזכיר בשואה רצחה לנשות שוב.

במקרה של תקיפת קמיןסקי, התוקף לא מנסה להשתלט על המתחובה ה-A, אלא על ה-NS. בולמר, במקרה לעונת השאלתא google.com עם ה-IP החדש, התוקף יענה עם תשובה NS שמובילה לשרת החדש.

**דרך הפשה:**

- הוא יבקש מה-resolver את הכתובות הרצiosa, google.com למשל, שכבלה הנראת בבר תיה ב-cache.
- ימשיך לבקש כתובות אחרות: com, 1.google.com, 2.google.com וכו'. בסופו הוא יצליח לבקש כתובות שלא נמצאת ב-cache, למשל ב-183.google.com. למה ישנה לנו אם הוא מצליח בכתובת שאף אחד לא משתמש בה?

מתישחו התוקף יצליח להשתלט על ה-cache, לא יצטרך להחות TTL. למה?

קמיןסקי הבהיר שייתכו שתי תשיבות אפשריות של-resolver לשאלתה של התוקף:

1. **A –** תחזר hostname לכתובת IP, בולמר הבקשת com 183.google.com תחזיר בכתובת IP חדשית.

m **google.com NS www.google.com**

m **www.google.com A 6.6.6.6**

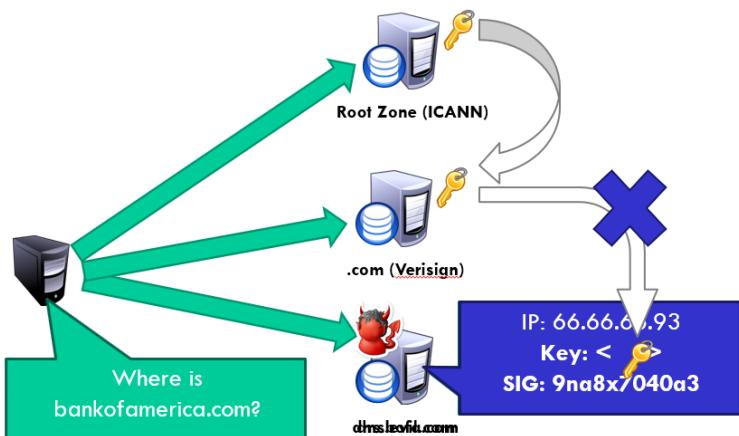
2. **NS –** התוקף ייצור תשובה NS במתואר למטרה. אומנם com 183.google.com לא תהיה תשובה סופית אבל הוא ייחזר את השרת הבא שצריך לפנות אליו בשם google.com וזה יהיה השירות החדש.

באשר הרשומה A עם ה-IP האמתי של com תפוג מה-cache ומיישהו יפנה לכתובת, השירות של התוקף יהיה ב-cache ויוחזק.

בנוסף, השירות קובע את TTL והתוקף יקבע אותו למקרים.

m **google.com NS www.badguy.google.com**

m **www.badguy.google.com A 6.6.6.6**



האגנות לתקיפה  
אופציה 1 - הגדלת אורך ה-ID transaction כדי להפוך את הניחוש ליותר קשה.

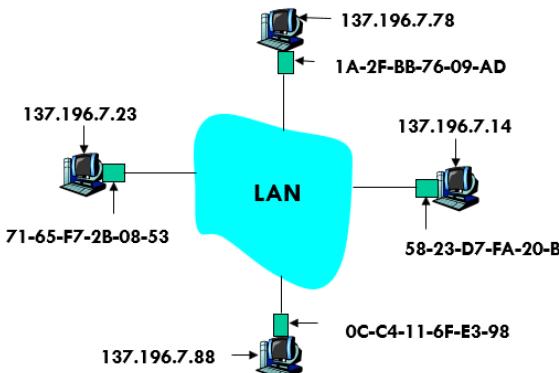
הבעיה: אין אקריאות, ואם לומדים את החוקיות של ההגירה אפשר לחקות אותה.

**DNSSEC – הפטון האידיאלי.**

כל פרוטוקול באינטרנט יש פרוטוקול SEC, שmbוסס על קרייפטוגרפיה. כל שרת, באשר הוא עונה תשובה, הוא יצטרך להוכיח שהוא הגורם הנכון ע"י שימוש במפתח ספציפי. היישום קשה ומחייב שיתוף פעולה מגורמים רבים.

- אבטחת DNS היא בעה פתוחה.

## – תרגום כתובות IP לכתובות MAC ולהפנן – Layer 2 vs. Layer 3

ARP – Address Resolution Protocol

הפרוטוקול שמתרגם כתובות IP לכתובות MAC ולהפנן. כל קודקוד end host יש טבלת ARP שmaps כתובות IP לכתובות MAC ולהפנן.

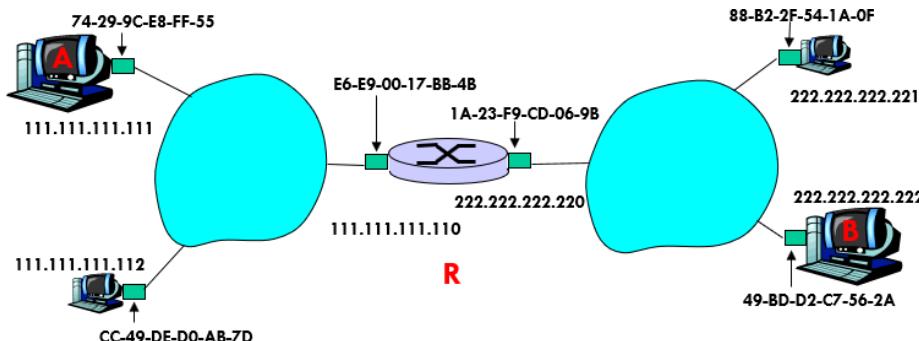
(שבאותה LAN) וכוללת בנוספ TTL (לרוב של 20 دق'). אם קודקוד רוצה לשלוח לגורם אחר ברשתה 3 (מחוץ ל-LAN), הוא שלוח כתובות IP ומוסיף לדאטיה של הפקטה בשכבה 2 את ה-MAC שלו ושל הנמען – הרואוטר הרלוונטי, לפי טבלת ARP. הרואוטר ישלח לנמען שmaps כתובות ה-IP, ויתרגם את כתובות זו לכתובות MAC של אותו גורם.

מה קורה כאשר אין בטבלה את כתובות ה-IP? ברגיל – נפנה לכלום. הקודקוד יעשה broadcast ויחפש אצל המשתמשים האחרים (נכון מדברים ברגע באוותה רשת) את הכתובת הרצוייה.

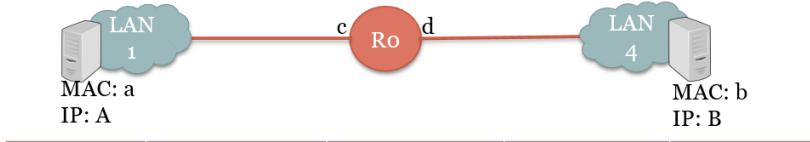
**דוגמה – שליחת פקטה דרך שכבות 2 ו-3.**

קודקוד A רוצה לשלוח לקודקוד B, אבל לא נמצא אותו באותו LAN, והנתב R מקשר ביניהם. נניח שהקודקוד A מכיר את כתובות ה-IP של B.

- A יוצר פקטה בשכבה ה-IP שבה ב-datagram מצוין ש-A הוא המוקור ו-B הוא היעד.
- A משתמש ב-ARP כדי לקבל את ה-MAC של הנתב כי אליו הוא שלוח (כתובות IP לפי הדוגמה 111.111.111.110).
- A יוצר פקטה של שכבת הילינק שבה מצוינת כתובות MAC של הנתב ב-dest (כדי שהסוויצ'ים בדרכיו ידעו לאן לשלוח), וכוללת את המידע בפקטה הקודמת (שליחה מ-A ל-B דרך R).
- A שלוח את הפקטה, ובאשר היא מגיעה לנtab, הנתב חזהה בפקטה של שכבת ה-IP שהוא הוא B (כתובות ה-IP של R). הנתב משתמש בטבלת ARP שלו כדי לתרגם את כתובות ה-IP של B לכתובות MAC ויזכר את הפקטות הרלוונטיות.



המחשה נחמדה מהתרגול:



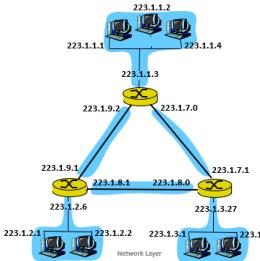
Step	SRC MAC	DST MAC	SRC IP	DST IP
A → Ro	a	c	A	B
Ro → B	d	b	A	B
B → Ro	b	d	B	A
Ro → A	c	a	B	A

**IPv6**

פרוטוקול שמנסה לאפשר יותר ביטים מאשר ה-32 שבפרוטוקול IPv4 ופתרור את מצוקת כתובות ה-IP ועל הדרך לפטור עוד בעיות.

הבעיה – היחסם של הפרוטוקול קשה למימוש ומעורב גורמים רבים. צריך לעדכן את כל הנתבים בעולם להסתכל על מרחב בתובות אחר, וכל עוד לא מעדכנים הפקות החדשות והיו חסורת שימושים שימוש חיב להיות הדרגתית. מה האינטראקט לא יכול להיות רשת IP אחת גודלה? הוא יכול.

## טופולוגיה



בשבבה 3 נתבים מחברים בין LANs (subnets) וצריך להסתכל על שני מאפיינים מרכזיים בশמראכיבים רשת:

1. **טופולוגיה:** איך הקודקודים ברשת מחוברים בצורה פיזית?

2. **Routing Algorithm:** איך מחליטים על פני הטעולה איך תעבורת תעבור בפועל?

## תכונות של טופולוגיה

איך נגיד טופולוגיה טובה? נסתכל על התכונות הבאות:

1. **קוטר –** אורך המסלול הכי קצר בין כל שני קודקודים, הקוטר יהיה המסלול הכי קצר בין שני הקודקודים ברשת.

2. **דרגת קודקוד ברשת –** עבור קודקוד ספציפי, כמה קודקודים מחוברים אליו קוודקו.

3. **Bisection Bandwidth** – נחלק את הקודקודים ברשת לשתי קבוצות בגודלים שווים ונמדד מה רוחב הפס ביניהן (המחשה בשרטוט).

אם יש הרבה תעבורת מידע אחד לשני היא תעבור בין הצלעות המחברות בין הקבוצות. הימין

חיצים שתמיד בשקבוצה אחת רוחצ'ה לדבר עם השניה יהיו הרבה צלעות שעוברות בחתק.

**Bisection bandwidth** יהיה המינימום, המקרה הכי גרוע לחילוק הקודקודים לקבוצות (אם נסתכל על הקיבולות עם ערך 1, BB-2B יהיה הכי מעט צלעות שחוצות את הקבוצות, כי באופן אופטימלי נעדיף שהיינו יותר צלעות שמקשרות בין שתי הקבוצות, יענה על השאלה מה הצוואר

בקבוק הימי נורא שיכל להיות לנו ברשת?).

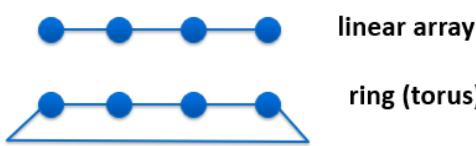
## Explicit vs. Non-explicit

**טופולוגיה לא מפורשת –** החוקיות של הטופולוגיה לא ידועה. הקודקודים מחוברים באופן שרירותי

**טופולוגיה מפורשת –** ניתן לדעת את חוקיות הטופולוגיה. יש שימושות ל"שמות", קואודינטות של הקודקודים (למשל יש צלע בין שני קודקודים רק אם השמות שלהם מקיימים תכונה מסוימת). יש חוקיות לטופולוגיה ונitin לגזר אותה ממשות הקודקודים. הניתוב יכול להסתמך על המבנה.

## דוגמה – טופולוגיה לינארית לעומת טופולוגיה טבעית

**טופולוגיה לינארית:**



linear array

ring (torus)

- קוטר: 3
- דרגה: חלק עם דרגה 2 וחלק עם 1
- 1 : Bisection bandwidth

**טבעת:**

- קוטר: 2
- דרגה: 2
- 2 : Bisection Bandwidth

**שתי דוגמאות אלה הן טופולוגיות מפורשות:**

במערך הלינארי ניתן למספר את הקודקודים החל מ-0 ועד  $N - 1$ , וכל קודקוד  $i < N$  מחובר לקודקוד  $i + 1$ . בטבעת ניתן למספר את הקודקודים באותו אופן, אבל הפעם כל קודקוד  $i$  מחובר לקודקוד  $(N + 1) \mod (N)$ .

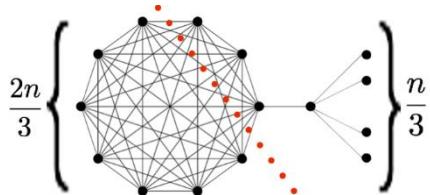
## Lecture 9

### טופולוגיה

#### Bisection Bandwidth

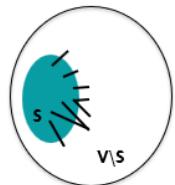
נחוור רגע לתוכנת ה-*bisection bandwidth*, מתי היא בעייתית בשנרצה להעריך רשתות?

נסתכל על הרשת הבאה –



3/2 מהקודקודים נמצאים בצד שמאל, 1/3 נמצאים בצד ימין. יש קודקוד בימין שמחבר בין שני הצדדים, ואם איזשהו קודקוד מצד ימין ירצה לדבר עם קודקוד מצד שמאל, המידע יהיה חייב לעבור בリンク היחיד שבין שתי הקבוצות. הצוואר בקבוק מוגבל מאוד ולכן זאת לא תהיה תשתובה.

ה-*bisection bandwidth* תמיד יחלק את הקודקודים כר' שתהיה קבוצה בה יהיו הרבה קודקודים משמאל, ככלומר הרבה צלעות שעוברות את החתך. לכן תמיד ייראה לנו כאילו ערך ה-*bisection bandwidth* ברשות הزادת הוא טוב.



במוקם להסתכל רק על חלוקות הקודקודים לשתי קבוצות שוות נסתכל על **בל האפשרויות**  $\frac{n}{2} < |S| < n$  לחלק קבוצת קודקודים ל-S ו- $V \setminus S$ .

תוכנות **edge expansion** על גרף תוגדר באופן הבא:

$$\min_{S \subseteq V, 0 < |S| < \frac{n}{2}} \frac{\text{EdgesBetween}(S, V \setminus S)}{|S|}$$

כלומר החלוקה שתתנו את היחס המינימי בין הצלעות החוצות את החתך לבין גודל הקבוצה הקטנה מבין השתיים  $S$ . – **Expanders** – גرافים שעבורם edge expansion-edge גבוה (קובע גדול או שווה מ-1). ← מספר הצלעות המומוצע שיוציא מקודקוד מהקבוצה הקטנה לגודלה הוא לפחות 1. – אם נרגיל גרף בדרجة קבועה, בהסתברות גבוהה הוא יהיה expander.

#### סוגי טופולוגיה נוספיט **Hypercube**

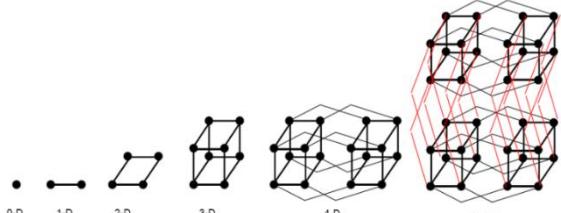
היפר-קוביות הם גרפים *cube* – א' בינארי ( $n$  הוא ממד הקובייה).

מספר הקודקודים יהיה  $2^n$  וכל קודקוד ממושפר לפי היצוג הבינארי שלו – א' ביטים – בניית מפורשת. בין שני קודקודים יש צלע אם יש רק קואורדינטה אחת שונה ביצוג הבינארי שלהם.

• דרגת הקודקודים:  $n$  לכל קודקוד יש א' ביטים لكن א' אופציות לקובורדינטה אחת שונה.

• קוטר:  $n$

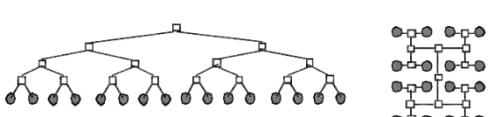
•  $n-2$  כר' שלמאל של כל הקודקודים המתחילה ב-0 לעומת  $2^{n-1}$ . בחלוקת לשתי קבוצות בגודל  $2^{n-1}$  הקודקודים המתחילה ב-1, לכל קודקוד יהיה בדיק שבן אחד שנמצא בקבוצה השנייה.



#### עץ בינארי

- דרגת הקודקודים: קבועה – 2
- קוטר:  $\log n$
- $n = 2^h$
- $n = 2^h - 1$

**Bisection bandwidth**: (1) אם נחלק את הgraf לשתי קבוצות (למשל קודקודים משמאל לשורש וקודקודים מימין) תהיה צלע אחת בחתך.



במה הגדירות מההרצאה ומהתרגול

**ניתוב –** תהליכי בחירת מסלול לתקשורת ברשות בין-/דרך מספר רשות.

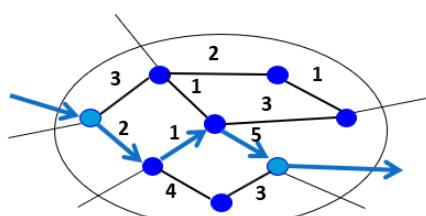
סוגי שליחה

- **Unicast** – שליחה לעד יחיד
- **Anycast** – שליחה לשכנים
- **Multicast** – שליחה למספר יעדים
- **Geocast** – שליחה לאזור גיאוגרפי
- **Broadcast** – שליחה לכל מי שברצה

**טבלת ניתוב –** טבלה שבל קדוק ברשות מחזיק, ששומרת את העליות להגעה ממנה לעד אחר ברשות (בהתבסס על מרחק, עובי פס-רוחב וכו').

– דבר תחיליה על **ניתוב פנים-ארגוני**. בניתוב זה הארגון יהיה אחראי על הרשת ועל ניתוב הפקודות.  
**inter-organization routing** – ניתוב בין ארגונים. נדון בו בהמשך הקורס.  
 נסתכל על הניתוב בשכבה 3 כאשרリンク מייצג subnet IP.

**Routing** – אלגוריטמי ניתוב, אלגוריתם מבוזר שמודיע את המסלולים להעברת פקודות ומיציר טבלאות בנתבים לפיהן הם שולחים את הפקודות המגיעות.  
**Forwarding** – המשימה של העברת פקודה הלאה. נעשו בחולוציה של מיקרו שניות, ברגע זה הטבלה חייבות להימצא בנתב כדי לא לעכב את התקשורת. לכן נרצה שאלגוריתמי הניתוב יפעלו לפני הגעת פקודה כדי לאפשר forwarding ועליל.

המודל הקלאסי – מציאת המסלולibi קצץ

נחשב לכלリンク משקל ובהינתן המשקל הזה נמצא את המסלול הקצר ביותר. המשקלים קבועים ע"י מנהל הרשת, דבר על כך בהמשך.  
 נעשה זאת כדי לקבוע את ה-"next hop", בהינתן פקודה שהגעה לנットב הנtentב ידע מי הבא אליו היא תישלח.

ישנן שתי סכימות עיקריות של ניתוב:

1. **Distance Vector Routing** – גרסה מבוזרת של בלמן-פורד. מעביר וידע את האינפורמציה רק של הקודקוד ושל השכנים.
2. **Link State Routing** – רואטרים שלוחים מידע – Link state לכל קודודי הרשת כדי למודד את טופולוגיה הרשת.  
 מעביר אינפורמציה גלובלית.  
 • לחוב ארגונים יעדיפו את 2.  
 • בשתי הסכימות החישוב הוא מבוזר – כל קודקוד מחשב בעצמו את הפלט.

Distance Vector Protocol

(מהתרגול) מבוסס על נוסחת בלמן-פורד:

$$Ds(y) = \min_{v \in \Gamma(s)} \{c(s, v) + Dv(y)\}$$

- כל קודקוד ברשות מחזיק בווקטור מרחוקים משאר הקודקודים ברשות (מרחקים לא ידועים ואוטחלו לאינסוף).
- האלגוריתם ירוץ עד להתייחסות הווקטור.
- אלגוריתם מבוזר.

העקרונות המרכזים (מנקודות מבט של רואטר):

- שמירה בנסוף וקטור של מרחקי השכנים.
- באשר הווקטור מתעדכן הוא ישלח לכל השכנים של הקודקוד.

- אם אחד השכנים התעדכן, אותו קודקוד יעדכן את הווקטור שלו בהתאם.

#### אתחל:

- כל קודקוד יאותחל את הווקטור שלו באופן הבא:  

$$D_s(v) = \begin{cases} c(s, v) & v \in \Gamma(s) \\ \infty & \text{else} \end{cases}$$
- יאותחל את הווקטורים של השכנים שלו:  
 $\forall w \in \Gamma(s), \forall v \in V: D_w(v) = \infty$
- הקודקוד ישלח את הווקטור שלו לכל השכנים.

#### עדכון:

אם אחד הווקטורים של השכנים השתנה נעדכן:

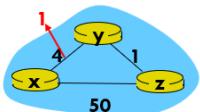
$$D_s(y) = \min_{v \in \Gamma(s)} \{c(s, v) + D_v(y)\}$$

ואם אחד הערכים אצלו השתנו, נשלח את הווקטור לשכנים.

- את המרחק של קודקוד עצמו מעצמו נתחל ל-0, אחרת  $\min_{v \in \Gamma(s)} \{c(s, v) + D_v(y)\}$  ייכל.

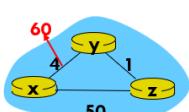
#### Count to infinity

(מהරצתה) נניח שאחד המשקלים בリンク השתנה. הנטבים שמחוברים לリンק יזהו את השינוי, יעדכוו את ה-DV שליהם וזה יתחל להתעדכן ברשות.



– נצפה שם יש עדכון משקלים לטובה ברשות שלם, העדכן יתרפסת מהר ברשות (רגעע לנצח הסופי מהר). נסתכל לדוגמה על הרשת הבאה עם השינוי בצלע בין  $y$  ל- $x$  משקל 4 למשקל 1, ומשקל 1 משקל 4. ברשות יkir הפעולות הבאות:

- ← בזמן  $t_0$  קודקוד  $y$  יזהה את השינוי, ישנה את ה-DV שלו (הגעה מ- $y$  ל- $x$  עולה 1 במקום 4) וישלח את הווקטור המעודכן לשכנים.
- ← בזמן  $t_1$  קודקוד  $z$  קיבל את הווקטור המעודכן, ישנה את ה-DV שלו (הגעה מ- $z$  ל- $x$  עולה 2 במקום 5) וישלח את הווקטור המעודכן לשכנים.
- ← בזמן  $t_2$  קודקוד  $x$  קיבל את הווקטור המעודכן, אבל לא ישנה את ה-DV שלו כי לפיו הוא יכול להגיע ל- $x$  בעלות 1 ובכל מקרה זה יותר זול מלחייב אליו דרכו  $z$ .



– שלא כמו המקרה הקודם, כאשר העדכון הוא לרעה, ייקח הרבה זמן לעדכן את הרשת. נניח שבאותה רשת נשנה את המשקל ל-60. המצב ההתחלתי הוא:

Distance Vectors:			Cost Vectors:			Routing Tables:		
	cost to							
	x	y	z					
from	x	0	4	5	<del>C(x,y)=4</del> <del>C(x,z)=50</del> C(y,z)=1			
	y	4	0	1		At x: to y on (x->y)		
	z	5	1	0		to z on (x->y)		

נצפה ש- $y$  יעדכן את הווקטור שלו בהתאם, אבל הוא יקבל ווקטורים מהשכנים שלו  $z$  עם דרך הגעה אטרקטיבי יותר מ-60 – הגעה ל- $x$  דרך  $z$  בעלות של 5. לכן  $y$  יעדכן את המרחק שלו מ- $x$  להיות 6.

נמשיך בהלאיר,  $z$  יעדכן את דרך הגעה שלו ל- $y$  להיות 7, אוח"ב  $y$  יעדכן ל-8 וכו'. מתי נעצור? אחרי 44 איטרציות באשר הגעה מ- $y$  ל- $x$  תעליה 51, מ- $z$  ל- $x$  תעליה 50, ו- $y$  יבין שאכן כדאי לו להגיע ל- $x$  דרך  $z$ . דברים י发生变化 מאוד ברשת, ובכל שנדיגיל את המשקל שהשתנה בכבה זה ייקח יותר זמן. זאת הבעיה בקשר שזה לוקלי.

cost to	x	y	z	
from	x	0 51 50		
y	51 0 1			
z	50 1 0			

הפתרון: – אם הדרך של קודקוד  $z$  לאיזשהו קודקוד  $x$  עוברת דרך שבן שלו  $y$ , הוא יעדכן את עשויה המרחק שלו לאותו קודקוד  $x$  להיות  $\infty$ . (כבה ע"ל עבריר מיידע ל- $x$  דרך  $z$ )

- יבעוד רק במקרה שהלולאות הן מוגדרות 2 גורמים.

## Link State Routing

(מהתרגול)  
globally. כל קודקוד שומר אצלו את המצב של כל ראות ברשות, ובאופן איטרטיבי מוצא את המסלול הקצר ביותר ע"י איזושה' וראיצה לדיקסטרה.

לכל קודקוד  $s$  נחזק טבלת מרחקים שתואתחל באופן הבא:

- לכל שכן של  $s$  נאותל ( $s, c$ )  $c$  ולכל השאר  $\infty$ .

- נחזק סט של קודקודים,  $N'$ , שהם הקודקודים שאנו ידועים את הדרך הבי קצורה אליהם בוודאות.
- בכל שלב נוסיף קודקוד לסט.

בניגוד ל-DV:

- נשלח את הטבלה שלנו לכל הקודקודים ברשות ולא רק לשכנים.
- מעದכנים בתדירות נמוכה יחסית (כלי 30 דקות).

הבעיות: איבוד פקודות, חורים שחורים, אי התכנסות.

**אתחול:**

$$\begin{aligned} \text{Set my } D(v) &= \begin{cases} c(s, v) & v \in \Gamma(s) \\ \infty & \text{else} \end{cases} \\ \text{Set } N' &= \{s\} \end{aligned} \quad \bullet$$

**כל עוד  $N \neq N'$ :**

נמצא קודקוד  $w$  שלא נמצא ב- $N'$  אך שהמරחק ( $w$ )  $D$  מינימלי.  
נוסיף אותו לו- $N'$ .

לכל שכן של הקודקוד, אם הוא לא נמצא ב- $N'$  נעדכן:

$$D(v) = \min\{D(v), D(w) + c(w, v)\}$$

### השוואה בין Link State לבין Vector Distance

(מהרצאה) בחירת האלגוריתם המתאים תלויה בתכונות הרשות. למשל נניח שבגרף יש קודקוד שמחובר ישירות לעד מסויים. DV-DS קודקוד יקבל הודעה מהיעד ויתחבר, לעומת זאת ב-LS הוא יצטרך לשלוח הודעות לכל הקודקודים, לחשב מרחקים וכו'. באופן כליל LS יותר נפוץ. נסתכל על כמה נקודות להשוואה:

**DV**

**LS**

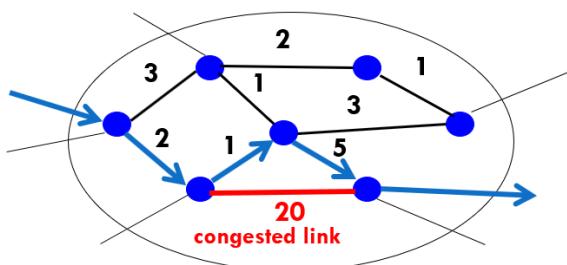
כמה הודיעות מועברות	מהירות ההתקנסות	robustness – מה קורה אם ראות נופל?
העברה בין כל הצלעות אליהן הקודקוד שליחה רק לשכנים, תלות בטופולוגיה הרשות. מחוור. עבור $n$ קודקודים ו- $E$ リンクים, יש $O(nE)$ הודיעות. האלגוריתם רץ ב- $O(n^2)$ וחושך בנוסף לא קבוע, תלוי בלולאות וב- $O(nE)$ הודיעות.	הראורט שනפל ועדיין את הקודקודים והם יפעלו בהתאם לאלגוריתם. קודקוד יכול לפרסם עדכון שגוי לגבי עלות של משקלリンク בודד אבל כל קודקוד מחשב את הטבלה שלו וזה יכול לגרום לפעוף של שגיאות ברשות.	דיברנו מוקדם על עדכנים במשקל צלעות במרקמים של שינויים. האלגוריתם יכול לפרסם עדכון שגוי לגבי עלות של משקלリンク בודד אבל כל קודקוד מחשב את הטבלה שלו בעצםו.

שני האלגוריתם ממומשים במצבים:

- פרוטוקול **Routing Information Protocol (RIP)** מմמש את DV
- פרוטוקול **Open Shortest Paths First (OSPF)** מממש את LS

**APRAnet**

איך נקבע את המשקלים על בסיסיהם המסלולים הקצרים ביותר ייקבעו? עם הינה ניתוב ברשות בשם **APRAnet** (1969). הניתוב היה באמצעות DV, אבל משקל הלינק היה נקבע בזמן אמת בהתאם לעומס. בהתאם הניתוב ניסה להיות דינמי ולפטור גם את שינוי הניתוב בזמן עומס (העדפת מסלולים אחרים על פני מסלולים עמוסים) בנוסף לפתרת בעיית ניתוב תüberות הפקטות.

**הבעיה:**

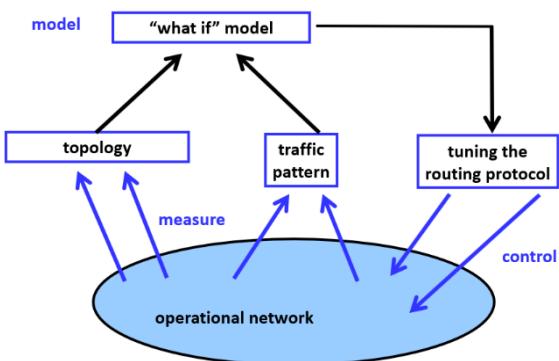
עם שינוי המשקלים גם המסלולים צריכים להשתנות בהתאם. זה יגרום לביעות רבות בתפקוד הרשת ולביצועים גורעים – במקרה הגורע ייווצרו לולאות של פקטות ונאבך קשריות, פקטות של אותו flow לא יגיעו בסדר המתאים. זה גורם למסלולים קצרים להיות לא אטרקטיביים ולבחרת מסלולים ארוכים במקום.

**הו מספר ניסיונות לפתרת הבעיה:**

- ב-1979 ניסו לעبور מ-DV ל-LS כדי להימנע מלולאות.
  - בנוסף ניסו למצער את שינוי המשקלים ולעדן לעיתים לא תכופות כדי להימנע משינויים תכופיים ומתחתיות.
  - בסוף הווחלט לא להסתבל על תעבורה בזמן אמת כאשר משנים את המשקלים. קודם המשקלים נקבעים, ולאחר זמן מה בוחנים אם יש צורך לשנות אותם.
- مكان נגיעה לנושא הבא – איך נקבעים המשקלים?

**Traffic Engineering**

Traffic Engineering היא קביעת הפרמטרים של אלגוריתמי הניתוב ברשות (המשקלים על הצלעות) כדי לשפר את הביצועים, נעשית ע"י מנהל הרשת ובשכבה 3. המשקלים יהיו סטטיים והמסלולים ישתנו בפרק זמן ארוכים. שינויים בזמן יהי' ברמת האפליקציה ויהיו שינויים בקצב שליחת הפקטות.

**מנהל הרשת מבצע מספר פעולות:**

המטרה היא להבין את טופולוגיה הרשת ואת ביקושי התüberה, בהתאם לה לשנות את הניתוב ע"י שינוי הפרמטרים והרצת פרוטוקולי ניתוב, וחזר חלילה.

1. **Measurement** - מדידת טופולוגיה הרשת ואת דפוסי התüberה.
2. **Network-wide models** - בניית מודל של if (מה יקרה אם נשנה את הפרמטרים) על בסיס ייצוג הטופולוגיה וההתüberה.
3. **Network Optimization** - הרצת אלגוריתם שיגרום לתüberה האופטימלית ברשת וחשב הפרמטרים והמגבלי על בסיס הנסיבות הקיימות.
4. נבדוק את דפוסי התüberה והטופולוגיה, נשנה בהתאם ונחזר לשלבים הקודמים.

**Multicommodity flow**

ראינו באלו  $\leftarrow \text{Max flow} = \text{Min cut}$ .

ברשת לא נרצה לפטור זרימה מלקסימלית כי בנגדוד לרשות זרימה שראינו באלו אין לנו מקור אחד ובור אחד, אלא הרבה גורמים שמדוברים עם הרבה גורמים. לכן נסתכל על הכללה של האלגוריתם – **Multicommodity flow**.

**הקלט:**

- גراف לא מקוון ממושקל  $G(V, E, c)$ .
- מטריצה  $\{d_{ij}\} = D$  שubah כל שני קודקודים  $i$  ו- $j$  מפרטת את ביקוש התüberה ביניהם.

**הפלט:** זרימה ברשת בהתאם לדרישות מסוימות.

ישן מטרות שונות שאפשר להשתמש באלגוריתם בשביל:

**Maximum-multicommodity-flow:** מיקסום התעבורה ברשות. באופן פורמלי נרצה למקסם את  $\sum_{v \in V} f_v$  אשר הוא סכום התעבורה שנשלחה ע"י קודקוד  $v$  ללא חריגה מהקibilitות ובהתאם לדרישות. יש ניצול מקסימלי של הרשות אבל תיתכן הרשבה.

- ברגע  $\text{Max flow} = \text{Min cut}$ , אין בעיה דואלית ל蹶ה זה.

**Minimize Congestion:** מינימזיר את העומס ברשות. נרצה להעביר את כל התעבורה דרך הרשות, כך שמנמזהר את הזרימה בリンク שמצוין היכן גרוע (הכי עמוס), ומצבו יהיה הכי טוב. פורמלית, נמזרע את  $\max_e f_e / c_e$ , אשר  $f_e$  היא הזרימה בצלע  $e$ . בעצם נרצה את הזרימה בה הניצול של הצלע ביחס לקיבולת הואה הכי גבוהה. ככל שהמספר הזה נמוך יותר כך הזרימה ברמת יותר טובה.  
**במקרה זה הזרימה יכולה לעבור את הקיבולות!**

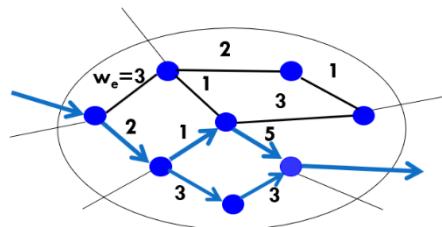
בגדול הבעיה ניתן לפתרה בזמן פולינומילי, בין היתר באמצעות תכנון לינארי. בפועל כשאנחנו פותרים את הבעיה אנחנו מחשבים את המסלולים ואת זירמת התעבורה במסלולים. אבל-ב-*IP routing* משלבים בלי להשתמש באלגוריתם כמו שאנחנו מכירים אותו.

## Lecture 10

### Optimizing (Static) Link Weights

בharaczaה הקומת דיברנו על Traffic Engineering בשבבה 3 – שינוי משקל הצלעות כדי להבטיח מעבר אופטימלי ברשת ונעשה ע"י מנהל הרשת בכל כמה זמן. בשבבה 4 מנסים לשנות בקצב השילחה ודבר על כך בהמשך. כמו כן דיברנו על **multicommodity flow** שהוא "הכללה" של **max flow** ומשיג זרימה אופטימלית ברשת שבה כל קודקוד יכול להיות מקור ובור, וניתנת מטריצה שמצוינת מה ביקוש התעבורה בין כל קודקוד  $i$  לכל קודקוד  $j$ ,  $\{d_{ij}\} = D$ . יש מי פלטים אפשריים ל-flow  $i$  שפהם מושגים ב- $D$ . בקרה שלנו לא יכולים להגיע לפתרון האופטימלי כי אנחנו בונים בדומה לוגיקת הטעורה ברשת ו踌ווער העומסים ברשת (ונצול הילוקום). בקרה שלנו לא יכולים להגיע לפתרון האופטימלי כי אנחנו בונים בדומה לוגיקת הטעורה ברשת ו踌ווער העומסים ברשת (ונצול הילוקום).

אנחנו יכולים לדעת מראש תעבורה תזרום דרך דרך הרשת שלנו, אבל לא נדע איך לקבוע את המשקלים על הצלעות כך שהמסלולים שיוציאו ע"י האלגוריתמים יהיו טובים. מנהל הרשת ירצה לקבוע משקלים על הצלעות באופן סטטי, ואוטם מסלולים יגידו מי יהיו הקודקודים אליהם נשלח תעבורה לכל יעד – next hop. next hop – next hop – next hop. ? יש מידע כדי לעבור במסלול הקצר ביותר לעד מסויים.

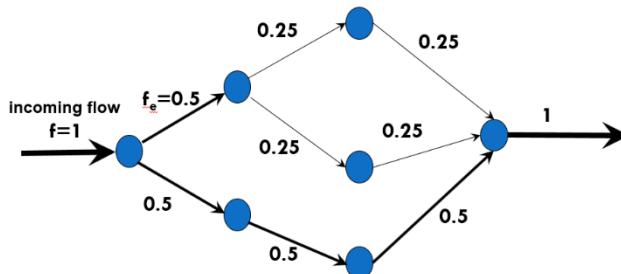


היררכיה אפשרית לקביעת משקלים סטטיות אבל לא אופטימלית היא קביעת המשקלים להיות 1 חלקו הקיבולת – כך שאם יש קיבולת נמוכה המסלול יהיה גבוה, ולהפוך.

### Equal Cost Multipath (ECMP)

ECMP הוא פרוטוקול שמנצל את העבודה שיכולים להיות כמה מסלולים הבין קצרים בין שני קודקודים ברשת, **ומפצל את התעבורה בין כמה next hops** על המסלולים הבין קצרים מוקודק אחד לשני, כך שההתעבורה עוברת בין כל המסלולים הבין קצרים.

- כל קודקוד ( $i$  נתב) מחשב את המרחק הבין קוצר לכל קודקוד  $j$  ומגדיר סט של קודקודים  $j_{i,xt}$  שהם  $next\ hops$  של קודקוד  $i$ .
  - קודקוד  $i$  יחלק את התעבורה אל  $j$  באופן שווה בין  $next\ hops$  ( $next\ hops$  –  $j_{i,xt}$ ).
- בדוגמה הבאה כל הקיבולות והמשקלים על הצלעות הם 1, והתעבורה שוחצים להעיר ברשת גם בגודל 1. יש שלושה מסלולים הבין קצרים מוקודק המקור לקודקן היעד באורך 3, וניתן לשימושם לבבדל בין חלוקת התעבורה ל- $next\ hops$  על מנת למסלולים הבין קצרים: בין הקודקן ליעד ישנן שתי  $next\ hops$ , אבל שלושה מסלולים הבין קצרים. ניתן לראות שהפיצול הוא אכן לפני  $next\ hops$  בהתעבורה מתפצלת לחצוי.



**קביעת משקלים לצלעות – בעית אופטימיזציה**

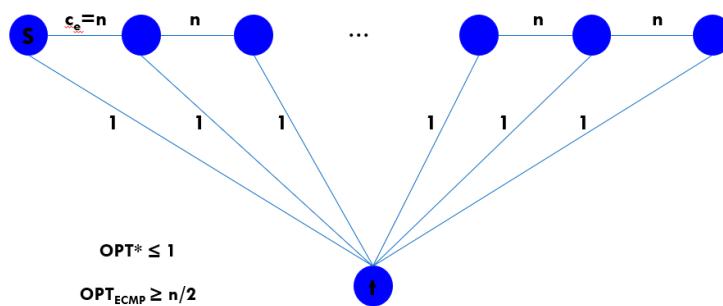
בהתנות:

- גראף עם קיבולות ( $c$ )
- מטריצת ביקוש תעבורות  $D = \{d_{ij}\}$

נרצה לקבוע משקלים לצלעות כך שהזרימה ברשת תהיה אופטימלית (ביחס למטרה של ה-flow  $\text{multicommodity}$ ) וגם תפוץל עפ"י ECMP.

הבעיה - לא תמיד זרימה אופטימלית ברשת תקיים ECMP. במקרה שבו הזירה ברשת צריכה לפחות  $c_e = n$  bytes congestion minimization, ככלומר זרימה מקסימלית שמנזעת את העומס המקסימלי על צלע באשר ניתן לחרוג מהקיבות, האם תמיד קיימים משקלים לצלעות כך שהזרימה ש-ECMP מייצרת היא אופטימלית? אם לא, ניתן לקרב את הזירה לזרימה אופטימלית? התשובה היא שלא.

בבחן במקרה הבא: נרצה להעביר ברשת מ- $s$  ל- $t$  תעבורת בגודל  $n$ . אבל ברשת זו, לא משנה איך נקבע את המשקלים על הצלעות, לא נגיע לזרימה המקסימלית ( $\text{OPT}$ ) ברשת עם ECMP.

 $n = \#\text{vertices}$ 

בזרימה אופטימלית בלי ECMP היינו מצליחים להעביר את כל ה- $n$  תעבורת, בלי לעבור את הקיבות כך שהעומס יהיה 100% (היאנו מעבירים מ- $s$  1 דרך הצלע הישירה ל- $t$  ועוד 1 דרך הצלע הסמוכה, ואז הקודקוד השני היה מעביר 1 דרך הצלע הישירה ל- $t$  ועוד 2 דרך הקודקוד שlid ובן הלאה).

$$\text{OPT}^* \leq 1$$

ב-flow max, במקרה שאנו אמורים לעבור את הקיבות, ישנו שלושה מקרים אפשריים עם ECMP:

1.  $t$  הוא ה-hop היחיד של  $s$ . במקרה זה יעביר  $s$  ו- $t$  זרימה בגודל 1.
  2.  $t$  הוא לא  $k$ hop next של  $s$ . הבעיה והמקרים תהיה זהה עבור הקודקוד הסמוך ל- $s$ .
  3. גם  $t$  וגם השכן השני של  $s$  הם  $k$ hop next. במקרה זה ECMP הזרימה תהיה 2 כי הפיצול נדרש להיות שווה.
- ← עבור כל המקרים לא הגיענו לזרימה מקסימלית לא משנה איך היינו קובעים את המשקלים, למרות שהצלע ה- $s$  עמוסה לא נושא יותר מהקיבולת שלה.

במקרה שאנו ב-congestion minimization, אנחנו לא מוגבלים לקיבות, אבל נסתכל על העומס בצלע ה- $s$  גרועה – כמה היא יכולה לשאת, נרצה שהוא לא תישא יותר מהקיבולת שלה.

נזכיר שוב שלושה מקרים אפשריים שהשליחה בהם תהיה כמו ב-flow max אבל עם שליחת כל יחידות הזרימה:

1.  $t$  הוא ה-hop היחיד של  $s$ . במקרה זה יעביר  $s$  ו- $t$  זרימת  $1/n$ .
2.  $t$  הוא לא  $k$ hop next של  $s$ . הבעיה והמקרים תהיה זהה עבור הקודקוד הסמוך ל- $s$ .
3. גם  $t$  וגם השכן השני של  $s$  הם  $k$ hop next. במקרה זה ECMP נעביר 2 /  $n$  דרך הצלע שבין  $s$  ל- $t$  וכך שיהיה עליה עומס של  $2/n$ .

כלומר הפתרון האופטימלי שבו יש ECMP יהיה  $2 / n \geq \text{OPT}_{\text{ECMP}}$ , זאת אומרת שם נכפה את ECMP ברשת בהיה חיבורים שתהייה צלע שתישא יותר מהקיבולת שלה.

נרצה שהמשקלים יישגו זרימת ECMP שבהיא קירוב של הזרימה האופטימלית. סיבוכיות אופטימיזציה משקלים לילינקים בגין עם זרימה מקסימלית – NP קשה 😞 בעצם כל כיסיון למציאת עומס מינימלי בראשת זרימה עם הגבלה על הקיבולת הוא NP קשה, לא נוכיח. בגלל אי קירוב גם לא נוכל למצוא משקלים שמקבילים את המשקלים האופטימליים בזמן פולינומי. לא קל להחליף את ה-ECMP ( מבחינות אבולוציוניות וכן בغالל יעילות, אין לולאות, משאים וטיפול בנפילות) ולכן נשתמש בהיוריסטיקות.

**היויריסטיקה אפשרית:** קביעת משקלים על בסיס שינויי מקומיים ובחינת התוצאה. למשל אם בリンク מסוים עברת יותר מדי תעבורת נשנה את המשקל שלו ונראה מה התוצאה – אם היא טובה נשאיר את המשקל. הפתרון לא מקרב אותו לפתרון האופטימלי אבל בהקשרים מסוימים יכול לעבוד היטב:

- חישוב מהיר של משקל הלינקים.
- החישוב מחדש מאפשר התמודדות עם תקלות.
- ביצועים טובים וחסית לפתרון האופטימלי.

#### ECMP Hashing

ב-ECMP במצבות אין-Robin RoundRobin – פיצול שווה של התעבורה, בغالל כל מינו אילוצים בשכבה האפליקציה. זה עלול להביא לחוסר סדר בהגעת הפקודות לעד. **נרצה שככל הפקות של תהליך כלשהו יעברו באותו מסלול כדי שיגיעו בסדר הנכון.** בשביל זה יש שימוש **בפונקציית Hash** שממפה פקודות (מייצגות בהרבה ביטים) ל-next hop (פחות ביטים) בהתאם להתפלגות שווה, עם מחיר חישוב נמוך ודרמטיוני.

מעבר לכך, כפי שציינו נרצה שפקות של אותו תהליך, flow יעברו באותו מסלול. לשם כך האש יהיה לכל flow. המיפוי יהיה על **חלקם ב-packet header** שקבועים אצל **פקות השירותים** לאותו flow. למשל IP source, dest IP, next hop.

#### Transport Layer

געבר לדבר על שכבה 4 – **שכבה ה-transport**. נתיחס למסלולים באופן סטטי שנקבעו מראש בשכבה 3, כאשר בשכבה זאת מנסים לפתור את הצפיפות של התעבורה. השכבה מאפשרת **לוגית בין תהליכיים** שרצים על משתמשי קצה (לעומת שכבת הרשות שמאפשרת תקשורת לוגית בין משתמשי קצה).

- **הצד השולח:** המידע שהשכבה שולחת מגע מהאפליקציה. השכבה מפרקת אותו לSEGMENTS, עוטפת אותו ומעבירה לשכבה 3.
- **הצד מקבל:** פרוטוקול בשכבה מקבל SEGMENTS של הודעה אחת, הוא מרכיב אותם בחזרה להודעה השלמה ומעביר אותה לאפליקציה.
- **הדרך בה האפליקציות מתALKות, כל אפליקציה פותחת socket ולו יש ID אליו המידע נשלח.** **Socket**

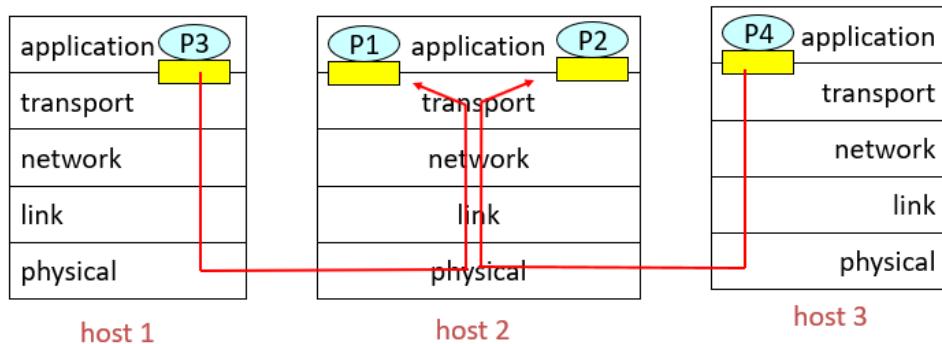
#### הפרוטוקולים המרכזיים:

1. **TCP** – ההפך-UDP. מה שמריצים משתמשי הקצה כדי להתמודד עם תקלות וחוסר וודאות איתן ה-UDP לא מתמודד.
  - אמין
  - בקרה על העומס
  - בקרה על נפילת פקודות וביעות בשילוחה
 מימוש עיקרונות הקצה לקצה.
2. **UDP** – שליחת פקטה מאפליקציה של משתמש קצה אחד לאפליקציה של משתמש קצה שני, בלי בדיקת הצלחה. אף פרוטוקול לא מבטיח שליחת פקטה בזמן מסוים ולא יעיכבים.

#### Multiplexing/demultiplexing

בכל רמה יש אנקפסולציה (Multiplexing) ודיקפסולציה (Demultiplexing) של פקודות. בשכבה 4: **Multiplexing** אצל השולח – איסוף מידע מכמה socket ועטיפתו בך שיופיע בסוף לפקטה לשילוחה. **Demultiplexing** אצל המქבל – העברת פקודות שהתקבלו ל-link הנכון.

מהתרשים לדוגמה, נניח שתהילך P3 ב-1 host מעביר מידע לתהילך P1 ב-2 host, ותוך כדי תהילך P4 ב-3 host גם מעביר מידע לתהילך אחר P2 ב-2 host. ההודעות נשלחות דרך ה-socket (המלבן הצהוב). שכבת ה-transport 2 עשויה את ה-**demultiplexing**, העברת פקודות מ-P3 ל-P1 ופקודות מ-P4 ל-P2.



### איך demultiplexing עובד?

הפקטה שSEGMENTATION-OF-DATAGRAM מגיעה משכבה האפליקציה ומכילה ב-**header**:

- כתובת IP (של-host) של המוקור והיעד.

SEGMENTATION-OF-DATAGRAM שבו ה-source port וה-destination port ברמת האפליקציה, ושכבה ה-transport צריכה להכיר את הפורט

.demultiplexing הרלוונטי כדי לאפשר

### Connectionless and Connection-Oriented Demultiplexing

- כאשר אפליקציה רוצה לתקשר בפרוטוקול ה-**UDP** היא פותחת UDP socket שמוגדר ע"י

Dest IP address .1

Dest port number .2

בלומר ב-UDP אין הבחנה בין המוקור ממנו המידע הגיע. עבור UDP ספציפי ששולח פקטה, הוא מאופיין רק ע"י היעד אליו הוא שולח ואין לו אפיון עצמי,ומי שקיבל את הפקטה לא ידע מי שלח אותה.

באשר מתקבל transport, שכבת ה-transport תעביר אותו לפורט הרלוונטי לפי Dest .2 בלי תלות במידע אחר שנמצא בפקטה (פקודות עם זהה אבל IP dest port/source port/source port/shall לאותו פורט).

- לעומת TCP, פרוטוקול ה-TCP כולל בהגדרת ה-socket גם מידע על המוקור:

Source IP address .1

Source port address .2

Dest IP address .3

Dest port number .4

המשתמש שמקבל פקחת TCP משתמש בכל המידע כדי להעביר אותה לעד הרצוי. ככלומר, תהליכיים שונים יריצו על חיבורים שונים ב-TCP, בניגוד ל-UDP, יש מידע על מה שקרה בשני הקצוות זה מה שמאפשר שיחה.

destination IP/port ← משותף לכל מי שחולק אותו IP/port .destination

TCP socket ← משותף לכל מי שונה בפרמטרים הראשונים.

### TCP Connection Management

תהליך השיחה ב-TCP.

יש שני משתמשים: TCP sender ו-TCP receiver .TCP

- לפני קיום השיחה שני המשתמשים "מבוססים" אותה – כל משתמש שומר מצב שקשור לשיחה.

- משתמשים מתחנים:

- # Seq - מציין את מספר הפקטה שעוברת

- Buffer

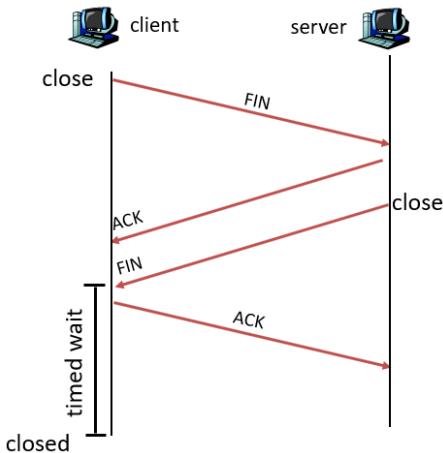
- ה-client מתחילה את השיחה (handshake) עם ה-server שמאשר את החיבור.

### Three Way Handshake

- אתחול השיחה לפני העברת המידע בפרוטוקול.
1. ה-client שולח סגמנט בשם **SYN TCP** ללא דטה ומתחל את המשתנה `# Seq`.
  2. ה-server מקבל את הסגמנט ומגיב ע"י שליחת סגמנט בשם **SYNACK**. הוא מקצה באפר לשיחה ומתחל את המשתנה `# Seq`.
  3. ה-client מקבל את הסגמנט ומחזיר סגמנט **ACK**, שיוביל לבסוף לשלול מידע.

# Lecture 11

TCP



## TCP Connection Management – סגירת השיחה

דברינו על כך שלעומת פרוטוקול ה-**TCP**, ב-**UDP** מתנהלת שיחה. כדי להתחיל שיחה צריך לאתחל את המצלב המתאים – ביסוס ע"י פניה למטרים אחרים, קיבלת אישור, שמירת מצב השיחה ופתחת באפר כדי לנהל אותה. המשוחחים יחויזקו משתנים: מזהה מספר הפקטה שנשלחת (לרוב יתחילה מספר רנדומלי מסיבות של אבטחת מידע), באפר לשמרות פקודות. תחילת השיחה מתחילה **three way handshake**.

- בສגירת השיחה, ה-client סוגר את ה-socket שלו. לאחר מכן:  
1. ה-client שולח TCP FIN סגמנט לשרת.

- .2. השרת מקבל את הסגמנט ומגיב עם ACK. הוא סגור א-client שולח ACK בחרזה.
- .3.

שני הצדדים מחררים את המשאים שהקצו בסיום השיחה.

## UDP overview

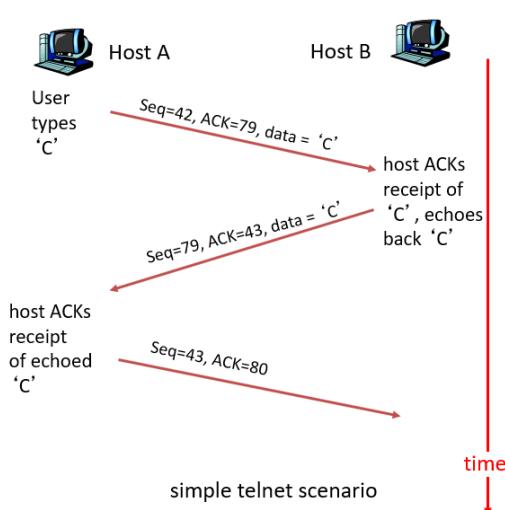
Connectionless, אין את שלב handshake כי לא מתנהלת שיחה בין שני גורמים. כל סגמנט של UDP הוא בפני עצמו, אין רצף של שליחת.

למרות ש-UDP פחות בטוח, למה בכלל זאת נרצה להשתמש בו?

1. בגל שהוא connectionless, הוא מותאים למנגנוןים מסוימים (כמו ה-DHCP).
  2. פרוטוקול פשוט לעומת TCP, אם רצים לשלוח פקטה אחת (למשל ב-DNS) אין צורך להקצות הרבה משבבים.
  3. ה-Header של הסגמנט קטן.
  4. אין ויסות של הקצב השיליחה ולבן הוא יכול "להפיצו" במידע בלי הגבלה.

## TCP overview

- תקשורת דו צדדי בין שני תהליכים.
  - תקשורת אמינה - פקודות שלא הגיעו לעדן ישלחו שוב. בנוסף, תנשה למכוון מצב שפקודות לא הגיעו עד (למשל ע"י האתת קצב השיליחה).
  - התמודדות עם拥塞 control & congestion control. flow control
  - duplex Full - בכל פקטה שנשלחת יש, בנוסף למידע שורצים להעיביר, גם מידע על השיחה עצמה (למשל ACK שקיבלו את ההודעה).
  - במתוך הדטה המועבר מוגבל ל-MSS maximum server side-length.



איזה מידע מועבר בPROTOCHOL TCP?

**בעת שיחה, מעבר להתחלה ולסיום שכבר ראיינו, כל צד מעביר לצד השני את החזון הבא:**

1. # Seq - מספר של הפקטה שנשלחה (נאמר מאוחר יותר שהוא בעצם סופר כמה bytes עברו בהודעה).
  2. ACK - בהודעת ACK נשלח מספר הפקטה (# Seq) שהצד ששולח את הودעתה-ACK קיבל עד כה מהצד המקלט.
  3. הדאתה עצמן.

כראת דוגמה:

אך A ו-ב מנהליים ישיכר

צד B מקבל את ההודעה ושולח 'C' seq = 79, ACK = 43, data = 'C'. הוא שולח את פקטה מס' 79 (A) לאחר לו שהוא קיבל עד 79), ומכיון שמדובר קיבל את פקטה 42 מ-A הוא שולח ב-ACK שקיבל את הפקטות עד המספר 43. אחר כך A שולח את פקטה מס' 43 ועם ACK = 80.

## TCP Reliable Data Transfer

TCP מבטיח הגעת מידע אמין – פקטות שלא הגיעו לעדין ישלחו שוב, ופקטות שלא הגיעו לפי הסדר הנכון יאורגנו לפה הסדר. ה-ACK מאפשר את הבסיס לכך.

פקטה תשליך שנית אם לא התקבל ACK עבורה, וזה ייקבע לפה:

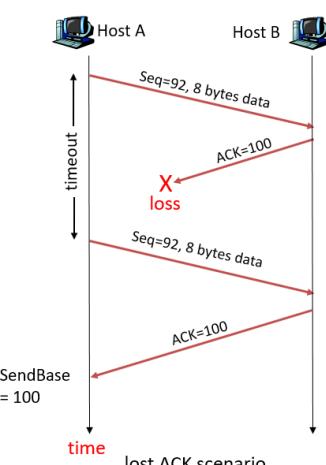
1. **Timeout** – טימר שמצין את הזמן שהינו רוצה לקבל בו את ה-ACK, התשובה שהפקטה הגיעה.

2. **Duplicate ACKs** – אם מתחילה לראות את אותו ACK שוב ושוב זה אינדייקציה לכך שיש פקטה שאבדה.

### Timeout – פעולה של הצד ששלח את הפקטה:

(נתעלם מרגע מזדים (duplicate ACKs))

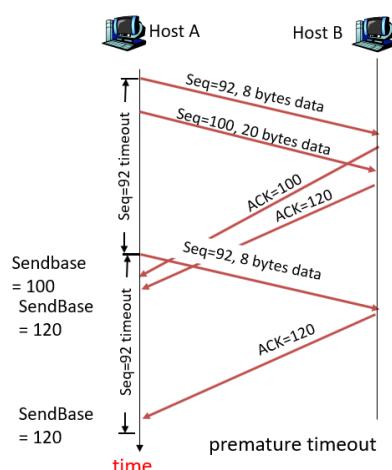
- כאשר נשלחת פקטה, אם הטימר לא התחיל לזרע, נאתחל אותו לאיזשהו timeout interval.
- אם עבר ה-timeout interval, קרה, ולא קיבלנו את ה-ACK עבורה הפקטה, זה "נורת אזהרה". הסוגמנט הביישן שלא קיבלנו ACK עבורה ישלח שוב תוך אתחול מחדש של הטימר.
- אם התקבל ACK עבורה אותו סוגמנט ששלחנו מחדש, נעדכן את המידע שלנו על פקטות עד אליו בר שמדד שהן התקבלו ביעד כמו שצריך.
- אם לא, מתחילה את הטימר שוב עבור פקטות שלא הגיעו.



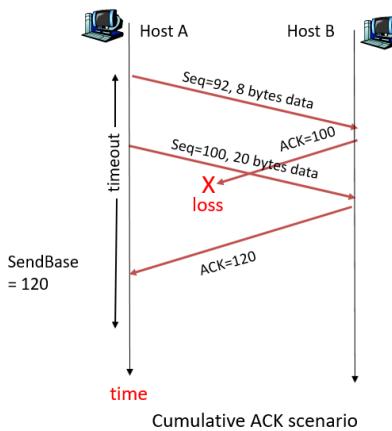
### דוגמאות:

A שולח סוגמנט במס' 92 עם 8 בתים.

B שולח ACK על 100 (כפי קיבל את הסוגמנט שהיה במס' 92 ועוד 8 בתים) אבל ה-ACK הזה נפל. מבחן A, הוא מתיישה וקובלtimeout על הסוגמנט הזה. בשזה קרה, והוא ישלח שוב את הסוגמנט ושוב יתחילה את הטימר (לא מופיע בשרטוט כי ה-ACK בן הגיע באינטראול זהה). הפעם B כן קיבל את הסוגמנט ו-A מקבל את ACK=100 וambil ששהוגメント הצליח להגיע ל-B, והוא-seq יעדכן אצל A ל-100.



וניה עבשו ש-A שולח שתי פקטות באותו זמן ומתחילה טימר. אחת (92) עם 8 בתים והשנייה (100) עם 20. שתי הפקטות הגיעו ל-B, ששלח ACK=100 וגם ACK=120 אבל היה עיכוב בהגעתם ל-A. הטימר ש-A הפעיל עבור שליחת הפקטות פקע בינייטים, והוא ישלח שוב את הסוגמנט הביישן שלא התקבל עבורה מידע שהגיעה – את 92. בזמן זה הוא קיבל תשובה מ-B, ACK=120 כי זה מספר הסוגמנט הביישן עדכני ש-B קיבל. למחרות שף פקטה לא נפלה, A ישלח שוב פקטות.



ובשיין, A שולח פקטה 92 שמתקבלת ב-B. בזמן שה-ACK=100 על אותה פקטה נשלח מ-B, A שולח פקטה נוספת נספחת 100. ה-ACK נופל בדרכו. B קיבל את הפקטה ה-100 ושלח עבורה ACK=120 ל-A. A יבין שככל הפקודות הגיעו למרוחת שלא התקבל ה-ACK עבור 92 ולא ישלח שוב פקטה חדשה.

#### Round Trip Time and Timeout

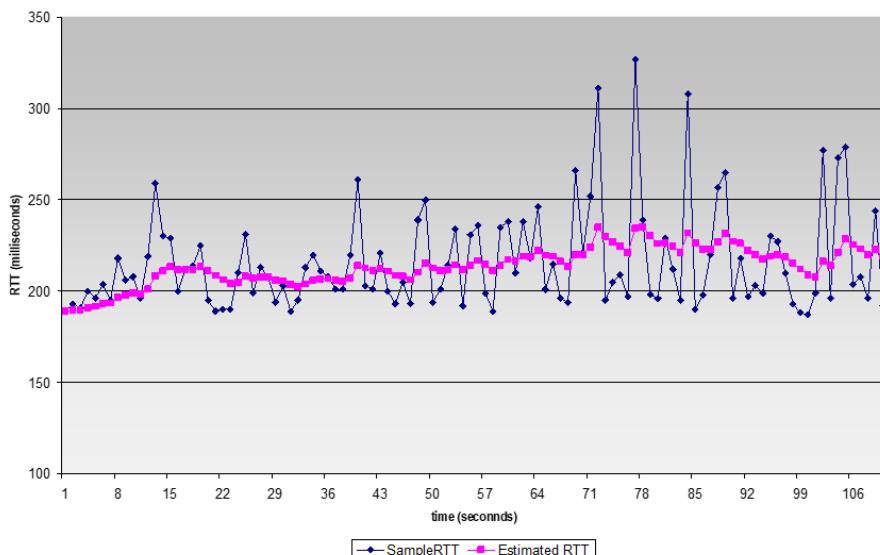
איך נקבע את ערך ה-timeout ?

- ערך timeout קצר מדי – יותר מדי יפקודות יישלחו מחדש.
- ערך timeout ארוך – הרבה זמן לשילוח מחדש.

הבעיה: RTT לא קבוע.  
בפועל TCP מחשב את ה-RTT המומוצע. כל פעם שנשלח סגמנט ומגיע עלייו ACK הוא מחשב את RTT, sampleRTT, וה-RTT הספציפי עבור הסגמנט, וביצע על הערכים ממוצע משוקלל - estimatedRTT.

חישוב ה-RTT יבוצע באופן הבא:

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{sampleRTT}$$



- בכל שזמן עבר ברז דעכת אקספוננציאלית ההשפעה של sampleRate.
- אלפא בדרכו יהיה 0.125 כדי לתת יותר משקל לממוצע המשוקלל מאשר לדגימה בודדת.

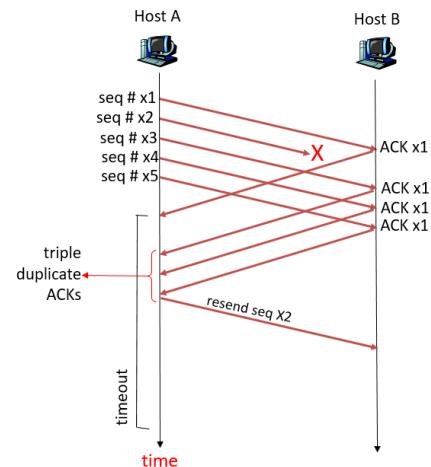
בנוסף נרצה להוסיף את סטיית התקן:

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

.0.25

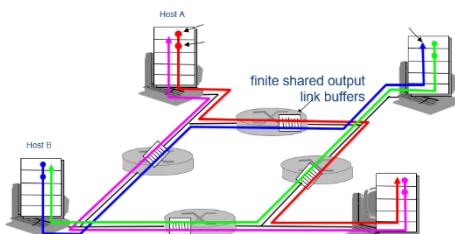
בר שלב סוף ה-timeout יקבע באופן הבא:

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

Duplicate ACKS

אם רואים שוב ושוב את אותו ACK, זאת אינדיקציה לנפילת פקטה.'Csudnv A שלוח 5 פקודות, ופקטה 2 נפלה בדרך. הוא קיבל על כל שאר הפקות את אותו ACK, שיצין ש-B קיבל נכון את הפקות עד  $x_1$ , ומשם סדר קבלת הפקות לא כמו שהוא צריך להיות. ההודעות האלה הן אינפורמציביות – נדע איזה פקטה נפלה, ונדע ששאר הפקות בן הגיעו. במקרה של timeout, יכול לנקח זמן עד שהפקטה שבאה תשלוח מחדש. במקרה של fast retransmit, הצד השולח ישלח ישר שוב ולא יচכה שהטיעימר יפוג – **duplicate ACKs**

## Congestion Control in the Internet

עקרונות של בקרה על עומס Congestion Control

**עומס** – קורה כאשר שלוחים יותר תעבורה ממה שהרשות יכולה לשאת. יכול להוביל לעיכובים ולאבדן של פקודות.

אם כמה קודקודים חולקים את אותו לינק, אם הם ישלחו הרבה תעבורה זה יגרום לעומס אבל אם ישלחו מעט מדי לא יהיה ניהול מרבי של הלינק.

Congestion Control vs. Flow Control

– בקרה על קצב השילחה כדי למנוע מהשולח לשלח יותר מידע مما שהמקבל מסוגל להכיל. הקצאות פותרים את הבעיה בינויהם כבר בשלב ביסוס השיחה.  
– בקרה על קצב השילחה כך שהשולח ישלח מידע באופן שמנצל את הרשות אבל לא מעmis עליה. **Congestion Control**

## Lecture 12

### TCP - Congestion Control in the Internet

#### עקרונות של בקרה על עומס

**עומס** – קורה כאשר שולחים יותר תעבורה ממה שהרשות יכולה לשאת.

**Congestion Control** – בקרה על קצב השילוח כך שהשלוח ישלח מידע באופן שמנצל את הרשות אבל לא מעmis עליה. **זכיר –**

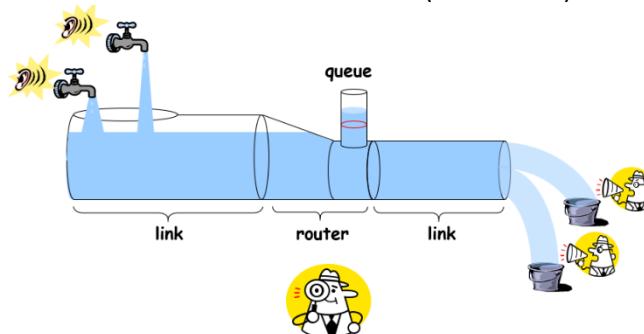
- המסלולים של הרשות קבועים (enschicht 3) ו-TCP מנסה לפתור את השאלה כמה מהר לשלוח. מפתרונות עבר שכרגע ה-TCP מיישם האינטרנט לא קורס תחת עומס, אבל בין קוראים ביצועים פחות טובים בגל כל. הבעיה היא שניתן לעשות caching למזה ששלוחים באינטרנט, ניתן לשימוש התוכן באיזשהו cache קרוב (למשל איזושה). סדרה).

שני גורמים שמשמעותם על ה-*end-to-end*:

1. הпротокולים הרצים בקצוות (TCP וכו')
2. מבנה הרשות שקובעת קיבולות, איזה פקודות נופות וכו'.

#### גישות לבקרה על עומס

1. **End-end congestion control** – אין מידע מהרשות על מה שקורה, קצה אחד מודע למה שקורה בתקשרות רק מהמידע שקיבל מהקצה השני (הודעות ACK). ממומשת ב-TCP.



2. **Network-assistant congestion control** – הרשות (נתבים) מעדכנת את הקצאות מה שקורה. ביטים מסויימים בפקטה מוקדשים לכך ומהווים אינדיקציה לעומס ולהזין קצב כדי לשולח, ומסומנים ע"י הנטב. קורה בד"כ בחווית שרתים, לא נדבר על כך בשיעור.

#### **End-end protocols**

בפרוטוקול זה המידע שהmidע שהמשתמשים בקצת מכירים הוא רק משתמשי קצה אחרים. אבל במקרים של עומס ברשות, מה שכך קורה ברטה ממושך במידניות של queuing, ככלים למכרה בהם הבادر עומס. למשל:

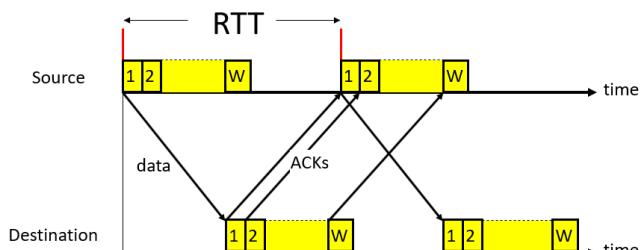
- **Droptail** – מתי שכבר לא נוכל לשולח פקודות חדשות יפלו.

- הפלת פקודות בצורה הסתברותית.

(לא עמוק מעבר)

#### Window Based Protocols

הדרך שבה TCP ממומש. בפרוטוקולים אלו יש חלון עם גודל מוגדר מראש, כך שהגודל מצין מה מספר הפקודות שניתן לשולח בלי לקבל עליהם ACK (פקודות באויר).



בشرطוט, המקור ישלח את פקודות 1 עד W אחת אחרי השניה, ולא ישלח עוד פקודה עד לסיום ה-RTT של החלון (הגעת ה-ACK). ברגע שהודעת ACK אחת הגיעה, זה אומר שיש לנו 1-W פקודות באוויר, ובגלל שגודל החלון הוא W נוכל לשולח את הפקודה הבא. • הפרוטוקול ידע שפקודה אבדה אם לא התקבלה עלייה • הودעת ACK. • ב-TCP משנים את גודל החלון בהתאם לעומס.

### קצב השילוח בהינתן גודל החלון:

$$\frac{W \times MSS}{RTT} bps$$

באשר MSS זה הגודל המקסימלי לפקטה ב-TCP.

השאלה היא איך קובעים את גודל החלון (W)?

- אם החalon קטן מדי,  $W < capacity$ , אז נצליח מלא של הרשות.
  - אם החalon גדול מדי,  $W > capacity$ , יש עומס.
- ← גודל החalon הוא דינامي כדי להתאים לשינוי העומס ברשות.

## TCP Protocols

-TCP יש כמה גרסאות שחלקו קיימות מסיבות היסטוריות, ועם הזמן נוספו מנגנונים שמתמודדים עם האינטראנט אחראית והם מותאמים לסביבות שונות או מיצגים אלגוריתמים שונים.

ישנם שני שלבים משותפים לכל הוראות אלה:

.1 Slow start (SS) – נרצה להגיע מהר לאוזור הנכון.

.2 Congestion avoidance (CA)

. – שלב לפיו נקבע מצב השילוח (SS או CA).

### TCP Tahoe

ב-TCP המקורי, הראשון עם בקרת עומס. מועד למנוע קריישה של האינטראנט, ומקיים את הרעיון הבסיסי:

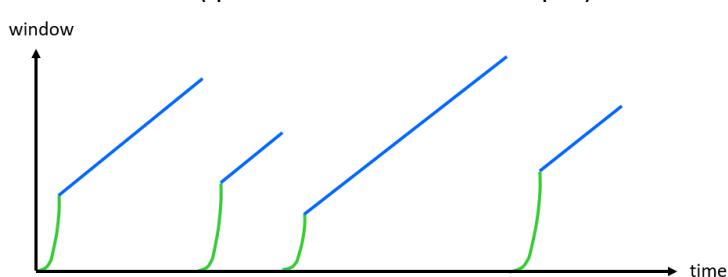
← כל עוד אין עומס, נעה לאט ובזהירות (באופן אדיטיבי) את גודל החalon.

← אם מתחילה לחות עומס, נוריד במהירות את החalon (באופן בפלי).

ונכנס למצבה כאשר W עולה מעל סף מסוים.

**Additive-increase-Multiplicative-Decrease (AIMD)** – פרוטוקולים שמקיימים את העקרון הזה נקראים

ב-Tahoe גודל החalon עולה בשלב SS אקספוננציאלית (גרף יירוק). אחרי שהוא עבר את הסף הוא נכנס למצב CA (כחול) והעליה נהיית לינארית. ברגע שנפללה הפקטה נהיה ירידה דרסטית וחזרה ל-SS (ניתן לראות מעבר מ-CA ל-SS בגרף).



SS: Slow Start  
CA: Congestion Avoidance

### שלב ה-Slow start:

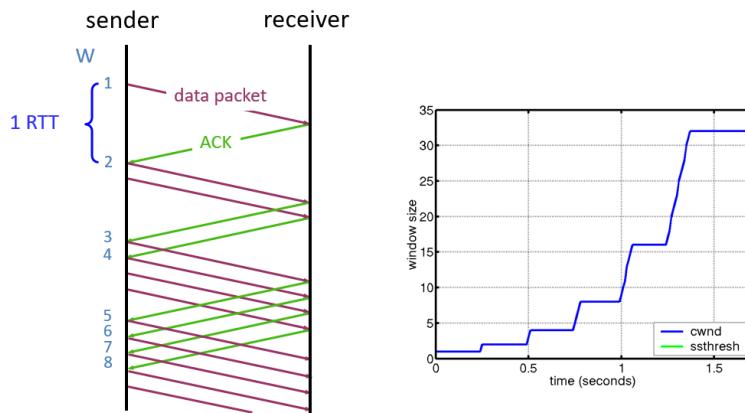
- מתחילה את גודל החalon להיות 1 = W.

- עם כל ACK שמתתקבל מעלים את גודל החalon ב-1 ( $W = W + 1$ )

אקספוננציאלי ב-RTT – בכל RTT מכפילים את גודל החalon. ב-RTT הראשון גודל החalon הוא 1, מוגעה הודעת ACK אחת וגודל החalon מתרעדם ל-2. ב-RTT השני גודל החalon הוא 2, אך מקבלים שתי הודעות

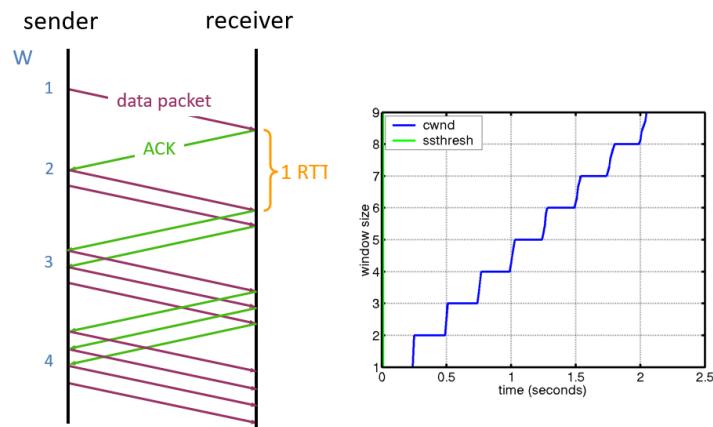
השני גודל החalon הוא 2, אך מקבלים שתי הודעות

- ACK, לכל הודהה zusätzlich מוסיפים +1 לגודל החלון כך שב-RTT הבא גודל החלון יהיה 4 וכן הלאה.
- באשר עברים את סף ssthresh נכנסים במצב CA.



### Congestion Avoidance

- על כל ACK ש מגיע, נעלם את ה-RTT בו- $\frac{1}{W}$ .  $W = W + \frac{1}{W}$ .
- גדילה לינארית ב-RTT (כל RTT, גודל החלון גדל ב-1).



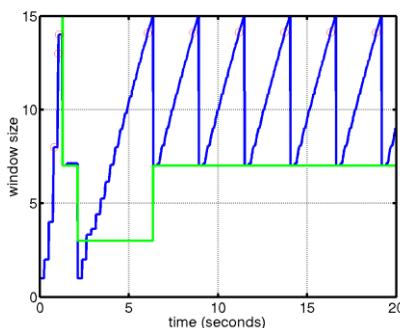
### נפילת פקודות

מכיוון שהמכנים שנפלו פקודות מעידה על עומס בראש, ברגע ש-Tahoe מזהה נפילת פקודות (ע"י timeout או duplicate ACKs) הוא מעדכן את החלון כך: מעדכנים את ssthresh להיות  $W/2$ . מתחלים שוב במצב של Slow Start שבו גודל החלון הוא 1. (ראינו בגרף הכהול יוק מוקדם שאחרי נפילה של פקטה, הגраф הכהול שמייצג את CA מפסיק והסתיים)

### TCP Reno

מנסה להתחמק מ-start slow במקרה של נפילת פקודות בשאיין הרבה עומס (בניגוד ל-Tahoe). **הרעיון:** הבחנה בין מקרים של Timeout לעומת מקרים של Duplicate ACKs. ב-timeout, התגובה תהיה כמו ב-Tahoe, אבל אם זיהינו Duplicate ACKs, זה אומר שפקטה אחת אבדה ובעצם העומס לא כזה חמור, אז ניתן אחרת.

← בגרף משמאלי ניתן להניח שיש רק Duplicate ACKs, העוקמה הירוקה היא ה-ssthresh.  
← התחלו ב-SS, עליינו עד שחוינו ואובדן פקטה וידנו דרישות חרזה במצב של SS. הסוף קטע בחצי כמו ש-Tahoe עושים.



- ← מנצח SS עליינו עד שעברנו את ה-ssthresh (בערך ב-2.5) והגענו לנצח CA.
- ← אבנו ב-CA וזיהינו נפילת פקטה (דרך ACKs duplicate): ירדנו בחצי בגודל החילון, אבל אחרי הירידה שלנו שיכינו גם את הסף, כך שעכשיו גודל החילון הוא מעל הסף בר שאנחנו עדין ב-CA.
- ← ההתחגות תהיה זהה בכל המקרים של הנפילות שזו ב-duplicate, אם הוא מתקיים timeout הימנו חוזרים לנצח SS.

אך באופן כללי, הпрוטוקול מתנהג באופן הבא:

$$1. \text{ לכל ACK נעדכן } W+ = \frac{1}{W}$$

2. על כל 3 duplicate שנקבל נעדכן:

$$\text{ssthresh} = W/2$$

$$W = W/2$$

:timeout

$$\text{ssthresh} = W/2$$

Enter slow start with  $W = 1$

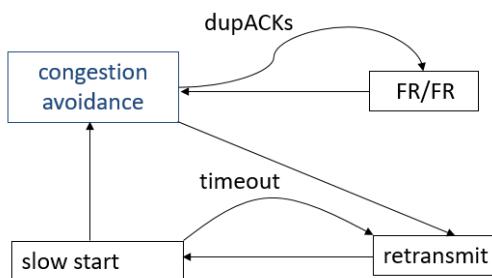
### Reno Overview – Basic Idea

AIMD probes available bandwidth –

fast recovery avoids slow start –

dupACKs: fast retransmit + fast recovery –

timeout: fast retransmit + slow start –



### Throughput Caponization of Window Size and RTT

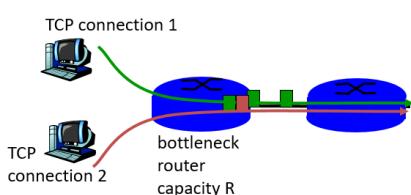
אינטואיזיה: מהגרף הכהול, ה throughput יהיה השטח שמתוחת לגרף חלקי כל הריבוע. בדוח את שלב SS.

נגיד  $W$  בתור גודל החילון כאשר פקטה אובדת, ו-RTT הוא הזמן שבו שלוחים את החילון.

- כאשר החילון הוא  $W$  הניתולתו הוא  $\frac{W}{RTT}$

- אחרי נפילת פקטה, גודל החילון יוזד ל- $2/W$  אז הניתולתו היא  $\frac{W}{2RTT}$

- הניתולות המומוצעת:  $0.75 \frac{W}{RTT}$



### Reno Fairness

בأنטראנט חולקים משאבם, אבל TCP מקבל החלטות לפי אותו גורם שמשתמש בו בלבד. נרצה לחלק רוחב פס בגודל  $R$  שווה בשווה בין  $K$  משתמשים קצה שהולכים אותו, אבל ה-TCP לא מתחשב בכך.

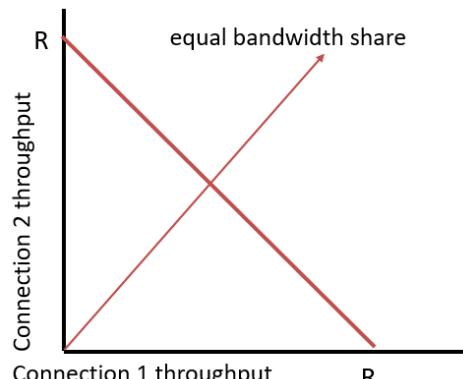
למה בכלל זאת Reno הוגן?

נסתכל על הגרף הבא -  $R$  מייצג את הקיבולת של הילינק. ציר ה- $x$  זה הקצב בו שלוח משתמש 1, ציר ה- $y$  הקצב של המשתמש השני.

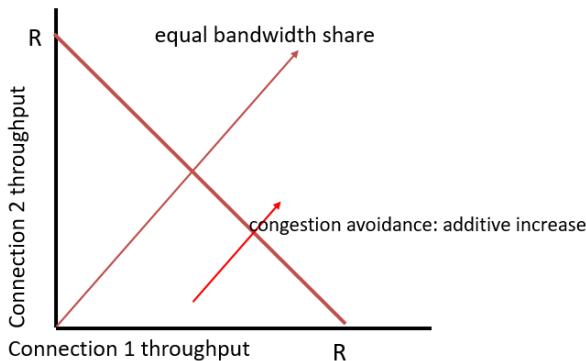
← הקו הדק הוא העקומה  $x = u$ , מייצגת הוגנות (שני הקודקודים שלוחים באותו קצב) אבל לא דוחוק אידיאלית מבחינהיעילות.

← הדק העבה הוא  $R = u + \alpha$  להפך – מייצג יעילות (ニカルו את כל הילינק) אבל לא הוגנות.

← מה שנרצה שיקורה יהיה מייצג בחיתוך שלהם – הוגנות ויעילות עם חילוקה של רוחב הפס לחצוי, ובנראה שלא נצליח להגיע לה במציאות.

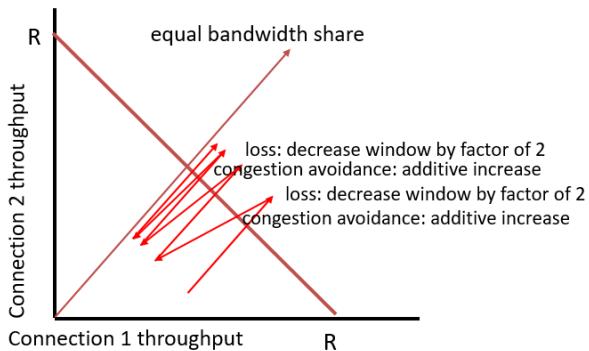


מה שכן, מה-AIMD, בגלל ה-CA, מי ששולח בקצב מהיר יותר "ישלם" יותר על נפילות וירידת הקצב מהר יותר, ובאייזהו שלב הקצב שלהם כמעט ישתווים.



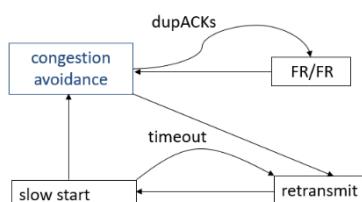
נסתכל על הדוגמה מוקדם כדי להמחיש את העניין:  
העוקמה שהთווסףה (בגרף משמאל) מתרת מה קורה במצבות. נניח  
שמשתמש 1 בgraf שולח מהר יותר מאשר 2, ושניהם במצב של CA.  
באייזהו שלב הקצב עולה את הקוו העבה, זה אומר שיש עומס, פקודות  
יפלו, והקצבים ירדו בחצי.

זה ימשיך וימשיך ומתוישה ותקרב לנקודת האידיאלית.



הבעיות בהוגנות:

- יש גורמים שלא משתמשים ב-TCP ויעדפו לשולח בקצב מהיר בלי הימנעות מעומס). אוטם גורמים יגרמו לביעות ולהוודה מיותרת של הקצב אצל גורמים שkn משתמשים ב-TCP.
- Parallel Connections – אפשר לעקוף את ההוגנות בכך שימושו שיריצה את רוחב הפס יפתח כמה חיבורים. נניח למשל שאת רוחב הפס חולקים שני משתמשים אחד פותח 10 חיבורים. משתמש 2 מקבל  $1/11$  מוחלט רוחב הפס, ומשתמש אחד  $10/11$  וזה בbijור לא הוגן.



**Vegas**  
**במה שונים וריאנטים של TCP?**  
בולום מחולקים ל-SS ו-CA זהים באוטומט המ מצבים, והם BD"כ זהים ב-SS.  
לרוב הם יהיו שונים בקרה ש-CA ממומש.  
אם נשווה את Vegas ל-Reno, ווגас מתיחס לא רק לאובדן פקודות אלא גם ובעיקר לעיכוב שלahn.

**הפרוטוקול:**

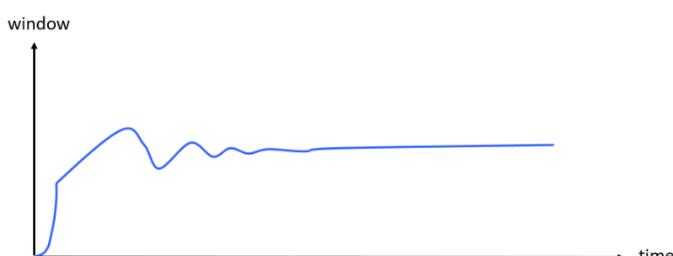
- CA במצב

- לכל ACK שמתתקבל:  
- אם  $\alpha < \frac{W}{RTT_{min}} - \frac{W}{RTT}$  (קצב השילחה ביחס ל-RTT היבי נמוך פחות קצב השילחה ביחס ל-RTT הנוכחי)  
 $W +$   
- אם  $\beta > \frac{W}{RTT_{min}} - \frac{W}{RTT}$  (קצב השילחה ביחס ל-RTT היבי נמוך פחות קצב השילחה ביחס ל-RTT הנוכחי)  
 $W -$

- לכל אובדן פקטה נעדכן

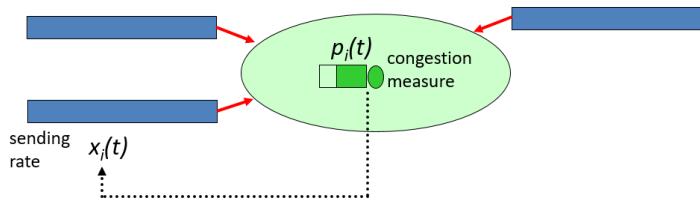
$$W = \frac{W}{2}$$

בחינת אינטואיציה, ה-RTT המינימלי מייצג את ה-RTT בעולם  
אידיאלי שבו אין עומס, וזה במאז קורה ברשות שלנו לאיזושהי  
תקופה. אם ההבדלים בין ה-RTT הנוכחי ל-RTT המינימלי גדולים  
משמעותית, אנחנו בעומס ונרצה להקטין את גודל החלון.



## Queuing at Routers

### Feedback Control



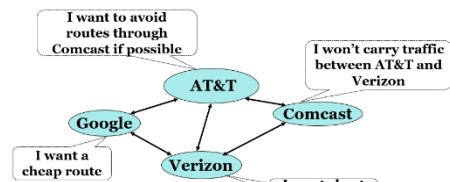
אמרנו שב-TCP הבקשה על עומס קורית בקצויות והרשות לא מעכנתה בימה שקורה בפנים. זה לא מדויק – לכל פורט בכל נתב יש באפר שפקעות שלא יכולות לצאת באופן מיידי צריבות להמתין בו. הקצויות יכולים לקבל סיגנלים על העומס באפר, אבל גם הנתב עצמו מתמודד עם העומס בדרכים שונות:

1. Drop Tail – כאשר הבאפר מתמלא פקודות פשוט נופלות, "לא עושים כלום".
2. RED – מפעילים פקודות איזושאי הסתברות.

**Implicit Feedback** – הרשות אומנם לא ספירה לקצויות חווים את השינויים שהנתב עשה בדרך עקיפה.  
**Explicit Feedback** – הרשות מחזירה פיידבק מפורש על המצב בה (למשל סימון פקודות בבייטים שמיעדים לבן).

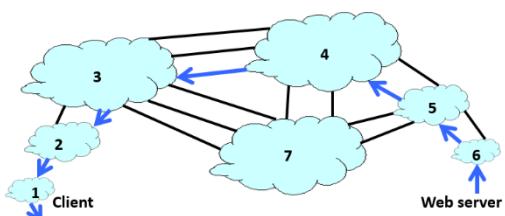
## Lecture 13

### מבנה האינטרנט



האינטרנט מורכב מרשתות קטנות ואוטונומיות של ארגונים **autonomous systems (ASes)**, כך שכל רשת כזו מנהלת את עצמה (ע"י מנהל הרשת) ובעלת מטרה שונה (רשתות של עסקים קטנים, של אוניברסיטאות, של תאגידי ענק וכו').

- הארגונים יכולים להיבדל אחד מהשני במדיניות ניתוב, מטרות בליליות ויחידות ניהול.
- כל רשת צריכה למשר פרוטוקול BGP כדי לתקשר עם שאר הרשתות ועם הדרישות השונות של כל רשת.



### :AS-Level Topology

1. כל קಡוק ברשת הוא AS, רשת עצמאית, ואנחנו נתעניין בקשריות שבין הקודקודים (בין הרשתות).
2. כל צלע היאLINK פיזי שמחבר בין נתב ברשת אחת לבין נתב ברשת אחרת. מידע שעובר מנתב אחד לשני יכול לעבור דרך בינה רשתות שונות.

← הרשתות מזוהות ע"י מספרם בני 16/32 ביטים – **AS numbers (ASNs)**.  
← ארגונים -AS זה לא אותו דבר, יכול להיות ארגון שאין לו בכלל ASes וארגון שיש לו במא.

### The Commercial Internet

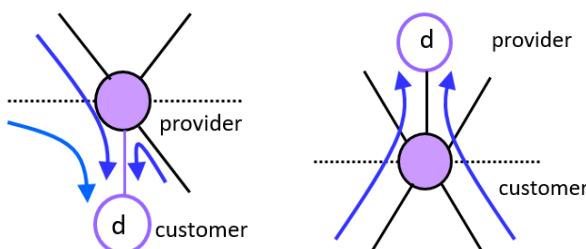
הקשריות באינטרנט היא מסחרית, ה-ASes חתומים על הסכמים מסחריים דו כיווניים ביניהם להעברת מידע שקבועים:

- איזה סוג תחבורה תעבור בהם
- מה יהיו היעדים
- סכום התשלומים

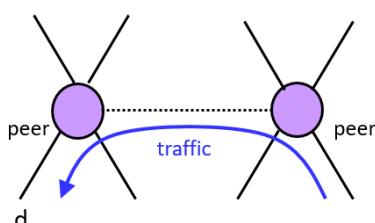
הלקוחות משלים ל-AS על קשריות אבל גם AS יכול לשלם ל-AS אחר על קשריות (בזק למשל לא מחוברת לכלום, גם צריכה לעبور דרך איזושהי AS).

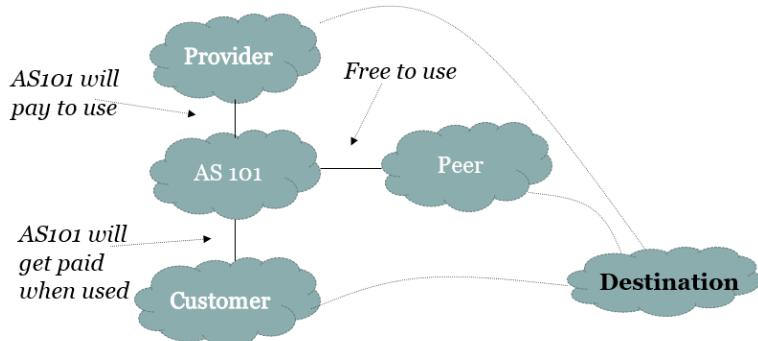
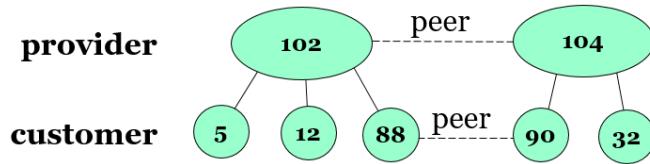
### שני סוגי יחסים כלכליים:

1. **Costumer-Provider**: לקוח ↔ ספק. הלקוח משלם ל-AS על קשריות (הלקוח צריך את הספק, יחסים א-סימטריים).  
הספק יעביר מידע מהלקוח לעד הרצוי, וכן יעביר ללקוח מידע שהגיע אליו ומוענד ללקוח.  
התקשורת יכולה להיות מלאה (לכל האינטרנט) או חלקית (לחלק מהינטרנט).



2. **Peering**: ספק ↔ ספק. מתאפשרת קישורויות בין ספקים שונים כך שהתחבורה יכולה לעבור בין הלקוחות שלהם.





בדוגמה באן, ה-costumer רוצה להגיע ל-destination-.  
הספק שלו הוא AS 101 והספק שלו הוא AS 102. Provider 101 משלם ל-Provider 102 כדי להגיע לעד אבל אופציה נוספת תהיה להعبر בחינם דרך ה-peer.

#### למה נרצה לאפשר peering?

- הפחיתה זמן המתנה
- הפחיתה עומסים
- הפחיתה עלויות
- מעלה את היתירות (הסיכוי לנפילות נמוך יותר והתקשרות בטוחה יותר)
- מעלה את הבקרה על הניתוב

#### היררכיה של ה-ASes

##### Tier-1 ISPs .1

- נמצאים בהיררכיה העליונה של האינטרנט
- לאף אחד מהם אין ספק
- מחוברים ביניהם בלבד עצםם
- מספקים קישוריות לספקים בשכבה שמתוחתם
- יש בינה שירות באלה

##### Tier-2 ISPs .2

- הספקים היחידים שלהם הם Tier-1
- מספקים שירותי לקוחות שלהם
- יש בהם אלפיים באלה

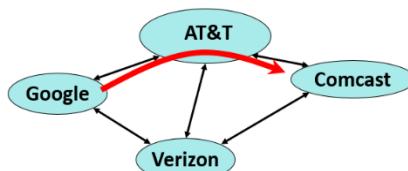
##### Stub ASes .3

- ASes- 85% מה-
- מחוברים לספקים
- אין לקוחות

בaprinciple העולם התקדם והמבנה יותר מורכב מזה, אבל לשם הפשטות נסתכל על המודל המתואר.

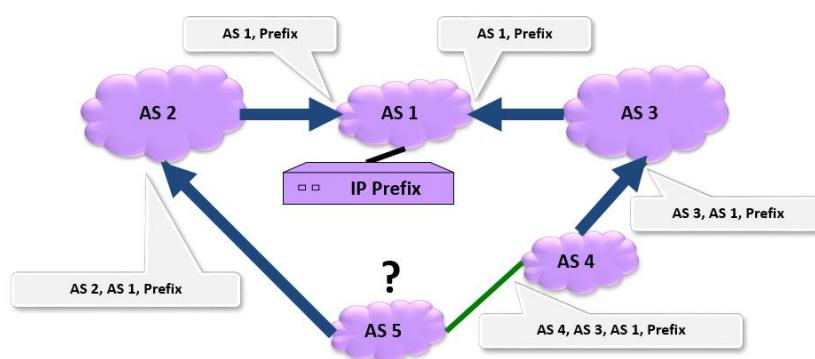
#### Interdomain Routing – BGP Protocol

– פרוטוקול שמקשר בין ASes ומאפשר ניתוב בין ארגונים. הוא יקבע את המסלול בו מידע יעבור מ-AS אחד לאחר.



בניגוד למה שראינו קודם, המסלול בין רשת אחת לשנייה לא מחושב לפי המסלול הימי קצר. מעורבים בעניין גם שיקולים כלכליים שיבולמים להשפיע על המסלול הרצוי, וחוץ מזה אין הגדרה חד משמעית למסלול הימי קצר – אנחנו לא יודעים מה קורה בארגונים עצםם וזה יכול להשפיע על המסלול.

גם כאן היעדים לשילוחו הם IP-prefix.



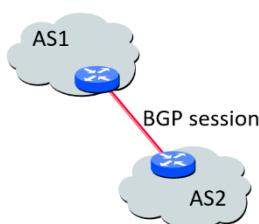
- נסתכל על דוגמה שמחישה את הניתוב בפרוטוקול:
- ← AS כלשהו (AS 1) מכריז על עצמו ועל תחילה ה-IP לשכנים שלו כך שידעו שאם הם רצים להעביר מידע למשהו עם בתובת שמתחלת במספר זהה הם יעבירו אליו.
- ← השכנים שלו (2 ו-3) יעבירו לשכנים שלהם שבדי להגיע לתחלת הרצiosa המסלול צריך לעבור דרך ואז דרך 1.
- ← באותו אופן פועל גם AS 4 כשהוא מקבל את ההודעה.
- ← נשים לב במספר 5 יכול ברגע לבחור בין שני מסלולים. לא יוכל לדעת באיזה מסלול בחר בגלל שימושם בבחירה גורמים שונים וככליים מלבד אורך המסלול.

#### באופן כללי, מה שביל קודקוד (AS) עושה:

- ← בבל רשת יש נתב BGP שמקבל מסלולי BGP מהשכנים.
- ← הוא בוחר למי מהם להתייחס ומסנן מסלולים לא רצויים – מדיניות ייבוא. את המסלולים הנבחרים ישמור ב-table.
- ← מתוך אלה שבחור יבחר את המסלול שהכי טוב עבורו.
- ← מפעיל את מדיניות הייצוא – למי מהשכנים יעביר את המידע על המסלול הבci טוב שבחור? איזה מסלולים ישן?
- ← שולח את המסלול לשכנים.

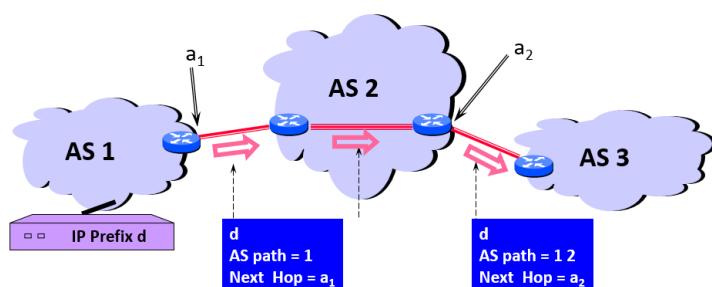
נקודות נוספות:

- פרוטוקול מבוזר.
- התהיליך עצמאי עברו כל IP-prefix.
- מועבר לשכנים כל המידע על המסלול כי יש גורמים נוספים שנרצה להתחשב בהם מעבר למספר הקפיצות.
- קודקוד מסוים יכול לבחור לא למספר לשכן שלו על מסלול מסוים.



ה-BGP קיים ברמת האפליקציה ורח מעל TCP. זה אומר שכארשר שני ASes רוצים לתקשר ביניהם, שני נתבי קצה בהם ידברו אחד עם השני ע"י פתיחת פורט (TCP port 179).

הודעת BGP כוללת את next hop שצריך כדי להגיע לאותו מסלול, לאיזה בתובת IP צריך לשולח.



כדי להימנע מלולאות, קודקודים יבדקו אם מסלול מסוים כבר עבר דרך מסוימת וכך יירידו אותו.

## BGP Routing Stability

### BGP Safety

בגלל שיש הרבה מילויים, לא כל פעם ידוחו המילויים המועדףים. במקרה זה כל AS ידוח רק על מילויים חדשים, שינויים במילויים או מחיקת מילויים.

הבחינה הזאת הפעוטוקול נחשב incremental, בהתחלה הקודקודים בוחרים את המילויים, מפיצים את הפירוט של המילויים (routing table) אם צריך וכאשר מילויים מתעדכנים הם מספרים לשכנים ורק על השינויים ולא על כל המילויים.

יש שני סוגי עדכוןים:

1. **Announcement:** החלטה על מילול חדש
2. **Withdrawal:** המילול לא זמין או לא רלוונטי זה גורם לכך שהפעוטוקול לא יציב.

נמchioש את זה בדוגמאות הבאות:

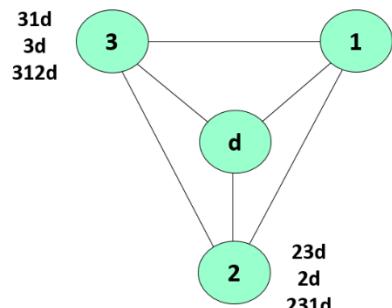
- יש איזשהו פורט עם תחילית d
- AS 1 שמעדיף מילויים דרך 2
- AS 2 שמעדיף מילויים דרך 1
- 1 ו-2 רוצים להעביר מידע ל-d

### מקרה 1:

- d מכיר על עצמו לשני השכנים, אבל ההודעה מגיעה ברגע רק ←  
ל-2.  
2. יבחר מילול שעובר דרך d (מחוסר ביריה). ←  
2 מסptr ל-1 על המילול ל-d, 1-2 יבחר ב-2 שהגדרכו אותו ←  
ל להיות המילול המועדף עליו.  
ברגע 1 מקבל את ההודעה ש-d שלח מקדם. 1 לא ישנה את ←  
המילול שלו כי הוא מעדיף מילויים שעוברים דרך 2, וכן גם ←  
2 לא ישנה את המילול, כי לא קיבל עדכון מהשכנים שלו.  
הגענו למצב יציב, אף קודקוד אין אינטראס לוז מהמילול שלו ←  
בלי שקדם אחר יוז.

### מקרה 2:

- d מכיר על עצמו לשני השכנים שלו שמקבלים את ההודעה בו זמן. ←  
1 ו-2 יבחרו במילול להיות דרך d ויעדכו אחד את השני:  
- 1 יעדכן את 2 שמלול שלו הוא 1d  
- 2 יעדכן את 1 שמלול שלו הוא 2d  
כל אחד יבחר בשני בgal העדפות שלהם:  
- 2 יבחר את המילול 21d  
- 1 יבחר את המילול 12d  
1 ו-2 יבינו שהmlinol שבחרו עובר דרכם ושוב יבחרו את המילול להיות ישירות ←  
ל-d:  
- 1 יבחר את המילול 1d  
- 2 יבחר את המילול 2d  
נקבל תנודתיות בין המילויים.

**מקרה 3:**

המקרים שראינו מוקדם תלוים בסנכרון של העברת ההודעות. נראה בעט מצב שבו לא נגיע ליציבות בשום מצב.  
בגרף הבא כל קודקוד יש מסלול מועדף ל-d (לפי הסדר).  
נניח בשלילה שהיא **מצב יציב** – כל קודקוד בווחר במסלול המועדף עליו. נניח שלפחות אחד הקודקודיים מחובר ישירות ל-d (לפי התרגול, המוחשה במצגת).  
 ← נניח שקודקood 2 מחובר ישירות ל-d והוא יעדכן את כולם שהמסלול ל-d הוא 2d.  
 ← 1 בווחר את המסלול 12d כי הוא המועדף עליו, הוא זמין ואנחנו במצב יציב.  
 ← ל-3 אין ברירה כי המסלול המועדף עליו לא זמין והוא יבחר את המסלול 3d שזאת האופציה השנייה, והראשונה לא זמינה.  
 ← 2 יבחר את המסלול 23d כי זה המסלול המועדף עליו והוא נהיה זמין.  
 ← סתייה **לכך שאנחנו במצב יציב** – זאת אומרת שיש אופציה יותר טובה ל-2 והוא לשולח דרך המסלול הראשון 2d, בגלל שאנחנו במצב יציב הוא בווחר בה זהה גורם ל-1 לשנות את המסלול שלו בהתאם, וזה היה גורם לעוד שינויים ולתנוודויות. המסלולים היו ממשיכים להתחלף בלתי התכוונות.  
 • קיבל את אותו דבר אם 1 מחובר ישירות או 3 (מופיע במצגת) ובכל מצב נקבל סתייה.

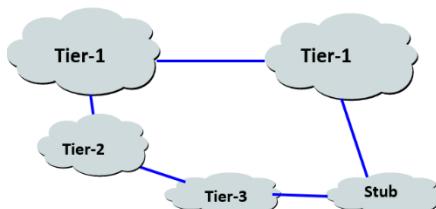
הבעיה בתנוודויות זו היא קשה לדיבוג, בלתי צפואה ויכולת להעמיס על האינטרנט (הרבה עדכוני BGP). זה יכול לפגוע באפליקציות שונות כמו סקייפ למשל.  
נראה שהייה תמיד ברשות (בסופו של דבר) מצב של **BGP safety** – התכנסות במצב יציב.  
אם יש יותר מצב יציב אחד ברשות, אז הרשות לא יציבה.

הוצעו כל מיני רעיונות כדי להגיע במצב זהה, שלא תמיד עובדים:

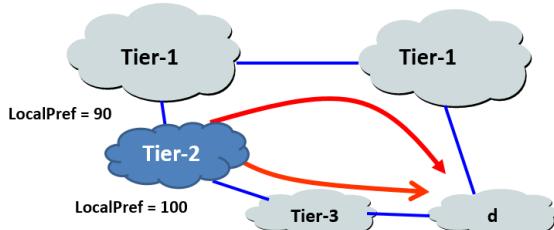
1. קנסות על תנודויות
2. מינימום של זמן עדכון על שינויים

**Gao-Rexford**

עונה השאלה למה האינטרנט כן יציב למחרות כל מיני מצבים כמו הדוגמה שראינו מוקדם?  
החוקרות Gao-*i*-Rexford מצאו שלושה תנאים שצרכיים להתקיים כדי לענות על השאלה:  
 1. **תנאי על הטופולוגיה:** אין מעגלים של לקוחות וספקים (ספק לא יהיה ל�� של הלוקו שלו).

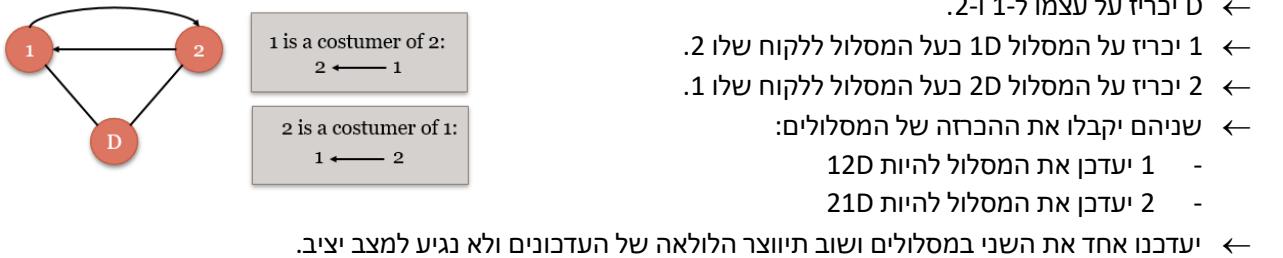


2. **תנאי על העדפות:** ספקים תמיד יעדיפו להעביר מידע דרך הלוקוחות שלהם ולא דרך ספקים או peers מהסיבה הפешטה שהлокוחות משלמים.

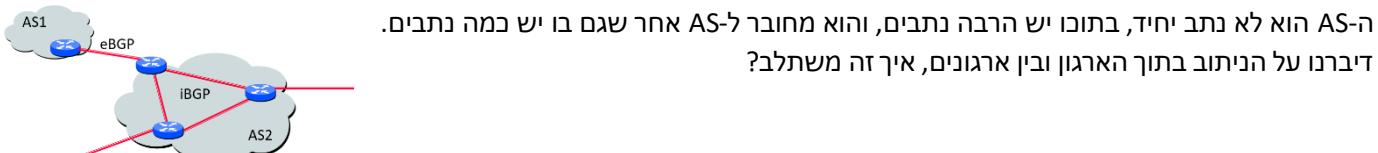


3. **תנאי על הייצוא:** ספקים לא יעדכנו במסלול את מי שלא משלם להם (providers, peers) אלא רק את הלוקוחות שלהם.

אם כל התנאים יתקיימו בראשת, מובטח שיש מצב יציב ו-BGP יתכנס אליו.  
ראינו בתרגול דוגמה לכך שאם התנאי על הטופולוגיה לא מתקיים אז אין מצב יציב –

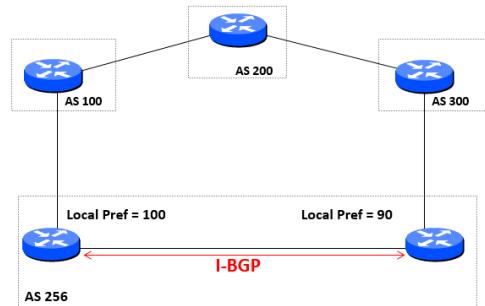


### Inter vs. Intra



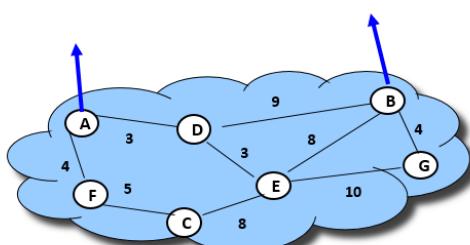
בדוגמה הבאה ל-256 AS יש שני נתבים שהוצרים להגיא לעד שנמצא ב-200 AS והעדפה של 256 היא ששניהם יעברו ל-200 דרך 300.

**כלל אכבע: תמיד שיקול חיצוני ועדף על פנוי שיקולים פנימיים.**  
נעדייף לנtab ב-256 דרך מסלול יחיד (השמאל), ושלח תמיד דרך הימני בשירה להגיא ל-200 (마שר השיקול הפנימי של כל נתב לשולח דרך הילוק שיצא ממנה מחוץ ל-AS).  
לben גם אם הנתב השמאלי ירצה לשולח ל-200 משיקול פנימי שלו, הוא יעביר קודם לנtab הימני שישלח את ההודעה ממנה.



### iBGP

**iBGP – Internal Border Gateway Protocol** הוא פרוטוקול שנמצא בתחום ה-AS ומעדכן כל נתב מה נקודת היציאה מהרשת, מאיפה הוא צריך להعبر מידע שהוא שאמור לצאת מהרשת.  
מalfa תחילה IP של יעד לנקודת היציאה שתוביל אל היעד הזה.  
**DV (Distance Vector) – IGP – Interior Gateway Protocols** הם פרוטוקולי ניתוב פנים ארגוניים שמוצאים את המסלולים הבי קצריים בתחום ה-AS (וכו'). היעדים של המסלולים יהיו הקודקודים ש-BGP ישייב.

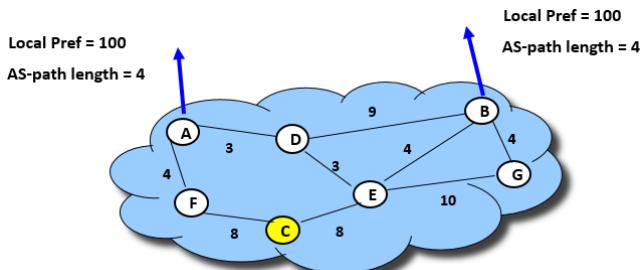


לרשת אחת יכולות להיות מספר נקודות יציאה עם ערך זהה, איך נדע לבחור את נקודת היציאה הבי טובה?

נשתמש בניתוב **Hot Potato Routing** – במקרה שמשהדים מסלולים זרים בראשת, העדפה היא לשולח את המידע כמה שייתר מהר. ככל מר בנתב יבחר להעביר את המידע דרך נקודת היציאה שהבי קרובה אליו בהtbס על המסלולים שחושבו ב-IGP.

באופן כללי נתב ב-AS אחד יעביר מידע ל-AS אחר באופן הבא:

1. העברה דרך נקודת היציאה שמוסדרת על ה-AS.
2. אם העדפות שוות, נבחר בדרך ה-AS הbi קצרה (פחות ASes בדרך ליעד).
3. אם עדין יש שוויון, נבחר בנקודת היציאה שהמסלול אליו הbi קצר (hot potato).
4. אם גם במקרה הזה יש שוויון, נבחר ע"י שיקולים אחרים.



**הערה חשובה:**  
ECMP מפצל את התעבורה בין מסלולים שווים **לאותו יעד**.  
במקרה זה, לנכון פנימי (למשל C) אין סיבה לפצל את המידע בין המסלולים ברגע (מסלול ל-A ומסלול ל-B) – לכל בתובת IP יש מסלול אחד עם יעד שונה ורק דרכו הנכון ישלח את המידע, ולאחר מכן יוכל לפצל כלומר C יבחר דרך איזה נתב להוציא את המידע, ולאחר מכן יוכל לפצל את המידע בין מסלולים שווים לאותו נתב.  
(AS-path length זה כמו רשותות עוברם בדרך ליעד)

## BGP Security

בש-BGP תוכנן ההנחה הייתה שהאינטרנט בטוח ולמן אין בו הרבה אבטחה. ההנחה אלה חמורות גם ביחס לשאר הפרטוקולים כי הוא הפרוטוקול הכי חשוב, ומגדיר את האינטרנט.

### BGP Session Security

הפרטוקול ריש להתקפות TCP. עברו איזשהו sessions בין נתבים בארגונים שונים, וכיון להשפעה על התקשרות ע"י התקפת TCP כי שיצינו BGP רץ מעיל TCP.  
התקפות לדוגמה יכולות להיות:

- השפעה על מדיניות הייזוא של המסלולים ודיווח על מסלולים שקריים.
- הפרעה לפקודות שעוברות בלינק (שינוי, הפללה וכו').
- העמסה על הרשת.

המקרה הזה פחות נוראי, התוקף צריך להיות באותו אזור גיאוגרפי וקל להגן על מתקפות כאלה:

- שתי נקודות קצה יכולות להשפיע על כתובות IP ופרטיהם קבועים לתקשרות, וכן על הצפנה והודעות.
- הגבלת גישה הפיזית ללינק (תהייה מאותנו בניין למשל).
- השפעה על תעבורת של פקודות: סיכון פקודות משולחים לא רצויים, תיעדו פקודות BGP.

בעצם שני הנתבים מחוברים פיזית ויכולם פיזית לעדכן אחד את השני.

## Manipulating BGP

התקפות מהסוג זהה הן בהיקפים גדולים ויכולות לגרום להיעלמות של ארגונים מהאינטרנט.

### Prefix Hijacking

נסתכל על מקרה שהוא בשנת 2008. ספק האינטרנט בפקיסטן Pakistan Telecom, התבקש מההמשלה בפקיסטן לבטל את הגישה ליוויו מהתושבים ונמסרו לו כתובות IP הביעתיות ש策יר למנוע תקשורת איתן.

כדי לטפל במקרה, פקיסטן טלקום הייתה צריכה להפעיל פקודות שמזעירות לבתוות האלה ומהבתובות האלה.

במקום זה, פקיסטן טלקום הכריזה על עצמה עם תחילת IP ששיכבת ליוויו. ההכרזה הייתה החוצה אל העולם (כגראה במקום הכרזה פנימית) וויטויב "נעלם" מהאינטרנט כי ההכרזה של פקיסטן טלקום הייתה יותר ספציפית. כל ההודעות שהיו אמרות להגיע ליוויו הגיעו לפקיסטן טלקום.

כדי לתקן בשיטה זו, התוקף צריך להיות בעל גישה לנット BGP (יכול להיות איזשהו ארגון או יכול להשלט על נתב BGP).

אותו תוקף מכריז על עצמו עם תחילית IP של יעד התקיפה שתהיה ספציפית יותר.

#### הבעיה:

1. קשה לשים לב לבעה ולדבג אותה.
2. קשה למנוע מהתוקף להפסיק.

#### פתרון אפשרי:

- **Origin Authentication** – נרצה שהשכנים של התוקף (AS 3) ידעו שהוא "משקר" ע"י מסד נתונים מאובטח שמאיפה תחילית לנתק. לא נוכל להבטיח שאין תקלות ושינויים במסד נתונים.

#### שינוי המסלול

- אם אם הפתרון הקודם הקודם היה עובד במאה אход, התוקף יהיה יכול להגיע אל הנתק ע"י בר שיעדכן את המסלול אל הנתקף להיות דרכו (בشرطו) ומודען את המסלול ל-v להיות xprefix (v, m) ויברץ על המסלול, ובשימושו ירצה לשלוח לנתקף הוא יוכל לעבור דרך התוקף. קשה לדיבוג. אף אחד לא יוכל לבדוק את האמינות של המסלול זהה. ההודעות יגיעו לתוקף שיוכל לעשות אותן מה שירצה.

- **Path-Shortening Attack** – התוקף יוכל להציג מסלול קצר יותר אל היעד ללא ASes מסוימים, ככה שהמסלול דרכו יוכל להירות יותר ואטרקטיבי גם מבחינת "מסלול הבני קצר" וגם למי שמנסה להימנע-m-ASes שהמסלול המוצע בביבול לא עבר דרכם.

- בעיה שקשה לדיבוג.

**BGPSEC** – דרך להתמודד עם הבעיה. כל נתב שמכריז על מסלול "יחתום" עליו. גם עם הדרך הזאת יש כל מיני בעיות, כמו שריאנו-ב-DNSSEC (קשה להטמע וכו'). בשורה התחתונה ה-BGP הוא פרוטוקול רגיון ומעוד להתקפות, ואחד הנושאים הבני נחקרים בתחום הוא אבטחת BGP.

בצלחה 😊