סיכום רשתות תקשורת

29 בינואר 2023

1 הרצאה 1:

1.1 רשת תקשורת:

- **המרכיבים ברשת תקשורת:**

1 קודקודים: ראוטרים, סוויצ'ים

2 לינקים: כבל המחבר בין שני קודקודים - כבל רשת, סיב אופטי

3 משתמשי קצה: כל מי ששולח מידע מעל הרשת ויש למידע משמעות סמנטית עבורו - טלוזיות חכמות, מקררים שמשדרים מעל הרשת ועוד.

1.2 רשת הטלפון:

- **רשת הטלפון:**

1 כשנרצה להתקשר נבדוק האם הרשת יכולה לתמוך בזה (לבסס $circuit$) - האם יש רשת עם קיבולת לשיחה. אם כן - השיחה תצא, אחרת - היא תיכשל.

2 לאחר שנמצא מסלול עם קיבולת - השיחה תתקיים עד לסופה.

3 בסוף השיחה יש פינוי של המשאבים.

- **סוויץ':** מקבל מידע מכל מיני כיוונים, ואמור להבין איזה מידע ממשיך לאן, ממפה כניסות ליציאות. למעשה הוא מפשר כמה שיחות במקביל מאחד לאחד.

- **איך מעבירים שתי שיחות שונות על אותו הלינק:**

חלוקת זמן: מחלקים את הזמן לסלוטים כך שכל שיחה מקבלת רק חלק מהזמן, לדגמה כל מילי שניה אי זוגית ל A וזוגית ל B.

חלוקת תדרים: נחלק את הקו לכמה תדרים, וכל שיחה תשדר בתדר אחר.

- **יתרונות של רשת הטלפון:**

הרשת יציבה, צפויה, קל לאתר תקלות, קלה לתפעול, הקו נשמר לנו מראש כך שאין עיקובים.

- **חסרונות של רשת הטלפון:**
הרשת לא תספק את הדרוש אם יש עומס - חלק מהמשתמשים לא יקבלו את השיחה כלל. ניצול גרוע של הרשת - אם אין עומס, חלק מהקו לא יהיה בשימוש כלל. אין תיקון תקלות עצמאי - במקרה של תקלה זזה באחריות המשתמש לחייג שוב.

- $Pick\ VS\ Avrage$: במערכת הטלפון נצטרך להסתכל על ה $Pick - P$ שהוא זמן הדיבור בפועל המהלך השיחה.
וכל $Avrage - A$ - הזמן הממוצע בו דיברנו במהלך השיחה כולה.
אם היחס $\frac{A}{P}$ גדול - אזי הרשת לא מנוצלת כראוי.
בטלפון הפער הוא 1:3 בדרך כלל.

1.3 ההבדלים בין רשת הטלפון לאינטרנט:

- מערכת הטלפון תומכת באפליקציה אחת בלבד - שיחת טלפון.

- אם היינו מבססים את רשת האינטרנט כמו רשת הטלפון הניצול של הרשת היה אחוזים בודדים בלבד.

- **העברת מידע על האינטרנט:** הפריצה הייתה שהחליטו שהרשת תהיה אחראית על שליחת המידע ותעביר פקטות. לא נשמור קיבולת מראש, אלא השידור ייעשה כשתהיה קיבולת פנויה.

- **רשת** $BroadCast$**:** רשת שבה רק משתמש אחד יכול להשתמש בכל רגע נתון, וכל שאר המשתמשים מאזינים לו. ברשת זו קיימת בעיית נצילות. בעיה נוספת היא בעיית הפרטיות ולכן נצטרך להצפין את המידע. רשת כזו מתקיימת בשכבות התחתונות של האינטרנט.

- **רשת עם סוויצ':** רשת שיכולה לחלק אותו לינק בין כמה משתמשים. הופכת את הרשת ללא $BroadCast$.

2 פקטות:

- באופן שונה מרשת הטלפוניה, ברשת האינטר נט אנו לא מקצים מקום מראש לכל דאטה משום שאז ניצול הרשת יהיה 1% בלבד. לכן נעשה זאת באמצעות שליחת פקטות.

- **מהן פקטות:** כשנרצה לשלוח מידע בינטרנט אנו נחלק אותו לפקטות כך שכל פקטה מכילה חלק מהמידע, ונשלח אותן לכיוון היעד. ללא כל הקצאת משאבים לפני השליחה.

- **מה פקטה כוללת:** פקטה של מידע תכלול את המידע עצמו, בנוסף היא תשמור מידע על הפקטה עצמה - לאן היא צריכה להגיע ועוד.

- **פרטים על פקטות:** כל פקטה תיאורתית יכולה לעבור במסלול שונה או לא להגיע כלל. אנו נרצה להבטיח שהיא תגיע כמו שנשלחה.

- **באפרים:** לכל קודקוד ברשת יהיה באפר שבו הוא ישמור פקטות למקרה והלינק שאליו הוא צריך להעביר את הפקטה לא יהיה פנוי.

3 מודולריות של רשתות ועקרונות:

- **מודולריות:** מבודדים פונקציוניליות אחת מהשניה אך אנו משלמים בביצועים.
- **מודולריות של רשתות:** האינטרנט בונה מודולריות שמבוזרת בין מקומות שונים, אנו נצטרך להחליט: כיצד לחלק את זה לשכבות. איך נממש את המודלים. איזה מידע לאחסן.
- **עקרון השכבתיות:** הצורה הספציפית של מודולריות באינטרנט.
- **עיקרון** *end to end*: הרשת לא תספק שום שירות בתוך הרשת למעט הקישוריות בין משתמשי הקצה. הצפנה אבדן פקטות וסדר הפקטות אינן באחריות הרשת.
  למשל - אנחנו צריכים להבדיל בין שיחות בזמן אמת שאז אין חשיבות לפקטה שאבדה לבין שליחת מסמכים שאז כן יש חשיבות לאובדן פקטה ולזה **המשתמש** צריך לדאוג. כלומר אובדן מידע והצפנה אינן הבעיות של הרשת.
  נשים לב כי חומות אש כן יכולות לחסום מידע מסויים למרות שעל פי העקרון הן לא אמורות לעשות זאת. כ"כ הן יכולות לטפל באובדן פקטות.
- **עיקרון** *Fate sharing*: הרשת אחראית על הקישור בין שני משתמשים, אך המידע על כך שהשיחה מתקיימת נמצאת רק אצל משתמשי הקצה. רק הקצוות מודעים לכך שהשיחה מתקיימת, לכן אם נתב באמצע נופל השיחה תימשך, והיא תיפסק רק אם משתמש קצה נופל מהשיחה.

3.1 שכבות - *Layers*:

- **הרעיון:** בכל שלב שבו נשלח פקטה, נוסיף עליה שכבה נוספת על מטא דאטה. בשליחה בכל שלב נוסיף שכבה כדי שהפקטה תגיע ליעדה הנכון.
  כל שכבה מתקשרת עם השכבה שאחריה ולפניה עם פקוטוקול מסויים.
- **עיקרון האנקפסולציה:** כל שכבה מוסיפה עוד מידע על הפקטה והכל נעטף יחד.
- **השכבות:** אנו נתעסק בשלש שכבות הביניים.
  1 השכבה הפיזית: מאפשרת לשכבה מעליה לתת פקודת שליחת ביט על הרשת הפיזית.
  2 שכבת הקישור - *Link*: כיצד נשלח פקטות דרך רשת מקומית.
  3 שכבת הרשת - *Network*: יכולה לתת פקודות לרשת המקומית - להעביר פקטות דרך הרשת המקומית.
  4 שכבת העברה - *Transpotr*: מגדירה שפקטה מגיעה בהצלחה מקצה אחד לשני, בודקת תקינות של חבילות ומידע.
  5 שכבת האפליקציה - *Application*: האפליקציות מתקשרות דרכה עם הרשת.
- באופן זה כל שכבה מתעסקת רק בתחום האחריות שלה.

4 הסתברות:

4.1 נוסחאות:

- **למה הסתברות:** אנחנו צריכים להעריך את ההסתברות להתנגשויות כאשר מספר מכשירים משדרים לאותו הראוטר על אותו לינק.

- **הסתברות מותנית:**

$$A, B \subseteq \Omega, P(B) > 0 : P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$
$$A, B \subseteq \Omega : P(A \cap B) = P(A \mid B) \cdot P(B)$$

- **מאורעות ב"ת:**

$$P(A \mid B) = P(A) \Rightarrow P(A \cap B) = P(A) \cdot P(B)$$

- **א"ש איחוד וסכום:**

$$P(A \cup B) \leq P(A) + P(B)$$

- **כלל בייס:**

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- **נוסחת ההסתברות השלמה:**

$$P(A) = \sum_{i=1}^{n} P\left(A \mid B_i\right) \cdot P\left(B_i\right)$$

- **נוסחת בייס השלמה:**

$$P(A \mid D) = \frac{P(D \mid A) \cdot P(A)}{\sum_{i=1}^{n} P\left(D \mid B_i\right) \cdot P\left(B_i\right)}$$

4.2 משתנים מקריים:

- **מהו מ"מ:** מ"מ היא פונקציה שמתארת לנו את מרחב המהסתברות

$$X : \Omega \Rightarrow R$$

- **תוחלת - E:**

$$\mathrm{E}(X) = \sum_i x_i \cdot P\left(X = x_i\right)$$

- **לינאריות התוחלת:**

$$\mathrm{E}(\sum_i a_i X_i) = \sum_i a_i \cdot \mathrm{E}(X_i)$$

- **שונות - Var:** כמה המדגם הספציפי שלקחנו רחוק מהתולת.

$$Var(X) = \mathrm{E}\left((X - \mathrm{E}(X))^2\right) = E\left[X^2\right] - E[X]^2$$

- **טבלת משתים מקריים בדידים מוכרים:**

Discrete Random Variables

25

	Bernoulli	Binomial	Geometric	Poisson
Intuition	We make an experiment, and we could either fail or succeed	We make n independent Bernoulli experiments. We mark by X the sum of successes	We do Bernoulli experiments until we succeed. We denote by X the number of tries	Counting the number of events that occurred during some period of time
Probability mass function	$P(X=1)=p$ $P(X=0)=1-p$	$P(X=k)$ $=\binom{n}{k}p^k(1-p)^{n-k}$	$P(X=k)$ $=(1-p)^{k-1}p$	$P(X=k)$ $=\frac{\lambda^k}{k!}e^{-\lambda}$
Notation	$X \sim Ber(p)$	$X \sim Bin(n,p)$	$X \sim Geo(p)$	$X \sim Pois(\lambda)$
Exp.	$E(X)=p$	$E(X)=np$	$E(X)=1/p$	$E(X)=\lambda$
Var.	$Var(X)=p(1-p)$	$Var(X)=np(1-p)$	$Var(X)=(1-p)/p^2$	$Var(X)=\lambda$

- **נוסחה שקולה עבור מ"מ פואסוני:** במקום להסתכל על המשתנה המקרי הבדיד - k. נוכל להסתכל על מספר האירועים שקרו בזמן הרציף t

$$P_{k=i}(t=T) = \frac{(\lambda T)^i}{i!}e^{-\lambda T}$$

4.2.1 משתנים מקריים רציפים:

- **טבלת מ"מ רציפים מוכרים:**

Continuous Random Variables

31

	Uniform	Exponential	Gaussian (Normal)
Notation	X~Uni(a, b)	X~Exp(λ)	X~N(μ,σ)
probability distribution function	$f(x)=\begin{cases}\frac{1}{b-a} & x\in[a,b]\\ 0 & else\end{cases}$	$f(x)=\begin{cases}\lambda e^{-\lambda x} & x>0\\ 0 & else\end{cases}$	$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-(x-\mu)^2}{2\sigma^2}}$
Exp.	$E(X)=\frac{a+b}{2}$	$E(X)=\frac{1}{\lambda}$	$E(X)=\mu$
Var.	$V(X)=\frac{(b-a)^2}{12}$	$V(X)=\frac{1}{\lambda^2}$	$V(X)=\sigma$

- **תכונת חוסר הזכרון של מ"מ גיאומטרי:**

$$P(X > s+t \mid x > t) = P(X > s)$$

- **משתנה פואסוני:** אנו שואלים מה התפלגות של קצב הגעת ההודעות, כלומר עבר טווח זמן T כיצד מתפלגים האירועים. התפלגות זו היא התפלגות פואסון.

5 פרוטוקולים:

פרוטוקולים מגדירים את הסינטקס והסמנטיקה של התקשורת בין שכבות האינטרנט. לכן כל רכיב צריך להסכים על הפרוטוקול של הרשת שמעליו ומתחתיו. בנוסף כל רכיב צריך להסכים על דרך השליחהשל המידע, כדי לדעת מהיכן לקחת את המידע שקשור אליו מתוך המטא דאטה, ובאיזה מיקום הוא נשמר.
לעיתים בשכבה אחת כול להיות יותר מאשר פרוטוקול אחד.

- **פרוטוקול** IP: פרוטוקול של שכבת הרשת - שכבה 3, ויש רק אחד כזה. פרוטוקול זה מייצג את כתובת משתמשי הקצה. זוהי למעשה השפה שממנה יש הסכמה משותפת לכולם והוא מכונה גם פרוטוקול שעון החול.

6 השכבות ברשת האינטרנט:

6.1 השכבה הפיזית:

השירות: מעבירה ביטים על פני מדיום פיזי בין שני גורמים.
ממשק: חושפת API של העברת ביט, כך השכבה מעליה יכולה לדבר בשפה של ביטים.
מה השכבה צריכה לעשות: קידוד, מתחים, סיבים ועוד..

6.2 שכבת הקישור - $Link$:

השירות: מספקת העברת הודעות בתוך רשת מקומית. לרשת המקומית יש כתובת של רשת מקומית (כתובות מק).
ממשק: שליחה הודעה לרשת מקטמית אחרת.
פרוטוקולים: פרוטוקולים למסלול ברשת, וחלוקת הלינק בין מספר גורמים.

6.2.1 תקשורת בתוך רשת מקומית:

הרשת המקומית היא רשת $broadcast$ פתוחה, לכן נצטרך לנהל את כל התקשורת ברשת בין כמה משתמשים במקביל.

- **הבעיה העיקרית היא:** כיצד מספר משתמשים שונים יוכלו לתקשר על אותו הלינק במקביל.
- **מה הלינק הבודד צריך לספק:** גישה ללינק ושפה שתגיד לנו למי המידע אמור להגיע בתוך הלינק. יש לינקים שיתקנו פקטות שהגיעו עם שגיאות וישלחו מחדש פקטה שאבדה (בניגוד לעיקרון $end\ to\ end$)
- **איפה שכבה זו ממומשת:** בכרטיס הרשת - הוא נמצא בין החומרה לבין ה CPU.
- **האלגוריתם האידיאלי:** עבור לינק שיכול להעביר R ביטים בשניה. בעולם אידיאלי היינו רוצים שכל מי שרוצה לדבר יקבל את כל הרשת.
  הוגנות: אם m קודקודים רוצים לדבר אזי כל אחר מהם יקבל $\frac{R}{m}$ מהתקשורת.
  אין מחשב מרכזי אחראי: משום שאם הינו רוצים לתקשר איתו אזי גם את התקשורת הזו היינו צריכים לנהל.
  זמן: אין סנכרון שעונים.
- **הפתרון - אקראיות:** כשאנו נכנסים לרשת אנו לא יודעים אם יש עוד משתמשים ברשת, לכן - נתחיל לשדר בקצב המלא.
  הבעיה: יכול להיות שתהיה התנגשות, לכן נרצה להבין שהתקיימה התנגשות ולפתור אותה.

האלגוריתם $ALOHA$:

- **האלגוריתם** $sloted\ ALOHA$:
  הנחות:
  1: כל הפקטות שמועברות הינן באותו הגודל.
  2: הזמן מחולק לסלוטים בגודל קבוע וזה הזמן שלוקח להעביר פקטה מסויימת, שלכולם לוקח אותו זמן לשלוח פקטה.
  3: אנחנו מתחילים לשדר רק בתחילת סלוט.
  4: המשתמשים מסונכרנים בזמן.
  5: אם יש התנגשות, כל משתמשי הרשת מודעים להתרחשותה.
  האלגוריתם:
  כשמגיעה פקטה חדשה: נסה לשלוח את הפקטה.
  אם אף אחד לא רוצה לשלוח במקביל - הפקטה נשלחה.
  אחרת - יש התנגשות.
  אם יש התנגשות - בהסתברות p נשלח את הפקטה בתור הבא. המשך כך עד שהפקטה תישלח.

- **שני סוגים של האלגוריתם -** $Sloted,\ Pure$**:** אפשר להריץ את האלגוריתם עם ההנחות - והוא ייקרא $Sloted$ משום שנניח כי כל הודעה נשלחת ב $slot$ שלה והשעונים מסונכרנים..
  מנגד יש את האלגוריתם בגרסת ה $Pure$ שבה כל משתמש יכול לשלוח הודעה בזמן שלו ואין סנכרון שעונים.

- **חסרונות של האלגוריתם:** קיים מרכיב הסתברותי שגורם לחוסר יעילות, לדוגמה - אם כולם שולחים יחד ויש התנגשות, ולכן קיימת הסתברות שכולם בסיבוב הבא לא ישלחו.
  כלומר פקטה תישלח רק אם היא יחידה בתור.

- **יתרונות של האלגוריתם:** אם אני לבד - הפקטה תשלח. מתקיים ביזור, ויש פשטות.

- **פרוטוקול נוסף:** יש לנו הסתברות p קבועה, וכל פקטה שמגיעה תוגרל עליה ההסתברות אם לשלוח אותה או לא. גם אם כל הלינק פנוי.
  הבעיה: ה $goodput$ (שימוש בקו) של האלגוריתם הזה יהיה 37% וזה נמוך.

- **בעיות עם הנחות האלגוריתם:** ההנחה כי כולם שולחים בתחילת כל סלוט אינה נכונה, בנוסף הם לא מסונכרנים כולם.

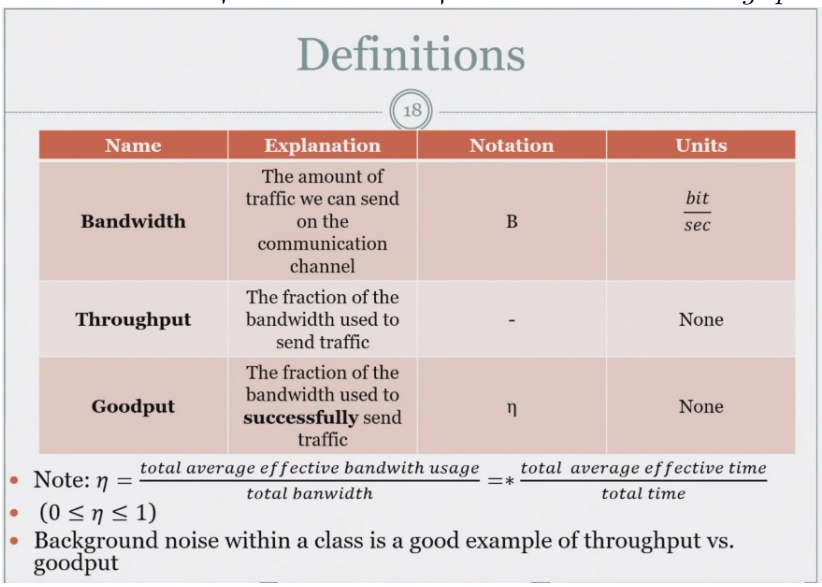
6.2.2 רוחב פס - $Bandwidth$:

- **הגדרה:** רוחב הפס מסומן ב B מוגדר כ $bit\ per\ second$, והוא מתאר את המהירות של העברת ביטים ברשת. כשאנו רוצים להעביר S ביטים אנו נכפיל את B ב T שזה הזמן, ונקבל את מספר הביטים S שניתן להעביר ביחידת זמן T.
  (משוואת מהירות זמן דרך $B \cdot T = S$).

- **מושגים על מעבר מידע על הלינק:**
  $goodput$**:** אחוז מהזמן שבו עבר מידע **בצורה מוצלחת** על הלינק - שהצלחנו לשדר בצורה מוצלחת.

$$goodput = \frac{E[T]}{Total\ Time}, \quad P_{success}$$

$throughput$**:** כמה אחוז מהזמן שהשתמשו בלינק.



Definitions

18

Name	Explanation	Notation	Units
Bandwidth	The amount of traffic we can send on the communication channel	B	$\frac{bit}{sec}$
Throughput	The fraction of the bandwidth used to send traffic	-	None
Goodput	The fraction of the bandwidth used to **successfully** send traffic	η	None

- Note: $\eta = \frac{total\ average\ effective\ bandwith\ usage}{total\ banwidth} =* \frac{total\ \ average\ effective\ time}{total\ time}$
- $(0 \leq \eta \leq 1)$
- Background noise within a class is a good example of throughput vs. goodput

6.2.3 עיקרון $CSMA$:

- **הרעיון:** עיקרון שאומר כי כל תחנה שרוצה לשדר ברשת ברודקאסט צריכה לפניכן להקשיב ללינק ולראות אם הוא פנוי. ורק אם הוא פנוי היא תשדר, אחרת היא סתם תפגע בשאר התחנות שמשדרות ואף אחת לא תצליח לשדר.

- **הבעיה:** אם שתי תחנות ישמעו שהקו פנוי ויתחילו לשדר בדיוק באותו הזמן הן ייתנגשו.
  נניח כי ההסתברות להתרחשות כזו היא נמוכה.

- **בעיה נוספת:** אם תחנה לא שומעת אף תחנה אחרת משדרת, זה יכול לנבוע מכך שהשידור טרם הגיע אליה, אך יש תחנה אחרת משדרת.

- **התמודדות:** נרצה שברגע שתחנה מודעת להתנגשות, היא תפסיק לשדר (ברשת שבה הנחנו כי כל התחנות מודעות לכך שיש התנגשות).
  כמובן שיש לנו את זמן התגובה. נגדיר את זמן זה להיות d.

- **האלגוריתם:** כאשר תחנה רוצה לשדר היא תבדוק שהקו ריק.
  אם הוא ריק - תשדר.
  אחרת - תחכה.
  אם התחלת לשדר וזיהית התנגשות - תפסיק לשדר.

- **אבחנה:** אם תחנה שלחה פקטה והיא לא שמעה התנגשות לאורך כל השידור, זה לא אומר שלא היית התהנגשות, אלא יכול להיות שהיא עדיין לא יודעת על ההתנגשות כי היא התרחשה רחוק ממנה.

- **זמן** $prop$: נגדיר את משך זמן השידור מהתחנה המשדרת עד למיקום הרחוק ביותר בו יכולה להתקיים התנגשות (התחנה הרחוקה ביותר) להיות - $prop$.

- **הסלוט המינימלי לזיהוי התנגשות:** תחנה שרוצה לוודא שלא התרחשה התנגשות, צריכה לשדר לפחות $2 \cdot prop$ כדי להיות בטוחה שלא התרחשה התנגשות. כי זה הזמן המקסימלי שיקח למידע על התנגשות להגיע אליה במידה והיתה התנגשות.
  לכן נגדיר את הסלוט להיות $2 \cdot prop$, וזה יגדיר לנו את גודל הפקטה.

- **זמן השליחה של פקטה (זמן שידור):** שווה לגודל הפקטה **חלקי** רוחב הפס.

- **חישוב ה** $prop$: שווה למרחק **חלקי** המהירות (קצב ההתפשטות).

- **מהו ה** $goodput$:

$$\eta = \frac{Time\ to\ send}{Time\ to\ send + overhead} = \frac{Time\ to\ send}{Time\ to\ send + 3 \cdot prop}$$

- **כיצד נעלה את ה** $goodput$: אם יהיה לנו יחס גדול בין זמן $prop$ לזמן השליחה. כי בנוסחה - זמן השליחה מתחלק ב $prop$.

- **מהו הערך של p:** אנו נרצה לבחור $p = \frac{1}{n}$ כדי למקסם את ההסתברות להצלחת שליחת הודעה.

- **מספר פרוטוקולים שונים לשליחת הודעה לפי עיקרון $CSMA$:**

CSMA

12

- A node should decide what to do if the medium is occupied. We will examine three protocols:

Protocol	Free?	Occupied?	Intuition
1-persistent	Send!	Wait until the channel becomes idle and then send immediately	High chance of collisions in loaded channels
non-persistent		Backoff and try again	Low utilization on low load
p-persistent	Send with prob. p	Wait until the channel becomes idle and then send with probability p.	A compromise between the two approaches

6.2.4 כתובת MAC:

- כתובת שמייצגת את המכשיר הנוכחי בתוך שכבת הלינק. לכל מכשיר יש כתובת קבועה שצרובה על כרטיס הרשת שלו, וזו הדרך בה הוא מקבל מידע מהראוטר.

6.2.5 פרוטוקול $Ethernet$:

- **התמודדות הפרוטוקול עם התנגשויות:** פרוטוקול שמקבל את הפקטה משכבת הרשת - שכבה 3, ומקשיב.
אם הקו פנוי הוא משדר, אחרת - מחכה.
אם הייתה בעיה בשידור - יפסיק לשדר, וישלח סיגנל שיש בעיה. לאחר מכן הוא מחשב את סך ההתנגשויות ולפי זה מסיק מה מספר הגורמים ברשת. ולכן לפי מספר ההתנגשויות הוא יחליט כמה זמן לחכות עד לשידור הבא.

- **כיצד הפרוטוקול מחליט כמה זמן לחכות:** אם היו m התנגשויות, הוא בוחר מספר בטווח $k \in [0, m-1]$ ומחכה k זמן. כך ההסתברות עולה אקספוננציאלית לכן שהיחידה תמתין לפני השידור הבא.

- $Jam\ signal$**:** סיגנל שהיחידה תשלח לפני שתפסיק לשדר, במקרה שזיהתה התנגשות, כדי שיהיה ברור לכל היחידות שהייתה התנגשות.

- **חישוב** $goodput$**:**

$$\eta = \frac{1}{1 + \frac{5T_{prop}}{T_{trans}}}$$

עבור T_{prop}: הפרופ המקסימלי שיש לחכות בין שתי תחנות.
ועבור T_{trans}: הזמן לשלוח פריים מקסימלי.

6.2.6 רשתות $wierless$:

- **מה שונה:** רשת ברודקאסט, אך ברשת כזו אם יחידה מסויימת משדרת הודעה היא דורסת את הגלים של היחידות האחרות, ולכן היא לא מודעת לכך שיש התנגשות.
  בנוסף קיים קודקוד ראשני שמנהל את הרשת - קדקוד מיוחס (AP).

- **אלגוריתם** 802.11:
  מחכה פרק זמן מסויים.
  אם לא שמעתי כלום - משדר את כל הפקטה.
  אחרת - נגריל כמה זמן נחכה. ונשדר בסוף הזמן.
  אם תחנת הבסיס שולחת ACK (מסמן שהמידע התקבל) נסיים. אחרת - נחזור להתחלה.

- **כיצד נמנע מהתנגשות:** לפני שנדבר נבקש רשות דיבור (RTS) מהקודקוד המיוחס (נניח כי היא לא מתנגשת כי היא קצרה).
  אם קיבלנו הודעה שמותר לנו לדבר - נדבר. בנוסף כולם יהיו מודעים לכך שיש לנו רשות דיבור.
  בסוף נקבל אישור שההודעה שלנו הועברה.

- **נשים לב:** כי הקדקוד המיוחס כן יודע על התנגשויות כי הוא רק מאזין, ושולח הודעות קצרות של אישורי דיבור וקבלת הודעות.

6.2.7 רשת מקומית וסוויצ'ים:

- **סוויץ':** יש לו ID, הוא מקבל הודעות מרשתות ברודקאסט, ההודעות הן פקטות של אינטרנט והוא צריך להחליט לאן להעביר את ההודעה (לפי כתובת MAC).
  לכל סוויץ' יש מספר יציאות ממסופרות שנקראות פורטים, ולכל פורט מחוברת רשת ברודקאסט.

- **איך הסוויץ' יודע לאיזו כתובת לשלוח:** הסוויץ' לומד את כתובות ה MAC של המכשירים ששולחים אליו הודעות.
  אלגוריתם $self\ learning$:
  בכל פעם שמכשיר מבקש ממנו לשלוח הודעה, הוא מזהה את הכתובת של השולח, ושומר אצלו מידע על הפורט הספציפי ששולח דרך הכתובת.
  אם הוא יודע לאן לשלוח את ההודעה: אם הכתובת אליה נשלחת ההודעה נמצאת באותו כיוון - הוא מתערב.
  אם הוא יודע את הכתובת - הוא שולח.
  אחרת - הוא שולח את ההודעה לכולם ($flood$).

- **הערה:** האלגוריתם יעבוד גם אם יש מספר סוויצ'ים שמחוברים אחד לשני. כי כל סוויץ' יודע את כתובות הקצה שמחוברות לכל לינק.

- **בעיה:** כשיש לנו שני סוויצ'ים שמחוברים במעגל, הם יכולים לטעות מהיכן מגיעה ההודעה. משום שכל הודעה שנשלחת מגיעה אל הסוויץ' גם מהצד השני ואז הוא לא יידע באיזה פורט נמצאת הכתובת השולחת.

- **פתרון הבעיה:** אלגוריתם לרישות של רשתות ברודקאסט שנקרא $Spannig\ tree$:
אנו רוצים שהרשת שלנו תכיל מעגלים כדי שתהיה יתירות, ואז במקרה של תקלה יהי לנו לינק חלופי לשלוח עליו.
לכן הפתרון הוא לעשות מעגלים, אך להשתמש רק בלינקים שהם עץ ללא מעגלים. וברגע שלינק מסויים נופל אנחנו מחליפים את הענף שנפל לענף אחר מהמעגל. כלומר - ננצל את היתירות רק במקרה של תקלה.

- **הערה:** אנו נרצה שאלגוריתם העץ הפורש ייתקיים באופן עצמאי. כל קדקוד יידע איך לקיים את העץ ובאיזה לינקים להשתמש.

- **כיצד נבחר שורש לעץ:** כל סוויץ' יוציא הודעת $Hellow$, לרשת הברודקאסט וישדר את מספר הסוויץ' הנמוך ביותר שהם שמעו עד כה (בהתחלה כל סוויץ' יטען שהוא השורש). הסוויץ' עם ה sid הנמוך ביותר "מנצח" ונבחר היות השורש. הסוויצ'ים ממשיכים לעדכן מי השורש עד שכולם יידעו על השורש.
לאחר מכן כל סוויץ' מחפש את המסלול הקצר ביותר (בעזרת אלגוריתם בלמן פורד) לשורש, כדי לזהות מי האבא שלו בעץ, אם יש כמה מסלולים באותו האורך הוא בוחר את זה עם sid הנמוך. ומסמן את הצלע הזאת להיות RP $(root\ port)$. כל רשת ברודקאסט תסמן את הצלע אל הסוויץ' עם המרחק הקצר ביותר לשורש, להיות FP.

- **אלגוריתם בלמן פורד מבוזר:** כל קוקוד שולח לכל השכנים שלו את המרחק שלו מהשורש (הקודקוד שמתחיל את התהליך הוא השורש כי הוא היחיד שיודע את המרחק שלו ושהוא שווה ל 0). כל קדקוד מסתכל על המרחק של כל שכניו מהשורש, מוסיף 1 (או כל משקל אחר שנחליט לצלע) ומעביר הלאה, הוא יבחר את הקודקוד עם המרחק הקצר ביותר לשורש להיות האבא שלו, ויסמן את הצלע אליו ב RP.

- **כיצד רשתות הברודקאסט בוחרות את הצלע** FP: לרשתות הברודקאסט אין את היכולת לבחור לינקים לשדר עדיהם כיאין להם cpu, לכן מי שבוחר את הלינקים עליהם הרשת תשדר הם הסוויצ'ים. כשהסוויצ'ים שולחים את ההודעות הם שומעים למי יש את המרחק הקצר ביותר לשורש, ובוחרים אותו להיות האחראי על הברודקאסט הזה, והוא מעביר את ההודעות. שאר הסוויצ'ים מאזינים להודעות אך לא מעבירות אותן.

6.3 שכבת הרשת - $Network$:

השירות: מעבירה מידע באמצעות כתובות גלובאליות - IP. כדי שהמידע יגיע הוא יעבור ברשתות מקומיות, המעבר בתור הרשת המקומית זאת לא דאגת שכבת הרשת אלא שכבת הלינק.
ממשק: מעבר הודעות בין כתובות גלובאליות.
פרוטוקול: IP קישור בין רשתות מקומיות (LAN) בעזרת נתבים, כדי שההודעה תעבור.

- **מהי כתובת** IP: 32 ביט שמיוצגים בעזרת 4 רבעיות של מספרים. כתובת זו מזהה את מי שמשתתף בשכבה זו - $end\ host$ ונתבים.

- **כתובות** IP **לנתבים:** כל נתב יחזיק כתובת לעצמו, וכתובת לכל פורט שלו.

- **מטא דאטה:** כשנשלח פקטה אנו נצטרך לשלוח את כתובות ה IP של המקור ושל היעד.

- $sub\ net$: לכל ארגון יש תת רשת שמיוצגת ע"י כתובות IP, כך כשנרצה לשלוח הודעה, הביטים הימניים בכתובת ייצגו את תת הרשת.

לכן כשנרצה לשלוח לכתובת שנמצאת תחת בזק אנו נדע זאת לפי הביטים הראשונים בכתובת, (בד"כ 23 ביטים ראשונים), הביטים הראשונים הם קבועים ומייצגים את הכתובת של הרשת, הביטים האחרונים מייצגים את הכתובת של המחשב ברשת.

- $sub\ net\ mask$: מגדירה לנו איזה ביטים מתוך כתובת ה IP הם ביטים קבועים לכל המחשבים ברשת. כל ביט שתופיעה בו הספרה 1 הוא קבוע לכל מחשבי הרשת, ואת שאר הביטים אנו מחלקים בין מחשבי הרשת. כתובות $mask$ ייראו כך בד"כ $255.255.255.0$ מסמנים כי 24 הביטים השמאליים קבועים לכל מחשבי הרשת.

- **שליחת הודעה בין מחשבים שנמצאים באותה הרשת:** תיעשה ישירות ללא מעבר דרך הראוטר, כי המשב השולח רואה את כתובת ה IP של היעד, ומבין שהם נמצאים באותה $sub\ net$. ושולח את ההודעה ישירות ליעד.

- **היררכיית כתובות** IP: הכתובת מחולקת לפי היררכיה מסויימת כך שהביטים הראשונים מייצגים את הארגון\ספק אליו שייכת הכתובת. לכן כשנרצה לשלוח הודעה לכתובת IP מסויימת, תחילה נעביר אותה לארגון שאחראי על הכתובת לפיהביטים הראשונים של הכתובת.
כל ארגון\ ספק מציג לעולם בחוץ את מרחב כתובות ה IP עליהן הוא אחראי. אם גוף מסויים המקבל שירות מהספק יחליט שהוא עוזב את הספק הנוכחי ועובר לקבל שירות מספק אחר, יכול להיות שהוא יעבור עם כתובות ה IP שהוקצו לו אל הספק החדש. במקרה זה הספק החדש יציג לעולם החיצון גם את מרחב הכתובות של הארגון החדש שהצטרף.

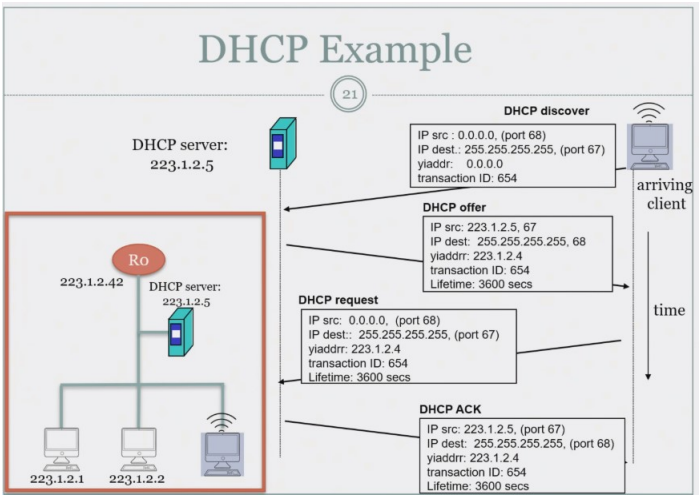
- **מה יקרה כשנרצה לשלוח הודעה לארגון שכתובותו נמצאת אצל כמה ספקים:** אם ארגון עבר מספק אחד לאחר, הספק החדש ייצג מרחב כתובות IP ספציפיות יותר. לכן ההודעה תועבר לספק שמכיל תיאור יותר ספציפי של הכתובת.

- **ארגון** $ICANN$: ארגון שאחראי על כתובות ה IP, והוא זה שחילק אותן לספקי האינטרנט.

6.3.1 פרוטוקול $DHCP$ - לקבלת כתובת IP:

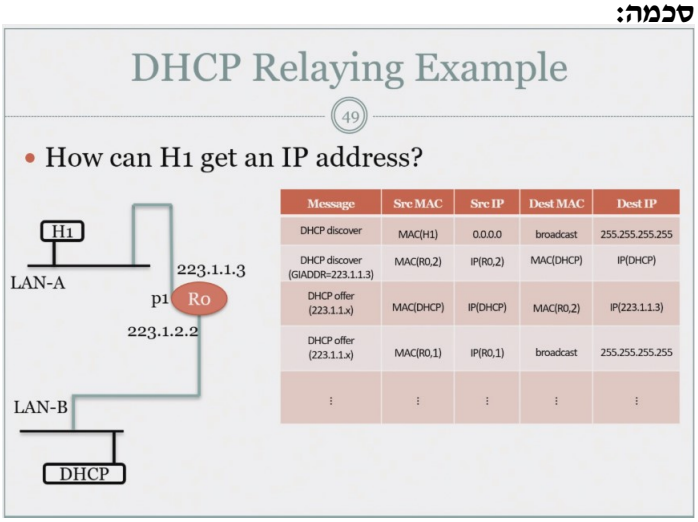
- **מה זה:** קיים שרת בתת הרשת המריץ את פרוטוקול $DHCP$, האחראי לחלוקת IP בתת הרשת למכשירים חדשים שמתחברים. נשלח לו בקשה, והוא יחזיר לנו הודעה עם כתובת IP.

- **הפרוטוקול:**
1: כשנתחבר לרשת נשלח הודעה לכל הרשת כדי לזהות היכן נמצא ה $DHCP$.
2: לאחר שההודעה תגיע ל $DHCP$ הוא ישלח בחזרה הודעה לכל הרשת, עם ה IP שלו, עם כתובת IP חדשה למחשב החדש, בנוסף נשלח מספר טרנזקציה כדי שנוכל לעקוב אחרי המחשב שקיבל את הכתובת. נשלח גם $life\ time$ כי כל כתובת IP מוגדרת בזמן.
3: נחזור שוב על אותן ההודעות כדי לוודא כי והכתובת שקיבלנו היא הכתובת שה $DHCP$ הציע.
4: שרת ה $DHCP$ יחזיר את התכתובת המתאימה שהציע לנו.
הפרוטוקול:



- **מה ה $DHCP$ שולח בנוסף:** הוא שולח את מסיכת הרשת, ואת ה IP של הראוטר.
- **תחנת ממסר - $DHCP\ Relay$:** כשיש לנו מספר רשתות ברודקאסט אך רק אחת מהן מחוברת ל $DHCP$, אנו נשים באמצע $Relay$ שהוא מעין תחנת ממסר שתעביר ל $DHCP$ את ההודעות משאר רשתות הברודקאסט. הוא ישמור אצלו את כתובת ה IP וה MAC של ה $DHCP$ ויוכל להעביר אליו ישירות את הבקשות.

סכמה:



6.3.2 פרוטוקול ARP לקבלת כתובת MAC של היעד:

- **טבלת address resolution protocol** $ARP -$: לכל מחשב, סוויץ' וראוטר תהיה טבלה שממפה כתובות IP לכתובות MAC. וכל שורה בטבלה תימחק כל זמן מסויים כי יש לה $TTL -$ time to leav.

  מה השימוש: למעשה טבלה זו עוזרת לנו לתרגם בין שכבה 2 לשכבה 3 של האינטרנט.

- **הפרוטוקול:** כשנרצה לשלוח הודעה למחשב מסויים נסתכל בטבלה, אם אין לנו את כתובת המאק של כתובת ה IP. נשלח לכולם בברודקאסט שאנו צריכים את כתובת ה mac של כתובת IP מסויימת. המחשב שיש לו את אותה כתובת IP ישמע את ההודעה ויחזיר ב $unicast$ את הכתובת MAC שלו למחשב השולח.

  למעשה מי שיחזיר את הכתובת הוא הנתב שאחרי על מחשב היעד, משום שרק נתבים יודעים להבין כתובות IP.

- **איך נמצא כתובת MAC של מחשב מחוץ לרשת שלנו:** ראשית נוודא כי היעד לא נמצא ברשת שלנו, ע"י כתובת ה IP. לאחר מכן נבקש את כתובת הראוטר של המחשב המרוחק, בעזרת הראוטר שלנו עם ה IP של הראוטר שלנו (שקיבלנו מ $DHCP$).

6.3.3 כיצד נעביר הודעה ברשת:

- ראשית נשים לב כי לכל רגל של הראוטר היא $sub\ net$ שיש לה כתובת MAC וכתובת IP משלה.
- כשנרצה להעביר הודעה ברשת, כתובות ה IP במטא דאטה לא יישתנו, אך כתובות ה MAC ישתנו בכל שלב, כי בהתחלה נשלח מהמחשב השולח אל כתובת ה MAC של הראוטר, ואחר מכתובת ה MAC של הראוטר אל כתובת היעד הבאה וכן הלאה.

6.3.4 שרת DNS:

- **שרת DNS:** הוא מסד נתונים מבוזר, שממפה כתובות אינטרנט לכתובות IP.
- כשאנו שולחים הודעה אנו שולחים אותה עם כתובת, אך אין לנו את כתובות ה IP. השרת רוצה לפצל את העומס, כך שיהיו מספר כתובות IP שונות לאותו השרת לפי הקירבה בין המקור ליעד, או לפי עומס. את זה נעשה באמצעות DNS.
  לדוגמה - כשנרצה להתחבר לגוגל, אנו נשלח ל DNS שאנו רוצים את גוגל, והוא ישלח לנו את כתובת ה IP של השרת הקרוב ביותר של גוגל.
- **היררכיית שרתי DNS:** יהיו מספר שרתי DNS כך שכל אחד מהם יודע כתובות IP עם סיומת מסויימת $(com, org, il, uk...)$. וכולם מחוברים לשרת שנקרא $root$.
  כל כתובת של אתר מחולקת לפי הנקודות "." לשרתי DNS, כל קטע עד לנקודה מסמן שרת.
  כשנשלח בקשה תחילה נשלח אותה לשרת ה $root$, והוא יחזיר לנו מי השרת שאחראי על הכתובות מהרמה הזאת (שם וכתובת IP של השרת). לאחר מכן נמשיך עד שנגיע לשרת שמאחסן את הדאטה.
- **$DNS\ Caching$:** כשאנחנו פונים לשרת DNS מסויים, הוא לא פונה ישירות לשרת ה $root$. אלא הוא בודק בקאש האם יש לו את אחת מהכתובות הללו, אם כן הוא פונה לשרת שמחזיק את הכתובת המתאימה.
  בנוסף יש למידע שנשמר בקאש תאריך תפוגה , שנקבע ע"י השרת של הכתובת (TTL).
- **רשומות של שרתי DNS:** קיימות שתי רשומות, NS ו A. הרשומה NS תחזיק את המיפוי של הדומיין לשם השרת. והרשומה A תחזיק מיפוי של שם השרת לכתובת ה IP.
- **איך נוסיף ארגון ל DNS:** תחילה נקבל כתובת IP, ולאחר מכן נפנה לארגון שאחרי על השרת (לדוגמה com.) כדי שיוסיף לשם את השרת שלנו ואת כתובת ה IP שלו. בנוסף נצטרל להקין שרת DNS שיהיה אחראי על הכתובת שלנו.

התקפות על DNS:

1. **מתקפת** $DDoS$**:** מתקפה בה התוקף מציף את השרת בבקשות לגיטימיות, כך שהשרת נאלץ לטפל בהן ולא מתפנה לטפל בשאר הבקשות האמיתיות.
מתקפות כאלה על שרתי ה $root$ כמעט ולא גרמו נזק משום שלשאר השרתים והמחשבים יש $cach$ וכל הכתובות החשובות היו שמורות שם.

2. מתקפה נוספת היא מתקפה בה התוקף נמצא איתנו על אותה רשת. וכשנשלח בקשה לשרת ה DNS התוקף יחזיר לנו כתובת IP שגויה, שתפנה אתנו אל האתר המפוקפק שלו.

3. **מתקפת** $Cach$**:** אם תוקף מצליח לשתול רשומה רעה בתוך הקאש שלנו, הכתובת הזו תישמר במערכת. והיא לא תפוג עד שלא יגמר ה TTL, שאותו התוקף יכול לקבוע למספר גבוה של שנים.
$Query\ ID$**:** עבור מתקפה זו קיימת אבטחה. כשנשלח הודעת בקשה לשרת DNS אנו נשלח 16 ביטים שיאמתו את ההודעה. כך שתוקף שמחזיר לנו הודעה יצטרך לנטר את הבקשה (להימצא על הקו) או לנחש נכון את הביטים.

4. **מתקפת** $cach$ **של קמינסקי:** נשלח בקשה לשרת של הספק שלנו $(resolver)$ ונבקש ממנו למצוא כתובת שלא שמורה אצלו בקאש, לדוגמה - $1.google.com$. במקביל, אנחנו נענה לבקשות שיתקבלו וננסה לנחש את ה $Query\ ID$, כנראה שאחרי מספר גבוה של פעמים אנו נצליח. התשובה שנשלח לשרת ה $resolver$ היא שם של שרת שמתחזה לשרת שאחרי על הכתובת $google.com$ עם כתובת ה IP שלו. השרת ישמור אותו בקאש, באופן זה לאחר שה TTL של השרת המקורי של גוגל יפוג, כל הבקשות יגיעו לשרת של התוקף.

6.3.5 טופולוגיה של הרשת ומדידת ביצועים:

אנו רוצים לדעת כיצד לבנות את הרשת שלנו. ולאחר שבנינו את הרדת למדוד כמה היא טובה ביחס לבניה אחרת. כך נוכל להעדיף בניה מסויימת על פני אחרות.

פרמטרים למדידת ביצועי רשת:

1. **קוטר:** המרחק הקצר ביותר בין שני קודקודים הכי רחוקים בגרף. כלומר, שני הקודקודים שיקח להם הכי הרבה זמן לתקשר בניהם.
נרצה שערך זה יהיה נמוך.

2. **Bisection bandwidth:** נרצה למצוא את צוואר הבקבוק של הרשת. נחלק את הרשת לחצי כך שמכל צד שלה יש $\frac{n}{2}$ קודקודים, ונמדוד מה המספר המקסימלי של הודעות שניתן להעביר בניהם. נעשה זאת עבור כל החלוקה של הגרף לשתי קבוצות שוות, ונקבע את הפרמטר הזה להיות הקבוצה עם הביצועים הנמוכים ביותר. כלומר החתך הזה מהווה מינימום על יכולות הרשת.
חסרונות של השיטה: אם יש לנו קבוצה של $\frac{n}{2}$ קודקודים שבתוכה יש צוואר בקבוק אנחנו לא נעלה על זה.
הפתרון הוא: לעבור על כל החלוקות של הגרף לקבוצות ולבדוק מה ממוצע הצלעות שיוצאות מכל קבוצת קודקודים.
נעשה זאת כך:

$$\min_{S\subset V, 0<|S|\leq\frac{n}{2}} \frac{\text{EdgesBetween }(S, V\backslash S)}{|S|}$$

נרצה שערך זה יהיה גבוה.

3. *Diameter*: דרגת הקודקודים, מספר הפורטים שיוצאים מהנתב.

תורת הגרפים והגדרות:

1. **גרף** *expander*: גרף שבו עבור כל קבוצת קודקודים, ממוצע מספר הקודקודים שיוצאים מהקבוצה הוא גבוה. באופן זה אין צוואר בקבוק קריטי שיעכב את התקשורת ברשת.

2. **בניות מפורשות לעומת לא מפורשות** *Explicit non Explicit*:
בניה מפורשת: לכל קודקוד בגרף יש שם, והשמות מגדירים את הצלעות והמסלולים בגרף. שינוי שמות הקודקודים ישנה את הגרף.

6.3.6 אלגוריתם *Routing* לניתוב ברשת:

נרצה אלגוריתם שיעזור לנו להעביר את הדאטה בין הנתבים. באותו האופן שהעברנו את המידע בין הסוויצ'ים, גם הנתבים צריכים לדעת מה המסלול הקצר ביותר ליעד וכיצד להעביר את המידע. לכל לינק יהיה משקל שהוגדר על ידי מנהל נרשת.
טבלת מידע: כל ראוטר יחזיק טבלה בה יש שלש עמודות - מספר היעד, דרך מי להעביר כדי להגיע ליעד והעלות. כל ראוטר ישמור טבלה המכילה את כל הראוטרים ברשת וכיצד הכי משתלם לו להעביר את המידע אליהם. הוא תמיד ישמור את הקודקוד הבא אליו הוא צריך להעביר את המידע.

יש שתי סכמות למציאת המסלול:

1. **Distance Vector**: כל נתב שולח ווקטור לשכנים שלו עם מידע על המרחקים שלו מכל יעד, והמחירים. לאחר מכן נריץ בלמן פורד.
באלגוריתם זה **המידע מבוזר**.
האלגוריתם: בשלב האיתחול כל ראוטר מחשב את המרחק שלו לכל הראוטרים שקרובים אליות מעדכן את הוקטור ושולח אותו לשכנים שלו. שני ראוטרים שאין בניהם לינק, מרחקם יוגדר להיות אינסוף. בשלב העדכון כולם יערכנו את הווקטורים שלהם לפי הוקטורים של שאר הראוטרים. האלגוריתם ייתכנס כששתי איטרציות יהיו אותו הדבר - כל הווקטורים שווים.
מימוש: כשנתב ירצה לשלוח מידע, הוא יחשב בעזרת הטבלה את המרחק בינו לבין השכן שהמרחק ממנו ליעד הוא הקצר ביותר, וישלח דרכו.
חסרון: כשנעדכן משקל של צלע למשקל גבוה יותר, יקח לרשת זמן לעדכן את המשקל החדש. כי יווצר מצב שבו שני קודקודים שרוצים להגיע לקודקוד שלישי, תמיד יעדיפו לעבור אחד דרך השני כי זה המרחק הקצר ביותר שעדין מעודכן מלפני השינוי של משקל הצלע. וניכנס ללולאה עד שנעבור את משקל הקשת החדשה.
פתרון - poisoned reverse: הבעיה נוצרת כששני קודקודים רוצים לעבור אחד דרך השני בכדי להגיע לקודקוד שלישי. לכן כשקודקוד a רוצה להגיע לקודקוד c דרך קודקוד b, הוא ישלח לקודקוד b כי משקלו ל c שווה לאינסוף. כך b ימשיך ל c מדרך אחרת ולא יבחר לעבור דרך a.

2. *Link State*: כל נתב שולח לכל הנתבים מי הנתבים שמחוברים אליו ומה המשקל של הצלעות. כך לכל נתב יש את המידע על כל הרשת והוא יוכל להריץ דייקסטרה.

באפשרות זו המידע על הרשת **גלובאלי**.

מימוש: כל נתב יודע את מצב כל הרשת. לאחר מכן הוא מריץ דייקסטרה, הוא מחשב את המרחק הקצר ביותר ממנו אל שאר הראוטרים, לאחר מכן כשהוא רוצה לשלוח הודעה הוא בודק בטבלה והוחר את המסלול הקצר ביותר.

6.3.7 אלגוריתם למציאת Max-Multicommodity Flow:

הרעיון: אלגוריתם הדומה ל $Max\ flow$ רק שהוא מיועד להעברה בין כמה קודקודי מקור לכמה קודקודי בור, ולא רק מקוד אחד ובור אחד.

נרצה להשתמש באלגוריתם כדי להחליט איך להעביר מידע ברשת. אנחנו לא נרצה זרימה מקסימלית, אלא אנחנו נרצה זרימה שבה כולם יכולים לשלוח ואין קודקוד שמורעב. כלומר - פיזור הזרימה והביקושים ברשת.

אותנו מעניין כי הזרימה על הצלע הכי עמוסה, תהיה מינימלית, ולא הזרימה המקסימלית ברשת. כך נפזר את התעבורה בכל הרשת.

מה האלגוריתם מחפש: האלגוריתם מקבל את כל המסלולים הכי קצרים בין שתי נקודות a, b. (המסלולים הכי קצרים חושבו בשלב קודם והתחשבו במשקולות w). בנוסף הוא מקבל את c - הקיבולת שכל צלע יכולה להעביר. בהתחשב באינפוט הזה הוא מוצא את הזרימה המקסימלית ברשת.

יש שתי בעיות שבהן נתעסק:

1. **המטרה:** להוריד את העומס מהצלע הכי עמוסה ברשת.

$$\text{minimize}\left(\max_{e}\left(f_e/c_e\right)\right), \quad \text{where} \quad f_e \ \text{ is the flow along edge e}$$

כלומר: נעבור על כל ההצבות האפשריות, וההצבה שבה הצלע הכי עמוסה היא מינימלית - נקח אותה.

מוטיבציה: באופן זה אנחנו יכולים להתמודד עם הפתעות כמו לינק שנפל, או תעבורה גבוה בפתאומיות. כך אף לינק לא מוצף.

נשים לב!! בבעיית המינימום - אין לנו אילוץ כי $f_e \leq c_e$, כלומר - אין הגבלה על הזרם.

2. $Max - MCF$:

המטרה: מקסום הזרם. נרצה שהזרימה ברשת תהיה מקסימלית.

נחפש את:

$$max\left(\sum_{v\in V}|f_v|\right)$$

כלומר: נסכום את כל הזרימות שיוצאות מקודקוד v, וזאת תהיה הזרימה של הקודקוד v.

איך נפתור את הבעיות:

נכתוב את שתי הבעיות כבעית תכנון לינארי עם אילוצים, יהיו לנו כמה אילוצים:

- **חוק שימור הזרם:** סך הזרימה שיוצאת מקודקוד v שווה לסך הזרימה שנכנסת אליו.

- **הזרימה תמיד גדולה מ 0:** $f_e \geq 0$
- **הזרימה על כל צלע תהיה קטנה שווה לקיבול שלה:** $f_e \leq c_e$.

נשים לב!! בבעיית המינימום - אין לנו אילוץ כי $f_e \leq c_e$, כלומר - אין הגבלה על הזרם.

6.3.8 מנגנון Equal cost multipath (ECMP):

אינטואציה: אם יש לנו שני מסלולים קצרים מ $s \Rightarrow t$ עם אותו המשקל, אנחנו נחלק את הזרימה חצי חצי בין שני המסלולים. באופן זה בכל פעם שקודקוד ימשיך לשני מסלולים בעלי אותו המשקל - הפקטות יפוצלו וחצי מהן יעברו בכל מסלול.
לכן נבצע שינוי: והאלגוריתם שמחזיר לנו את המסלול הכי קצר בין שני קודקודים, יחזיר לנו את **כל** המסלולים הכי קצרים בניהם.
משקל הלינק: יוגדר להיות $\frac{1}{\text{link capacity}}$.
מי מגדיר את משקל הצלעות: מנהל הרשת צריך להגדיר את משקלי הצלעות כך שיהיו לו מלא מסלולים הכי קצרישווים במשקלם, כדי שהאלגוריתם יפצל את הפקטות בין המסלולים.
חסרונות: אם נעביר חבילות של שידור לייב, אנחנו לא נרצה לפצל את הפקטות כי אז נחווה ניתוקים וחבילות לא יגיעו באותו הסדר. לכן נרצה שחבילות ששייכות לאותה האפליקציה יגיעו באותו המסלול.
כיצד נדאג שחבילות יגיעו דרך אותו הלינק: נשתמש בהאש. נמפה חבילות ששייכות לאותה באפליקציה לאותו הלינק. נעשה את ההאש על ה $nexst\ hop$ שנמצא ב $header$ של הפקטה, באופן זה לחבילות של אותה אפליקציה יש את אותו ה $header$.
הבעיה עדיין קיימת: מכיוון שחבילות שהגיעו מאותו הראוטר אך מאפליקציות שונות, יגיעו לאותו $next\ hop$ ולכן לא יפוצלו. למרות שהן היו יכולות להתפצל כי הן מאפליקציות שונות.

6.4 שכבת ההעברה - $Transport$:

השירות: שכבה המאפשרת תקשורת בין תהליכים במחשבים של משתמשי קצה, ולא בין משתמשי קצה עצמם (פתיחת סוקט).
ממשק: שליחת הודעות בין תהליכים.
פרוטוקולים: אחראית לכך שמידע יעבור כמו שצריך, שולחת בקצב הקיבול, דואגת שהחבילות יגיעו ולפי הסדר.
מי מדבר בשכבה זו: היא ממומשת באפליקציות ולכן רק משתמשי הקצה מדברים עם הפרוטוקול הזה.
מטרת הפרוטוקול: הוא מקשר בין תהליכים במחשב. כך הוא מבדיל בין שיחת לייב להעברת קבצים. (הוא לא מקשר בין כתובות IP).
מיקום השכבה: שכבה זוממוקמת בין שכבה 3 לשכבת האפליקציה.
מה היא באה לפתור: שכבה זו באה לוודא כי החבילות ששלחנו אכן יגיעו.
מה לא ניתן להבטיח: את זמן ההעברה של הפקטה.

6.4.1 פרוטוקול UDP - connectionless:

פרוטוקול שמאפשר את ההעברה בין מכשירי קצה, אך הוא **לא מבטיח העברת מידע ללא תקלות**. כלומר אין שחזור של חבילות שנפלו או מעקב אחר פקטות.

אין אפשרות לנהל שיחה כי המקור לא מזוהה.

כיצד נזהה את היעד - $connectionless$**:** השולח שולח עם קובץ ה $header$ של הפקטה את מספר ה IP של היעד, ואת מספר הפורט של האפליקציה ביעד שאליה אמור המידע להגיע.
כלומר - אם נשלח מאפליקציות שונות במחשב המקור, אל אותה אפליקציה במחשב היעד, אנו נעשה הכל דרך $socket$ אחד.

יתרונות של UDP**:**

- פרוטוקול זה לא מחייב את השולח לכתוב את כתובת ה IP שלו, לכן ה פרוטוקול שהאינטרנט צריכה כדי לתפקד. לדוגמה כאשר אנחנו מחברים מכשיר חדש לרדת, עדיין אין לו כתובת IP, ואם היינו משתמשים בפרוטוקול TCP לא היינו יכולים לחבר את המכשיר לרשת, כי הוא לא היה יכול לתקשר עם שרת ה $DHCP$.
- הוא יעיל יותר כשאנחנו רוצים להעביר קצת מידע, אנחנו שולחין רק את המידע בלי חבילות מיותרות.
- הוא יעיל בשליחת מידע חייב, כשאנחנו לא צריכים שחזור של חבילות שנפלו.

6.4.2 פרוטוקול TCP - connection oriented:

פרוטוקול זה מספק **שידור אמין -** מספק שירות של שחזור פקטות, שמירה על סדר פקטות, קצב שליחה ועוד.
הפרוטוקול מתנהל כמעין שיחה, על כל פקטה המקבל צריך להודיע לשולח כי הוא קיבל את ההודעה. בנוסף המקבל צריך להקצות מקום לחבילות שמגיעות. לעקוב אחר הפקטות שמגיעות ועוד.

בפרוטוקול זה, בניגוד ל UDP, כהשולח ישלח את החבילה, ויפתח את ה $socket$ - הוא יוגדר לפי כתובת ה IP וה $port$ של היעד, אך **גם של המקור** כך הם יוכלו לתקשר.

כיצד נתקשר בפרוטוקול זה - לחיצת יד משולשת:

- השולח שולח הודעת SYN שמגדירה את מספר הפקטה הראשונה.
- המקבל שולח הודעה לשולח, ומגדיר גם כן מה מספר הפקטה הראשונה שהוא ישלח.
- השולח שולחח למקבל פקטה.

סיום התקשרות:

- המקבל שולח הודעת FIN שמודיעה שהוא סיים.
- השולח שולח הודעה שבה הוא מודיע שקיבל את ההודעה.

השולח, שומר את הפקטות אצלו כל עוד הוא לא קיבל הודעה מהמקבל שהן התקבלו. אם הוא לא קיבל אישור (הודעת ACK) שהן התקבלו ועבר ה $timeout$, הוא שולח אותן שוב.

הודעת ה ACK**:** המקבל, שולח הודעת ACK בה מופיע **מספר הביטים שהוא קיבל** עד כה כמו שצריך. אם הוא קיבל פקטה מספר 1 ואחכ פקטה מספר 3, הוא ישלח ACK שמציין את מספר הביטים שיש בפקטה 1.

הגדרות: גודל הדאטה עצמו יסומן ב MSS, והראטה עם ה $header$ יסומן ב MTU.

איך נגדיר את הזמן של $timeout$**:** בזמן שיש את לחיצת הידיים המשולשת, השולח מחשב מה זמן ה RTT של הרשת. כלומר כמה זמן לוקח לשלוח הודעה ולקבל תשובה.
נעשה זאת ע"י הוספת משתנה, ל $timeout$ שכבר שמרנו.

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

בד"כ נגדיר $\alpha = 0.125,\ \beta = 0.25$.
נרצה להתחשב גם בשונות של הרשת, לכן נגדיר כך:

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

לבסוף נגדיר את ה : $timeout$

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

duplicate ACKs: כשהשולח מקבל אותה הודעת ACK מספר פעמים, הוא צריך להסיק מזה שפקטה נפלה לו בדרך, ולכן על כל פקטה שהוא שולח הוא מקבל ACK על ההודעה שהתקבלה לפני שהפקטה נפלה, הוא צריך להסיק מזה שיש פקטה שנפלה, אך זה מנחם כי הוא יודע שהרשת מעבירה מידע. לכן הוא צריך לנצל את האינדיקציה לכך שנפה פקטה ולטפל בבעיה.

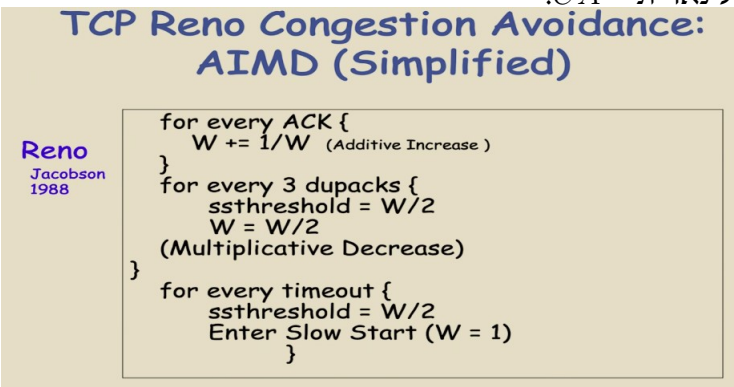
כיצד הפרוטוקול קובע כמה פקטות לשלוח: הפרוטוקול מתחיל מלשלוח פקטה אחת (גודל החלון $w = 1$), ועבור כל הודעת ACK שהוא מקבל הוא מעלה את מספר הפקטות שוא שולח ב 1 $(w+=1)$. כך הוא גדל אקספוננציאלית כל עוד הרשת מאפשרת את השליחה.
השלב של הגדילה האקספוננציאלית נקרא $SS-$ slow start, והוא חסום ע"י $SS\ threshold$. מרגע שאנחנו מגיעים לסף העלויון, אנחנו עוברים לשלב של $CA-$ Congestion Avoidance, ועבור כל ACK שמתקבל, מגדילים את החלון ב $\frac{1}{w}$, - עליה לינארית.
לאחר מכן אנחנו נתחיל להקטין את גודל החלון.
יש שני פרוטוקולים להקטנת החלון:

1. **פרוטוקול** $Tahoe$**:** ברגע שקיבלנו $timeout$ או שנפלו פקטות (קיבלנו duplicate ACKs) - גודל החלון יורד ל 1. ואנחנו מעדכנים את $SS\ threshold$ להיות חצי מגודל החלון שהיינו בו.

2. **פרוטוקול** $Reno$**:** אם יש $timeout$ - הוא מתנהג כמו $Tahoe$.

אם יש duplicate ACKs- נקטין את גודל החלון ואת $SS\ threshold$ להיות חצי מגודל החלון הנוכחי, ונמשיך בעליה

לינארית - CA.



שרת NAT**:** שרת שעוזר לנו להתמודד עם חוסר של כתובות IP. בנוסף הוא מאובטח יותר כי אנחנו לא חושפים את הכתובת ה IP שלנו לעולם.

ביצוע: נגדיר שרת NAT שיש לו כתובת IP אמיתית, לכל שאר המחשבים שתחתיו הוא יחלק כתובות וירטואליות. כלומר הכתובתשלהם לא תהיה כתובת IP עולמית. כשהמחשבים שתחת השרת ירצו לשלוח הודעות, הם יפתחו סוקט עם פרוטוקול TCP וישלחו את ההודעה דרך שרת ה NAT. השר יפתח סוקט עם TCP גם כן ויתכתב עם מחשב היעד. כשהוא יקבל הודעה מהיעד הוא ישלח אותה למחשב השולח בעזרת טבלה שהוא שומר בה הוא ממפה את הסוקטים למחשבים.

6.4.3 בעיית העומס - congestion comtrol:

בעיית העומס בשכבת ההעברה ברשת, היא בעיה שנוצרת בעקבות כך שמלא משתמשי קצה משתמשים בכל המשאבים.במצב כזה הרשת תוצף, והיא יכולה ליפול. לכן נרצה שכל משתמש יווסת את השליחה שלו.

בעיה נוספת, בעיית $flow\ control$**:** בעיה דומה אך דומה, הבעיה הזאת קלה יותר, והיא מתרחשת כשמשתמש קצה אחד שולח במהירות מידע שהמקבל לא מסוגל לעבד.

פתרון הבעיה: המקבל יגיד לשולח באיזה מהירות לשלוח.

קצב השליחה של TCP**:** הפרוטוקול מגדיר חלון בגודל W, ובכל שלב הוא דואג שיהיו Wפקטות באוויר. ברגע שהוא מקבל ACK על הפקטה הראשונה, הוא שולח עוד אחת.

קצב השליחה כזה:

$$\frac{W \cdot \text{packet size}}{RTT}$$

פתרון בעיית congestion comtrol: הפתרון הוא לשחק עם הגודל של W. באופן זה אנחנו נשנה את קצב השליחה.

איך נבחר את הגודל של W **- TCP congestion comtrol:**

פרוטוקול $Reno$ או פרוטוקול $Tahoe$ שהזכרנו בחלק של TCP.

6.4.4 פרוטוקול BGP:

הפרוטוקול: אנחנו רוצים לחבר ארגונים קטנים לרשת אחת גדולה. כל ארגון רוצה להעביר את המידע שלו ליעד דרך רשתות שונותת כשהאתגר הוא לשלם כמה שפחות כסף. כלומר, אנחנו לא מחפשים את המסלול הקצר ביותר, אלא את המסלול המשתלם והזול ביותר.
ספק אינטרנט קטן שרוצה להעביר מידע מישראל לארה"ב, יעדיף להעביר במסלול ארוך יותר דרך ספק גדול יותר שיגבה לו פחות כסף על תעבור המידע.

אלגוריתם למציאת מסלולים:
בשונה מהאלגוריתם לניתוב ברשת, שבו כל קודקוד ידע רק למי הוא מעביר את החבילה, ולא ידע מהו כל המסלול. באלגוריתם זה כל קדקודי שרוצה לשלוח מידע ליעד, יודע מה המסלול המלא.

1. כל קודקוד מקבל הודעת BGP מהשכנים, שאומרים לו לאיזה קודקודים הם זמינים.
בפועל: כל ארגון מדווח לעולם מה מרחב הכתובות שלו, ומה מרחבי הכתובות שיש לו גישה אליהן. כל מי שרוצה להעביר דרכו מידע, ישלם לו על ההעברה.

2. הוא ישמור רק את המסלולים שהוא מעדיף, וישליך את השאר.
בפועל: יש שני סוגי יחסים בהעברת מידע, ספק - לקוח: הלקוח משלם לספק על זה שהספק מעביר לו מידע. עמית: שניהם מעבירים אחד לשני ולכן אף אחד לא משלם, עד כמות מסויימת שהם הסכימו בניהם.
ארגונים יבחרו דרך מי להעביר את המידע לפי המחיר הזול ביותר.
איך נימנע מלולאות: באלגוריתם ניתוב ברשת, דאגנו שהשכן שאחנו עוברים דרכו לא יידע על המסלול שלנו, וכך לא יעבור דרכנו בחזרה. באלגוריתם הזה, אנחנו שולחים את כל המסלול, ולכן אם אחד השכנים רואה שהוא מופיע במסלול הוא לא יבחר אותו.

3. יבחר מסלול מועדף.
פרוטוקול $iBGP$**:** תחילה נבחר לשדר דרך הבחירה המוגדרת (לרוב זה יהיה המסלול הסול ביותר). אם יש שוויון בין שני מסלולים, נבחר את הקצר ביותר.

4. יעדכן את השכנים על המסלול.
בפועל: הקודקוד לא שולח לשכנים את כל המסלולים שלו, אלא רק את המסלולים שרווחי להם שיעברו דרכם.
אם המסלול עובר דרך לקוח: הוא יספר עליו לכל השכנים (ספקים, לקוחות, עמיתים).
אם המסלול עובר דרך ספק: הקודקוד יעביר את המידע רק לקודקודים שהם לקוחות שלו.
אם המסלול עובר דרך עמית: הקודקוד יעביר את המידע רק לקודקודים שהם לקוחות שלו. כדי שהעמית לא יחרוג מהכמות בהסכם בניהם.

תנאי Gao Rexford: תנאי שאומר לנו מה התנאים שצריכים להתקיים כדי ש BGP יהיה יציב.

1. אין מעגל בין ספק ללקוח (מתבצע בשלב 2, נימנע מלולאות).

2. נעדיף להעביר דרך לקוח (מתבע בשלבים 3,4).

3. תנאי הייצוג (אוטומטי).

מפגעי אבטחה:

1. **מתקפה על ידי הכרזת מרחב כתובות** IP**:** משתמש יכול להכריז מרחב כתובות IP של ארגון אחר, כך כל התעבורה שמיועדת להארגון השני תגיע אליו.

2. **מתקפה על ה** $Data\ plan$**:** ניתן לשלוח את הדאטה למקום אחר מהמקום שכיוונו אליו, ולבחור מסלול אחר.

6.5 שכבת האפליקציה:

השירות: יצירת המידע ושימוש במידע.
ממשק: תלוי באפליקציה.
פרוטוקול: תלוי באפליקציה

7 תיקון שגיאות:

- **הבעיה:** אנו שולחים פקטה וכתוצאה מרעש (לדוג' מגנט שמשנה את אחד הביטים) הביטים משתנים, ואנו צריכים לתקן את הבטים ששונו.

פתרונות:

7.1 $Repetition\ Code$:

- $Repetition\ Code$**:** נשכפל את הקוד, ונשלח אותו 3 פעמים, כך ברגע שביט משתנה המחשב המקבל יזהה שיש טעות ויחליט לפי החלטת הרוב.
הבעיה: אם ביט אחד השתנה המחשב המקבל יתקן נכון, אם 2 ביטים השתנו - הוא יזהה שיש תקלה אך יתקן לא נכון.

- **כיצד השינוי מתבצע בפועל:** יש לנו שני ווקטורים אפשריים $(0,0,0),(1,1,1)$. כך כשאנו משכפלים את הקוד אנו עוברים מחד לתלת מימד. כשמגיע ביט ואנו מזהים שיש בו טעות למעשה מה שקרה זה סיבוב של הווקטור, לכן אנו נעשה הטלה של הווקטור החדש ונבדוק איזה ערך גבוה יותר - $(0,0,0)$ או $(1,1,1)$ ונתקן.

- $Overhead$**:** מספר הביטים הנוסף שהשתמשנו בהם שווה ל: סך הביטים שהשתמשנו בהם חלקי הביטים ששלחנו ומהתוצאה נחסיר 1

$$\frac{3N}{N} - 1 = 2$$

כלומר השתמשנו ב 200 אחוז ביטים נוספים.

7.2 $1D\ Parity$:

- **הרעיון:** נוסיף ביט נוסף לכל פקטה שמשלים למספר זוגי של אחדות. כלומר - אם היו בפקטה $2n$ אחדות אזי הביט הנוסף יהיה 0. ואם היו $2n+1$ אחדות - הביט הנוסף יהיה 1.
- $Overhead$:

$$\frac{N+1}{N} - 1 = \frac{1}{N}$$

- **החסרון:** נוכל לזהות עד שגיאה אחת בלבד, כי אם יתהפכו שני ביטים לא נוכל לדעת זאת. בנוסף לא ניתן לשחזר.

7.3 $2D\ Parity$:

- **הרעיון:** נהפוך את הפקטה לריבוע $i \times j = N$ כך שכל שורה היא $1D\ Parity$ כך שיש ביט נוסף לכל שורה ולכל עמודה. והביט הנוסף בודק את הבדיקות.
- $Overhead$:

$$\frac{i \cdot j + i + j + 1}{i \cdot j} - 1 = \frac{i + j + 1}{N}$$

- **היתרון:** יכול לזהות עד 3 שגיאות, 4 הוא כבר לא יוכל לזהות (כולל שגיאה של הבדיקות).
  יכול לשחזר - רק אם ביט 1 שונה. אחרת הוא ישחזר, אך לא בהכרח שהוא יתקן כמו שצריך.

8 אימות שליחת פקטות:

8.1 פרוטוקול$Stop\ and\ Wait$:

פרוטוקול שמטפל בשליחת פקטות ובקבלתן. הוא יוודאכי הפקטות הגיעו בסדר הנכון ושלא התפספסה אף פקטה בדרך. האלגוריתם ירוץ מצד שרת וצד לקוח בשתי ווריאציו שונות לכל אחד מהם.

צד לקוח (מקבל): כשהמחשב מקבל את הפקטה הוא שולח פקטה שקוראים לה ACK, שמסמנת לשלולח כי הפקטה התקבלה והיא תקינה. אם יש שגיאה בפקטה (זוהה באמצעות תיקון שגיאות) נשלח $NACK$.

צד שרת (שולח): יש לו שלש אפשרויות -

1. **התקבל** ACK: שולח את הפקטה הבאה.
2. **לא התקבל** ACK: מחכה זמן מסויים, ושולח שוב את אותה הפקטה.

3. **התקבל** $NACK$**:** הפקטה פגומה, נשלח את הפקטה מחדש.

מספר בעיות:

- **נשים לב:** כי המחשב השולח יכול לא לקבל ACK משתי סיבות. או שהפקטה לא נשלחה, או שהפקטה התקבלה אך הודעה ה ACK לא התקבלה אצל השולח.
  לכן: אנו נמספר את הפקטות, כך שהמקבל יידע אם הוא מקבל את אותה הפקטה פעמיים.
- **בעיה נוספת:** אם הודעת ה ACK מתעקבת והשולח שלח את הפקטה פעם נוספת לאחר ה $time\ out$, יכול להיווצר מצב שהוא יקבל שתי הודעות ACKעל הודעה 1 ברצף, והוא יחשוב ששתי הפקטות התקבלו.
  לכן: אנו נמספר את הודעות ה ACK כך שנדע עבור איזו פקטה כל הודעה מתייחסת.

איך נחשב את ה $time\ out$ **האידיאלי:**
נסמן את הזמן לשלוח את כל ההודעה ב T_{packet} (הזמן שלוקח להודעה לצאת מהשולח).
את הזמן שלוקח להודעה להגיע ליעד נסמן ב T_{prop} (הזמן שלוקח להודעה להגיע למקבל).
את הזמן שלוקח לשלוח את הודעת ה ACK נסמן ב T_{ACK} (הזמן שלוקח להודעה לצאת מהמקבל).
סה"כ: נתחיל לספור מרגע שההודעה יצאה מהשולח (לאחר זמן T_{oacket})

$$time\ out = 2 \cdot T_{prop} + T_{ACK} + TPT$$

הערה: יש זמן עיבור TPT אז לרוב נזניח אותו.

ניזכר כי נחשב כך:

$$T_{prop} = \frac{S}{V_{prop}} = \frac{\text{distance between stations}}{\text{prop speed}}$$

$$T_{ACK} = \frac{S}{B} = \frac{\text{ACK packet size}}{\text{transmission rate}}$$

$$gootput = \frac{E\left[p_{succ}\right]}{\text{relavant total time}} = \frac{T_{packet}}{E(T_{suc})}$$

כאשר סך הזמן הכולל הרלוונטי הוא כל הפעמים שניסינו לשלוח את הפקטה גם ללא הצלחה (הזמן המבוזבז).
אפשרות שניה לחישוב ה $goodput$**:** במקום להסתכל על פקטה מסויימת ולשאול כמה זמן לקח לה להישלח. נוכל להסתכל על יחידת זמן מסויימת ולשאול כמה פקטות בממוצע נשלחו בה.

8.2 פרוטוקול (GBN) Go Back N:

הפרוטוקול: נגדיר חלון בגודל מסויים, בד"כ גודל החלון יהיה $timeout + 1$.
לאחר שהפקטה השמאלית ביותר התקבלה (התקבלה עבורה הודעת Ack), אנחנו נזיז את החלון ימינה.

המקבל ישלח הודעת Ack_i המסמלת כי כל הפקטות **עד** הפקטה ה i (לא כולל הפקטה ה i) התקבלו בהצלחה.
החלון יסמן לשולח מהן ההודעות שטרם התקבל עליהן Ack.

הפרוטוקול מצד השולח: נשלח את הפקטות שבתוך החלון, נזיז את החלון כאשר נקבל על הפקטה הודעת Ack. אם לא קיבלנו על פקטה מסויימת הודעת Ack - בסוף ה $timeout$ נחזור לפקטה הזו ונשלח את כל הפקטות החל ממנה מחדש.
הפרוטוקול מצד המקבל: אם קיבלנו פקטה נשלח עליה הודעת Ack.
נשים לב: כי אם פקטה ה i התקבלה, והמקבל שלח עליה הודעת Ack שנפלה באמצע. מבחינת המקבל הוא לא יודע כי הודעת ה Ack נפלה ולכן הוא ימשיך לשלוח הודעות Ack על כל הפקטות שהוא יקבל. ברגע שהשולח יקבל הודעת Ack על הפקטה ה $i+1$ הוא יידע כי כל הפקטות עד הפקטה ה $i+1$ התקבלו ולכן הוא יזיז את החלון 2 ימינה.

חסרונות של הפרוטוקול: הפרוטוקול הזה מבזבז לנו מלא ביטים, כי הוא שולח את מספר הפקטה. ואם החלון גדול מאד גדול אנחנו נשלח $log(2)$ ביטים נוספים בכל פעם. לכן נרצה לאפס את הקאונטר של הפקטות.
כיצד נפתור את הבעיה: נגדיר את מספור הפקטות להיות מודולו של $n+1$ עבור חלון בגודל n. ולאחר מכן הקאונטר מתאפס. כך המקבל בודק את מספר ההודעה, והוא יכול לדעת אם ההודעה היא מהחלון החדש או עדיין זאת הודעה מתחילת החלון הנוכחי.

8.3 פרוטוקול Selective Repeat (SR):

הפרוטוקול: המקבל יחזיק באפר, כך שאם ההודעה ה i נפלה וההודעה ה $i+1$ התקבלה, הוא ישמור את ההודעה ה $i+1$ וכשתגיע הפקטה ה i בפעם השניה הוא ימקם אותה לפני הסקטה ה $i+1$.
והוא ישלח Ack_i על הפקטה שהתקבלה, כאשר Ack_i מזמן כי הפקטה ה i התקבלה.

חסרונות של הפרוטוקול: תהיה לנו בעיה עם המספור של החלון, לכן נקח מספור בגודל $2n$ עבור חלון בגודל n.