תאריך הבחינה: 20 ביוני 2023

שעת הבחינה: 13:30

משך הבחינה: שעתיים וחצי

מחלקה: מדעי המחשב

שם הקורס: מבוא לרשתות תקשורת

מספר הקורס: 30-1301025-1

שם המרצה: ד"ר עמי האופטמן

שנה: תשפ"ג סמסטר: א' מועד: א'

חומר עזר: מחשבון כיס (קיים נספח בסוף הבחינה)

סוג הבחינה: חסויה

הוראות:

- במבחן זה **5** שאלות
  - **שאלה 1 (מכילה סעיפים מרובי ברירה - אמריקאיים) היא שאלת חובה**
  - מבין 4 השאלות 2-5, בחרו 2 שאלות וענו עליהן
- הניקוד עבור כל שאלה מופיע בסמוך למספר השאלה. הניקוד לכל סעיף מופיע ליד מספר הסעיף, ואינו בהכרח שווה בין הסעיפים השונים.
- כתבו את תשובותיכם **על גבי טופס הבחינה**, במקומות המיועדים לכך. **מחברת הטיוטא שקיבלתם לא תיבדק**.

מומלץ לפתור קודם במחברת הטיוטא, ורק לאחר מכן להעתיק את הפיתרון לטופס הבחינה. שימו לב לדברים הבאים:

בכל אחד מהסעיפים בהם נדרש חישוב, **יש להראות בקצרה את דרך החישוב**, אלא אם כן נאמר במפורש לא לעשות זאת.

בסוף הבחינה קיימים 2 **עמודים נוספים** (לגיבוי), למקרה שהמקום לא יספיק לכם באחד הסעיפים. אם אתם משתמשים בעמוד זה, כתבו בברור את מספר השאלה ומספר הסעיף עליהם אתם עונים.

למרות הבונוסים בחלק השאלות, ניתן לצבור **עד 100 נקודות בלבד**. ניתן לענות על סעיפי הבונוס בשאלות **רק** אם בחרתם לענות על **כל** השאלה.

במבחן **-32-** עמודים כולל עמוד זה וכולל הנספח. ודאו זאת!

- הנספח (עמ' 23-32) מכיל דפי נוסחאות, **שנכונותם באחריותכם בלבד**.

אם אינכם יודעים את התשובה לשאלה או סעיף כתבו " לא יודע " ותקבלו 20% מהניקוד עבור אותה שאלה/סעיף, מעוגל **מטה**. ניתן לבצע זאת רק בסעיפים בהם הניקוד הוא 5 נק' ומעלה.

בהצלחה !!!

שאלה 1 (50 נקודות) שאלת חובה

לגבי 30 הסעיפים הבאים:

ענו על **כל** 20 הסעיפים 1-20. מתוך הסעיפים 21-30 ענו על 5 לבחירתכם.

בחלק מהשאלות מופיעה תשובה בנוסח: "יש יותר מתשובה נכונה אחת: _____"

אם בחרתם בתשובה כזו, כתבו על הקו את מספרי הסעיפים הנכונים. שימו לב כי בסעיפים אלה **לא בהכרח** קיימת יותר מתשובה נכונה אחת. מאידך, ניתן לקבל חלק מהניקוד עבור תשובה נכונה חלקית (למשל אם קיימים 2 סעיפים נכונים וסימנתם רק אחד מהם).

ראו **בהמשך** הוראות לגבי סעיפי הבחירה (בסמוך לסעיף 21)

.

1. כיצד נקראת שכבה 5 במודל 5 השכבות?
   - א. שכבת האפליקציה
   - ב. שכבת הרשת
   - ג. שכבת התעבורה
   - ד. שכבת ה-Web
   - ה. יש יותר מתשובה נכונה אחת: _____

2. מה נכון לגבי ספקי האינטרנט (ISPs)?
   - א. קיימת ביניהם היראכיה והם לא מעבירים מידע זה לזה
   - ב. לא קיימת ביניהם היראכיה והם לא מעבירים מידע זה לזה
   - ג. קיימת ביניהם היראכיה והם מעבירים מידע זה לזה
   - ד. לא קיימת ביניהם היראכיה והם מעבירים מידע זה לזה

3. מה **אינו** קשור לקצה הרשת?
   - א. Host
   - ב. Access Network
   - ג. מיתוג מנות
   - ד. Router
   - ה. יש יותר מתשובה נכונה אחת: _____

4. נניח שמשתמש יחיד לשלוח קובץ בגודל 100MB, ברוחב פס של 100MBps בשיטת TDM (או TDMA) כאשר יש 10 חריצים (slots) ו-10 משתמשים שונים, כמה MB יעברו בשניה?
   - א. 1
   - ב. 10
   - ג. 20
   - ד. אף תשובה אינה מדויקת

5. מה נכון לגבי לקוח-שרת?
   א. זו ארכיטקטורת תוכנה
   ב. זו ארכיטקטורה של הרשת
   ג. זו ארכיטקטורה של אפליקציות רשת
   ד. חייבים להיות ממוקמים באותה רשת
   ה. יש יותר מתשובה נכונה אחת : _____

6. מה נכון לגבי פרוטוקולים ואפליקציות?
   א. אין קשר – אלה דברים שונים
   ב. פרוטוקול יכול להכיל לעתים יותר מאפליקציה אחת
   ג. אפליקציה יכולה להכיל לעתים יותר מפרוטוקול אחד
   ד. פרוטוקול תמיד מכיל אפליקציה אחת ואפליקציה תמיד מכילה פרוטוקול אחד

7. מה נכון לגבי הודעות TCP והודעות HTTP?
   א. הודעות TCP ניתן לשלוח במקביל ו HTTP – לא
   ב. את שני סוגי ההודעות אפשר לשלוח במקביל
   ג. הודעת HTTP יכולה להכיל הודעת TCP
   ד. בהודעת HTTP חלק מהשדות הם טקסט אך לא בהודעת TCP
   ה. יש יותר מתשובה נכונה אחת :_____

8. מה נכון לגבי רשומות (records) ב- DNS?
   א. עוברות בהודעות, אך בהודעות יש גם מידע נוסף
   ב. שמורות בשרתי DNS בדרך כלל
   ג. בשדה Type מופיע טיפוס הנתון השמור
   ד. תמיד מכילות IP
   ה. יש יותר מתשובה נכונה אחת : _____

9. בין מי למי מקשרת שכבה 4?
   א. Hosts
   ב. TCP Stations
   ג. UDP Services
   ד. Client\server OR peer-2-peer
   ה. אף תשובה אינה נכונה

10. מה מהבאים נכון לגבי קוד הסוקטים (sockets) שלמדנו?
   א. השרת חייב ליצור את הסוקט לפני שהלקוח שולח, אך לא חייב להאזין בו
   ב. הלקוח חייב ליצור את הסוקט לפני שהשרת מאזין לו
   ג. פורט הלקוח חייב להיות ידוע לשרת בזמן ההאזנה
   ד. השרת חייב גם ליצור את הסוקט לפני שהלקוח שולח, וגם להאזין בו
   ה. יש יותר מתשובה נכונה אחת : _____

11. מה מתבצע בריבוב (mux) ודה-ריבוב (de-mux)?
א. ריבוב מאפשר העברה מקבוצה גדולה לקבוצה קטנה
ב. ריבוב מאפשר העברה מקבוצה קטנה לקבוצה גדולה
ג. דה-ריבוב מאפשר העברה מקבוצה גדולה לקבוצה קטנה
ד. דה-ריבוב מאפשר העברה מקבוצה קטנה לקבוצה גדולה
ה. יש יותר מתשובה נכונה אחת : _____

12. על איזו בעיה מתגבר פרוטוקול GBN אשר הפרוטוקולים RDT1.0, 2.1, 3.0 **אינם** מתגברים עליה?
א. בעיה של שיבוש חבילות
ב. בעיה של ניצולת ערוץ נמוכה
ג. בעיה של אבדן חבילות
ד. בעיה של שיבוש ACK בלבד
ה. בעיה של שיבוש ACK או NAK

13. מה נכון לגבי הנוסחא $(1-\alpha) * EstimatedRTT + \alpha * SampleRTT$ ב TCP?
א. כאשר $\alpha = 0.5$ ההודעה ה- n תהיה במשקל זהה להודעה ה- $(n-1)$
ב. מהווה חלק מחישוב ה safety margin עבור הטיימר
ג. נשתמש בה רק אם הטיימר פקע
ד. נשתמש בה רק אם הטיימר לא פקע
ה. יש יותר מתשובה נכונה אחת :_____

14. היכן ממומשת שכבה 3 באינטרנט?
א. בכל router
ב. בכל מכשיר המחובר לרשת האינטרנט
ג. בכל host ובכל router
ד. בכל host
ה. אף תשובה אינה נכונה

15. בשיטת IP classful addressing, פי כמה גדול מספר ה subnets ב class A בהשוואה למספר ה subnets ב class C?
א. פי 256
ב. פי 256^2
ג. פי 256^3
ד. אף תשובה אינה מדויקת

16. מה נכון לגבי טבלאות זרימה (flow tables)?
א. מאפשרות לחסום הודעות בפורטים מסוימים
ב. אינן מאפשרות להגדיר התנהגות של NAT
ג. המידע עליו הן פועלות מכיל גם מידע מרמה 2 בחבילה
ד. מאפשרות להגדיר התנהגות גלובאלית של מספר תחנות
ה. יש יותר מתשובה נכונה אחת : _____

17. מה נכון לגבי Network Address Translation?
א. מספר הקישורים בו הנתב יכול לתמוך שווה למספר הפורטים
ב. מספר הקישורים בו הנתב יכול לתמול שווה למספר הפורטים מוכפל במספר כתובות ה IP שלו
ג. מאפשרת שימוש בכתובות לוקאליות זהות ברשתות ביתיות רבות
ד. מאפשרת לתחנה לקבל כתובת IP באופן אוטומאטי
ה. יש יותר מתשובה נכונה אחת :_____

18. מה נכון לגבי שכבה 2?
א. בגלל שמדובר בשכבה נמוכה יחסית, יש בה מעט שיטות ופרוטוקולים, בהשוואה לשכבה 3
ב. הטכנולוגיות שבה מניחות קשר קווי ולא אלחוטי
ג. יש בה זיהוי שגיאות אך פחות מאשר בשכבה 4
ד. אחת הבעיות המרכזית שרוצים לפתור בה הוא כיצד לשלוח ביטים על קו פיזי
ה. יש יותר מתשובה נכונה אחת :_____

19. נניח שבשיטת CRC רוצים לשלוח D ביטים של מידע, בעזרת מקודד G כשהשארית של פעולות ה XOR היא R, מה נכון?
א. G נקבע ע"י R,D
ב. R נקבע ע"י D,G
ג. $Len(G) < Len(D)$
ד. $Len(R) > Len(G)$
ה. יש יותר מתשובה נכונה אחת : _____

20. מה נכון לגבי שיטה כמו polling לניהול תקשורת בערוץ משותף ?
א. זהו נסיון להתגבר על החסרונות של השיטות האחרות (כגון חלוקת הערוץ)
ב. התחנה יכולה לשלוח מעט מידע בזמן קצר בלבד
ג. זו שיטה עמידה באופן יחסי
ד. לא נדרשת תחנה מנהלת (master)

ענו על 5 מבין השאלות הבאות (30-21). ניתן לבחור 2 שאלות נוספות כבונוס אך חובה לסמן ליד כל אחת מהן "**בונוס**" (מימין לשאלה). תשובה מלאה לשאלת בונוס תוסיף נקודה לציון, ושגיאה בשאלת בונוס לא תפגע בניקוד. אם לא תסמנו, או תבחרו יותר מהמוגדר, יבדקו 5 שאלות באופן אקראי.

<u>סימון בונוס</u>:

21. מה נכון לגבי שירותם ואבני בנין באינטרנט?
   א. השירותים חשובים יותר מאבני בנין
   ב. הם חשובים באותה מידה
   ג. השירותים משפיעים על אבני הבנין אך לא להפך
   ד. העיוורים שבדקו את הפיל זיהו אבני בנין אך לא שירותים
   ה. יש יותר מתשובה נכונה אחת: _____

22. מה נכון לגבי לקוח ושרת בתהליך תקשורת?
   א. קיים רק בארכיטקטורת לקוח-שרת
   ב. קיים רק בארכיטקטורת peer-to-peer
   ג. קיים בשתי הארכיטקטורות
   ד. השרת הוא תוכנה, ואין קשר לארכיטקטורה או לתהליך תקשורת

23. מה נכון לגבי עיכוב בתורים בהנחה שמגיעות בממוצע n חבילות בשניה לבאפר בגודל B, והשונות בזמני ההגעה היא s (למשל כאשר s גדולה יתכן שיהיו שניות ללא חבילות)?
   א. כאשר s גדולה הבאפר עלול להתמלא גם אם $n < B$
   ב. כאשר s קטנה הבאפר עלול להתמלא גם אם $n < B$
   ג. אורך התור קשור ל n אך כמעט אינו מושפע מ-s
   ד. אורך התור קשור ל s אך כמעט אינו מושפע מ n
   ה. אף תשובה אינה נכונה

24. מה חדש ב HTTP גרסא 2?
   א. קודי המצב (status codes) עודכנו
   ב. תמיכה בפרוטוקולים חיצוניים כגון ZOOM
   ג. חלוקת אובייקטים ל frames כדי למנוע חסימות בתור
   ד. מבנה ההודעה עודכן
   ה. יש יותר מתשובה נכונה אחת: _____

25. מה מהבאים משותף ל $RDT2.0, RDT1.0$?
   א. אין דבר המשותף להם
   ב. בשניהם אין התיחסות לחבילות היכולות להשתבש מצד השולח
   ג. בשניהם אין התיחסות לחבילות שאובדות
   ד. אף תשובה אינה נכונה

סימון בונוס :

26. מה מהבאים נכון לגבי SR,GBN?
א. בשניהם קיים חלון בצד המקבל
ב. בשניהם קיים חלון בצד השולח
ג. בשניהם יתכן מצב שבו אינדקס החבילה הצפויה הבאה גדל בצד המקבל ביותר מ-1
ד. בשניהם יתכן מצב שבו האינדקס הבא למשלוח גדל ביותר מ-1 בצד השולח
ה. יש יותר מתשובה נכונה אחת :_____

27. איזה מהשכלולים הבאים קיים במקבל של TCP ?
א. אם קיבלנו חבילה כאשר יש ACK ממתין, נשלח מיד
ב. הכפלת האינטרבל אם חבילה לא מגיעה
ג. אם זיהינו שלושה ACKs רצופים או יותר, נשלח מיד את החבילה
ד. אם קיבלנו חבילה תקינה לא נשלח מיד ACK
ה. יש יותר מתשובה נכונה אחת :_____

28. מה מהבאים **לא** ניתן לקבל בפרוטוקול DHCP ?
א. כתובת IP חיצונית של ספק האינטרנט
ב. Subnet mask
ג. כתובת IP של שרת ה DNS אם אינו נמצא באותה רשת פיזית
ד. כתובת IP של הנתב
ה. יש יותר מתשובה נכונה אחת :_____

29. מה נכון לגבי IPv6 Tunneling ?
א. כאשר שולחים חבילה דרך מנהרה, כתובת היעד בחבילה העוטפת היא סוף המנהרה
ב. כאשר שולחים חבילה דרך מנהרה כתובת היעד בחבילה העוטפת היא היעד המקורי
ג. חבילה של IPv4 נכנסת לתוך חבילה של IPv6
ד. חבילה של IPv6 נכנסת לתוך חבילה של IPv4
ה. יש יותר מתשובה נכונה אחת :_____

30. מהו ההבדל המרכזי בין Slotted ALOHA ל Pure ALOHA ?
א. ב-Slotted אפשר להתחיל לשדר בכל רגע נתון וב-Pure רק בזמנים מוגדרים
ב. ב-Pure אפשר להתחיל לשדר בכל רגע נתון וב-Slotted רק בזמנים מוגדרים
ג. בSlotted מחכים ל token וב- Pure לא מחכים
ד. ב Pure מחכים ל token וב-Slotted לא מחכים
ה. יש יותר מתשובה נכונה אחת :_____

יש לענות על 2 מבין 4 השאלות הבאות (במשקל 25 נקודות לשאלה)

שאלה 2 (25 נקודות)

(4 נק') בחרו 3 שכבות במודל 5 השכבות. לכל שכבה כתבו את המטרה שלה, פרוטוקול אחד הנמצא בה ומה תפקידו. לא ניתן לבחור את הפרוטוקולים הבאים : (HTTP, TCP, IP)

שיכבה : _____

שיכבה : _____

שיכבה : _____

(6 נק') למדנו כי רשת האינטרנט היא רשת הטרוגנית. בחרו 3 מתוך הבאים: מערכות הפעלה שונות, פרוטוקולים שונים, קישורים בקצבים שונים, שפות תכנות שונות. לכל אחד מהדברים שבחרתם, כתבו אם הוא מגדיל את ההטרוגניות או לא. הוסיפו הסבר קצר

בחירה 1: _____

מגדיל את ההטרוגניות? : _____

הסבר

בחירה 2: _____

מגדיל את ההטרוגניות? : _____

הסבר

בחירה 3: _____

מגדיל את ההטרוגניות? : _____

הסבר

(5 נק') Bob רוצה להמציא מכשיר חדש בשם 4-Station. מכשיר זה יממש את כל השכבות עד שכבה 4, ללא שכבה 5 וטוען שיש בו מספר שימושים. Alice טוענת שאין טעם במכשיר כזה. מי מהם צודק? הסבירו

מי צודק : _____

הסבר

(6 נק') במהלך הקורס נתקלנו במושגים TDM (בפרק המבוא) ו TDMA (בפרק האחרון) ששניהם קשורים לחלוקת ערוץ תקשורת לחריצי זמן. הניחו כי TDMA מתייחס לשכבה 2 בלבד, וכי TDM מתייחס לשכבות גבוהות יותר בלבד. לגבי כל אחד מהמושגים הסבירו אם הוא רלוונטי לרשתות טלפוניה, לרשת האינטרנט, או לשניהם. נמקו בקצרה.

TDM

TDMA

סעיף אחרון בעמוד הבא

(4 נק') התייחסו לכל אחת משתי הטענות שלהלן לגבי במודל השכבות **באינטרנט**, לכל אחת כתבו אם היא נכונה או לא, והסבירו בקצרה:

(1) פרוטוקול הנמצא בשכבה N מסוגל להעביר הודעה לפרוטוקול בשכבה N+1
(2) אם שירות מסויים **אינו** קיים בשכבה N אז הוא (או שירות דומה) **אינו** קיים בשכבה מעליה

טענה 1

טענה 2

שאלה 3 (25 נקודות)

מ-15 תחנות קצה רוצים לשלוח בו זמנית n קבצים: כל אחת מהתחנות שולחת קובץ (כך שנשלחים 15 קבצים במקביל), מסיימת ומתחילה לשלוח קובץ חדש, וכך הלאה. הקבצים נשלחים דרך נתב יחיד, המקושר לכל התחנות בקישור ישיר, ולו יציאה בודדת לרשת האינטרנט. גודל כל אחד מהקבצים הוא 2Mb. נניח כי כל תחנה יכולה לשדר בקצב של עד 1Mbps, וכי זמן ההגעה של חבילה ששודרה לנתב, ועיכוב-העיבוד שלה הם זניחים. קצב השידור של הנתב הוא 16Mbps. גודל הבאפר לאיחסון החבילות הממתינות לשידור בנתב **אינו** ידוע. הניחו כי העיכובים **שאינם** מתוארים בתרחיש הם זניחים.

(5 נק') מהו ערכו של עומס התעבורה (traffic intensity) בתרחיש זה? האם נגיע למצב של אובדן חבילות בסבירות גבוהה בנתב (הניחו כי זמן העיבוד זניח)? נמקו.

(6 נק') לאחר השניה הראשונה, בשל תקלה בנתב, קצב השידור שלו ירד ל 8Mbps למשך t שניות. חשבו את מקדם העומס החדש, וכתבו מהו גודל הבאפר **המינימאלי** הנדרש לכך שב-t השניות הללו לא נאבד חבילות (השתמשו בt בנוסחא).

(4 נק') נניח כי גודל הבאפר הוא $\frac{1}{2}$ מהגודל שמצאתם בסעיף הקודם וקצב השידור הוא 8Mbps כאשר נתון ש $t = 3$. הסבירו בקצרה מה יתרחש , ללא חישוב.

(5 נק') לאחר $t = 3$ שניות הנתב חוזר לקצב השידור המקורי, והתחנות ממשיכות לשדר. חשבו כזה זמן ידרש לכך שהבאפר יתרוקן.

סעיף נוסף בעמוד הבא

(5 נק') בתרחיש אחר, שולחים 50 חבילות בזו אחר זו דרך 4 נתבים המחוברים בטור כך :
$A - R1 - R2 - R3 - R4 - B$. לכל חבילה נדרשת שניה לעבור כל מקטע כזה. חשבו כמה זמן ידרש עד שהחבילה האחרונה תגיע ל B.

שאלה 4 (25 נקודות + בונוס עד 2 נקודות)

(8 נק') שתי בנות: Alice, Clara ושני בנים: Bob, Dimitri כותבים אפליקציית רשת, שחלקה רץ מעל UDP וחלקה מעל TCP. לשם כך, האפליקציה מקבלת פורט נפרד ל UDP ופורט נפרד ל TCP בכל תחנה. הארבעה סיכמו שAlice תריץ במחשב הביתי שלה שני שרתים, כאשר שרת ה UDP ירוץ מעל פורט 2222 ושרת ה TCP ירוץ מעל פורט 3333 (נניח בשאלה כי הפורטים פנויים).

כתובות ה IP של הארבעה הן:

Alice – 210.220.230.240 Bob- 30.40.50.60
Clara – 110.120.130.140 Dimitri - 40.50.60.70

מספרי הפורטים הזמינים הם:
עבור Bob הם 3003, 3090 (3090 עבור TCP)
עבור Clara הם 2090, 1090 (שניהם עבור TCP)
עבור Dimitri הם 4004, 4090 (3090 עבור TCP)

נניח כי כל אחד מהבנים יוצר קישור UDP וקישור TCP עם המחשב של Alice, ואילו Clara יוצרת שני קישורי TCP עם Alice.

כתבו כמה סוקטים יווצרו, ואז את פרטי כל הסוקטים שיווצרו – אצל כל אחד מ 4 החברים. אם קיים סוקט שהוא סימטרי לסוקט אחר ניתן לכתוב זאת מבלי לכתוב את כל הפרטים של הסוקט (אך כיתבו בדיוק מהו הסוקט הסימטרי)

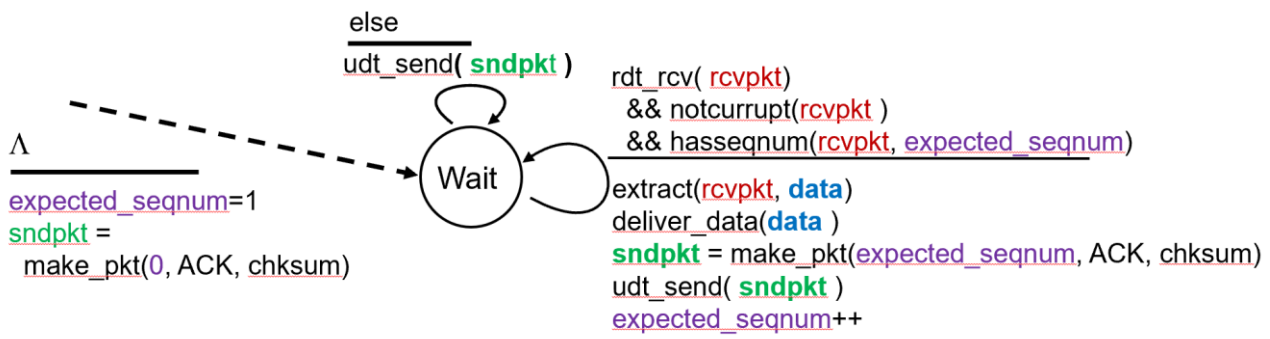
מספר הסוקטים שיווצרו בסך הכל: _____

פרטי הסוקטים:

ניתן להמשיך בעמוד הבא

(4 נק') ללא קשר לסעיף הקודם, תארו תרחיש אפשרי שהשולח ב Rdt2.1 יקבל NAK במצב השמאלי-תחתון שבאוטומאט. **אין** לחרוג מהשורות המוקצות לתשובה!

(6 נק') נתון התרשים הבא של אוטומאט צד המקבל ב GBN:



(1) מה יעשה המקבל כשתגיע חבילה משובשת? מדוע הוא עושה זאת?

(2) מדוע מופיע expected_seqnum++ בתחתית התרשים, וההגדלה אינה לפי המספר הסידורי המופיע בחבילה?

(ענו בעמוד הבא)

(1) הפעולות בחבילה משובשת ומדוע:

(2) expected_seqnum++ בתחתית התרשים

(3 נק') מהם המספרים הסידוריים האפשריים ב TCP? אם נתחיל קישור חדש, איזה מספר נקבל? הסבירו.

(4 נק') האם יתכן שנשלח 3 הודעות TCP עם אותו מספר seq, שאינן נשלחות במקביל? אם כן – כיצד זה אפשרי? אם לא – מדוע?.

שאלה 5 (25 נקודות + 2 נק' בונוס)

שאלה זו עוסקת בנושאים הקשורים בשכבת הרשת ובשכבת הקו.

בשאלה זו, **ענו על 5** מתוך הסעיפים הבאים. לא ניתן לענות על סעיפים נוספים כ"בונוס" !

א. (5 נק') נתונה טבלת הקידום הבאה:

Address Range:	Link:
11001000 00010111 ******** *********	1
11001000 00010111 11110000 *********	2
11001000 11101110 00011*** *********	3
11001000 11101110 0001111* *********	4
Else	5

כתבו, **ללא נימוק**, לאיזה מה-Links תישלח כל אחת מהחבילות שלהלן:

חבילה הממוענת לכתובת 11001000 00010111 11110000 00001101:

תישלח לקישור מספר : _____

חבילה הממוענת לכתובת 11111111 01010101 00011111 00000001:

תישלח לקישור מספר : _____

חבילה הממוענת לכתובת 10000000 00010111 00011000 00000001:

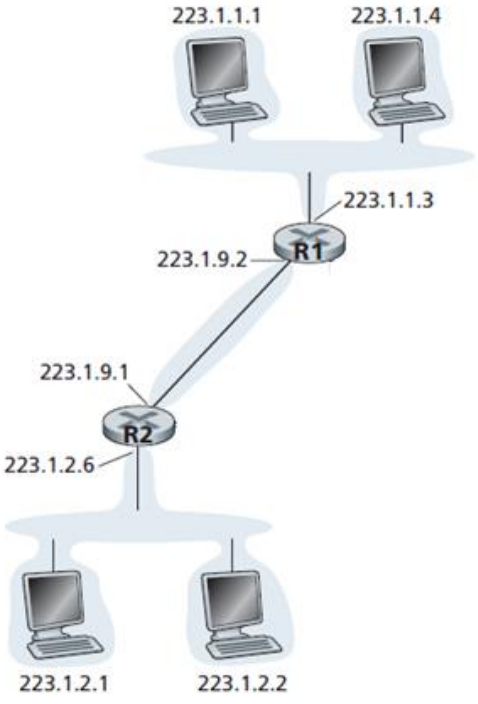
תישלח לקישור מספר : _____

חבילה הממוענת לכתובת 11001000 11101110 00011100 11111110:

תישלח לקישור מספר : _____

ב. (5 נק') חבילת IP מגודל 5000 בתים צריכה לעבור בערוץ שהגודל המקסימלי שאפשר לשדר בו הוא 2000 בתים. כתבו כיצד תתבצע הפרגמנטציה: לכמה חבילות, מה כמות הנתונים וה-offset בכל חבילה?

ג. (5 נק') כמה תתי רשתות (subnets) קיימות בתרשים זה? כמה ביטים מוקצים ל subnet בכל אחת מהרשתות?



מספר תתי הרשתות: _____

מספר הביטים המוקצים ל subnet : _____

ד. (5 נק') מהו ההבדל בין קידום לבין ניתוב? אם כשסטודנט מגיע ליום הקוד המחלקתי, ובכניסה למעבדה מכוונים אותו לאיזור המתאים לשנה ב' – האם זו פעולת קידום או ניתוב? הסבירו.

ה. (5 נק') כתבו יתרון וחיסרון של CRC בהשוואה ל parity-bit דו ממדי. מדוע משתמשים ב XOR ב CRC?

ו. (5 נק') תנו 2 דוגמאות לחריגות ממודל השכבות הקשורות לשכבה 2

בהצלחה !

2 עמודים לגיבוי – בעמודים הבאים

עמוד תשובות (גיבוי) ראשון – תשובה לשאלה _____

עמוד תשובות (גיבוי) שני – תשובה לשאלה _____

בהצלחה !

2 גישות לניהול האינטרנט: גישת אבני הבניין(פרוטוקולים, תחנות קצה, ראוטר, לינק, חבילה, ISP..) -ממה בנוי האינטרנט. מסתכלת על המון מכשירים שמחוברים אחד לשני. שולחים מידע, מחלקים את המידע לביטים ולמספר חבילות נפרדות, ושולחים את החבילות על קישור מסוים(לינק) והקצב העברה נמדד בביטים בשנייה. **פאקטה** – חבילה של ביטים שמורכבת מהמידע ומheader. **ראוטר(נתב)** - מקבל את החבילה, לפי ההדר יודע לאן להפנות את החבילה ומבצע את ההפנייה. מוגדר בו אלגוריתם ניתוב שמייצר טבלת ניתוב שהיא מגדירה מסלול לכל אחד מהיעדים. **Hop** – פעולת המעבר בין ראוטר לראוטר. פעולת הניתוב נקראת גם forwarding. **ISP** – ספק השירות, מבטיח שנתחבר לאינטרנט, יש כל מיני ספקים כמו בייתים, למקומות עבודה וכו. בנוי בצורה שיש ספק ברמה מסחרית כמו at&t, יש פרטיות כמו google שנותנות אינטרנט לספק אזורי שנותן אינטרנט למישהו ברמה נמוכה יותר כמו נגיד פרטנר שמביאה לנו אינטרנט. **פרוטוקולים** – שפה משותפת, שולטים בזרימה של המידע. כמו למשל TCP או IP. מגדירים פורמט הודעות, סדר ההודעות ומה לעשות בקבלה ושליחה. לא מגדירים התנהגויות חריגות. כללי תקשורת. מגדירים אותם ב **RFC** = סטנדרטים באינטרנט, מגדירים חוקים שיתנו אחידות בסטנדרטים, כולם צריכים לממש. **גישת השירותים** – מה האינטרנט עושה, לא מסתכל על האלמנטים עצמם. אפליקציות מבוזרות, עם הרבה תחנות קצה. לא מדברים פה על ראוטרים. ראוטר זה אבן בניין זה לא שירות. לדוגמה API שמשתמשים בו לבצע פעולות מסוימות, הוא שירות.

קצה הרשת : **תחנות קצה** – מארחים, כל מי שמתחבר לאינטרנט, לנקודת גישה רשת מקומית כלשהי, 2 תחנות שמדברות אחת עם השנייה, מריצות אפליקציה כלשהי, אפשר במכשירים שונים, מדברים בשפה סטנדרטית ומוגדרת. **מודל לקוח ושרת** – בתקשורת יש 2 צדדים : לקוח ושרת. שניהם תוכנות. לקוח רץ בתחנת קצה ומבקש שירותים מהשרת. השרת גם רץ בתחנת קצה ומספק שירותים מסוימים. הם מתקשרים בהודעות. לקוח תמיד יהיה מי שיוזם את ההתקשרות. תמיד תשובה מהשרת. (כמו אתר אינטרנט).שרת תמיד עובד. **מודל P2P** – peer to peer, כמה תחנות קצה, אין שרת אחד שמטפל בהכל, כולם יכולים לפנות לכולם, ניתן להרחבה- אם יש הרבה לקוחות על שרת אולי צריך שרת נוסף. כולם שווים אחד להשני ,הלקוחות נותנים שירותים אחד לשני, מפזר עומסים, פחות עומס על שרת אחד. יותר מורכב עובד בצורה מקבילית. קשה לסנכרון, אבטחה קשה יותר להבטיח. (כמו הורדה מטורנט).קשה לניהול. הוא יעיל מבחינת עלויות.ספקי אינטרנט עובדים גם ב P2P וגם בשרת לקוח.לא ידידותי לספקי אינטרנט, אין תמריץ למשתמשים, לרוב לא קוד פתוח. **סקייפ משלב את שני המודלים-** יש שרת המאפשר למצוא כתובות של לקוחות אבל יש חיבור ישיר בין לקוח ללקוח. **Access network** – הרשת הלוקאלית, כל הרשת עד הראוטר שיוצא לאינטרנט(ל ISP). קו אסימטרי **ASDL** כי ההעלאה יותר איטית מההורדה. בנוי אסימטרית כי רוב האינטראקציה היא הורדת מידע ולא העלאה. **מדיה פיזית** –יש קישורים GUIDED ו UNGUIDED – כלומר פיזי ואלחוטי. פיזי יותר מהיר. **ליבת הרשת** – הראוטרים שמחברים את הכל וכולם, רשת של רשתות. מה שמחוץ לרשתות המקומיות.

2 גישות למימוש העברת המידע בליבת הרשת בסוויצים ובלינקים: מיתוג מעגלי – תקשורת בין שני צדדים. חלק מהמשאבים שייכים באופן מוחלט למי שמנהל תקשורת לאורך כל השיחה (להזמין מקום במסעדה). בעיקר לטלפוניה, לא לאינטרנט. שלב 1 : יצירת קישור בין השולח והמקבל. שלב 2 : שימור ותחזוק הקישור, תקשורת דו כיוונית קצב שידור קבוע. שלב 3 : סיום החיבור. אין שיתוף משאבים אי אפשר להכנס לקו תקשורת בין שני צדדים, כל המתגים מודעים לשיחה ומקצים לה משאבים. **בתוך מיתוג מעגלי יש 2 גישות לחלוקת הקו: חלוקה לפי תדרים** – מקצים תדר מסוים לכל קו וככה הם לא מתנגשים, כמו רדיו. **חלוקה לפי זמן** – כל קו מקבל הקצאת זמן מסוימת בה הוא מעביר ,מחלקים את הזמן למסגרות וכל מסגרת מחלקים לחריצים. בכל חריץ יכנס משתמש מסוים. ואז זה חוזר בצורה מעגלית. **חישובים:** כמה זמן לשלוח 640kb בשנייה, כשיש 24 חריצים בתוך מסגרת, וקצב ההעברה הוא 1.536mbps, ולוקח 500ms ליצור את המעגל: קצב העברה(רוחב פס) / כמות החריצים = **כמה משתמש יכול להעביר במסגרת**(מסגרת זו שנייה). (64kbps = 24 / 1.536). כמות מידע להעברה(גודל קובץ) / כמה משתמש יכול להעביר בשנייה = **זמן העברה של הקובץ**.(640 / 64 = 10 sec). נשאר לחבר את הזמן ליצור מעגל : 0.5 + 10 = 10.5 sec. מספר הלינקים לא משנה. **חסרונות** : יכול להיות לא פעיל(בהמתנה) הרבה פעמים, בזבזני. מסובך למימוש, קשה לתחזוק. **מיתוג חבילות** – לא מקצים משאבים, הם מוגדרים פר הודעה וכל פעם ששולחים אותה, לעיתים מחכים בתור (להגיע למסעדה בלי להזמין ולהיכנס על בסיס מקום פנוי). בעיקר באינטרנט, שולחים חבילות מנקודה לנקודה יכולה לנוע בכל מסלול אין "מסלול אחד" כמו במעגלי, אין הבטחה על שימור מידע. שלבים : שלב 1 : מקבלים מידע שרוצים לשלוח. שלב 2 : מחלקים אותו לחבילות. שלב 3 : מעבירים את החבילות האלו והן דורשות את כל הרחוב פס לזמן קצר. משתמשים בעקרון של **store and forward** – כשמקבלים ביטים של חבילה ראשונה מחכים לקבל את כל שאר המידע ורק אז מעבירים הכל. העקרון הזה יוצר עיכובים בכל לינק בדרך. **חישוב קצב העברה של חבילה בלינק בשניות:** גודל חבילה להעברה / קצב שידור בלינק. חבילות נשלחות במקביל בלינקים.***תמיד לתרגם לביטים. זה בסדר נניח קילו ביט אבל שלא יהיה בבייט*.**

עיכובים ואיבודים: ישנם 4 סוגי עיכובים : 1. **עיכוב בתורים** – אם ראוטר מקבל יותר ממה שהוא יכול לשדר יהיה תור שליחה שיגרום לעיכובים. אם יתמלא יותר ממה שהוא יכול להכיל יהיו גם איבודי חבילות, תלוי בעומסים לכן קשה לנבא, יחסית מסובך, יכול להיות זניח או גדול תלוי בתור. 2. **עיכוב בעיבוד** – כשמתקבלת חבילה צריך להקצות לה משאבים ולעבד אותה שאין שגיאות וכאלה(יחסית מהיר). 3. **עיכוב בשידור** – החישוב קצב העברה של קובץ, החישוב למעלה של גודל חבילה / קצב שידור בלינק. מה שמשפיע עליו זה חבילה גדולה או קצב שידור איטי. 4. **עיכוב בפעפוע** – התזוזה של החבילה על הקו מראוטר אחד לשני. מה שמשפיע עליו זה מרחק בראוטרים. זמן עיכוב מקצה לקצה זה סכימה של כל העיכובים ביחד. **פעפוע מול שידור** : שידור – בכמה חבילות הראוטר יכול לטפל ביחידת זמן. פעפוע – הזמן שחבילה מטיילת על לינק מסוים עד שמגיעה לראוטר הבא. **הסברים יותר מעמיקים על עיכוב בתורים** : תלוי בקצב הגעה. קצב הגעה מהיר קצב שידור איטי יהיה עיכובים גדולים יותר. יכולים להיות פיקים. רגע אחד שהתור ריק כי קצב הגעה איטי, רגע אחר שהתור כמעט מלא כי קצב הגעה מהיר. **חישוב מקדם עומס** : קצב הגעה ממוצע של חבילה(או מספר בקשות בשנייה) * גודל חבילה ממוצע בביטים(L) / רוחב פס(קצב שידור)(R) בביטים. יהיה תור אם מקדם עומס גדול מ1. יקרה אם קצב שידור קטן מקצב הגעה. **חישוב זמן המתנה לחבילה ה n:** (n – 1) * L / R. הצמיחה אקספוננציאלית : כשיש קצת המתנה עלייה קטנה תוסיף קצת המתנה. כשיש הרבה המתנה עלייה קטנה תוסיף הרבה המתנה. **אובדן חבילות** – כשהתור מתמלא מעבר לקיבולת שלו חבילות חדשות שנכנסות ילכו לאיבוד. מקדם עומס גדול יותר אומר יותר איבוד חבילות. **הספק(קצב הורדה לדוגמה, כמה ביטים יורדים בשנייה)** –הקצב שידור של הלינק המינימלי(בעיות זרימה וקיבולת). גודל קובץ בביטים / שניות להעברה של הלינק המינימלי.

ארכיטקטורת השכבות באינטרנט: כל שכבה מודעת רק לשכבה מעלייה ומתחתיה. פועלת ולא אכפת לה מהשכבות האחרות, חוסר תלות. שלב i מצד אחד צריך להיות מתואם עם שלב i בצד השני. לדוגמה השולח והמקבל צריכים לדבר אותה שפה .**שכבת האפליקציה (5)**– אפליקציות הרשת - עבודה מול המשתמש. תהליכי התקשורת, לקוח\שרת .מטרתה לאפשר למשתמש לעבוד באינטרנט וגם לספק שירותים שונים. **שכבת התעבורה(4)** –תקשורת בין תהליכים בתחנות מרוחקות. ריבוב ודה-ריבוב. העברת נתונים אמינה (TCP) או פשוטה (UDP). **שכבת הרשת(3)** – מטרתה לנווט באינטרנט, לספק כתובות ולדווח על תקלות. **שכבת הקו(2):** תקשורת לוקאלית ב LAN.מטרתה לבצע קישור פיזי בין מכשירים, העברת הודעות בתווך משותף, ולבדוק תקלות בהודעות **השכבה הפיזית(1):** העברת ביטים בערוץ תקשורת פיזי שכבות 3+2+1 ממומשות בחומרה(לדוגמה בראוטר). השאר ממומשות בתוכנה + שכבה 3 שגם בתוכנה. **העברת הודעה משכבה לשכבה** : הודעה של שכבה 5 על ההדר שלה מגיעה לשכבה 4 והיא מלבישה הדר של שכבה 4 ומקדמת אותה ל3 וכך הלאה. הודעה שמגיעה ליעד מגיעה לשכבה 1 וכדי להעביר אותה לשכבה 5 בכל שכבה מקלפים את ההדר של אותה השכבה, עד שההודעה מגיעה לשכבה 5 ביעד בלי כל ההדרים שנוספו. פעם *היו עוד 2 שכבות, שכבת ייצוג* שאחראית לדחיסה והמרות ושכבת הסשן שאחראית לסנכרונים. לא קיימים באינטרנט יותר, פיזרו את התפקידים בין שכבות אחרות. **חסרונות של מודל השכבות** – פונקציונאליות כפולות, פונקציה שמופיעה בכמה שכבות כי כל שכבה בלתי תלויה באחרת. הפרה של עיקרון ההפרדה – יש מקומות שההפרדה לא עובדת, שצריך לשתף מידע. (לדוגמא קביעתMMS)

אבטחה ורשתות: איך יתקפו, איך אפשר להגן, ואיך מונעים מראש תקיפות. **וירוס** – קובץ שצריך להפעיל אותו והוא משכפל את עצמו. **תולעת** – משכפלת את עצמה בצורה פאסיבית, לא צריך להפעיל. **רוגלה** –מתעדת מה הקלדנו איפה ביקרנו וכו. **Dos** – דחייה של שירות, לדוגמה ששרת לא יוכל לתפקד, מבוצע עם מלא בקשות ומלא בקשת משאבים עם תעבורה מזויפת מסיבית. **רחרוח** – לקחת חבילות ולראות מה הן מכילות. **IP spoofing** – לשלוח חבילה עם כתובת מקור שקרית. **Botnet** – להשתלט על מלא תחנות ברשת בוטים שרצים על התחנות האלה וכולם ביחד יפציצו רשת בהודעה ביחד. **המתווך** – מידע מועבר דרך ההאקר. **כופר** – פשוט כופר. **Rootkit/bootkit** – משהו שנדבק ממש לביוס.

שכבת האפליקציה – שכבה 5: אפליקציות רשת יושבות בתחנות קצה, מדברות ביניהן בעזרת האינטרנט. אפשר כאן לממש ב2 הגישות של שרת לקוח או P2P. **שרת** : פעיל תמיד, כתובת IP קבועה, קשה להוסיף שרתים נוספים וזה יקר. **לקוח** : מתקשר עם השרת, כתובות IP דינאמיות, לא מתקשרים ישירות אחד עם השני. **סוקט** – מעיין דלת, לכל תהליך יש סוקט שדרכו הוא מתקשר. מעבירים חבילה לסוקט מצד אחד ואומרים לאיזה סוקט הוא צריך לשלוח מצד שני. האם אפליקציות שולטות על התעבורה? – שליטה מאוד מוגבלת, כמו למשל לבחור איך להעביר אם UDP או TCP. איך לדעת כתובת של תהליך : כתובת IP עם 32 ביטים שמחולקים ל 4 מספרים. כדי לשלוח משהו בין תהליכים צריך את הכתובת IP וגם את מספר הפורט. IP מייצג את המחשב בכללי, מספר הפורט מייצג את התהליך הספציפי אליו רוצים לשלוח, הדלת הספציפית. לדוגמה שליחת הודעת HTTP צריך כתובת כמו למשל 128.119.245.12 ופורט למשל 80(פורט 80 ל HTTP). **שירותים להעברה (TCP / UDP): מימדי השירות:** אמינות – שכל מה ששלחנו יגיע. הספק – כמה ביטים אפשר להעביר ביחידת זמן. תזמון – מה שדורש מהירות כמו משחקים לדוגמה, כל חבילה מגיעה בזמן מסוים. אבטחה – הצפנות. **תפוקה ותזמון לא מובטחים ברשת האינטרנט אך ניתן לקבל אותם ע"י שליחת מידע דחוס או איכות שידור נמוכה כאשר יש עומס על הקו. SSL** – הצפנה על TCP בשכבה מעל TCP. TLS זה גרסה חדשה שלו. שני הצדדים צריכים לדעת לתרגם את ההצפנה. אסור להצפין את היעד, איך נדע לאן לשלוח. יושב בין HTTP ל TCP, מולבש לפני TCP. **פרוטוקולים בשכבת האפליקציה:** מגדיר את סוגי ההודעות שמעבירים, מה הם חלקי ההודעה והמבנה של השדות, מה המשמעות של הערכים, מה לעשות איתה וכו. פרוטוקול פרטי – יש לדוגמה בסקייפ. פרוטוקול ציבורי מוגדר ב RFC וזמין לכולם, כמו HTTP. אפליקציות ופרוטוקולים הם לא אותו הדבר, אפליקציות מממשות ומשתמשות בפרוטוקול לדוגמה אפליקציית ווב משתמשת בפרוטוקול HTTP. **נתרכז ב HTTP ואפליקציות ווב כדי להמחיש את שכבת האפליקציה: HTTP** זה פרוטוקול להעברת היפר-טקסט(שזה טקס ואובייקטים כמו תמונות), מסמך HTML עם קישורים לאובייקטים.**URL** – כתובת אתר מכילה 2 חלקים : הכתובת עצמה בעולם(כתובת של שרת), ומיקום של אובייקט/ קובץ בתוך אותה הכתובת(לדוגמה /home/pic.jpg). יכול להכיל גם שם משתמש ומספר פורט. **URI** אותו הדבר יותר כללי. איך עובד : משתמש מבקש דף, דפדפן שולח בקשת HTTP לשרת, השרת מקבל את הבקשה ומביא את המידע, השרת שולח תשובת HTTP, הדפדפן מקבל את התשובה ומציג את הדף. HTTP משתמש בשירותי TCP. יוצרים שיחה, יוצרים התקשרות מסוג TCP, חיבור שרת לקוח. הלקוח זה הדפדפן, השרת שולח את התשובות לפי הבקשות, מתקשרים דרך סוקט TCP. פרוטוקול TCP עובד בצורה שקופה בלי קשר ל HTTP בזכות מודל השכבות. HTTP חסר מצב, אין לו אוטומט, אין לו היסטוריה, לא זוכר אחורה מצבים. התשובה תהיה זהה לאותה הבקשה, יכול לנהל כמה לקוחות בו זמנית. מי שמקים את ההתקשרות וזוכר ויודע מה להגיב ושומר מעין מצב זה TCP, לא HTTP. **חיבור עקבי/ רציף** – משאירים את הקישור פעיל. TCP לא נסגר, נוצרים שני סוקטים בלקוח ושרת כמו בלא רציף, לקוח שולח בקשה השרת מחזיר תשובה, והחיבור לא נסגר בסוף התשובה. החיבור נסגר כשנגמרת התקשורת (timeout). יתרונות שאין חיבורים חדשים לכל אובייקט, פחות כבד, שולח את כל הקבצים במקביל על אותו החיבור, מחכים לתשובה רק אחרי ששולחים הכל, דיפולטיבי עבור HTTP. **חיבור לא עקבי/ לא רציף** – יוצרים חיבור חדש עבור כל בקשה. לקוח מתחיל חיבור ונוצרים 2 סוקטים אחד ללקוח אחד לשרת. לקוח שולח בקשה, השרת מחזיר את התשובה. לאחר הקבלה מסיימים את החיבור וסוגרים את הסוקטים בשני הצדדים עד הבקשה הבאה. round trip time – RTT, הזמן של שליחת בקשה וקבלת תשובה, מפתיחת קישור עד סגירה. לכל אובייקט צריך 2 RTT, אחד לבקשה השני לתשובה. יוצר עומס בשרת כשיש הרבה בקשות, TCP חדש לכל בקשה. יכול להיות גם מקבילי. סוקט שונה לכל בקשה .סך הכל זמן תגובה לבקשת HTTP בחיבור לא רציף = 2RTT + זמן שידור של הקובץ(זמן להוריד את הקובץ = גודל הקובץ / רוחב פס). **פורמט כללי לבקשת HTTP** : בנוי מ header וגוף ההודעה עצמה. ההדר מכיל שדות כמו ה URL כתובת להגיע אלייה, גרסת הפרוטוקול, מתודה, connection שאם השדה הוא close = לא רציף, keep alive = רציף. מתודות לבקשות HTTP – post : יש גוף לשיטה זו, שולחים מידע עם הבקשה בגוף ההודעה, לדוגמה פרטים שמזינים בטופס מסוים. Get : אין גוף, מה שמבקשים יופיע בשורת URL, בקשה לאובייקט. Head : דומה לget בלי גוף, משתמשים בעיקר לדיבאג, אין צורך בתגובה. Put : העלאה של אובייקט לשרת(כמו הרשמה של משתמש). Delete : מחיקה של אובייקט מהשרת. **פורמט כללי לתשובת HTTP** : מחזיר סטטוס תשובה, אורך הודעה, זמן תגובה, זמן אחרון לעדכון האובייקט וכו. Status code – משפחות

של סטטוסים(100 – 199 : תגובת מידע, 200 – 299 : תגובת הצלחה, 300 – 399 : הפניות, 400 – 499 : שגיאת לקוח, 500 – 599 : שגיאת שרת). Entity body יהיה 0 או 1, קיים גוף או לא. **עוגיות** – HTTP חסר מצב אבל לא באמת ב 100 אחוז, עוגיות שומרות מידע על המשתמש. 4 רכיבים של עוגייה : קוקי הדר בבקשה(כדי להגיד מי אנחנו), קוקי הדר בתשובה, קובץ קוקי במחשב של המשתמש לוקאלית והדפדפן מנהל אותו, וקובץ קוקי בבסיס נתונים בצד השרת באתר. שולחים בקשה עם מזהה לקוח(עוגייה), השרת ניגש למזהה בבסיס נתונים ומחזיר תשובה על סמך זה. איך שומרים על המצב : הלקוח יוצר קובץ עוגייה, השרת מייצר שורה על העוגייה הזאת בבסיס נתונים שלו. עוגיות מאוד נוחות אבל מסוכנות לפרטיות ופרצות אבטחה. **מטמון רשת/ שרת פרוקסי** – **פרוקסי** : מתווך, **קאש**(מטמון) : משהו מהיר אבל מוגבל בגודל, עוזר לחסוך בזמן. במקום לפנות לשרת אפשר לפנות לשרת פרוקסי, זה מחשב עם זיכרון לוקאלי ששומר מידע שביקשו לאחרונה. פחות זמן כי לא פונים לרשת עצמה אלא למתווך. פחות עומס ומהיר יותר, משהו שהספק אינטרנט מספק וצריך להגדיר גם בדפדפן. לדוגמה לקוח מבקש אובייקט, דפדפן יוצר התקשרות עם שרת פרוקסי עם TCP, השרת פרוקסי בודק אם הקובץ קיים, אם כן שולח ללקוח, אם לא מוציא בקשת HTTP לשרת המקורי, ושומר אותו לוקאלית בפרוקסי לפעם הבאה. ישנו פרוקסי גם בלקוח וגם בשרת. חסרונות : אם שרת המקור מתעדכן והפרוקסי לא נוכל לקבל משהו ישן, בעיות אבטחה, קל יותר לתקוף את הפרוקסי. פרוקסי יותר יעיל וזול מלהגדיל רוחב פס. Get תחת תנאי – תביא משהו מהפרוקסי אם הוא לא השתנה לפי תאריך מסוים, אם השתנה תביא מהשרת המקורי.

עדיין בשכבת האפליקציה, עוברים ל **DNS** : דרך למיפוי של כתובות URL לכתובות IP, מתרגם מכתובות אתר ל IP ולהפך. משתמש בבסיס נתונים שמחזיק את התרגומים. פרוטוקול בשכבת האפליקציה. לכל מכונה יש לקוח DNS, הדפדפן מחלץ כתובת URL ומעביר אותה ללקוח DNS. הלקוח שולח בקשה לשרת DNS, והוא מחזיר תשובה עם כתובת IP בחזרה ללקוח DNS. הלקוח מחזיר את התשובה לדפדפן והדפדפן מייצר חיבור TCP בעזרת ה IP. נותן אפשרות גם לתת כינויים עבור מארח, כמה כתובות לאותו האתר(למשל www.facebook.com, וגם facebook.com), כינויים לכתובות של שרתי מייל, חלוקת עומסים בין שרתים אם יש רשימה של כמה כתובות לאותו מארח, אותה כתובת URL תתורגם לכתובת IP מתוך הרשימה לפי סבב מסוים ככה שיפזר עומס על כל הכתובות בצורה שווה(לדוגמה כמה מנסים לגשת לנטפליקס כל אחד מקבל כתובת אחרת בצורה מעגלית מתוך רשימה). אם היה שרת DNS אחד אם היה כישלון הכל היה נופל, יותר מידיי תעבורה, זמני בקשות ארוכים ותחזוקה קשה. זה מוגדר בצורה היררכית של עץ, יש את השורש, יש צומת של com. צומת של org. וכך הלאה, עד שמגיעים לשרת DNS ספציפי לבקשה. לדוגמה מבקש כתובת של אמזון, הולך לשורש, אחרי זה לצומת של com. אחרי זה לשרת DNS של אמזון ואז מקבל כתובת IP. שרת ברמת שורש – יש 13 עם כתובות קבועות, צריך לדעת למי לפנות. שרת ברמה גבוהה(צומת) – אלו שמנתבים בצורה כללית כמו com. לדוגמה. שרת בר סמכא(עלים) – קשורים לחברה מסוימת, ממפה בתוך החברה עצמה, נותן את התשובה, חייב להיות זמין בצורה ציבורית. **שרת DNS** – הוא לוקאלי לא נמצא בהיררכיה, יש אחד בכל ספק אינטרנט, **לקוח DNS** פונה לשרת הלוקאלי והוא מבצע פנייה חיצונית. אפשר לבצע בקשה איטרטיבית ורקורסיבית – הרקורסיבית פחות טובה בגלל איבוד מידע ועומסים. לפעמים שרת ברמה נמוכה לא מוכר וצריך מתווך אליו. גם כאן יש קאשינג. **DNS resource records** מאחסן את המיפויים והתרגומים מכתובת IP ל URL. מאוחסן בטבלאות ונשלח כהודעות. מכיל שדות של : שם – כתובת URL, ערך – כתובת IP, טיפוס(A – IP ושם, AAAA - IPV6, NS – שרת DNS אליו פונים, MX – מייל, Cname – כינויים, ועוד), TTL – זמן למחיקה. NS – הערך שלו יהיה שרת DNS שאליו צריך לפנות כדי לקבל את התשובה. CNAME עובד בצורה דומה, בשניהם חייב לשמור גם טיפוס A כדי שיחזיר בסוף את הכתובת. פורמט הודעת DNS : מזהה בקשה, דגל שאומר אם זו בקשה או תשובה, כמה תשובות עונים אם יש יותר מאחת, שדה של השאלה, שדה של התשובה(יהיה ריק כשמדובר בבקשה מין הסתם) אם נניח ניצור חברה חדשה : אפשר לתקוף DNS עם תקיפות DDOS שיוצרות עומס לשרתים, הרעלת DNS, המתווך כדי לתקוף בקשות וכולי.

תכנות בסוקטים: חשוב בממשק בין שכבה 5 ל4. השרת חייב לפעול בהתחלה, מבחינת האפליקציה מקבלים שירות לא אמין של דאטהגרם ב UDP.**UDP** : יוצרים סוקט בשרת, רק אז הלקוח יכול לשלוח הודעה, הלקוח שולח הודעה השרת קורא ושולח תשובה לסוקט של הלקוח. אין לחיצת יד. בסוקט UDP הלקוח צריך בכל חבילה להגדיר כתובת ופורט כי אין לחיצת יד, מעבירים מידע מבלי להגדיר התקשרות מראש, פשוט שולחים. לא מבטיח אמינות, אין תפעול עומסים, לא מבטיח הספקים ותזמונים. טוב כי הוא דורש פחות. טוב לאפליקציות בזמן אמת כמו סטרימינג או משחקים. אפשר להרשות את ההפסד.

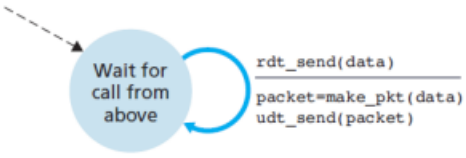
הגדרות אצל הלקוח : מגדירים את הכתובת IP ומספר פורט שניהם של השרת, יוצרים סוקט, לא מגדירים ביצירה כתובת IP או PORT, כי אפשר לשלוח למי שרוצים ב UDP. כששולחים את ההודעה שולחים את ההודעה וגם את הכתובת IP והפורט אליהם שולחים את ההודעה(מה שולחים ולאן שולחים). **הגדרות אצל השרת:** מגדירים מספר פורט של השרת, ביצירת הסוקט גם לא רושמים IP, אבל לאחר היצירה מקשרים את הסוקט למספר פורט שהגדרנו כדי שהשרת יהיה מקושר לפורט הזה. ממתינים לבקשות וכשמקבלים מקבלים הודעה ואת כתובת הלקוח, כדי שהוא ידע לאן להחזיר את התשובה ואז מחזירים.**TCP:** שירות אמין. יש לחיצת יד, מקבלים אישור כשהמידע נשלח והתקבל. מעביר מידע בצורה אמינה. יש יצירת התקשרות מוגדרת לפני שמתחילים שיחה. שליטה על עומסים, מגדיר לשלוח פחות כשיש עומסים. מגבלות : אין תזמון או יכולת להבטיח הספק מינימלי, שליטה על עומסים מאטה את האפליקציה. אין מנגנון אבטחה מובנה. אפשר להלביש עליו SSL לאבטחה אבל זה לא מובנה. השרת חייב ליצור סוקט לכתובת ספציפית של הלקוח, הלקוח יוצר סוקט וצריך להגדיר כתובת IP וסוקט ספציפי של הלקוח כי יש לחיצת יד. בשרת יש סוקט שממתין לבקשות, כשהוא מקבל הוא יוצר סוקט חדש שמטפל בבקשה, כשהסוקט המקורי ממשיך להמתין לבקשות חדשות. **הגדרות אצל הלקוח:** מגדירים שם ופורט של השרת, אחרי שיוצרים סוקט חייב לחבר אותו לשם ופורט של השרת בצורה ספציפית, אחרי זה שולחים וממתינים לתשובה, ובסוף סוגרים את הסוקט. **הגדרות אצל השרת:** מגדירים פורט, וגם כאן לאחר יצירת סוקט מקשרים אותו לפורט. הסוקט מאזין תמיד, ברגע שיש בקשה פותח סוקט חדש והסוקט החדש מטפל בבקשה, מחזיר תשובה וסוגר את עצמו, ובזמן הזה הסוקט המקורי ממשיך להאזין לבקשות חדשות.

שכבה 4 – שכבת התעבורה: איך אפליקציות מדברות אחת עם השנייה מאחוריי הקלעים. יוצרת תקשורת לוגית בין אפליקציות. רצה בתחנות הקצה. שתי שיטות

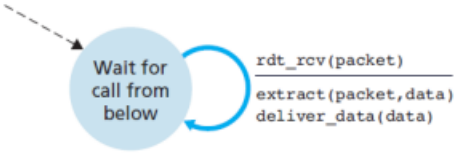
- registrar inserts two RRs into .com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

לתעבורה כמו שדיברנו הן TCP ו UDP. שכבה 3(הרשת) מקשרת בין שני מארחים(בין כתובות שונות). שכבה 4 מסתמכת על השירותים של שכבה 3 ומרחיבה אותם. IP יוצר קישור בין מארחים מרוחקים ומגדיר לוגיקה של מציאת מסלולים וכאלה.
ריבוב ודה-ריבוב: להריץ כמה הודעות על אותו ערוץ(ריבוב, mux), ומאותו הערוץ לפצל כל הודעה ליעד שלה(דה-ריבוב, demux). המספרי פורט עוזרים לדה-ריבוב. סוקט מזהה את התהליך, ID של תהליך זה המספר סוקט, פורט מוכל בתוך הסוקט, אבל בסוקט יש עוד דברים. סוקט מבחינתנו זה PORT + IP. בשליחת חבילה מציינים PORT יעד, בהגעה לשרת הוא יודע לפי המספר פורט לאן לנתב. כל מי ששולח לפורט 50 יכנס לאותו הפורט ב UDP. בTCP זה לא יהיה ככה. **UDP** הוא connectionles אין לחיצת יד, בשליחה מציינים IP ופורט של השולח כדי לדעת למי להחזיר את התשובה. עושים מינימום, הכל קורה על אותו הסוקט. יתרונות: יותר ישיר, אפשר לעשות העברה אמינה, לא צריך לשמור מצבים, הדרים קטנים(מכילים רק פורט מקור ויעד, אורך הודעה כולל ההדר, מתודת checksum לבדיקת שגיאות והמטען עצמו של האפליקציה, IP לא מועבר בחבילה). **Checksum**: סוכמים מספרים בינאריים, כל השדות שיש בהדר. אם יש overflow מחברים אותו לתוצאה. לאחר מכן הופכים את הביטים ומכניסים לשדה של checksum, וכך השולח שולח את ההודעה. המקבל עושה בעצמו את החישוב של checksum מבלי להפוך את הביטים ומחבר עם השדה שהשולח עדכן. אם התוצאה שווה אחדות בכל הביטים הכל תקין, אם לא אז יש שיבוש(יכולה להיות שגיאה ב checksum עצמו זו בעיה). גם שכבה 2 עושה checksum, בשכבה 2 בודקת לוקאלי, בשכבה 4 בודק מעבר חיצוני בין השולח למקבל(נקרא end to end principle, לעשות בדיקות בשני הקצוות). **TCP** הוא connection-oriented, כלומר יש לחיצת יד. כל סוקט מגדיר IP ופורט של השולח + IP ופורט של היעד. הדה-מרבב מקבל את כל ה4 נתונים האלה ומשתמש בכולם, כי מייצרים קישור ולחיצת יד לכל סוקט ששולח בקשה כלומר ריבוי חיבורים, לכן כל רביעייה תסמל קישור אחר. **עקרונות להעברה אמינה של מידע, הכנה ל TCP, פה עוברים על RDT פרוטוקולים להעברה אמינה מעל ערוץ לא אמין:** RDT – reliable data transfer, UDT – unreliable data transfer: יש ערוץ לא אמין, ועליו שולחים בעזרת ערוץ אמין. מניחים שחבילה אחת לא תעקוף חבילה אחרת. גרסה 1: שום דבר לא יתקלקל. גרסה 2: ביטים שגויים. גרסה 3: איבוד חבילות. מידע בקרה יעבור לשני הכיוונים. המידע שהלקוח מעביר תמיד יעבור מהשולח למקבל בכיוון אחד.
Rdt 1.0: אין טיפול בשגיאות, אין כאן בקרה כי אין טיפול בשגיאות. רק מצב אחד. השולח שולח חבילה למקבל. המקבל מקבל את החבילה, מוריד את ה header ומעביר לשכבת האפליקציה.
Rdt 2.0: מטפל בשגיאות בביטים: ניתן לזהות שיבוש בביטים עם checksum. מוסיפים הודעות ACK/NAK כדי לדעת אם חבילה לא הגיעה תקינה למקבל. לשולח יש 2 מצבים: שולח את החבילה ומחכה להודעת NAK או ACK. אם מתקבלת הודעת ACK הוא שולח את החבילה הבאה, אם מתקבלת הודעת NAK הוא שולח את אותה החבילה ששלח מקודם. למקבל יש מצב אחד – ממתין לקבלת חבילה. אם הגיעה שגויה שולח NAK וממשיך לחכות, אם הגיעה תקינה, מעביר לשכבת האפליקציה וממשיך לחכות לבאה. יש במבנה הזה פגם: מה אם ההודעות של ACK או NAK לא תקינות? איך נדע שהן השתבשו? **יש שדרוג של Rdt 2.1 שמטפלת בזה:** הפתרון הוא למספר הודעות כדי לדעת את הסדר של ההודעות ולדעת אם משהו השתבש או לא, כרגע מספיק ביט אחד למספר הודעות של 0 או 1, כי מבחינתנו אין איבוד חבילות אז מבחינתנו זה מספיק. יש checksum.**עבור המקבל:** מצב התחלתי מחכה לשלוח הודעה עם ביט 0, כשהוא רוצה לשלוח עובר מצב, מכין הודעה עם ביט 0 ושולח אותה, ולאחר מכן עובר למצב המתנה של ACK או NAK עבור ביט 0. אם התקבל ACK לא עושה כלום ועובר למצב הבא שבו הוא מחכה לשלוח הודעה עם ביט 1, ומשם הכל זהה. **עבור השולח:** מצב התחלתי ממתין לקבל הודעה עם ביט 0, אם הוא מקבל והיא משובשת הוא שולח NAK וממשיך להמתין לביט 0 באותו המצב, אם קיבל הודעה עם ביט 1 הוא שולח תשובה ו ACK בחזרה לשולח אבל נשאר באותו המצב כי עכשיו הוא ימתין באמת ל0. אם קיבל הודעה לא משובשת עם ביט 0 שולח לו ACK ותשובה, ועובר למצב הבא שבו הוא ממתין לביט 1, שם הכל זהה רק עבור ביט 1. **עכשיו יש לנו עוד שדרוג מגרסה 2.1 לגרסה 2.2** שמוריד לנו את ה NAK, אין צורך בו.

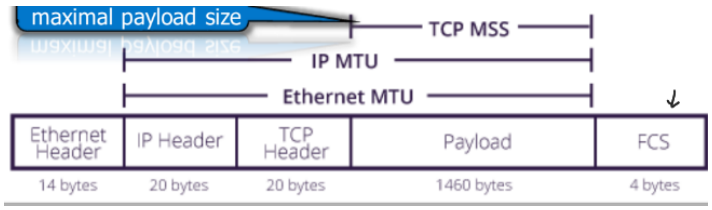


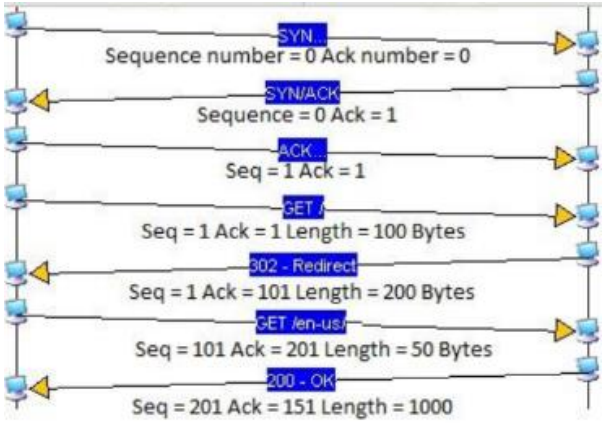
a. rdt1.0: sending side



במקום לשלוח ACK או NAK נשלח על הודעת 0 האק0 והודעת 1 האק1 , אם קיבלנו על הודעת 0 האק1 או הפוך, יש שגיאה ונשלח מחדש, יש לשים לב שאם הוא צריך לשלוח פעמיים הודעת ACK עם אותו מספר סידורי אז בפעם השניה הוא לא צריך "להכין" את החבילה מחדש, אלא רק שולח את הACK ששלח קודם).
Rdt 3.0:מטפל באיבוד חבילות: כאן השולח אחראי לעדכן אם הודעה הלכה לאיבוד. השולח מפעיל טיימר עם זמן סביר(זמן סביר = K * RTT + זמן העיבוד של החבילה, כש K יהיה גדול שווה מ1 אם הרשת איטית), אם לאחר שהטיימר נגמר הוא עדיין לא קיבל ACK או NAK משמע החבילה הלכה לאיבוד, ובמקרה כזה הוא צריך לשלוח מחדש את החבילה ולאפס את הטיימר. במידה והיה ACK שהגיע אחריי הטיימר יהיה שידור כפול, הביט שמתאר את מספר החבילה מטפל בבעיה הזאת. לא טוב שהמקבל יהיה אחראי על הזמנים, כי יכול להיות שהשולח בכלל לא שלח הודעה, לא הגיוני שהוא יהיה אחראי על זה. **השולח**: מצב התחלתי של המתנה לשליחת חבילה עם ביט 0, שולח את החבילה ומאתחל את הטיימר, ועובר למצב הבא בו ממתינים ל ACK על ביט 0. אם קיבלנו חבילה: 1. קיבלנו ACK על חבילה 1 החבילה ששלחנו הלכה לאיבוד. 2. קיבלנו חבילה משובשת, בשני התרחישים האלו לא עושים כלום וממשיכים להמתין ל ACK על 0. אם הטיימר נגמר הוא שולח שוב את החבילה ומתחיל את הטיימר שוב. במקרה בו הוא קיבל ACK על הודעה 0 הוא עוצר את הטיימר, ועובר למצב הבא שם הוא ממתין עד שיישלח חבילה חדשה עם סידורי 1, שם הכל זהה. רואים שבמצבים בהם הוא ממתין לשלוח הודעה אם מקבלים הודעה מתעלמים ממנה – זה מטפל במצב שבו קיבלנו ACK בדיליי ומתעלם ממנו. **המקבל**: ברובו נשאר זהה כל מה שהשתנה זה איך רושמים את ה ACK עם 0 או 1. מצב התחלתי שממתין להודעה עם ביט 0, אם קיבל הודעה תקינה עם ביט 1 שולח ACK 1(כלומר כמו NAK) וממתין באותו המצב להודעה עם ביט 0. אם קיבל הודעה משובשת גם אז שולח ACK עם 1 כלומר NAK וממשיך

להמתין להודעה תקינה. כשהוא מקבל הודעה תקינה עם ביט 0 הוא שולח בחזרה תשובה עם ACK עם 0 כלומר ACK רגיל, ועובר למצב הבא שם הוא ממתין להודעה עם ביט 1, ומשם הכל זהה. הביצועים של הגרסה השלישית לא טובים כי יוצא שממתינים הרבה. ניצולת של ערוץ מסוים(שולח לדוגמה) = זמן לשליחה(שזה שווה גודל חבילה / קצב שידור) / (RTT + זמן לשליחה), חישוב ניצולת עוזר להבין את הביצועים.**Pipelining** – השולח מאפשר שליחה של כמה חבילות בו זמנית, עוזר לשפר את הניצולת, שני פרוטוקולים מהסוג הזה : Go-back-N, ו Selective repeat. הפייפליין עוזר לשפר את הניצולת כי אנחנו מכפילים את המונה בנוסחה של הניצולת בכמות החבילות שניתן לשלוח בו זמנית(ניצולת גבוהה יותר = יותר טוב).**Go-back-N:** השולח שולח עד N חבילות שהוא לא קיבל עליהן עדיין ACK, המקבל שולח ACK נצבר, כלומר אם הוא שלח ACK על חבילה 5 הוא מאשר את כל החבילות עד חבילה 5. לשולח יש טיימר אחד שכשהוא נגמר הוא שולח את כל החלון הנוכחי שהוא מחכה לACK נצבר עליו, כלומר ישלח את כל החבילות עד 5 שוב. יש 4 סוגי חבילות : חבילות שהגיעו ויש עליהן ACK, חבילות שנשלחו ואין עדיין תשובה, חבילות שאפשר לשלוח ועדיין לא שלחנו(לא בטוח שיש לשלוח), וחבילות שעדיין אי אפשר לשלוח. ההסבר לשם של הפרוטוקול הוא שהוא חוזר אחורה לשלוח N חבילות אם אין ACK. מגדירים כאן חלון, גודל החלון הוא יהיה ה N. החלון זז לפי הACK שהתקבל. מגבילים רק עד N כדי שלא יהיה עומס על הקו. Base = החבילה הראשונה בחלון הנוכחי. Nextseqnum = החבילה הבאה שצריך לשלוח מהחלון. יש רק מצב אחד גם למקבל וגם לשולח כי יש N אינדקסים, יש לנו משתנים ששומרים את ה state וזה פותר אותנו מעוד מצבים. למקבל אין זיכרון, ולכן אם התקבלו חבילות אחרי שחבילה מסוימת הלכה לאיבוד הוא זורק אותן. **האוטומט של השולח** : ישנו מצב אחד של המתנה שהוא תמיד נשאר בו. אם הוא שולח מידע הוא בודק אם החבילה הבאה שהוא רוצה לשלוח נמצאת במסגרת הנוכחית(base + N) נותן את המספר של החבילה האחרונה במסגרת). אם כן הוא בונה חבילה ושולח אותה, ולאחר מכן בודק אם החבילה הראשונה במסגרת הנוכחית שווה לחבילה ששלחנו עכשיו(כלומר האם אנחנו בתחילת המסגרת), אם כן הוא מתחיל את הטיימר ומגדיל את המספר של החבילה הבאה לשליחה ב1. אחרת, אם החבילה הנוכחית לשליחה לא נמצאת במסגרת דוחים את השליחה הזאת. אם נגמר הטיימר הוא מאתחל אותו מחדש ושולח מחדש את כל החלון הנוכחי(בדיוק הפעולה שמגדירה Go back N. במידה והתקבלה חבילה והיא לא משובשת הוא מזיז את החלון קדימה, ככה שה Base של המסגרת החדשה יהיה שווה למספר הACK שהוא שלח ועוד 1(כלומר שלח ACK על 5 אז הבסיס החדש יהיה 6). אם אחרי זה הבסיס הוא החבילה הבאה לשליחה עוצרים את הטיימר כי שלחנו כבר הכל, סיימנו לשלוח את החלון, אחרת מאתחלים את הטיימר כי יש עוד חבילות לשלוח בחלון. בשיבוש חבילה אין פעולה, פשוט מחכים שהטיימר ייגמר. **אוטומט של המקבל:** גם מצב אחד של המתנה. אם התקבלה הודעה בלי שיבוש ויש לה מספר חבילה ששווה למספר חבילה שהוא ציפה לקבל הוא מחלץ את המידע, מחזיר תשובה עם ה ACK על המספר חבילה הזאת, ומגדיל את המספר חבילה הבאה שהוא מצפה לקבל ב1. אחרת אם משהו לא תקין שולח את ההודעה האחרונה שהוא הכין עם האינדקס הקודם שהיה(נניח שלח ACK על 5 אבל לא על 4, אז ימשיך לשלוח ACK על 3). ישנו מצב התחלתי אם כלום לא קורה מצפה להודעה עם מספר חבילה 1, ומכין תשובה על מספר חבילה 0 רק עבור אתחול. **Selective-Repeat :** גם כאן השולח שולח עד N חבילות שהוא לא קיבל עליהן עדיין ACK. כאן אבל המקבל שולח ACK על כל הודעה בנפרד, לכן גם יש טיימר על כל הודעה בנפרד, שהוא עדיין לא שלח עלייה ACK, ולכן אם פג הטיימר הוא ישלח רק את החבילה הספציפית עלייה פג הטיימר. גם פה יש חלון בגודל N, לשולח יש אותם 4 סוגי חבילות, למקבל יש 4 סוגים מנקודת המבט שלו : חבילה לא בסדר הנכון אבל עדיין נשלח ACK, חבילה שניתן לקבל ונמצאת בתוך החלון, חבילה שהוא מצפה לקבל אבל עדיין לא התקבלה, וחבילה שלא ניתן להשתמש בזה(מחוץ לחלון). השולח שולח חבילה אם היא נמצאת בתוך החלון, ויש לחבילה הספציפית הזאת טיימר, מקדמים את החלון לחבילה עם המזהה הכי נמוך שעדיין לא קיבלנו עלייה ACK. אם המקבל מקבל משהו לא בסדר נכון הוא שומר אותו בבאפר ורק אז מעביר לאפליקציה. פה הוא יכול להזיז את החלון ביותר מ1, ב Go back N הוא מזיז רק אחד.**TCP (ממומש רק בלקוח, לא בראוטרים, ראוטר מממש רק שכבה 3):** שולח כמה הודעות במקביל(פייפליין, משלב כאן גם Go back N וגם Selective-Repeat), ושולט על הזרימה, לא מפציץ את המקבל אלא שולח לו כמה שהוא יכול לקבל לפי גודל החלון בלחיצת יד יש 3 הודעות, 2 ההודעות הראשונות לא מכילות נתונים, הלקוח(יכול להיות גם PEER כאן) שולח הודעת SYN שהוא רוצה להתחיל התקשרות, השולח שולח SYN ACK שמאשר תחילת תקשורת, ורק אז בהודעה השלישית הלקוח כבר שולח מידע עם ACK. מעביר את המידע לסוקט, הסוקט הוא באפר שליחה. **MMS** – maximum segment size, גודל ההודעה עצמה שרוצים להעביר, maximal transmission unit – MTU, מקסימום העברה ביחידת זמן (מסתמך על שכבה 2, מפר את מודל השכבות). גודל הדר ב TCP הוא 20 בייטים(בדרך כלל). **חשוב לשים לב** ש MMS הוא לא כולל ההדר, לכן כל פעם צריך להתחשב בזה שצריכים לשמור 20 בייטים להדר בכל הודעה. גם אצל הלקוח וגם אצל השרת יש באפר לשליחה ובאפר לקבלה, משתני מצב וסוקט לחיבור. שום דבר לא בראוטר כי כל מה שממומש שם נמצא בשכבה 3. **מבנה הודעת TCP** : פורט מקור ויעד, מספר seq,ומספר ACK, יש דגלים שיכולים להיות 1 או 0, כמו URG שאומר להעביר הודעה לפני שבאפר מתמלא, PUSH אותו הדבר, דגל ACK, אורך ההדר(בדרך כלל בייטים20) אם מתווספים options יהיה גדול מ20, דגל איתחול, ריסטקט ומחיקת חיבור, checksum כמו שכבר ראינו, receive window שמגדיר לשני הצדדים את גודל החלון, *<u>התוכן לא יהיה 20 – MMS כי MMS זה כבר רק התוכן עצמו בלי ההדר, זה אחרי החיסור של ה 20, גודל התוכן יהיה קטן</u>*



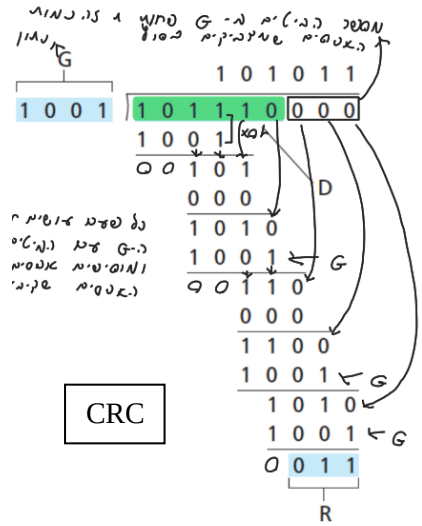


שווה ל MMS. TCP מסתכל על הנתונים שלו כרצף של ביטים, לא במבנה מסוים אבל הם מסודרים(ממוספרים) בסדר, אסור להחליף בין בייטים, המבנה אבל לא חשוב. המספרים הסידוריים הם של הביטים עצמם ולא של החבילות. לדוגמה גודל קובץ 500kb, ו 1kb MMS, כל חבילה תיהיה בגודל 1kb, ויהיו 500 חבילות כאלה(חבילה 1 בייט 0 עד 999, וכך הלאה). מספר seq יהיה הבייט הראשון, ולא לפי מספר החבילה. כלומר מספר ה seq יהיה הביט הראשון בחבילה. בדוגמה seq של חבילה 1 יהיה 0, של חבילה 2 יהיה 1000 וכך הלאה. ישנו קשר full duplex, קשר דו כיווני על אותו החיבור, כלומר שולחים בהודעה גם מספר seq על ההודעה הנוכחית(שיהיה הACK הבא שנצפה לקבל מהצד השני), וגם מספר ack על ההודעה האחרונה שקיבלנו מהצד השני. מספר ACK זה המספר seq הבא שמצפים לקבל מהצד השני. לדוגמה צד א' מקבל בייטים 535 – 0 מצד ב', עכשיו צד א' ישלח לצד ב' הודעה עם מספר ACK ששווה 536. או נניח צד א' קיבל 535 – 0 וגם 1000 – 900 כלומר יש איבוד חבילות, לכן הוא ישלח בחזרה ACK ששווה 536, TCP ישמור את ההודעות של 1000 – 900(זה יותר מורכב, שמירה, ובדרך כלל נלך לשמירה של הנתונים, מצמצם את הטראפיק), לעומת זאת GBN יזרוק אותן, לכן לרוב לא משתמשים בו. seq ו ACK לא מתחילים מ0 הם מאותחלים רנדומלית כדי לא לבלבל בין שידורים שונים. הגודל שלהם הוא 4 בייטים unsigned, כלומר הערך המקסימלי הוא 4294967294. מספר **seq שנשלח יהיה ה ACK שהתקבל**(נסתכל על ההודעת ok ונראה שבהודעה לפני ACK 201 =, לכן מספר ה seq החדש יהיה שווה ל ACK הזה, 201), **והמספר ACK שנשלח יהיה מספר ה seq + אורך ההודעה מהשלב הקודם**(נסתכל לדוגמה על הודעת ה Redirect, בהודעה הקודמת seq = 1, והאורך שווה 100, אז מספר ה ACK יהיה 1 + 100 = 101). **איך מגדירים את אורך הטיימרים שיש לנו ב TCP?** הזמן שצריך לחכות יהיה RTT, כלומר הזמן להעברת חבילה ועוד טיפה זמן ביטחון, הבעיה שהזמן הזה יכול להשתנות לפי עומסים ודברים כאלה. אם נחכה מעט מידיי זמן נעשה מלא שליחות חוזרות, אם נחכה יותר מידיי לא נבין שחבילות הלכו לאיבוד. הפתרון הוא לתת ערך טיימר לפי דגימות של זמן הגעה של חבילה. מחשבים בעזרת ממוצע משוקלל: $\boldsymbol{EstimatedRTT} = (1 - \alpha) * EstimatedRTT + \alpha *$ $SampleRTT$, כאשר אלפא תיתן משקל לכל ערך, יהיה נתון, EstimatedRTT מתוך הנוסחה זה החישוב של השלב הקודם, וה SampleRTT הוא הדגימה הנוכחית. אלפא מומלץ שיהיה בין 0.125 לחצי כדי שייתן משקל גדול יותר להערכה האחרונה כי היא יותר מייצגת. עבור הESTIMATED הראשון לא מחשבים את הנוסחה, ה estimated יהיה שווה ל sample הראשון. צריך לחשב גם זמן ביטחון: $\boldsymbol{DevRTT} = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$ מודד את השינוי בין הדגימה הנוכחית לבין החישוב הנוכחי שעשינו ולראות את הסטייה, וגם כאן הוא מתחשב בחישוב של הסטייה מהשלב הקודם. בדרך כלל בטא יהיה שווה 0.25, גם זה יהיה נתון לנו. ואז נוכל לחשב את הטיימר עם החישוב הבא:
TimeOutInterval = EstimatedRTT + 4 * DevRTT. TCP משתמש בנוסחה הזאת(בדרך כלל זמן התחלתי יהיה שנייה).
לשים לב שכל החישובים האלו הם איטרטיבים, כל חישוב מסתכל על שלב מסוים, מספר דגימה מסוימת, כלומר עבור כל דגימה מקבלים טיימר ספציפי עבורה.
עכשיו נסתכל ספציפית על TCP והשכלולים שלו מנגד ל RDT שראינו: משתמש בפייפליינים, בACK נצבר, בטיימר יחיד(נבדק וטיימר יחיד יותר יעיל). **השולח TCP** הפשוט מטפל ב3 אירועים: **אירוע 1** – קבלת מידע מהאפליקציה: אם הטיימר לא פעיל תפעיל אותו. יוצר חבילה עם nextseqnum ומעביר אותו ל IP. ומגדיר את nextseqsum להיות המספר האחרון של nextseqnum ועוד אורך החבילה שנשלחה. **אירוע 2** – פקיעת הטיימר: שולחים שוב את החבילה עם המספר seq הכי קטן בחלון, כלומר שומר את ההודעות האחרות בחלון לא זורק אותן, לא שולח את כל החלון בגלל זה, ומאתחל את הטיימר. **אירוע 3:** התקבל ACK: אם ה ACK קטן מהבסיס מתעלמים, אם גדול ממנו מגדירים את הבסיס להיות שווה אליו. ומאתחלים מחדש את הטיימר. **תרחישים אפשריים:** אם מקבלים כמה ACKים רצופים על אותה החבילה המקבל TCP מתייחס אליהם כNAK. נניח שיש timeout, זו לא בעיה כי הוא פשוט ישלח שוב כמו שאמרנו. התקבל ACK על חבילה אחרי שנגמר הטיימר – אין בעיה, הוא ישלח את ההודעה בשנית ובכל מקרה יעדכן את ה base להיות הערך החדש כשמתקבל ה ACK, וההודעה שנשלחה שוב פשוט לא תשנה כלום. הלך לאיבוד ACK על חבילה עם מספר קטן, אבל התקבל ACK על חבילה עם מספר גדול יותר – הבסיס יתעדכן לפי ה ACK על המספר הגדול יותר, לכן גם אם הACK על החבילה הקודמת הלך לאיבוד יש cumulative ack לכן ACK על החבילה עם המספר הגדול יותר מאשר שגם החבילה עם המספר הקטן יותר התקבלה. **עוד כמה שכלולים במקבל:** הוא לא זורק חבילות, הוא שומר אותן בבאפר ומשתמש בהמשך. לא מיד שולח ACK, הוא מעכב ACK כדי לא להציף את השולח בACKים, הוא צובר יותר חבילות כדי לשלוח ACK על כולן. אם מקבלים הודעה עם מספר seq שציפו לה והחבילה הקודמת עדיין ממתינה ל ACK מיידית שולח ACK על שתיהן. מקבלים הודעה שהיא לא בסדר הנכון, שולח ACK כפול על ההודעה הקודמת כלומר עושים מעיין NAK עם ACKים כפולים(השולח יזהה שיש בעיה לפני שיגמר הטיימר ולכן ישלח מחדש). אם מתקבלת חבילה שמשלימה "חור" שנוצר בין 2 חבילות שולחים מיידית ACK כדי למלא את החור הזה.
שכלול נוסף בשולח: אם לא מקבלים ACK אפשר להכפיל את ה timeoutinterval ב2, אחרי שקיבלנו ACK להחזיר אותו לערך המקורי. זה עוזר לבקרת עומסים, לרעת השולח כי מחכה יותר אבל לטובת המקבל כי מורידים ממנו עומס. TCP משתמש גם ב GBN (לדוגמה cumulative acks) וגם ב SR(לדוגמה כי שומר את המידע). **TCP flow control** – שליטת זרימה, בקרה על העברת הנתונים, דואג שהמקבל לא יוצף בהודעות לכן המקבל שולט על השולח. לשולח יש משתנה שנקרא Rwnd(receive window, חלון קבלה), שומר על איזון בין מהירות השידור של השולח ליכולת העיבוד של המקבל. שולח א' שולח קובץ גדול למקבל ב'. ב' שומר באפר קבלה ושני משתנים: lastbyteread – מספר הבייט האחרון שהוא קרא מתוך הבאפר, lastbytercvd – מספר הבייט האחרון שנכנס לבאפר שלו. וכך Rwnd = rcvBuffer – [lastbytercvd – lastbyteread], כלומר גודל החלון קבלה של השולח יהיה שווה לבאפר של המקבל(סך כל המקום בבאפר), פחות הבייט האחרון שנקלט לתוך הבאפר פחות הבייט האחרון שב' קרא מהבאפר(כלומר המקום שנוצל מתוך הבאפר) וכך מקבלים את סך כל המקום הפנוי בבאפר. המקבל שולח את ה rwnd לשולח בכל חבילה, גם אם אין עוד חבילות לשליחה. השולח זוכר את הבייט האחרון שהוא שלח ואת הבייט האחרון שקיבל עליו ACK. ישנו difference – מספר הבייטים שנשלחו וטרם התקבל עליהם ACK. ב UDP אין בקרת זרימה.
שכבה 3 – שכבת הרשת: יש 2 פונקציות מרכזיות: **forwarding** – קידום, איזה דרך לבחור לפי היעד הסופי, **routing** – ניתוב, תהליך ההקמה של כל עניין הקידום, להגדיר את המסלולים עצמם והכל. קידום מוכל בתוך ניתוב. ישנם אלגוריתמי ניתוב. Data plane – לוקאלי מה שמכיל את הפונקציית קידום. Control plane – איך שולטים בכל הניתובים בחלק של הרשת,

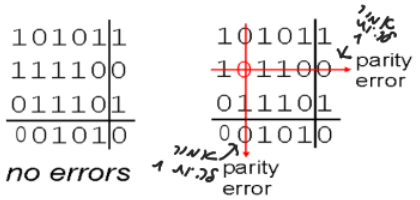
בו יש את האלגוריתמי ניתוב שממומשים בנתבים. אלגוריתם ניתוב מבצע מיפוי של הרשת לגרפים ובונה טבלאות ניתוב שיושבות בתוך ראוטר ספציפי, לפי טבלת הניתוב ממשיכים לאן לקדם את החבילה(קידום). **Packet switch** – מיתוג חבילות, להעביר חבילה ממקום למקום. הראוטר עושה packet switching. גם בשכבה 2 יש משהו שנקרא link layer switch שמבצע את המיתוג חבילות, בשכבה 3 זה הראוטר. באופן כללי שכבה 3 יכולה להציע עוד שירותים של אמינות הצפנה וכאלה, בטלפוניה מציעים אותם ובאינטרנט לא, כי מקבלים אותם בשכבות אחרות, וגם כי רוצים לאפשר התממשקות של פרוטוקולים שונים, לכן לא מבטיחים כלום. Connection – טלפוניה. Connection-less – אינטרנט. בטלפון יש לנו circuit, תפיסת קו, באינטרנט יש לנו העברה של חבילות פשוט. בטלפוניה כשיש התקשרות כל התחנות בדרך מודעות לשיחה ומקצות משאבים לקישור. לעומת זאת ברשתות של חבילות(אינטרנט) אין הקמה של שיחה, אין שמירת מצב, פשוט שולחים חבילות. **טבלאות ניתוב** – ישנו אלגוריתם שקובע את הטבלה, ובטבלה לפי כתובת IP של היעד מנתבים את החבילה ללינק מסוים. הטבלאות קצרות כי עובדים לפי prefixes, לא מנתבים כל כתובת ספציפית, משתמשים במשהו שנקרא longest prefix matching – הכתובת מתורגמת לרצף של ביטים, מנתב ללינק שיש לו את ההתאמה הכי מדויקת ליעד. **IP** – יש 2 גרסאות, IPv6, IPv4. **פורמט הדר של IP**: גרסה(האם 6 או 4), אורך ההדר(לרוב 20), **DSCP** שנותן שליטה על עומסים, אורך כל החבילה, **TTL** – כמות של פעמים שחבילה התקבלה בראוטר, בודק אם חבילה מסתובבת הרבה ברשת ולא מוצאת את היעד, כל קפיצה בין ראוטר הTTL קטן ב1 ואם הוא מגיע ל0 זורקים את החבילה. **Upper layer** – לאיזה פרוטוקול מעבירים את החבילה(הפרה של מודל השכבות, מועבר לפעמים כן לפרוטוקול בשכבה 3). כתובת IP מקור באורך 32 ביטים, כתובת IP יעד ב32 ביטים. **פרגמנטציה** – חלוקה של הודעה לחבילות קטנות יותר. MTU גודל מקסימלי של חבילה, אם הוא יהיה קטן מגודל חבילת IP צריך לבצע חלוקה של החבילה הזאת(חבילת IP לא גדולה מספיק). מפרקים וממספרים כל חבילה שפוצלה לנו מהחבילה המקורית. לא מתנגש אחרי מה שעשינו עם TCP כי אין מנגנון של שליחה מחדש וכל זה. משתמשים במספור כדי להרכיב את החבילה מחדש בסוף. עושים את זה ביעד ולא בראוטר כדי להמנע ממצב של איבוד מידע או צורך לפרק שוב בראוטר הבא, לכן עדיף להרכיב את החבילות רק ביעד. ישנו דגל שנקרא fragflag שמתאר אם מדובר בחבילה שפוצלה. הוא יהיה 1 אם כן, והחבילה האחרונה שפוצלה תקבל ערך 0 כדי לדעת עד איזו חבילה צריך לאחד. יש שדה ID שמזהה את החבילה, אורך החבילה(כאן חשוב לשים לב שאורך אם נניח יהיה 1500 זה כולל ההדר, כלומר המטען יהיה בגודל 1480), ו offset שמתאר לאיפה מחברים את החבילה בשחזור ואיחוד לאחר מכן(מתואר במספר בייטים לכן צריך גם לכפול אותו ב8 כדי לקבל מיקום בביטים ולדעת לאיפה לחבר). **איזה שגיאות יכולות להיות**: חלק מהחבילות שפיצלנו לא הגיעו, איך נזהה: לא קיבלנו את החבילה האחרונה שיש לה fragflag 0, אומר שעדיין משהו חסר. אפשר גם לפי הoffset לדעת אם יש משהו שלא קיבלנו. מה IP יעשה במקרה של שגיאה: יזרוק את החבילה, מבחינת TCP פשוט הודעה שנשלחה ולא הגיעה אז הוא יתמודד עם זה בצורה טובה. עולה זמן לפצל ולאחד, פתח להתקפות DOS. ב IPV6 אין יותר פרגמנטציה. **כתובות IP:** כתובות IP בגודל 32 ביטים(4 בייטים), יש 256 בחזקת 4 כתובות אפשריות, בערך 4 ביליון, אין מספיק כתובות בשביל זה יש נניח NAT. **ממשק** – מחבר רשת לאינטרנט, כתובת IP מזהה ממשק. **Subnet** – קבוצה של כתובות עם אותו prefix. מורכב משני חלקים: Subnet – החלק הגדול, מה שמופיע משמאל. Host – החלק הקטן שמופיע מימין. סבנט מזהה את הרשת, מארח מזהה את המכשיר בתוך הרשת.**IP classful addressing** – במקור מחלקים סבנט ל3 סוגים של כתובות: Class A – הסבנט בגודל 8. Class B – סבנט בגודל 16. Class C – סבנט בגודל 24 (כתובות IP סהכ יהיה 256 בחזקת 4 כפול מספר הקלאסים – 3). **הבעיות עם חלוקה למחלקות:** אין משהו באמצע, אם צריך 2000 כתובות מחלקה C קטנה מידיי מחלקה B עצומה מידיי, בזבזנית, לא גמיש.(לשים לב לא הכל מוקצה יש כתובות שנניח הכל 255 זו כתובת broadcast ששולחת הודעה לכל התחנות, כתובות שמורות עם מטרה). פתרון חלקי: **CIDR** (כתובות IP סהכ יהיה 30(סבנטים אפשריים) כפול 256 בחזקת 4)– classless interdomain Routing, הסבנט לא יהיה רק בקפיצות של 8, הוא יכול לקבל כל ערך וככה נוסיף גמישות ומאפשר למחזר כתובות כתלות בגודל הסבנט. בגלל שאפשר לחתוך באמצע לוקחים את כל החלק של הסבנט ועושים לו padding מימין, כדי לתרגם אותו לבסיס 10. ספק יכול לעשות סבנט על סבנט. הספק מקבל סבנט, כלומר מרחב כתובות מסוים, ואותו הספק מחלק גם לחלקים, כלומר עושה סבנט בעצמו, וכל חלק הוא מרחב כתובות שונה שהוא יכול להקצות ללקוחות שלו. השיטות האלו מאפשרות לספקי אינטרנט שונים להשתמש במרחבי כתובות שלא הוקצו להם מלכתחילה, מה שמקל על לקוחות לעבור בין ספק לספק, יכול להשאר עם אותה הכתובת ורק משנה את הסבנט. איך מקצים את הכתובות IP עצמן: ארגון ICANN. **flow table:** מוגדר לפי שדות ההדר. מגדיר קידום כללי לפי חוקים: התאמה – מציאת תבניות בערכי השדות בהדרים. פעולות – ניתן לזרוק, לקדם, לערוך או לארוז ולהעביר לבקר חבילה מסוימת. להגדיר דחיפות, ולמספר לפי מספר בייטים ומספר חבילות. לדוגמה האבסטרקציה הזאת ממומשת בנתב. ההתאמה עובדת בעזרת ה prefix IP, והפעולה היא לקדם את החבילה דרך לינק.

פרוטוקולים בשכבת הרשת עבור IP: כל מכשיר(מארח) על הרשת צריך לקבל כתובת IP משלו. אפשרות פחות טובה קידוד ידני. אפשרות טובה יותר: **DHCP** – dynamic host configuration protocol, מקבל בצורה דינמית כתובת IP בלי לעשות שום דבר. הכתובת שמקבלים היא קבועה כל עוד לא יקרה כלום לרשת, אבל בפעם הבאה שנתחבר נקבל משהו רנדומלי חדש, זה עובד כמעין השכרה של כתובת מתוך מאגר של כתובות שניתן לקבל. התהליך עובד ב4 שלבים עיקריים: ההוסט שולח בDHCP discover broadcast – אם יש שרת DHCP ואם הוא יכול לקבל בקשה. לאחר מכן שרת ה DHCP מגיב עם DHCP offer שמציע לו כתובת. ההוסט מבקש את הכתובת עם DHCP request, ולאחר מכן השרת עונה ב DHCP ack לאישור קבלה של הכתובת(אם יהיה בונוס הודעה חמישית תיהיה DHCP release, לשחרר את הכתובת הזאת). שרת DHCP יכול להיות: באותה הרשת של ההוסט, הראוטר עצמו משמש כשרת DHCP, שרת ה DHCP לא יהיה באותה הרשת. שדות בהודעות: כתובת מקור ויעד כולל סבנט, מתחיל כברודקאסט, שדה yiaddr שברגע שהשרת מציע כתובת היא מאוחסנת שם, ומספר אקראי שעוזר לעקוב אחר מי שלח את הבקשה. הלקוח צריך לענות והשרת צריך לאשר כי מישהו יכול להיות שתפס כבר את הכתובת הזאת/ איבוד חבילות. עובד מעל UDP כי לא צריך לציין כתובת ספציפית של לקוח ואפשר לעשות ברודקאסט, בTCP זה לא אפשרי. DHCP (אם יהיה בונוס זה הפרוטוקול היחידי עם אותו פורט מוגדר מראש גם בלקוח וגם בשרת) עושה יותר מרק להקצות כתובות, הוא גם אומר מי הראוטר הראשון, גם אומר מי ה DNS SERVER הראשון ומספק את הסבנט. **NAT** – הפתרון שמשתמשים בו בפועל במקום CIDR. Network address translation, מוגדר בראוטר, יש לו כתובת IP חיצונית שיוצאת לרשת, וכתובת IP פנימית(לוקאלית), הראוטר יודע להמיר בין 2 הכתובות האלו. עוזר לאבטחה שלא רואים את הIP

הפנימית, קל להחליף ספק אין אפקט לוקאלי. יש 4 סוגי NAT, הכי חשוב והכי טוב זה PAT – port address translation. יש static nat – לא חוסך כתובות אין לו יתרון, תורם רק לאבטחה. Dynamic nat – מאגר של כתובות עדיין לא חוסך כתובות IP. Port forwarding – להשתמש בכתובת חיצונית ופורט כמו PAT ולפי הפורט יודע לנתב לIP מסוים, גם פתרון מקובל, בעיקר לתעבורה נכנסת. **הרחבה על PAT:** משתמשים באותה הכתובת IP של הראוטר אבל בעזרת הפורטים הראוטר מוציא כל שיחה על פורט אחר. עוזר לחסוך כתובות ולהסתמך על מספר פורט שהראוטר מקצה וכתובת IP אחת. כמה תחנות יוצאת על אותו IP לרשת, אבל עם מספר פורט שונה. יש כתובת לוקאלית וכתובת חיצונית מהראוטר, פורט של המכשיר ופורט שהראוטר הקצה. הראוטר שומר את ה 2 זוגות האלה, זוג של כתובת ופורט פנימיים, זוג של כתובת ופורט חיצוניים בטבלה. כשמגיעה חבילה מהאינטרנט לפי הטבלה מנתב למי שצריך להגיע אליו לוקאלית. גודל מספר פורט הוא 16 ביטים. **בעיות עם PAT:** מפר את מודל השכבות, מכניסים מושג של פורטים משכבות 4 או 5 לתוך ראוטר משכבה 3. בלבול בין IP ופורט מדויק לתקשורת כי משתמשים בפורט שהראוטר הקצה. הפתרון הבאמת טוב הוא להשתמש ב IPV6. לא פונים ישירות לתחנות הראוטר מסתיר אותן ואין איך לעקוף את זה. **ICMP** – internet control message protocol, בעיקר מדווח על שגיאות ברמת IP לדוגמה ראוטר מפנה חבילות לאיזור ברשת שיש שם ראוטר שלא מתפקד, לא חלק מ IP אבל בפועל יושב מעל IP באותה השכבה. פרוטוקול לדיווח שגיאות, אפשר גם לדבג, לראות מסלולים וכו.**IPv6** – אורך ההדר עכשיו 40 בייטים. יש פי 4 ביטים לכתובת (במקום 32 יש 128 ביטים לכתובת) כלומר במקום 2 בחזקת 32 כתובות יהיה 2 בחזקת 32 בחזקת 4(2 בחזקת 128) כתובות שונות. הפורמט להדר: priority ו flow label עוזרים לתעדף חבילות על פני חבילות אחרות, אורך המטען, next hdr – לדעת מה הפרוטוקול בשכבה 4(שכבה למעלה), אם יש options, ו hop limit שהוא כמו TTL בהדר של IPV4 – מספר מעברים בין ראוטרים. אין פה checksum או פרגמנטציה, ראו שזה מיותר. Anycast – נניח לשרת מסוים יש כמה כתובות ברשת עם אותו IP, עוזר לגלות מה הכי קרובה ולפנות אלייה. גרסה חדשה של ICMPV6 שפשוט תומכת בעוד דברים. **Tunneling** – לא כל הראוטרים מעודכנים להשתמש ב IPv6, לכן צריך תמיכה מלאה גם ב4. מה שעושים זה שחבילה בגרסת IPv6 מוכלת כמטען בחבילה בגרסת IPv4. כלומר עוטפים בהדר של IPv4, מעבירים עד ליעד ואז שם מחלצים את המטען שהוא הודעה בפורמט IPv6.



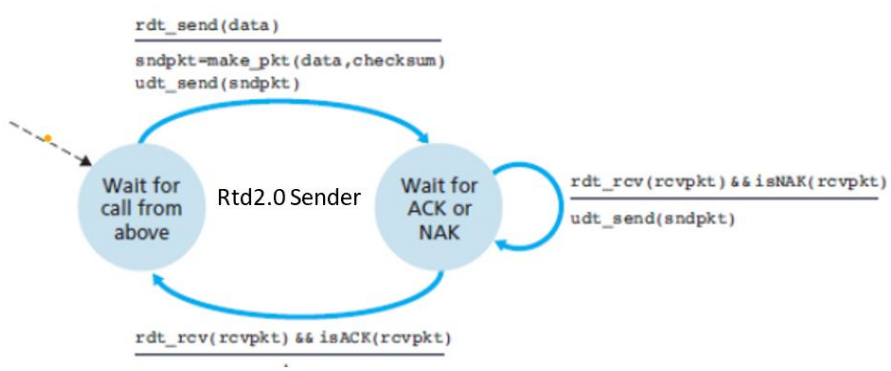
שכבה 2 – שכבת הערוץ: מארחים וראוטרים נקראים נודים(צמתים), ומתחברים אחד עם השני. התפקיד שלה היא שכולם יצליחו לדבר אחד עם השני. בשכבה 2 יש גם trailer שמופיע בסוף החבילה(כמו הדר רק בסוף). בשכבה 2 מתחייבים לאמינות בגלל שזה מקומי, בשכבה 3 לא אפשרי כי הרשת הטרוגנית ולא ידוע מה קורה, בלוקאלית כן ידוע. עוד שירותים: שליטה בזרימה, קצב העברת נתונים, גם זיהוי וגם תיקון שגיאות. שכבת הרשת קיימת בכרטיס הרשת בכל מכשיר(חומרה) הצד השולח שולח חבילות בתוך פריים, יש גם הדר וגם טריילר, וגם מוסיף ביטים לבדיקת שגיאות וכולי, הצד המקבל בודק לשגיאות, מחלץ את החבילות ומעביר לשכבה הבאה.

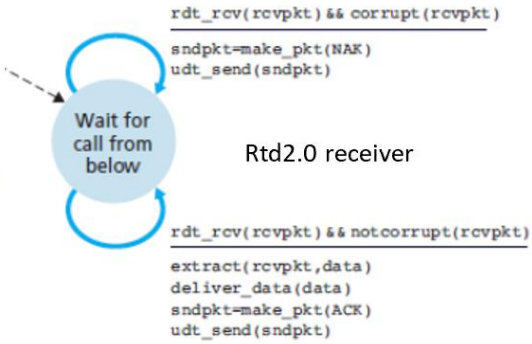


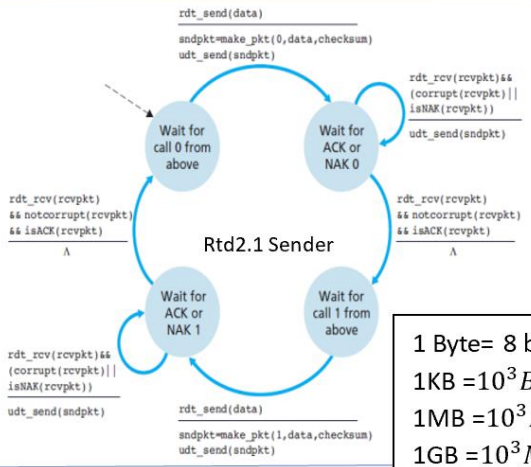
זיהוי ותיקון שגיאות – EDC error detection and correction bit, ביטים לזיהוי ותיקון שגיאות, כמו למשל checksum. **שיטות נוספות לזיהוי שגיאות: ביט זוגיות יחיד** – מגדיר ביט זוגיות יחיד עבור כל החבילה, מזהה רק אם יש שגיאה או לא, משלים את מספר האחדות למספר זוגי של אחדות(אפשר גם להגדיר למספר אי זוגי זה לא באמת משנה). **ביט זוגיות דו מימדי** – ישנו מערך דו מימדי, מגדירים ביט זוגיות לכל אחת מהעמודות ולכל אחת מהשורות. בצורה הזאת אם יש שגיאה ניתן גם לזהות איפה בדיוק השגיאה לפי השורה והעמודה ולתקן אותה. **שיטה יותר מסובכת נקראת CRC, cyclic redundancy check** – ישנו G כלשהו שנתנו לנו ומחרוזת שרוצים לשלוח. לוקחים את המחרוזת ומצמידים לה מימין אפסים כמספר הביטים ב G פחות אחד. אם ב G 4 ביטים מצמידים שלושה אפסים. מתחת למחרוזת החדשה הזאת רושמים את G צמוד לשמאל ועושים xor לכל אחד מהביטים, ומורידים את הביטים הבאים למטה(כל עוד הערך שקיבלנו לא באורך של G נוריד ביטים מלמעלה), עד שבסוף אין עוד ביטים להוריד וקיבלנו תוצאה שנקרא לה R. אפשר ללכת בכמה כיוונים כאן, אחד מהם הוא לשלוח ביחד עם החבילה את R, ולהגיד למקבל "תעשה את החישב הזה ותראה שאתה מקבל את R" – אם לא קיבל את R יש שגיאה.

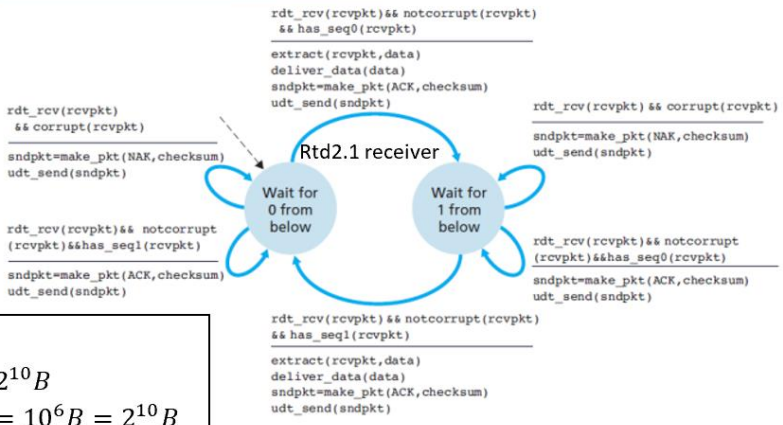
פרוטוקולים multiple access – כמה אנשים שמדברים בו זמנית, ערוץ שידור יחיד וכמה רוצים לדבר עליו בו זמנית, יתכן collision, התנגשות כששניים או יותר מנסים לדבר בו זמנית. מגדירים פרוטוקולים שאומרים מי כן יכול לדבר ומתי. יש 3 שיטות: **שיטה 1 – חלוקת ערוץ:** לחלק את הערוץ בעזרת TDM או FDM שאנחנו כבר מכירים(כאן הם נקראים FDMA, TDMA). אם נסתכל על TDMA גודל כל פרוסה קבוע והגודל יהיה קצב השידור בכל איטרציה. מי שלא פעיל(לא מדבר) ממתין, כלומר יש בזבוז של פיסות. FDMA בדיוק אותו הדבר רק לתדרים. **שיטה 2 – חלוקה רנדומלית:** יש כמה דרכים. אחת היא slotted aloha, יש פריימים שכולם באותו הגודל והזמן מחולק, כל צומת מנסה לשדר בצורה רנדומלית, אם יותר מ2 צמתים ניסו ביחד כל הצמתים מזהים התנגשות. זה כמו הטלת מטבע על מי מדבר מתי, ויש 3 תרחישים: התנגשות – יותר מ2 ניסו לדבר ביחד. הצלחה – רק אחד שידר לכן הצליח. ריק – אף אחד לא שידר. יתרונות: רק אחד יכול לשדר בקצב שידור מלא, פשוט. חסרונות: יש התנגשויות ובזבוזים, תאים לא מנוצלים. יש לשיטה הזאת גרסה טהורה – **pure aloha** – אותו הדבר רק שפה אין תאים מוגדרים, אין סנכרון יותר פשוט, אבל זה גורר שיש יותר הסתברות להתנגשויות לכן שיטה פחות טובה. דרך אחרת היא **CSMA** – תאזין לפני שידור, אם הערוץ בהמתנה תשדר פריים מלא, אם הערוץ עסוק תדחה את השידור(אל תפריע לאחרים). התאוששות מהתנגשות תיהיה איטית. יש גרסה CSMA/CD שהיא עם זיהוי התנגשויות. הצומת תבדוק אם תיהיה התנגשות, אם תיהיה לא משדרים(שיחה מנומסת). עדיין יש סיכוי להתנגשויות וההתאששות תיהיה מהירה. **שיטה 3 – הגדרת**

תורים: שיטה מחמירה יותר, מעיין round robin. שיטה 1 טובה לעומסים כבדים אבל בעומס נמוך יהיה הרבה בזבזו. שיטה 2 טובה לעומסים קלים אבל בעומסים כבדים יהיו יותר התנגשויות הטלות מטבע ודברים כאלה. שיטת התורים אמורה לשלב את הטוב משניהם. אחת השיטות כאן נקראת תשאול – יש מאסטר מסוים ששואל כל אחת מהתחנות אם הוא רוצה לדבר, פשוט מתווך. שיטה נוספת נקראת העברת טוקן – "טוקן שליטה" מועבר לכל אחת מהצמתים, רק מי שמחזיק בטוקן יכול לדבר.
כתובות MAC: כתובת MAC צרובה בכרטיס רשת ומשומשת בצורה לוקאלית. בגודל 48 ביטים, עוזרות לזהות מכשיר ברשת לוקאלית, לכל מכשיר יש כתובת ייחודית שהוא מקבל מהכרטיס רשת(MAC לא יכול להשתנות, כתובת IP כן יכולה להשתנות). **ARP:** address resolution protocol – מחבר בין הכתובת IP לבין הכתובת MAC. זו טבלה שמחזיקה רשומות כשכל רשומה קושרת בין כתובת IP לכתובת MAC של המכשיר. לכל רשומה יש כתובת IP, כתובת MAC ו TTL – אחרי TTL מסוים, לרוב 20 דקות, הרשומה הזאת נמחקת מהטבלת ARP. לכל מכשיר יש טבלת ARP כדי שיוכל להגיע למכשיר הספציפי. נניח A רוצה לשלוח הודעה לB, הוא מחפש בטבלת ARP, נניח ולא מצב הוא ישלח הודעת broadcast לבירור של מי הIP שהוא מחזיק שייך. מי שהכתובת IP שייכת לו יגיב עם הכתובת MAC שלו, וכך A ישמור בטבלת ARP שלו את הכתובת IP עם הכתובת MAC שקיבל. אם הודעה לא מיועדת ל MAC ספציפי הוא יזרוק אותה, אלא אם כן זו הודעת broadcast, שזה מופנה לכולם.
נקודה חשובה: דיברנו על האינטרנט אבל שאלות במבחן לאו דווקא ידברו על אינטרנט אלא על רשתות בכללי, לשים לב!









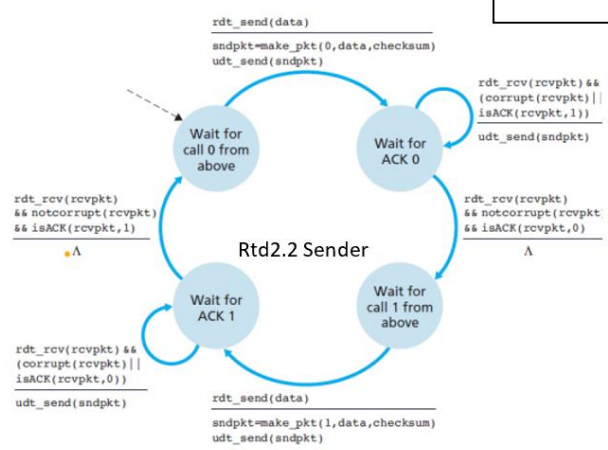
1 Byte= 8 bit

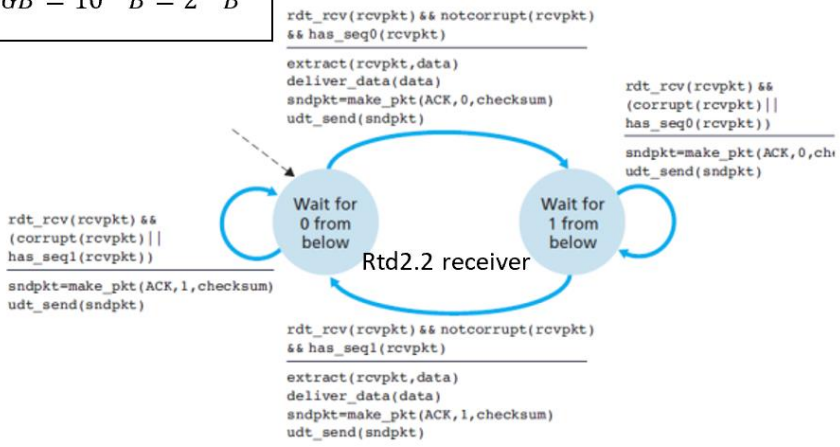
1KB $=10^3B = 2^{10}B$

1MB $=10^3KB = 10^6B = 2^{10}B$

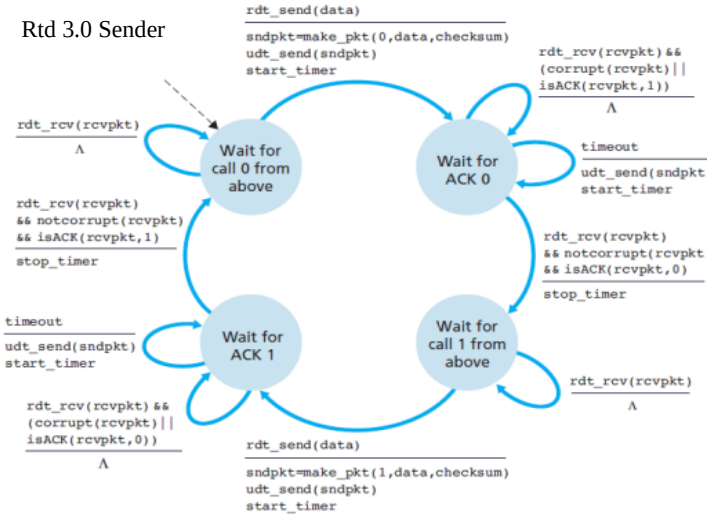
1GB $=10^3MB = 10^9B = 2^{30}B$

1TB $=10^3GB = 10^{12}B = 2^{40}B$

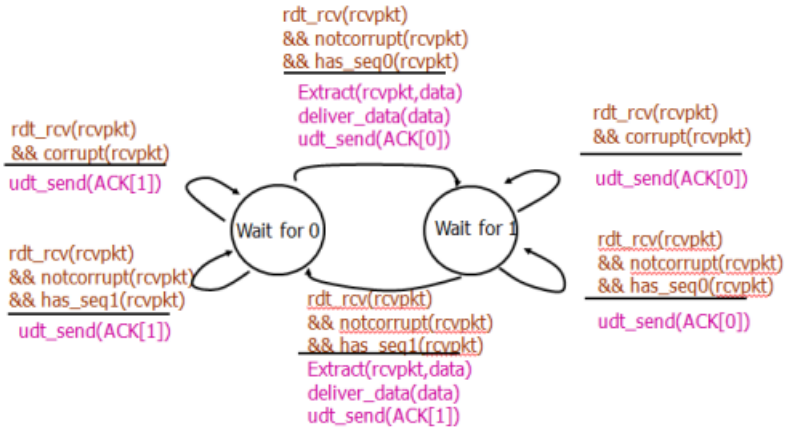




Rtd 3.0 Sender



Rtd3.0 receiver



חישובים- תמיד נרצה להמיר יחידות לbit, כאשר המרה מ-Byte ל-bit יש להכפיל את המספר ב 8 וכך הלאה

הפיכת ממילישניות לשניות -> מחלקים ב1000.sec=1000ms