סיכום רשתות

<u>איזה בעיות עלולות לצוץ כששולחים לשרת משהו מהמכשיר שלנו?</u>

Different hosts

Different media

Far away networks

Not same languages

Different message formats

Different topologies(כמו מהירות)

הודעות לא יכולות להתבטל (אי אפשר למחוק מה שעלה לרשת)

רעש בערוצים

המון ערוצים פועלים במקביל

מתקפות סייבר

<u>פתרונות לכך</u>

תקשורת סטנדרטית = פרוטוקולים, מספרים ולא אותיות(נשלחים ביטים)

כתובת סטנדרטית=כתובת IP

הצפנה

מודל השכבות

בצד השולח כל שכבה שמקבלת חבילה מוסיפה header כלומר מידע משלה ומעבירה לשכבה מתחתיה .ובצד המקבל כל שכבה "מקלפת" את השכבה המקבילה לה.

<u>יתרונות</u> => לכל שכבה תפקיד משלה

<u>חסרונות</u> => יש הפרות, דברים שחוזרים על עצמם

למדנו רק את מודל 5 השכבות ולא הגרסה המורחבת של ה7 כי במציאות, מרבית היישומים שנדרשים לשירותים שנכללים בשכבות אלה מממשים אותם בעצמם

שכבות 3,4,5 הן שכבות תוכנה ושכבות 1,2,3 הן שכבות חומרה

לא קיים מצב שבו חבילה תסתובב לנצח לכן בתוך החבילה יש time to live counter

ואם היא מגיעה לאפס אז זורקים את החבילה לפח

לכן header של שכבה יכול להשתנות

למה אינטרנט היא רשת הטרוגנית?

קשרים לא אחידים

תחנות שונות

מערכות הפעלה שונות

שינויים לאורך זמן

בעיות שצריך לפתור באינטרנט

תחנות מרוחקות שצריכות לתקשר ביניהן ונפתור זאת על ידי אלגוריתמי ניתוב

שליחת מידע אמין על גבי ערוץ לא אמין נפתור על ידי פרוטוקולים

תקיפות סייבר נפתור על ידי הצפנה אבל לא רק

אינטרנט

אבני בניין שירותים

<u>אבני בניין</u> => כל מה שמריץ=host, את האבני בניין מחברים לאינטרנט, אפליקציות רשת

<u>שירותים</u> => אפליקציה נותנת שירותים, נמצאות על תחנות שונות, השירות מתבצע דרך Routing applications (פה זה הAPI וpacket switching)

תחנות קצה\end systems = לקוח, סמארטפון, peer, שרת = אפליקציות רשת שמתחברות לאינטרנט

<u>ראוטר</u> => מקבל חבילות ומחליט לאן צריך להעביר

<u>Switch</u> => הוא כמו ראוטר אבל עובד רק לוקלית והחיבור תמיד בתוך אותה רשת

<u>ISP</u> => ספק אינטרנט- ארגון או חברה שמחברת את הלקוחות לאינטרנט

יש אפשרות שכמה ISP מחוברים לISP גלובלי = זול אך אם תוקפים את המרכז הכל נופל

יש גם אפשרות שכל כמה ISP מחוברים לתת רשת ויש פה יתרונות וחסרונות

היתרון מדיניות אחידה, החיסרון נקודות כשל מקומיות

חיבור בין ISP נקרא peering link

<u>ליבת הרשת</u> => ראוטרים שמחוברים אליהם הרבה מכשירים, יותר מידע עובר, נמצא יותר גבוה בהיררכיה

<u>איך תחנות קצה מתחברות לראוטר?</u>

דרך וויפי או כל חיבור (נגיד שכונתי, של המכללה)

לכל חיבור קצב משלו

יכול להיות shared משותף כמו וויפי או משהו ספציפי

IXP => מקום בו משתפים מידע, תחנה שמתחברים אליה ספקים שונים שמשתפים מידע

שירות בתחנת הקצה למשתמש – קישוריות לאינטרנט, גלישה בWWW שליחת דוא"ל

שירותים בתחנת הקצה לאפליקציות – העברת נתונים אמינה – בסדר נכון וללא שגיאות או חוסרים, קישור עם אפליקציה מרוחקת – העברת נתונים לאפליקציה שיכול להמצא במקום מרוחק

שירות בנתב – קידום וניתוב – העברת החבילה למסלול הקרוב יותר ליעד שלה. וגם – הקטנת ה TTL למניעת מצב שבו חבילה נודדת ברשת ללא סוף

פרוטוקולים

אוסף כללי תקשורת, כמו שפה אחידה שכולם משתמשים כדי שכולם יוכלו לתקשר

בפרוטוקול מוגדר מבנה ההודעה, סדר ההודעה ומה עושים עם כל סוג של הודעה

<u>העברה משכבה N לשכבה N+1</u>

לא באופן ישיר, מתבצעת בשכבות הנמוכות על ידי מערכת ההפעלה, קילוף הheader

<u>העברה משכבה N לשכבה N-1</u>

באופן ישיר, נוסיף header ונעביר את כל הנתונים

סטנדרטים של אינטרנט

פותח על ידי IEIF שהיא ועדת תקנה של האינטרנט, כל פרוטוקול מוגדר במסמך, כל המידע עליו כל הדרישות

לכל אחד יש מספר rfc וכל אחד מתייחס לפרוטוקול אחד בלבד ואפשר לחדש RFC ישן

P2P

תקשורת ישירה בין לקוח ללקוח, כל ישות מהווה גם לקוח גם שרת בו זמנית

יתרונות=> אין צורך להשקיע בשרתים, גם אם יש עומס זה אומר שיש יותר לקוחות וזה אומר שזה יותר כוח עבודה, סימטרי

החסרונות=> אין גורם מתווך כלומר אין מי שיפקח על התהליך אז עלול לגרום לבעיות פרטיות ואבטחה, יכול להשפיע לדוגמא על קצב הורדה כי תיתנן עדיפות למשהו שמתבצע מהר חלק מהפרוטוקולים בשביל לממש אותם נצטרך לשלם כי אין פה קוד פתוח, קשיים באבטחה ובאחריות

מודל שרת לקוח

שרת=משהו שתמיד עובד, בדרך כלל נמצא בכתובת קבועה , יש חוות שרתים, שרתים וירטואליים

לקוח= יוצר את הקשר בדרך כלל, ה ipיכול להשתנות בתדירות גבוהה, לקוחות לא מדברים אחד עם השני

בדרך כלל לכל אפליקציה מורץ שרת משלה .היתרון שהוא אמין, תמיד זמין, וקל להגן החיסרון שקל לתקוף את השרת ובכך להקריס נקודתית .התקשורת נעשית באמצעות בקשות

לא סימטרי, תחנה אחת יוזמת קישור והשנייה זמינה רוב הזמן

<u>שילוב של P2P ושרת לקוח</u>

קישור על ידי שרת אך תקשורת בין עמיתים באופן עצמאי

לדוגמא סקייפ

Host => שולח חבילות שבתוכן L ביטים ומלביש על הקו, קצב השידור R

מהירות הקו היא L\R

רשת של ראוטרים מחוברים.

החבילה מגיעה לראוטר והראוטר מקדם אותה עד שבסוף היא תגיע ליעד.

Forwarding => לקדם את החבילה

פעולה לוקלית, מזיזה את החבילה מהלינק שממנה התקבלה ללינק שאליה צריך לשלוח.

Routing => פעולה גלובלית, כל האלגוריתמי ניתוב שאחראים למצוא את הדרך הכי קצרה שאפשר להעביר בה את החבילה

Store and Forward:

בעיקרון זה מחכים שכל החבילה תעבור לפני שמתחילים לשדר אותה על מנת לא להגיע למצב שפתאום החבילה מפסיקה להגיע ונתקענו עם חצי חבילה

גם זה עיכוב השידור שלוקח L\R seconds

Packet switching

שולחים מידע על בסיס מקום פנוי, יכול להיווצר מצב של תור אם קצב השידור קטן מקצב ההגעה

המקום של התורים בראוטר מוגבלים אס אם אין מקום החבילות שיבואו נזרקות פשוט

בpacket switching ניתן לשתף, פשוט יותר אך לפעמים יכול להיווצר מצב של עומס.

יכול לתמוך ביותר משתמשים, אין צורך לעשות קישורים, חלוקת תדרים, אין הבטחות.

Circuit switching

מיתוג מעגלים, מקצים קו לשיחה, משאב קבוע לאורך כל השיחה

אין פה שיתוף

מראש כל התחנות בדרך צריכות להסכים (כי הן גם מקצות משאבים לכך)

משתמשים בזה יותר בטלפונים מאשר באינטרנט

לדוגמא שיחת טלפון למסעדה כדי לבצע הזמנת מקום לפני ההגעה

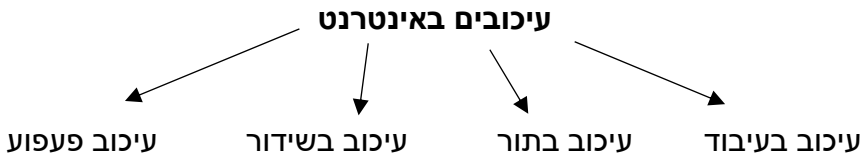
אפשר לממש על ידי TDM או FDM

<u>TDM</u>

מחלקים את הזמן לקבוצות, ניתן כל פעם לזמן קצר את כל רוחב הפס למשתמש אחד והוא יוכל לשדר הרבה יותר מידע ואז זה עובר להבא בתור וחוזר כך שוב ושוב.

<u>FDM</u>

מחלקים לפי תדרים, לכל משתמש תדר משלו לאורך כל השיחה(משמש בשידורי רדיו)



עיכוב בעיבוד proc => כשמגיעה חבילה עושים בדיקות כמו שגיאות ביטים וזה הזמן הזה

עיכוב בתור queue => הזמן מאז שהחבילה הגיעה ועד שמתחילים לשדר, יכול להיות הכי גבוה מכל השאר

עיכוב בשידור trans => הזמן שלוקח לשדר את כל החבילה L\R

עיכוב פעפוע prop => כמה זמן לוקח לחבילה להגיע לצד השני D\S

כאשר D אורך הקו וS מהירות הפעפוע

(באיורים חץ מאונך פעפוע זניח, חץ בשיפוע פעפוע לא זניח)

הספק

<u>הספק</u> => הקצב (ביטים ליחידת זמן) שבו ביטים נשלחים מהשולח ועד הרגע שזה מגיע למקבל.

הספק רגעי = הקצב בנקודה מסוימת בזמן

הספק ממוצע = הקצב לכמות מסוימת ארוכה יותר של זמן

<u>צוואר בקבוק:</u>

נוצר כאשר קצב הזרימה איטי

נגיד יש חלק אחד איטי אז קצב הזרימה הכללי גם יהיה איטי

עומס תעבורה

L= packet length

a= avg packet arrival rate

R = bit transmission rate

$$\text{עומס תעבורה} = \frac{L * a}{R}$$

עומס תעבורה ~ 0 => קצב ממוצע הדיליי קטן

עומס תעבורה = 1 => קצב ממוצע הדיליי גדול

עומס תעבורה > 1 => קצב עבודה שמגיע הוא גדול יותר ממה שאפשר לטפל, עיכוב אינסופי יותר מידי עבודה ואין איך לספק, יותר סיכוי לאיבוד חבילות

אבטחה

וירוס => הדבקה על ידי קובץ המשתכפל בזמן הפעלה(מעורבות המשתמש בלי ידיעתו שזה מה שיקרה)

תולעת => הדבקה על ידי קובץ המשתכפל גם ללא הפעלה(ללא מעורבות המשתמש)

Man in the middle => ההאקר מתחזה גם לשרת וגם ללקוח

Botnet => ההאקר תוקף בעזרת רשת מחשבים אליהם הצליח לפרוץ

Denial of service =>הפצצת מידע ובכך מניעה מהאתר לתת שירות למשל הצפת השרת בבקשות בזמן קצר

Poising => ההאקר מכניס מידע שגוי למערכת

Packey sniffing => ההאקר קורא הודעות בערוצים הלוקאליים

Ip spoofing => התחזות לכתובת IP של מישהו אחר

Ransomware => מתקפות שוחד

Bootkit => יודע לשכפל בעצמו את הקוד של ההפעלה

Rootkit => מקבל הרשאות של השורש ואז ככה ישלו גישה לדברים אישיים

רוב ההתקפות היום מסתמכות על שילוב מגנונים

שכבת האפליקציה

האפליקציות יושבות בקצה הרשת ונמצאות בתחנות שונות ומדברת דרך השרת

דוגמא לאפליקציית רשת: אימייל, רשת חברתית, משחקים וכדומה

<u>גישות עיקריות לבנייה של אפליקציות רשת</u> => מודל שרת לקוח P2P שילוב של שניהם

<u>תפקידים</u> => שירותים למשתמש כגון Web, Email

מציאת כתובת IP עבור אפליקציות

ממשק מול שכבה 4 ע"י סוקטים

תקשורת בין פרוססים מרוחקים

הפרוססים מעבירים הודעות אחד לשני באמצעות שרתי תעבורה

גם הלקוח וגם השרת הם פרוססים

<u>איך מתבצע?</u>

משתמשים בסוקטים שהם כמו דלת כזאת לדוגמא חברת הובלה שמה מחוץ לדלת את

החבילה אך לא דופקת כי אי אפשר לשכבה מעל

הסוקט סוג של מקשר בין שכבת האפליקציה לשכבת התעבורה

לשני הצדדים יש סוקט כלומר לשולח ולמקבל

כל תהליך צריך סוקט כדי לדבר עם תהליך מרוחק ברשת

הכתיבה מתבצעת דרך API

ובדרך כלל הכתיבה והקריאה קשורות לבאפר

<u>כדי ליצור קשר עם process בתחנה מרוחקת צריך:</u>

IP של התחנה מספר הפורט הפרוסס צריך לרוץ

אפליקציה VS פרוטוקולים

אפליקציה היא דרך לממש פרוטוקול

אפליקציה מממשת מה שכתוב בפרוטוקול ויודעת לממש בכל סוגי הדפדפנים\מערכות הפעלה\מחשבים

פרוטוקול מוגדר במסמך RFC ואפליקציה לא באופן מלא

אפליקציה יכולה לממש מספר פרוטוקולים

השירותים שאפליקציה צריכה

אמינות | תזמון | הספק | אבטחה

אמינות => כל המידע שנשלח יגיע ללא שגיאות

תזמון => זמן ההגעה של כל חבילה מובטח(יש חסם עליון)

הספק => קצב השידור מובטח(N ביטים פר שנייה)

אפליקציות שדורשות הספק הן bandwidth-sensitive ואז או שהן קורסות

או שתרד האיכות (פחות חשוב אמינות כי אי אפשר גם וגם)

אפליקציות אלסטיות לא דורשות הספק אך דורשות אמינות

אבטחה => קשה לבצע תקיפות או לגנוב מידע

TCP and UDP

TCP => שנותן לנו אמינות בין השולח למקבל

ונותן עוד דברים שעוזרים לא להציף את תחנת הקצה והרשת= בקרת זרימה ובקרת עומס

אין תזמון, אין אבטחה

בקרת עומס- יכול להגיד ללקוח שכרגע עמוס ואין מענה כלומר מטרתה היא הפחתת העומס

באינטרנט כאשר קיימים שידורים רבים מדי

השיחה מתקיימת על אותו קישור

טוב עבור העברת קבצים

UDP => נותן את המינימום הדרוש לתקשורת בין 2 פרוססים, קל זול

אין אמינות בין השולח למקבל, מעביר את המידע מבלי לוודא כלום

לא ניתן להבטיח הספק ותזמון, אין אבטחה

טוב עבור אפליקציות וידאו

TLS\SSL

פרוטוקול אבטחה בשביל הTCP

ניתנת הודעת אימות שאכן ההודעה הגיע ממי שלח

שומר על שלמות המידע

משתמש באלגוריתמי קידוד על מנת לאבטח את המידע

נמצא כזה בין שכבת האפליקציה לתעבורה, לא בדיוק בשכבה מסויימת

בשבילו יוצרים סוקט ייחודי שהקלט שלו זה הודעה והפלט הודעה מוצפנת

לא מצפינים את כל הheader משום שיש שם את כתובת הip שאנחנו צריכים

מצפינים את כל הbody, הלקוח והשרת צריכים מפתח בשביל זה

מה שנותן את הs לhttps

HTTP

URL => איפה שנמצא מה שאנחנו מחפשים אפשר להגיד גם URI שזה יותר גנרי

HTTP עובד במודל של שרת לקוח

יצירת הקשר בhttp:

הלקוח יוצר קישור tcp מעל פורט 80

השרת מסכים ליצור קשר

אחרי שיש קישור אפשר להתחיל להעביר מידע

ובסוף לסגור את הקישור

<u>Stateless http</u>:

הוא פרוטוקול שלא שומר את ההיסטוריה של הלקוח, מי הלקוח

הפרוטוקול צריך פחות באפרים משום שהוא לא צריך לזכור כלום

היתרון שזה יכול לשרת הרבה לקוחות

Stateful protocols יותר מסובכים בהרבה.

הקישורים שhttp יוצר:

הקישור יכול להיות קצר או ארוך

ויש לזה שתי גישות persistent ו non-presistent שנעבוד איתן עכשו

Non-persistent => יוצרים קישור חדש עבור כל דבר, רק אובייקט אחד עובר כל פעם

Persistent => באותו קישור מעבירים הכל, תופס מקום להרבה זמן

Concurrent => הרבה קישורים באופן מקביל

Pipelined => לשלוח כמה בקשות לפני שאני מקבל תשובות

RTT= round trip time

הזמן שלוקח לאובייקט לטייל בין לקוח לשרת וחזרה

וזה לוקח 2RTT

בקישור לא רציף הזמן הכולל הוא:

2RTT + fille transmission time

http request message:

חשוב לזכור שentity body יש רק בput וpost

Cookies

סוג של שמירת state ללקוח

יש מזהה ייחודי, קובץ שנשמר אצל השרת והלקוח, נמצא בDB של הדפדפן, ללקוח זה נוח

ניתן לפרוץ לשם, צריך header line אצל הלקוח ואצל השרת

כשאני פעם ראשונה פונה לאיזה אתר, האתר מבין שזה לקוח חדש ויוצר עבורו כניסה בdb

ואז המזהה מתקבל מהאתר ונשמר אצל הלקוח.

כל פעם שאני פונה לשרת אז המזהה של העוגייה נשלח גם בבקשה והאתר שמקבל את המספר המזהה ניגש לדאטה בייס ושומר את הדברים הרלוונטיים ובכך יודע לשלוח גם דברים שרלוונטיים לנו

משתמשים בזה בשביל לקבל מידע, המלצות

Web caches

לספק את הבקשות של הלקוח ללא התערבות של השרת המרוחק

אם אני יודע שיש דבר שהרבה אנשים מבקשים אז אני אשמור אותו במקום מסוים לוקאלי במחשב שומרים עותק של אתר ווב

אז נגיד אני מחפשת אתר כלשהו

קודם נבדוק בשרת פרוקסי אם האתר קיים שם

אם כן תחזיר לי אותו אם לא הפרוקסי ילך לשרת ויחפש את האתר הנתון וישמור אצלו בזיכרון ואז יביא ללקוח

הפרוקסי הוא גם שרת וגם לקוח

לכל נתון בפרוקסי יש תאריך תפוגה

אז למה נשתמש בפרוקסי?

מוריד את זמן ההמתנה של הלקוח (פרוקסי יותר קרוב ללקוח)

פחות עומס תעבורה

לפרוקסי דאטה בייס משלו

החיסרון יכול להיות שלפעמים השרתים לא מעודכנים ובנוסף אנשים יכולים להפיץ איזה מידע שגוי

יתרונות = מהיר, מהר לקבל בו תשובה כי המידע עובר מהר יותר ברשת הלוקלית

חוסך עומסים באינטרנט וגם בשרת המקורי, וזול

hit => כשמה שאנחנו מחפשים כן נמצא בדאטה בייס של הפרוקסי

Cache miss => כשמה שאנחנו מחפשים לא נמצא בפרוקסי ואז הוא לוקח את זה מהשרת

Revalidation\freshness check => כל כמה זמן הפרוקסי צריך לבדוק ששמה ששמור אצלו מעודכן

Hit rate => מה הסיכוי שביקשתי משהו שכבר יש

Conditional get

אפשר לבקש עם תנאי של תן לי את האוביקט רק אם הוא התעדכן מאז תאריך מסוים

If modified-since: <date>

אם האתר לא השתנה הוא יעדכן לנו שזה לא השתנה

ואם יש עדכון הוא כן ישלח לי את האתר ונקבל הודעה עם entity body

איך זה מתבצע?

שליחת בקשה לשרת המקומי

↓

השרת בודק אם היה שינוי

אם היה שינוי מחזיר אובייקט מעודכן

אם לא היה שינוי לא מחזיר דבר

- ניתן לעשות חישובים מראש כך שכאשר הלקוח פונה המידע מוכן ומעודכן.
- השרת פרוקסי מקונפג כך שכל זמן איקס מסוים הוא בודק מול השרת האם יש עדכונים.
- Get condition אופייני לפרוקסי.

Http connection types

קישורים לא רציפים:

לכל אובייקט קישור של יצירה ואחד של קבלת האובייקטים

אז הזמן שייקח לי זה מספר האובייקטים כפול RTT2

קישורים רציפים:

מספר הבקשות זה מספר האובייקטים פלוס אחד של היצירת קישור הראשוני

(number of objects +1)RTT

בקשות במקביל:

לא תמיד מגיעות בדיוק באותו זמן

בערך הקמת קישור במקביל פעם אחת RTT אחד

לרוב זה קישורים לא רציפים בבקשות במקביל כי זה יותר קל מאשר לעבוד בקישורים רציפים

M = קשרים מקבילים

$$\sim 2\lceil n/m \rceil \, RTT$$

DNS

שרת DNS נותן מענה של מיפוי השם פלוס IP

מבוסס על כך שקל יותר לזהות שמות מאשר מספרים

כי לא נצפה מאנשים לזכור את כתובת הIP אבל התעבורה באינטרנט מתבסס על הIP ולא

על כתובות ווב(מה שלאדם נוח)

יש פה הפרה במודל השכבות של שכבה 3 ושכבה 5 שעובדות "יחד" כביכול

עדיף שנשים יותר מורכבות בקצה הרשת מאשר באמצע הרשת

כי ככל שהלקוח יותר מתוחכם יש פחות עומס על הליבה ולא יהיה עומס תעבורה

DNS נותן לנו:

מעבר משם לIP

נגיד כתבנו google.com הוא מעביר את זה לwww.google.com

שמות קנוניים = שם האמיתי של האתר

שמות alias = שם בדוי (לא הכתובת המלאה)

מאפשר לפזר גם עומס בין המשתמשים

איך DNS עובד?

רץ הDNS-app client

כותבים כתובת, הדפדפן לוקח את הכתובת שכתבנו ומעביר אותה לdns-client

הdns-client מקבל את הבקשה ושולח אותה לdns-server

ואז הserver מחזיר את הip

וברגע שהדפדפן מקבל את הכתובת הוא כבר יכול ליצור קישור TCP באמצעות שימוש בIP

DNS הוא גם לכתובות אימייל לא רק לווב

מתחלק ל3 קטגוריות: שרתי השורש

Top level domain

Authoritative

שרתי הroot:

בלעדיהם לא יהיה ווב

מנוהלים על ידי כל מיני ארגונים

יש 13 חוות שרתים שכל אחד מהם מכיל עשרות עד מאות שרתים גדולים

כל שרת יודע את הכתובת IP של הroot server

הוא מפנה אותנו לשרת הTLD המתאים.

שרתי הTLD:

אחראים על חלקים מסוימים, שרתי השורש יודעים להפנות לtld המתאים

שרתי מדינות, שרתים של סיומות ספציפיות והוא יודע לפנות לאיזה authoritative המתאים

והוא זה שיתן את התשובה

שרתי הauthoritative:

שרתים ספציפיים לכתובות מסוימות, מחזיקים את הכתובות הכי מעודכנות, מיפוי מעודכן

ויש שם שרת לוקלי שהוא לא חלק מההיררכיה (באחריותו לעשות את הקישור בין הדפדפן לעץ השרתים)

כשאנחנו עושים פנייה לDNS אנחנו לא פונים ישירות אלא אנחנו פונים לשרת הלוקלי (באזור הISP)

כל שאילתה נשלחת לשרת הלוקלי והוא זה שעושה את כל העבודה מאחורי הקלעים כדי להשיג את כל מה שאני רוצה

השרתים האלו בדרך כלל מחזיקים cache

השרת הלוקלי הוא כמו פרוקסי כזה

Routing schemes

Unicast => פנייה למישהו עם כתובת מסוימת, פנייה רק אליו

Broadcast => פנייה לכל מי שנמצא ברשת

Multicast => פנייה לקבוצה

Anycast => פנייה למישהו אבל יש כמה שרתים שעונים על הקריטריון ואז הם יודעים

לכוון אותי לאן צריך

DNS name resolution

איטרטיבית רקורסיבית

<u>איטרטיבית</u> => נגיד אני רוצה לבקש כתובת IP של אתר מסוים אז זה יהיה התהליך:

שולח את הבקשה לשרת הלוקלי

השרת הלוקלי פונה לשרת השורש שאומר לו לאיזה שרת הוא צריך לפנות בTLD

לאחר מכן הTLD מחזיר לו תשובה לאיזה שרת הוא צריך לפנות בauthoritative

ואז הוא מחזיר למשתמש את הIP

השרת DNS הלוקלי אחראי על כל הפניות

אם יש זיכרון בקאש זה יכול לחסוך פניות

<u>רקורסיבית</u> => אני פונה למחשב והוא שולח רק הודעה אחת לשורש ומחכה

אחרי זה השורש הולך לTLD ואז הTLD הולך לשרת הauthoritative ואז הוא מחזיר

תשובה לTLD ומשם לשורש ומשם לשרת הלוקלי ומשם למשתמש

החיסרון פה שזה יוצר עומס על הroot

מבנה הנתונים בDNS:

RR(name, value, type, ttl)

הטיפוס של השם והערך משתנה לפי הטייפ

<u>A</u> => הרקורד הכי בסיסי, כאשר השם הוא כתובת ווב והערך הוא כתובת הIP

<u>NS</u> => השם הוא domain והערך הוא הפנייה שיודעת לתת את הכתובת של הdomain

והוא בא יחד עם רשומה מסוג A כי אנחנו צריכים לדעת כתובת IP בשביל להפנות מישהו

לכתובת מסוימת

<u>CNAME</u> => ספציפי למשהו קנוני, כלומר מישהו פונה לאיקס אבל למעשה הוא צריך לפנות לזד

שזה השם האמיתי

<u>MX</u> => רלוונטי למייל

לכל שאילתה יש מזהה(16 ביט), דגלים

יש שאילתות ומענה

דגלים נועדו להגיד אם יש תשובה או לא, רקורסיבי או לא וממי קיבלנו את התשובה(ממישהו שהדאטה שלו מעודכן או לא)

לשאלה יש אפשרות שנקבל כמה תשובות

Socket programming

<u>UDP</u>

אין קישור בין הלקוח לשרת

אין לחיצת יד לפני ששולחים מידע

לא אומר מה הכתובת מקור שלו

כל פעם שהלקוח שולח משהו הוא כותב לאן הוא שולח ומעביר דרך הסוקט לכל חבילה מחדש

מי שיכתוב את כתובת המקור על החבילה היא מערכת ההפעלה

המקבל יקבל את הIP והפורט והוא יכול לדבר על אותו סוקט עם הרבה לקוחות

הלקוח חייב להתחיל את הקישור וכשהוא עושה את הפנייה השרת כבר צריך לרוץ

והוא צריך לדעת את הIP והפורט של השרת ולהשתמש בהם בכל פנייה

צריכים לדעת באיזה פורט השרת נמצא

הלקוח והשרת יוצרים סוקט(פה לא משנה מי יוצר קודם)

בסוקט יש 2 פרמטרים(השני אומר שמדובר בUDP)

הלקוח יוצר דאטהגרם עם הIP והפורט של השרת שולח הודעה לשרת

השרת קורא את ההודעה עושה עליה עיבוד ושולח ללקוח

הלקוח קורא אותה ומחליט מה הלאה(לסגור קישור, לשלוח עוד משהו)

<u>TCP</u>

כשהשרת פונה ללקוח הוא יוצר סוקט שהוא ספציפי ללקוח ואז כל פעם שהלקוח פונה לשרת הוא

מדבר על אותו קישור\ערוץ

השרת צריך לפעול לפני שהלקוח פונה

השרת בTCP יכול לדבר עם הרבה לקוחות והוא משתמש באותו מספר פורט

השרת מאזין לפורט יוצר סוקט

ופה יש שיטה שנקראת accept והיא יוצרת סוקט חדש שעליו מתנהלת השיחה

לקוח גם יוצר סוקט לפני שמתחיל לדבר והוא מדבר דרך הclientsocket

והשרת מדבר על הסוקט שנוצר ספציפי ללקוח(שנוצר על ידי accept)

אצל השרת לפחות 2 סוקטים אחד שמאזין לשיחה ואחד לבקשות

(וישלהם אותו מספר פורט)

שכבת התעבורה

יוצרת קשר בין פרוססים

בצד השולח => הופך את ההודעות לסגמנטים ומעביר לשכבת הnetwork

בצד המקבל => מתרגם את הסגמנטים להודעות ומעביר לשכבת האפליקציה

נמצא רק בתחנות הקצה בhostים שונים

נותנים אמינות כאשר שכבת האפליקציה לא נותנת אמינות

לגבי מהירות לא (אם זה איטי אז איטי)

Multiplexing and de-multiplexing

שולח עושה multi => לקחת כמה לשים על אותו ערוץ, מוסיפים header

מקבל עושה de-multi => לפזר הודעות, משתמשים בheader כדי לדעת לאיזה סוקט זה מתאים

de-multiplexing

מקבלים חבילה ומסתכלים על הIP והפורט

לכל datagram יש IP מקור ומספר פורט של היעד(כי צריך להבין לאן הולכת החבילה)

ישלה גם סגמנט של שכבת התעבורה

הhost משתמש בכתובת IP ומספר הפורט כדי לדעת לאיזה סוקט צריך לשלוח את הסגמנט

De-multi in tcp

מוגדר על ידי רביעייה: IP של המקור פורט של המקור

IP של היעד פורט של היעד

פה הסוקט מזוהה על ידי כל הרביעייה הזאת וצריכים את כל ה4 כי אנחנו יכולים לדבר עם יותר מאחד.

עקרונות של תעבורה אמינה

הפרוססים מניחים שהם עובדים מעל ערוץ אמין כשבפועל הערוץ לא אמין = בעיה רצינית וחשובה ככל שהערוץ לא אמין ככה נצטרך להתאמץ הרבה יותר

פרוטוקולי RDT

המטרה להעביר נתונים בצורה אמינה כדי ששום דבר לא ילך לאיבוד, שלא ייאבדו חבילות וכדומה

גרסה 1: reliable communication

אוטומט לכל אחד בנפרד

השולח מחכה, מקבל הודעה להעברה, שולח אותה וחוזר לאותו מצב

המקבל ממתין לחבילה, מעביר לשכבת האפליקציה וחוזר לאותו מצב

הצורה הכי פשוטה ולמעשה אם כל הדאטה עובר בצורה תקינה אני לא עושה כלום

Stop and wait

נשלח הודעה ונצפה לשמוע מה שאנחנו רוצים לשמוע

המקבל מאזין לקשר ומקבל בצורה תקינה ונניח אומר קיבלתי שולח ACK ושולח את ההודעה לאפליקציה

השולח לא שולח את ההודעה הבאה לפני שמקבל את האישור מהמקבל

Channel with bit errors and losses

מישהו שלח הודעה ולא הבנו את ההודעה כי ביטים יכולים להשתבש בערוץ

נגיד בעזרת checksum אפשר לבדוק

חבילות גם יכולות ללכת לאיבוד

כל עוד קיבלתי הודעה תקינה אני שולח ACK

ואם לא אני יכול להשתמש בטיימר ואחרי שהזמן הזה נגמר ולא קיבלתי מהמקבל ידיעה שהוא קיבל אני שולח שוב את ההודעה

Reliable stop and wait mechanism

המקבל עדיין לא משתנה, מחכה להודעה ואז שולח את הACK אם הוא מקבל

אצל השולח יש תוספת של הטיימר

נותנים נגיד איקס שניות וברגע שהזמן חולף שולחים מחדש את ההודעה.

השולח הוא היוזם כי הוא זה שיודע מתי להפעיל את הטיימר כי הוא זה ששולח את ההודעה

לכן יש פה יותר אחריות על השולח

Modified sender

מספיק לשלוח מספרים רק של 0 ו1 כדי להבדיל בין הודעות עוקבות

נגיד הודעה 1=1, הודעה 2=0, הודעה 3=1 וכך הלאה (כל עוד שולחים רק הודעה 1 במקביל)

כל state שומר את מה שהוא אמור לקבל כלומר 0\1

לכל הודעה אני מוסיף את המספר 0\1 וזה הsequence number

צריך לבדוק אם הACK שקיבלנו תקין ולא

עקרונית כל state אפשר לשמור בתור משתנה

מחכה להודעה מאינדקס 0 מעביר הלאה ומפעיל את הטיימר, ומחכה לACK

קיבלנו את ACK של אפס שהכל תקין והseq הוא 0

ואז עוברים לחכות להודעה של אינדקס 1 ואותו דבר

(הטיימר הוא רק כשחבילות יכולות ללכת לאיבוד)

תקלות => יכול להיות מצב שבו הטיימר יסתיים

מקבלים הודעה שלא ציפינו לקבל נגיד ציפינו לseq1 וקיבלנו seq 0 או ההפך

אם ההודעה היא לא מה שציפיתי לקבל ולא נגמר הטיימר אני לא עושה כלום

ואז בסוף הטיימר תישלח ההודעה מחדש(יכול להיות שקיבלתי ACK קודם פשוט

המקבל => מזהה שאין כפילויות

יודע שהוא כרגע מחכה ל0 ואם הוא מקבל הודעה עם 0 והיא תקינה הוא שולח ACK

ועובר למצב השני ועכשו כל דבר שהוא יקבל עם אפס הוא יודע שהוא לא תקין

ההודעה הבאה חייבת להיות עם אינדקס 1

יכול להיות מצב שחיכיתי להודעה עם 0 שלחתי ACK עברתי למצב השני אבל השולח לא קיבל

את הACK ואז הוא שוב שולח הודעה אז אני יודע שזה כבר היה זורק אבל שולח שוב ACK 0

וככה מכסים את המקרה שהוא שלח פעמיים ולא קיבל את הACK בפעם הראשונה וזה יכול להמשיך תיאורטית עד אין סוף

U_{sender}: utilization

מתוך סך הזמן שההודעה חזרה, כמה באמת זמן לקח לי לשדר, הזמן שלקח לי להיות פעיל

Pipelined protocols

נשלח הרבה חבילות ונקבל הרבה ACKים וזה ניצול הרבה יותר טוב

אז כדי להגיע לזה צריך לשכלל את הפרוטוקול

ככל ששולחים יותר חבילות מכפילים את ניצולת הערוץ

יש 2 פרוטוקולים: go back n and selective repeat

Go-back-n

יותר חבילות במקביל- ניצולת עולה

<u>אז מה עושים?</u>

נעשה את המינימום ההכרחי, נעקוב אחרי החבילות, יהיה לנו טיימר 1 בלבד

כשהמקבל שולח נגיד ACK מאה זה אומר שהוא קיבל את כל החבילות עד 100

ואם הוא יקבל כל חבילה שהוא לא מצפה לה, הוא יזרוק אותה

השולח צריך רק לזכור

הטיימר הוא על החבילה הכי ישנה ששלח(שלח חבילה 1 2 3 4 מפעיל טיימר על 1)

לשולח יש חלון כזה על האינדקסים של ההודעות

הירוקות אלו הודעות שהוא כבר קיבל אישור

הצהובות אלו ששלח ועוד לא קיבל עליהן ACK

הכחולות הודעות שהוא יכול לשלוח בחלון

ואלה שבצד הודעות עם אינדקסים שעדיין לא הגיעו אליהם

ברגע שאני מקבלת ACK על הודעה חמש

אני יכולה להזיז את החלון ימינה ולשכוח את כל ההודעות שהיו לפני 5

ומפעילים טיימר על הודעה 6 ואפשר לקבל כל דבר שהוא 6 ומעלה

כי כל מה ש5 ומטה אושר לי

אצל המקבל:

אם הוא מקבל הודעה שהוא ציפה לה הוא שולח ACK על ההודעה שהוא ציפה לה

אבל אם הוא קיבל משהו שהוא לא ציפה לו הוא עדיין ישלח ACK על המספר שהוא ציפה עד שהוא יקבל את מה שהוא רוצה וזורק את מה שהוא קיבל

כשהוא מקבל הודעה תקינה הוא מעלה מספר שנקרא rcv_base (ההודעה שעכשו הוא מצפה לה)

תעבורה בTCP

MSS => גודל מקסימלי להודעה (סגמנט=הודעה)

מתייחס לאינדקס של דאטה בתוך הסגמנט

רוצים לעקוב אחרי הדאטה

כי אותנו מעניין מה הביית האחרון שהתקבל

לא מעניין אותנו מה המבנה בתוך הקובץ

רוצים להעביר את כל הבתים כמו שהם

הצד השני יענה עם ACK של האינדקס הבא

נגיד שלחתי עד 1000 יחזור לי ACK של 1001

נגיד גודל הקובץ הוא 500 KB והMSS הוא 1

אז כמה סגמנטים יהיו? 500

נגיד האינדקס הראשון הוא 0

הסגמנט הראשון יכיל 0...999 והseq num=0

הסגמנט השני יכיל 1000....1999 והseq num=1000

וכך הלאה..

אבל בדרך כלל לא נתחיל עם מספר 0 אלא עם מספר אקראי בטווח גדול כדי להוריד סיכוי לכפילויות

Sent_seq# = received ack

Sent_ack# = (seq+ len) above

שכבת הרשת

מדברים על משתמשים לאורך כל הדרך

פה קורה הforwarding והrouting

שכבת הרשת מתחלקת ל2 חלקים: data plane and control plane

מה נמצא בראוטר

IP

Data plane

פונקציונליות מקומית ששייכת רק לראוטר מסוים

דברים יותר לוקאליים כולל forwarding

כשמגיעה חבילה לראוטר מסתכלים על השדות שיש בטבלה ואז מחליטים לאן לשלוח

לפי מה שכתוב זה השלב של הrouting וזה נמצא ב**control plane**.

<u>נראה מה קורה כשחבילה מגיעה לראוטר אחד:</u>

יש routing table מסתכלים על שדות היעד וככה מחליטים לאן לשלוח

וזה קורה בכל הראוטרים שנמצאים לאורך כל הדרך

שכבת הרשת יכולה להבטיח שהdatagram יגיעו בוודאות לאן שצריך או שיגיעו בתוך זמן מסוים

אם אני שולחת הרבה הודעות נרצה לשמור על סדר, לצרוך רוחב פס נמוך יחסית

מה קורה בתוך ראוטר?

יש כניסות (פורטים)

הדאטה נכנס לתוך הראוטר

בפנים יש משהו שיודע למתג את החבילות מהכניסות ליציאות וכל התהליך הזה נקרא

ויש מעבד של הראוטר שיושב בתוך הראוטר

וכשהוא יודע איך הוא צריך לעשות את הforwarding שנצרב לחומרה

כשהתכונה נצרבת אל תוך החומרה אז הפעולות נעשות בננו שניות כי זה בחומרה

יש תורים גם בכניסה וגם ביציאה

Forwarding table

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

יש טווחים של כתובות

אם הודעה בטווח איקס היא תצא באינטרפייס זד ותמיד יש משהו דיפולטי

לפעמים החלוקה של הטווחים היא לא אחידה ויפה ויכולה להיות קצת יותר בעייתית

נגיד תת תחום בטווח מסוים שילך למקום אחר

העיקרון שתקף פה הוא longest prefix matching

יכול להיות מצב שבו יהיו לי כמה התאמות אז נלך למקום שבו ההתאמה הייתה הארוכה ביותר

Head of the line blocking

יכול להיות מצב שאני מחכה למקום והוא פנוי

אך יש מישהו לפניי והוא מחכה שיסיימו לטפל בו והוא חוסם אותי למרות שהיציאה שלי פנויה

שלבים של הforwarding

1.חבילה מגיעה לראוטר

2.הראוטר מוצא את שדות היעד (32 ביט)

3.הראוטר מחפש את הlongest prefix match

4.האינטרפייס שאליו צריך לצאת נלקח מהטבלה

5.הראוטר שם את החבילה בתור של אותו אינטרפייס

6.ברגע שהתור מגיע החבילה יוצאת לשידור

IP fragmentation/reassembly

ללינק יש MTU שזה הגודל המקסימלי שהוא יכול להעביר (הקיבולת שלו)

אז כאשר גודל הדאטה גרם שלנו יותר גדול נצטרך לחלק את החבילה לכמה חבילות

ונבנה את החבילה חזרה רק כשנגיע ליעד

One datagram ----- > many datagrams

DHCP

פרוטוקול שמקבל כתובת באופן דינאמי

נכנסתי לאתר, יצאתי עכשו הכתובת פנויה ומישהו אחר יכול לקבל

יש 4 שלבים:

1.כשhost נכנס לרשת הוא שולח discover

2.עונים לו עם כתובת , מציעים לו כתובת

3.הhost שולח בקשה לכתובת

4.והDHCP עונה לו ACK ואת הכתובת

הוא יכול להיות באותה רשת ויכול להיות גם ברשת אחרת

הוא שולח broadcast לכולם

גם נותן את כתובת הראוטר

יכול לתת לנו מידע של DNS

יכול לתת לנו network mask

Network address translation

הרבה כתובות ביתיות נמצאות בטווח מאוד דומה (כתובות מאוד דומות אחת לשנייה)

הרעיון הוא שיש לי כתובת לוקאלית וכתובת חיצונית

ברשת המקומית נקרא אחד לשני בכינויים וברשת החיצונית לא אדבר כמו עם האנשים שלי

אלא אדבר דרך מתווך וזה הNAT

והוא ישים בכתובת השולח את הכתובת שלו

איך הראוטר ידע ברגע של שליחת הודעה חזרה שההודעה מיועדת אלי ולא למישהו אחר?

הראוטר צריך לשמור מיפוי, לדעת שהוא שלח בשם מישהו איקס

וכשתגיע חבילה חזרה שמיועדת לכתובת של איקס הוא צריך להבין שלאיקס הוא צריך להחזיר את החבילה

לראוטר תהיה טבלה שתשמור את המידע : שלחתי הודעה בשם איקס לכתובת זד

וכשהוא יקבל הודעה חזרה מכתובת זד הוא ישלח לאיקס

הוא שומר את הקישור הזה

ISP ישמור כתובת אחת לכל המחשבים

נותן יתרון של אבטחה כי אין את הכתובת הספציפית שלי

<u>איך זה עובד?</u>

כל פעם שהראוטר שולח הודעה הוא צריך לשמור את הIP והפורט ולאן הוא שולח אותה

וכשחבילה מגיעה חזרה הוא צריך פשוט להפוך את סדר הדברים

<u>בעיית הNAT traversal</u>

לקוח רוצה להתחבר לשרת אבל הכתובת היא של שרת לוקאלית בLAN

דרכי פתרון=>

קונפיג סטטי = כשראוטר מדבר מעל פורט איקס זה תמיד ישלח לשרת זד

אז במקרה שלנו דרך הראוטר תמיד יעבור לשרת 10.0.0.1

UPnP = משפחות של פרוטוקולים שנועדו לקנפג ראוטרים ולשמור את המידע

Relaying = אם הלקוח נמצא מחוץ לרשת ורוצה לדבר עם לקוח שנמצא בתוך הרשת אז השרת יכול לחבר ביניהם, הלקוח שנמצא בתוך הרשת פונה לשרת מסוים שיכול לקשר אותם

IPv6

כתובות גדולות יותרת של 128 ביט

בגרסה הזאת יש דברים חדשים שלא היו בגרסה הקודמת

מכניסים דברים שמאפשרים לטפל בflow- כמו מידע שהוא יותר דחוף, פחות דחוף, להגדיר עדיפות

Icnp אחראי לבדוק על כך שTTL נגמר

Next header = מה הפרוטוקול שנמצא בתוך החבילה

פה אין checksum(הוא מאט את החישובים)

אין פה פרגמנטציה ואין אופציות נוספות שהיו בגרסה הקודמת

Tunneling

מה יקרה כאשר נגיע לאזור מסוים שעובד לפי גרסה 4 ונגיע עם חבילה של גרסה 6?

אז נכניס את החבילה לחבילה של גרסה 4 כדי להצליח לעבור באזור זה

פרוטוקול שעוזר לתחזק דברים שקשורים למסלולים, שגיאות, דיווחי שגיאות ועוד כל מיני דיווחין

נמצא מעל הIP אך עדיין באותה שכבה

Generalized forwarding: match plus action

טבלת forwarding עושה match plus action כלומר מגיעה חבילה, בודקים בטבלה מה מתאים

ומוציאים לפורט המתאים

אך ההתאמה יכולה גם להיות על בסיס דברים שונים ולא רק לפי היעד

כלומר אפשר לבדוק התאמה על פי ערכים כמו מי שלח, באיזה פרוטוקול ועל פי כל דבר שלפיו

אני ארצה להחליט

Match => יכול להיות על המון ערכים בתוך החבילה

Actions => אפשר לזרוק חבילה, להעביר חבילה, לעדכן משהו בheader, לדווח

לשמור כל מיני סטטיסטיקות או להעביר למישהו שנמצא בהיררכיה מעליי

openflow מתבצע על ידי איזשהו controller והוא חיצוני

כלומר admin חיצוני ששולט על כל הרשת מלמעלה

שכבת הLINK

שכבה 2 נמצאת בכל מכשיר והיא בדרך כלל אחראית על התקשורת באותו אזור פיזי באינטרנט

לכל מכשיר יש את הכתובת שלו mac address

נדבר פה על איך חבילה בנויה

ויש פה כמה שיטות להעברת נתונים בצורה אמינה

אפשר להפעיל יותר אלגוריתמים מתוחכמים משום שזה נמצא בתוך החומרה

יש מנגנונים לזיהוי ותיקון שגיאות

ובנוסף יש flow control

זיהוי שגיאות

מוסיפים ביטים נוספים בשביל זה בסוף החבילה

ופה גם יש מנגנון של checksum

EDC

מאפשר לשלוח דרך ערוץ רועש ולנסות לנטרל את רעשי הרקע

שדה EDC ארוך יותר יצליח לגלות יותר שגיאות

Parity bit

ביט זוגיות והוא מסמן 1 כאשר מספר האחדות הוא זוגי

ו0 כאשר מספר האחדות לא זוגי

מסוגל לזהות שגיאה רק של ביט אחד

החיסרון בשיטה זו שיכול להיות שיש לנו מספר שגיאות זוגי וזה לא יגלה לנו איפה בדיוק

אז נהוג יותר לעבוד עם parity bit דו ממדי

parity bit דו ממדי

סורק כל שורה ועמודה מזהה שגיאה ומתקן אותה

מבצעים בדיקת זוגיות לפי קו גובה וקו אנכי

CRC

שיטה שמאפשרת לגלות את כל השגיאות עד אורך R

שולחים הודעה D שמספר הביטים שלה צריך להיות חזקה של 2 אז במידת הצורך נשלים על ידי הוספת אפסים מימין

D זה המידע שאני רוצה לשלוח

ואז לוקחים את G (generator) ועושים XOR

ואז התוצאה נכנסת לי לסוף החבילה

וברגע שהחבילה תגיע לצד השני גם לו יש את D G אז הוא גם יבצע XOR והוא כך יבדוק אם מה שהוא קיבל בחבילה זהה למה שהוא חישב בעצמו

Random access protocols

פרוטוקולים של גישה אקראית

לא מחלקים לא מתאמים, מי שרוצה לשדר אז הוא משדר

בכל פרוטוקול כזה יש מנגנון שמזהה התנגשות ומנגנון ואיך מטפלים בו

משפחת הALOHA

<u>ALOHA- מנגנון לזיהוי התנגשות</u>

כשכל תחנה רוצה לשדר היא משדרת

אם היא מזהה שיש התנגשות היא מחכה ואז מנסה לשדר שוב

אם 2 או יותר מנסים לשדר ויש התנגשות אז הם מחכים ושניהם צריכים לשדר מחדש

כל אחד מחכה זמן אקראי על מנת שלא יתנגשו שוב

(ככל שאני נכשל יותר אני מחכה יותר)

התחנות עצמן מזהות את ההתנגשויות

כשיש לי מסגרת לשידור אני משדר ומתחיל להקשיב

אם אני מזהה התנגשות אני מחכה זמן אקראי ושוב חוזר לשליחה והקשבה

פרוטוקול מאוד פשוט והוא טוב עבור מצב שבו host אחד רוצה לשדר

הוא די מבוזר

יעילות pure aloha=>

לאורך זמן נבדוק כמה מהזמן הצלחתי לשדר

איזה חלק מהזמן באמת אפשר לשדר בממוצע

או איזה חלק מהערוץ אני מנצל

<u>Slotted aloha</u>

יחידות זמן קבועות שאני יכול לשדר וכשאני משדר אז לאורך כל אותה יחידת זמן

כל המסגרות מאותו גודל

וכל אחד משדר מתי שבאלו אבל בתחילת יחידת זמן

כל יחידה יודעת לזהות התנגשות

אם הוא זיהה התנגשות הוא ישדר מחדש בהסתברות P עד שהוא יצליח

ואם אין התנגשות הוא ישדר ביחידת זמן הבאה

משפחת הCSMA

קודם אני אאזין לערוץ ואם אני רואה שהערוץ פנוי אני משדר

אם הוא לא פנוי אני אחכה

אבל זה לא פתרון מושלם כי אם הוא היה פנוי הוא היה פנוי באותו שבריר שניה

ובדיוק ברגע שארצה לשדר יש מישהו שכבר התחיל לשדר את החבילה שלו

אז אני יכול שיהיה פנוי או לחכות זמן אקראי ואז לשדר

<u>כדי לשפר נעשה גם CD</u>

בזמן שאני אשדר אני גם אקשיב כדי לנסות לזהות התנגשות

ואם אזהה שיש התנגשות אז אני יכולה לעצור את השידור

(ההתנגשות מתחילה בזמן פעפוע)

משפחת הtaking turns

מנסה למצוא את הטוב ב2 העולמות

כשיש משתמש אחד אז שהוא יקבל הרבה מהרוחב פס

וכשיש הרבה משתמשים נרצה שהם לא יפריעו אחד לשני

<u>Polling</u>

שואלים כל אחד אם יש משהו להגיד

אפשר על ידי טוקן ואפשר על ידי מאסטר

המאסטר עובר אחד אחד ושואל

ועל ידי הטוקן מעבירים אותו מאחד לשני, וכשיש לך את הטוקן אתה יכול לדבר

אבל מה שכן הוא יכול ללכת לאיבוד

כתובות MAC

כתובות פיזיות, צרובות בכרטיס, לא משתנה לעולם

יותר ארוכה, מכילה 48 ביטים

לכל IP שיושב בLAN יש גם כתובת MAC

ושיהיה ייחודי לאותה רשת

<u>פרוטוקול ARP</u>

יודעת שיש לי חבילה שמגיעה לIP ואני צריכה לגלות לאיזה כתובת פיזית היא צריכה להגיע

יש טבלת ARP ולכל IP יש מיפוי:

< IP address; MAC address; TTL>

הן טבלאות לוקאליות ונבנות בצורה דינאמית

Ethernet

הטכנולוגיה הכי משמעותית בLAN, זולה נפוצה

(מכשירים מהירים ואיטיים יכולים לדבר אחד עם השני)

(אין TTL)

Dest address =>אם זה הMAC שלי אני לוקח אותו, אם זה broadcast גם לוקח

ואם לא שלי זורק

Hubs and switches

המכשירים הפשוטים נקראים repeater and hub

Repeater שומע משהו בצד אחד ושולח לצד השני

Hub יודעת לעשות את מה שהריפיטר עושה בכמה כיוונים

כדי למנוע התנגשויות עדיף להשתמש בhub

לכל אחד כבל משלו ואין התנגשויות על אותו כבל

Hubs

אפשר לקחת כמה כאלה ולעשות מבנה היררכי שלהם

Bridges and switches

מכשירים ברמה 2 שהיום בדרך כלל משתמשים בהם

הם פעילים

גשר => מחבר בין מסגרות(מחבר בין חלקים של אתרנט)

סוויץ => גשר שיש לו כמה כניסות, יותר משוכלל

יודע לעשות גם broadcasting ושולח לכולם חוץ ממי שזה הגיע ממנו

כשהם מקבלים הודעה הם מסתכלים על הכתובת MAC ושולחים רק לפורט שהם יודעים

שהMAC נמצא שם

<u>אז איך זה עובד?</u>

יש switch table ושם כתוב כתובת MAC ובאיזה יציאה הוא נמצא

הסוויץ יודע ללמוד בעצמו

נגיד מקבל הודעה הוא לומד מאיזה host זה הגיע

ולא יודע לאן להעביר

שולח הודעת broadcast מקבל תשובה ומבין לאן להעביר