רשתות תקשורת – סיכום

<u>תמונה כללית</u>:

בהתחלה רשתות היו בעיקר רשתות טלפוניה שעבדו על תדרים אנלוגים, אולם עם הזמן הגיעה המהפכה הדיגטלית, שהפכה כול פיסת מידע לביטים, אותם ניתן לשנע בקלות וביעילות ממקום למקום.

רשת תקשורת צריכה להחליט על כמה מרכיבים עיקריים:

- <u>כתובות:</u> איך מזהים באופן ייחודי לקוח ברשת, למשל בטלפוניה שהן הרשתות הראשונות שהיו, משתמשים במספר טלפון שהוא מחולק לחלקים המייצגים כול אחד תת קבוצה של המספרים. באינטרנט משתמשים בכתובות IP.
- <u>ניתוב:</u> איך גורמים למידע להגיע מצד אחד לצד שני. בטלפוניה עושה Circuit switchin כלומר מייצרים קישור קווי ישיר בין המקור ליעד, על ידי מרכזיות. באינטרנט משתמשים באלגוריתמי ניתוב שעובדים על Routers במהלך הדרך.
- <u>יחידות מידע:</u> איך לחלק את המידע שעובר מצד לצד. בטלפוניה דוגמים את הקול של השולח באינטרוולי זמן קבועים ושולחים אותם אחד אחרי השני, יש חשיבות לסדר! באינטרנט מחלקים את מקטע המידע לבלוקים של מידע הנקראים פקטות, וכול חבילה נשלחת באופן אינדיבידואלי אל היעד.
- <u>שירות:</u> איזה סוג שירות הרשת מספקת ללקוחות שלה. למשל טלפוניה מבטיחה קו תקשורת אמין בין קצה לקצה, אבל זו רשת יקרה להקמה. לעומתה האינטרנט לא מספקת שום הבטחת שירות, אבל זו רשת פשוטה להקמה ולתחזוק.

וגם צריכה לעמוד בכמה עקרונות חשובים:

- פשטות – אין צורך בשינויים ענקיים כדי להרחיב אותה או למזג אותה עם רשתות אחרות
- אוטונומיות – אין צורך לבקש אישור מאף אחד כדי להוסיף רכיב לרשת
- הבטחת Best Effort
- רשת מבוזרת, אין מרכז כוח ושליטה לרשת.

רשתות בעולם צריכות לעבוד לפי סטנדרטים שמכתיבים את כול העקרונות שמופיעים לעיל כדי לתאם ציפיות ברשת ולאפשר מעבר בטוח של המידע בין הלקוחות, באופן שהלקוח המקבל ידע איך לפרש את המידע שקיבל. לשם כך מחלקים את תהליך העברת המידע ברשת לשכבות של פעילות, שכול שכבה מתקשרת עם זו שמעליה וזו שמתחתיה. החלוקה לשכבות פוגעת ביעילות אבל מאפשרת מודולריות ברשת, וורסטיליות ומסדרת את המחשבה באופן שנוח לתפעול. מאפשר לשנות את המימוש של שכבה אחת, מבלי להצטרך לשנות דבר בשאר השכבות.

כול שתי יישויות הנמצאות באותה השכבה אך בצדדים שונים של התקשורת מדברות בינהן בפרוטוקול, ולכן יש פרוטוקולים שונים לכול שכבה של התקשורת.

כול שכבה מספקת שירות לשכבה שמעליה, ומקבלת שירות מהשכבה שמתחתיה. חשוב להבין כי בסופו של דבר כול התהליך של השכבות שקוף עבור המשתמש. הקונספט הוא שפיסת מידע מתחילה בצד השולח, יורדת במעלה השכבות מהעליונה ביותר לתחתונה ביותר, נשלחת לאורך הערוץ, ובצד המקבל היא נפתחת בסדר ההפוך, כלומר מהתחתונה ביותר עד לעליונה ביותר שמקבלת את המידע כפי שהוא נשלח.

מודל 7 השכבות: המודל הנפוץ ביותר למימוש התאוריה שהוסברה לעיל הוא מודל 7 השכבות של ISO. (Physical – Data Link – Network – Transport – (Session – Presentation – Application כאשר את שלוש השכבות העליונות נהוג לחבר לאחת, שכבת האפליקציה.

המודל בשימוש נרחב הוא מודל TCP/IP שנוצר בעקבות הפרקטיקה והוא אכן מכיל רק חמש שכבות, סקירה קצרה שלהן:

- **Physical:** החיבור הפיזי בין רכיבים ברשת. אחראי על העברת הביטים בעולם הפיזי
- **Data link:** מסתיר את המימוש הפיזי של העברה המידע, ומספק ערוץ למעבר מידע בין שתי תחנות סמוכות ברשת. מבודד את הרעש על הכבלים לבעל משמעות, מסמן סוף והתחלה של המידע ומוודא אמינות של המעבר של המידה על הקו.
- **Network:** אחראי על ניתוב המידע ברשת מהמקור ליעד כאשר הם לא סמוכים אחד לשני. מספקת לכול לקוח ברשת כתובת יחודית המזהה אותו.
- **Transport:** מספק אבסטרקציה של מעבר מידע אמין מקצה לקצה של התקשורת תוך הסתרת כול הדרך הארוכה עבור השכבה מעליו. השכבה הזו בא לידי מימוש רק במשתמשי הקצה של הרשת, ומספקת תיקון שגיאות ובקרת עומסים.
- **App:** השכבה של האפליקציה המשתמשת בתקשורת ככלי לפעולתה התקינה, עבורה כול המתרחש מתחת הוא שקוף.

Physical Layer: **(פחות ניגע בקורס הנ"ל)**

מתחלק לשתיים, אות מוכוון ולא מוכוון. כאשר מוכוון הכוונה באות העובר על גבי כבל, למשל על גבי Twisted pair או סיב אופטי או קבל $Co-Axial$. ולא מוכוון הכוונה באל-חוטי כמו רשתות סלולריות, תקשורת לווינית ורשת אלחוטית מקומית כמו Wi-fi.

- Reapeter: רכיב חומרתי בשכבה הזו שמשכפל כול ביט שהוא שומע וככה מעצים את האות ומרחיב את טווח השידור.

Data – link Layer:

מטרת השכבה הזו היא לספק זיהוי ותיקון שגיאות על הקו, אפשור תקשורת מרובה על פני תווך משותף, שליטת עומסים, ויצירת כתובות המאפשרות הכוונה של המידע בתוך רשת מקומית בה כול הלקוחות מחוברים פיזית. שכבה זו מספקת העברת מידע אמינה בין שני צמתים סמוכים ברשת, ופעולת שכבה זו מתבצעת בכול צומת ברשת, כך שיכול להיות שבין כול שתיים סמוכות מתבצע פרוטוקול Data link שונה.

שכבה זו מפרקת את זרם המידע לחתיכות שנקראות Frame.

- **זיהוי שגיאות:** הוספה של ביטים "מיותרים" למידע אותו אנו שולחים כדי לאפשר לצד המקבל לוודא את אמינות המידע שהגיע אליו, תוך הנחה שכמות השגיאות שהתבצעה בתווך חסומה.
  - Parity: למשל ניתן להוסיף ביט בודד שמשלים את הקסור של כול ההודעה לאפס. זה מאפשר לזהות שגיאה בודדת. או אפשר להרחיב את הרעיון לזוגיות דו ממדית, כך שעושים זוגיות לכול שורה ועמודה במטריצה שהיא ההודעה שנשלחה. זה מאפשר לזהות שתי שגיאות ולתקן אחת.
  - CRC: מסכמים מראש על מספר G בעל $r+1$ ביטים, וכול הודעה ששולחים, מוסיפים לה r ביטים כך שלאחר ההוספה ההודעה D משורשרת ל r הביטים הללו תתחלק ללא שארית ב

G. באופן שקול רוצים שאם מחלקים את D ב G מקבלים את הביטים שהוספנו כשארית החלוקה. (לא פשוט לנתח את כמות השגיאות שניתן להזות באופן זה). באופן שקול אפשר לחשוב על מחרוזות בינאריות כפולינומים מעל Z_2, ושני הצדדים מתאמים פולינום יוצר G וכששולח רוצה לשלוח מידע D הוא שולח את $D * x^r + R$ כאשר R זה שארית החלוקה של $x^r D$ ב G (כי קסור צמוד לעצמו מה שישלח יתחלק ב G ללא שארית)

- **Multiple Access:** במקרה של תווך משותף להרבה קישורים שונים, אם שניים ינסו לשדר במקביל תיווצר התנגשות והמידע לא יעבור כמו שצריך. לכן צריך פרוטוקול שמסדיר את התקשורת על פני התווך המשותף, והפרוטוקול עצמו גם כן עובר על פני התווך. נרצה שהפרוטוקול יהיה הוגן(אם N לקוחות משדרים אז כול אחד מקבל $\frac{R}{N}$), יעיל (אם לקוח אחד משדר הוא מקבל R) פשוט ומבוזר.
  - **Channel Partition:** קונספט של חלוקת המשאב לחתיכות קטנות וכול לקוח יקבל חתיכה מהמשאב הזה וכך נחלק את התווך המשותף. שימושי כאשר הקו מאוד עמוס ואז הנצילות מלאה והוגנת. כאשר הקו ריק הנצילות גרועה מאוד.
    - TDMA: חלוקה של הערוץ לפי זמן, כלומר מחלקים את הזמן ל N מקטעים, כאשר כול חלוק לקוח מקבל את המקטע ה N ואם יש לו מה לשדר הוא משדר את זה בתורו. הוגן, אבל לא יעיל כאשר יש כמות נמוכה של לקוחות פעילים, וכמו כן דורש עמדת פיקוד שתחלק את המקטעים.
    - FDMA: חלוקה לפי תדר. כול תחנה ברשת מקבלת טווח תדרים עבורה ברשת, והוא שייך רק לה. אם תחנה לא משדרת אז טווח התדרים הזה יהיה שקט. שוב זה הוגן מאוד, אך לא יעיל אם לא כולם משדרים יחד, ודורש עמדת בקרה.
    - CDMA: לכול תחנה ברשת יהיה קוד יחודי לה, וכול פעם שתרצה לשלוח מידע כלשהו היא קודם תכפיל אותו (מכפלה פנימית) עם הקוד שלה, תשלח ברשת, וכדי לשחזר את המידע מבצעים שוב מכפלה פנימית עם הקוד של התחנה השולחת. כלומר כעת הביטים ברשת יעברו לא כ 0,1 אלא כקידוד מתוחכם C_i עבור 1 ו $-C_i$ עבור אפס (עבור התחנה ה i) אם נבחר את C_i להיות אורתונורמלים אז השידורים השונים לא יפריעו אחד לשני. נשים לב אבל שאם הקודים באמת אורתונורמלים אז לא הרווחנו כלום על פני FDMA/TDMA לכן נשתמש בקודים "כמעט אורתונורמלים" כדי להרוויח ביעילות ולשלם באמינות.
  - **Random Access:** הרעיון הוא שכאשר תחנה רוצה לשדר משהו היא משדרת מיד ובקצב מלא, ואם היא זיהתה שיש התנגשות עם שידור של תחנה אחרת אז הפרוטוקול מסדיר איך מתאמים שליחה מחדש. בגדול אין שום תיאום ראשוני בין התחנות. יהיה טוב כאשר העומס נמוך, ויסבול כאשר העומס גבוהה ויהיו הרבה התנגשויות.
    - Sloted aloha: מחלקים את הזמן למקטעים קבועים, כול מקטע בגודל הזמן שלוקח לשדר חבילה בודדת. כול תחנה שרוצה לשדר משדרת בתחילת המקטע הבא, ואם זיהתה התנגשות היא תנסה לשדר שוב במקטעים הבאים בסיכוי P עד אשר היא מצליחה. נשים לב שכדי שתחנה תשדר היא צריכה שכולם ישתקו ולכן $E[X] = Np(1-p)^{N-1}$ כאשר X זה מספר הפקטות ששודרו בהצלחה במרווח זמן בודד, N מספר התחנות. לכן אם בוחרים את האופטימום להיות $p = \frac{1}{N}$ (לא ריאלסטי, כי זה דורש מראש לדעת כמה הולכות לשדר) אז הנצילות של הערוץ תהיה בערך $\frac{1}{e}$ (לא מדהים...). בעולם ריאליסטי p מקובע ו N משתנה, וככול ש N גדל, ו NP עובר את 1 כך הנצילות יורדת, ונכנסים למעגל אינסופי של דעיכת הנצילות לאפס, כי פקטות לא מצליחות לעבור, לכן יש יותר תחנות שרוצות לשדר, ולכן הסיכוי להצליח לשדר קטן ולכן התור מצטבר עוד ועוד עד שהנצילות הופכת לאפס.

- Pure Aloha: בדומה ל sloted aloha ברגע שיש לתחנה מידע לשדר היא פשוט ישר משדרת אותו, ואם במהלך השידור היא זיהתה התנגשות היא תנסה לשדר אותו (במרווחי זמן של שידור פקטה) בעתיד בסיכוי p. היתרון הוא שלא צריך לסנכרן שעון בין התחנות, וכמו כן עם הקו ריק ורק לקוח אחר משדר אז הוא מקבל נצילות מלאה. החיסרון הוא שיהיו יותר התנגשויות כי כעת ברגע שתחנה התחילה לשדר היא יכולה להתנגש עם מישהו שהתחיל לשדר אחריה. האינטרוול שבו פקטה מתנגשת גדל פי שתיים ולכן הנצילות קטנה פי שתיים.
- CSMA(Carrier Sense Multiple Access): לפני שתחנה משדרת את המידע שלה היא תחילה בודקת האם הקו ריק, אם הוא ריק היא תתחיל לשדר, אם לא היא תחכה שהוא יתפנה ואז תשדר את המידע שלה (או מיד לאחר שהקו התפנה, או לאחר אינטרוול זמן שנבחר באקראי). נשים לב שעדיין יכולות להיות התנגשויות מסיבות של Propagation delay, כלומר לוקח זמן למידע להגיע מתחנה לתחנה ויכול להיות ששתיים החלו לשדר ביחד. הסיכוי להתנגשות מושפע מהזמן שלוקח למידע לעבור בין התחנות. נשים לב שהתנגשות היא בזבוז של כול זמן השידור של הפקטה.
- CSMA/CD: כמו לעיל, אולם אם באמצע שידור מידע נתגלתה התנגשות השולח לא מסיים את השליחה אלא קוטע אותה באמצע כדי לא לבזבז זמן. (בעייתי לביצוע ב wireless כי אנטנות לא יכולות גם לשלוח וגם להאזין במקביל). לאחר זיהוי התנגשות השולח יחכה $2\mathrm{RTT}$ לפני שינסה שוב כדי לוודא שהרשת ריקה. כאשר p אופטימלי, כלומר $p=\frac{1}{N}$ אז הסיכוי להצלחה בסלוט בודד היא $\frac{1}{e}$ ולכן לאחר בתוחלת e נסיונות תצליח לעבור פקטה ברשת. לכן נצילות הרשת, שזה הזמן שבו שידרתי מתוך סך כול הזמן היא $\frac{\frac{P}{C}}{\frac{P}{C}+2eT}=\frac{1}{1+\frac{2eTC}{P}}$ כאשר P זה גודל פקטה ברשת ו C זה הקיבול המירבי של הרשת. את $\frac{TC}{P}$ נסמן ב α ונרצה להקטין ביטוי זה, אבל לא נרצה לפגוע ב C, ואת T לא ניתן ממש לשנות, לכן נגדיל את P. פרוטוקול זה מספק נצילות מלאה ללקוח בודד, הוגנת לא בטוח, זה תלוי בזמן שלוקח לזהות התנגשות, והוא מבוזר לחלוטין.
  - נשים לב שחשוב מאוד שהזמן שלוקח לשדר פקטה בודדת הוא גדול מ RTT כדי שאם התחלתי לשדר ומישהו התחיל לשדר ממש קצת לפני שהשידור שלי הגיע אליו, אז עד שהשידור שלו יגיע אלי יעבור RTT מאז שהתחלתי לשדר, ולכן חשוב שאני עדיין אהיה בשידור כדי לזהות את ההתנגשות.
- **Taking Turns:** נסיון להיות יעילים גם בעומסים גבוהים וגם בנמוכים. הרעיון הכללי הוא לקיחת תורת בשידור, אך רק צמתים שרוצים לשדר משתתפים בתור הנ"ל.
- **Polling:** יש צומת מאסטר שמזמין תחנות כשתורן לשדר, ותחנה לא יכול לשדר ללא אישור של המסטר. בעיה כי מכיל SPOF וגם יש overhead של לשאול מי רוצה לשדר.
- **Token Passing:** יש מעין Token שרק לתחנה שבידה הוא נמצא יש אישור לשדר ברשת, והוא עובר במעגל בין התחנות ומי שלא רוצה לשדר מעבירה אותו הלאה. שוב מכיל SPOF וזה ה Token, וגם יש תשלום על טיפול בכול העניין.
- **Reservation:** לפני כול סבב שליחת מידע יש סבב הזמנת מקום, שבו עובר וקטור ביטים ברשת וכול אחד מדליק את הביט המתאים לו במידה והוא רוצה לשדר. בסבב שיבוא לאחר מכן כול תחנה תשדר לפי הסדר כשיגיע תורה, וככה כולם יודעים את הסדר של מי משדר לפניהם ומתי תורם לשדר.

- ***LAN:*** רשת אזורית, מקומית, כך שכול הרכיבים ברשת מחוברים פיזית בניהם.
  - בתוך רשת כזו מזהים את הלקוחות לפי כתובת MAC שזה מספר 48 ביטים שצרוב על גבי המכשיר. (כיום אין שני מכשירים בעולם בעלי אותו MAC, למרות שאין בכך צורך, כול מה שחשוב הוא ייחודיות בתוך הרשת המקומית) (למבנה של הכתובת MAC אין משמעות, בניגוד ל IP, זה פשוט מספר)

- ARP: פרוטוקול בתוך הרשת המקומית שמקשר בין כתובות IP לבין כתובות MAC. פשוט כול צומת ברשת מחזיקה טבלת המרה מIP ל MAC, והשאילתות נעשות ב Broadcat, והיעד, כלומר מי שה IP הזה שייך לו עונה ב Unicast לשואל.
- פרוטוקול Ethernet: שולט כיום בשוק ה LAN משום היותו זול. פקטות הן מהצורה $Preamable \mid Dest\ addr \mid Source\ add \mid type \mid Data \mid CRC$ כאשר החלק הראשון הוא נועד לסנכרון השעונים בשני הצדדים, Type זה פרטוקול שרץ מעל, למשל IP, ו CRC זה קוד תיקון שגיאות שראינו. כמו כן הכתובות הן כתובות MAC. כמו כן אלגוריתם זה מבצע CSMA/CD כמו שראינו לעיל, כך שבמקרה שהתגלתה התנגשות השולח מזין Jam signal לרשת כדי לוודא שכולם זיהו את ההתנגשות, ולאחר מכן מבצע Exponential backoff, שזה אומר שזמן ההמתנה לפני נסיון שליחה חדש הוא נבחר באקראי מתוך 2^n כאשר n זה מספר ההתנגשויות הרצופות שהיו עד עכשיו, כאשר 1024 הינו חסם עליון, והזמן שהשולח ימתין הוא המספר שנבחר באקראי * 512 זמן שליחת ביט בודד. כמו כן הפרוטוקול מחליט שהקו שקט אם לאחר 96 מרווחי ביט היה שקט על הקו.

- ***Hubs/Bridges/Switches:*** לא נרצה שכול האינטרנט יהיה LAN אחד גדול, כי מרחב ההתנגשויות יהיה פשוט גדול מדיי, הדיליי בהודעות יהיה עצום, וזה ידרוש מכולם להחזיק טבלת MAC לכול אחד מהמשתמשים באינטרנט! לכן נרצה לפרק את הרשת להרבה LAN מקומיים, שכול אחד מהם יהיה בעל Collision domain משלו.
  - ***HUB:*** מכשיר שלמעשה עובד בשכבת הפיזיקלית, פשוט משכפל את הביטים כל הקו ומגביר אותם. משמש להגדלת הטווח של רשת מקומית, אבל לא עוזר להגדיר את ה collision domain, ולא עוזר להגדיל את הקצב ברשת, למעשה כשמחברים שתי רשתות ב HUB התפוקה של שתיהן יחד היא כמו התפוקה של אחת! שקול למעשה לחיבור קווי ישיר, עד כדי הארכת הטווח. כמו כן לא מאפשר לחבר שתי רשתות בקצבים שונים כי הוא פשוט מעתיק את מה שהוא קולט. היתרון הוא שהוא פשוט וזול ואינו מהווה SPOF, אם הוא קורס כול חלק של הרשת יכול להמשיך בנפרד.
  - ***Bridge:*** רכיב מתוחכם יותר היושב בשכבת ה Data Link. עובד על פקטות שלמות של Ethernet ולא על ביטים בודדים. הוא למעשה מהווה חוצץ בין שתי הרשתות ובורר אילו חבילות עוברות בין הצדדים. במידה ויש צורך להעביר חבילה לצד השני אז הוא משדר את החבילה בצד השני בהתאם לחוקי ה $CSMA/CD$ ומהווה שחקן לכול דבר ברשת (אבל מבחינת שאר הצמתים זה שקוף, אין לו MAC משלו). מאפשר לבודד שני $Collision\ Domain$ ומאפשר תפוקה כפולה, כלומר שתי שיחות יכולות להתקיים במקביל במידה והן בשני צדדים שונים של הגשר, והוא לא יעביר פקטות לצד השני אם הן לא מיועדות לשם. כמו כן מאפשר לחבר רשתות בקצבים שונים, כי בכול צד הוא משחק את משחק הרשת המתאים.
    - ***סינון החבילות:*** איך הגשר ידע בהנתן חבילה האם להעביר אותה לצד השני או לא? (לא בהכרח רק שני LANs מחוברים אליו, יכולים להיות הרבה). כול פעם שחבילה מגיעה לגשר מיעד מסוים דרך רגל כלשהי הוא לומד על כך ומוסיף לטבלת הסינון שלו רשומה שמציינת את כתובת ה MAC של הלקוח ששלח הודעה זו, והרגל שדרכה ניתן להגיע אליו, בתוספת TTL. אם הגיעה פקטה לשליחה שהגשר עוד לא יודע את המיקום המדויק של היעד שלה, הוא פשוט יעשה flood וישלח לכול הרגלים שלו במקביל(חוץ מלרגל שממנה הגיעה ההודעה)
    - ***בעיות באלגוריתם הלמידה:*** כדי שאלגוריתם הלמידה יעבוד אנחנו צריכים שהרשת תהיה עץ, כלומר ללא מעגלים כי אחרת אלגוריתם הלמידה פשוט יטעה. אבל עץ זה רע, כי כול צומת מלבד העלים הוא SPOF לניתוק הרשת, לכן דווקא כן נרצה יתירות ומעגלים ברמה הפיזית אבל עץ ברמה הלוגית.
    - ***STP:*** אלגוריתם מבוזר לבניית עץ פורש לוגי לרשת שיש בה מעגלים, כאשר מטרתנו היא לבנות עץ פורש ל LAN Segments, אך למעשה אנחנו בונים עץ פורש ל Bridges.

- באלגוריתם הצמתים מנסים יחד למצוא שורש לעץ אשר יהיה הצומת עם המזהה הנמוך ביותר (לכול Bridge ברשת יש מזהה יחודי), כאשר כול גשר תחילה מניח שהוא שורש וכול פעם מעדכן את השורש להיות הצומת בעל המזהה הנמוך ביותר שהוא שמע עליו עד עכשיו, וכול כמה זמן מודיע לכול שכניו מי הוא השורש עבורו.
- כמו כן מבצעים מעין Bellman ford מבוזר. כול גשר מחפש את המסלול הקצר ביותר ממנו לשורש (מי שהוא חושב שהוא השורש) על ידי המידע משכניו, ובסופו של דבר נקבל עץ מסלולים קצרים ביותר ובפרט עץ. (זה לא צריך להיות סינכרוני, בסוף זה יתכנס). לכן הצלחנו ליצור עץ פורש לגשרים.
- עכשיו נייצר עץ פורש ל LAN Segments כך שבכול LAN Segment רק גשר אחד יעביר החוצה את ההודעות ממנו, והגשר שיבחר הוא זה שקרוב ביותר לשורש(במקרה של תיקו יבחרו לפי מזהה מינמלי). כול שאר הגשרים על אותו ה Segment יוותרו על הזכות להעביר את הפקטות, והם ידעו זאת כי הם ישמעו הודעה מהגשר המנצח שהוא קרוב יותר לשורש מהם. לכן לכול LAN יהיה פורט יציאה יחיד שיקרא Designated port. נשים לב שאת המרחקים על הקשתות ניתן לבחור לפי עלות/קיבולת/ קצב/מרחק וכו'.
- לכול גשר ברשת שיודע שהוא אינו השורש יהיה פורט שיקרא Root port וזה הפורט שדרכו הגשר מסוגל להגיע לשורש במסלול הקצר ביותר (במקרה של שוויון בוררים לפי מזהה נמוך יותר)
- לאחר סיום הליך הבנייה רק ה Designated port וה Root Port של כול גשר נשארים פעילים להעברת תקשורת, וכול שאר הפורטים רדומים לוגית(ומעבירים רק הודעות BDPU) ויתעוררו רק במקרה של תקלה. (נשים לב שלאחר התייצבות האלגוריתם אין באמת חשיבות להאם פורט הוא Designated או Root מבחינת העברת הפקטות)
- כאשר צומת מקבל על עצמו שורש אחר, הוא מפסיק לשדר הודעות BDPU משל עצמו, ורק מעביר את אלה שמגיעות מהשורש שאותו הוא קיבל, או אם מגיעה הודעה טובה יותר.
- אם מתישהו השורש מת, הצמתים יזהו את זה ויתחילו לבצע האלגוריתם מחדש וימצאו שורש חדש. כול עוד הרשת בהתייצבות יכולים להיות נתקים, העיקר שלא יהיה לולאות.

- לסיכום: גשר שימושי ברשתות קטנות, הוא פשוט וקל למימוש, דורש מעט כוח חישוב, אבל מכריח את הרשת להיות בטופולוגית עץ, ולא מספק שום הגנות לרשת עליה הוא רץ.

- **Ethernet Switch**: מכשיר שמייצר ארכטיקטורת כוכב ברשת ומאפשר מספר שיחות רב במקביל בין כול זוג מהרגלים שלו מבלי ליצור התנגשות, למרות שהצמתים עצמם חושבים שהם כולם נמצאים על אותו הקו, כמו שethernet מצפה, ולמעשה משחקים ב CSMA/CD למרות שאין התנגשויות. מאפשר לחבר מכשירים מכול מיני קצבים שונים.
- **רשת אלחוטית:** בדרך כלל ברשת אלחוטית יש מנהלים של הרשת שנקראים AP ולקוחות ברשת שנקראים Hosts. יש כול מיני פרוטוקולים שמסדרים את ה MAC ברשתות כאלה, אבל הבעיה היא שכשעושים CSMA/CD יכול להיות ששני Hosts לא שומעים אחד את השני בגלל מחסומים פיזים כמו קירות/ הרים וכו'. לכן מממשים CSMA/CA שהו כול לקוח שרוצה לתקשר עם AP צריך תחילה לבקש אישור לתקשר, ואם ה AP אישר לו הוא מתחיל תקשורת וה AP אחראי להודיע לכול התחנות הנסתרות שכרגע הוא עסוק ושלא ישלחו ויפריעו לו. נשים לב שעדין יש מלחמה על להעביר את פקטות ברשת התקשורת ל AP ולכן צריך שגודל פקטות המידע יהיה גדול בהרבה מגודל חבילות המלחמה על אישור ה AP.

באותה מידה אפשר להוריד הצורך ב AP מרכזי, ופשוט כול צומת שירצו לפנות אליו יממש את המנגנון הזה עבור שיחות נכנסות אליו.

- **Point to Point:** מודל תקשורת של שולח אחד ומקבל אחד, אין צורך ב MAC כי אין התנגשויות בכלל (למשל חייגן). דורשים יכולת לזיהוי שגיאות, יכולת לזיהוי חיות הקישור, ויכולת לשדר כול רצף ביטים כ Data מבלי להתבלבל עם ביטים של הפרוטוקול. לא דורשים היכולת לתקן שגיאות, לא דורשים שליטה בקצב ובזרם ולא דורשים לתמוך בריבוי משתמשים, כמו כן לא דורשים סדר לחבילות, כול הפעולות הללו מועברות לשכבות למעלה.
  - **Bit transparency:** היכולת לשלוח כול רצף ביטים. בפרוטוקול יש רצף ביטים שמור שמייצג דגל כלשהו, ולכן כדי שיהיה ניתן לשדר רצף זה גם כData בלי להתבלבל, נבחר רצף ביטים כלשהו אחר שייצג שהרצף הבא שמגיע הינו רצף Data וככה נוכל לשים את הרצף הזה לפני רצפים שנרצה שיחשבו כ Data ולא כסמנים לפרוטוקול (כמו \ בשפות תכנות)

- ***Scheduling:*** התרחיש הוא מכשיר שתפקידו לקבל אליו הרבה חבילות מהרבה מקורות שונים ולנתב אותן לזרמי הפלט הרצויים, תוך שהוא מבצע את החלטות הניתוב הללו בזמנים מינמלים, מספק הוגנות לכול הזרמים השונים שנכנסים אליו, ומספק הגנה לזרמים קטנים מפני זרמים שחונקים את התקשורת, ומביא למינימום את זמן ההמתנה של חבילה בתור (Buffer). (לא ברור איך מגדירים הוגנות, ולרוב היא באה על חשבון יעילות!)
  - ***Max Min Fairness:*** המטרה היא לחלק באופן שווה בשווה את המשאבים כך שאף אחד לא מקבל יותר ממה שהוא ביקש, ואת העודף מחלקים באופן שווה. מביאים למקסימום את הערך המינמלי שקיבל קישור אשר לא קיבל את מה שהוא ביקש מהרשת. אפשר לחשוב על זה כעל כלי מים שממלאים אותם בנוזל (רוחב הפס), כאשר אחד מתמלא מים מפסיקים לזרום אליו.
  - ***FIFO:*** מימוש נאיבי לחלוטין שבו החבילות מטופלות לפי סדר הגעתן. לא מספק הגנה לזרמים חלשים, לא מספק הוגנות וחשוף להתקפות DOS. אם הבאפר התמלא פשוט זורקים חבילות שמגיעות.
  - ***Priority Queuing:*** כול חבילה שמגיעה מקבל מספר המייצג את העדיפות שלה. חבילות נשלחות לפי סדר העדיפויות שלהן. חבילה מעדיפות i תשלח רק אם כול הזרמים של עדיפויות קטנות מ i ריקים כולם. זה יוצר קצב גבוה מאוד עבור חבילות מעדיפות גבוהה, ויכול ליצור הרעבה של חבילות מעדיפות נמוכנה. (בתוך אותה העדיפות חבילות מטופלות ב FIFO). למעשה יש תור נפרד לכול עדיפות.
  - ***Round Robin:*** יהיו מספר רב של תורים, כאשר המכשיר משרת אותם באופן סדרתי. כלומר כאשר הוא מסיים לטפל בטור מסוים, כלומר הוא שלח את הפקטה הראשונה בטור הזה, אז הוא יעבור לתור הבא שאינו ריק בסדר המעגלי ויטפל בו. זה מספק הגנה לזרמים קטנים מפני זרמים גדולים, ומספק הוגנות. נשים לב אבל שהוא לא מתייחס לעדיפויות של התורים, ולא לאורך הפקטות. (כמו כן דורש מהחומרה להיות מסוגלת לקפוץ לתור הלא ריק הבא)
  - ***Weighted Round Robin:*** כמו לעיל רק שלכול תור יש משקל שמציין כמה פקטות ישלחו מהתור הזה כול פעם ש RR יגיע לטפל בו. זה מאפשר לתת עדיפויות לתורים, אבל נשים לב שההוגנות בתהליך באה לידי ביטוי רק לאורך פרק זמן מתמשך, ועבור פרקי זמן קצרים מאורך סיבוב (שכעת אורך סיבוב הוא סכום המשקלים) אין הוגנות. כמו כן עדיין לא פותר את בעיית אורך הפקטות.
  - ***Deficit Round Robin:*** מבצע RR תוך התחשבות בגודל של החבילות. לכול תור יהיה מונה שיספור כמה קרדיט התור הזה צבר. בכול סבב של ה RR כשהוא בא לטפל בתור מסוים הוא מוסיף לו עוד קרדיט למונה שלו. אם הקרדיט עבר את גודל החבילה הראשונה בתור אז שולחים אותה ומפחיתים מהקרדיט את גודלה, אם עוד לא עבר אז התור הזה יצטרף

לחכות לסבב הבא של RR. כמו כן חשוב מאוד שתורים שאין להם מה לשלוח לא צוברים קרדיט כך סתם, ובאופן כללי לתור מסוים לא יכול להיות יותר קרדיט מכמה שעוד נשאר לו לשלוח, כדי למנוע כול מיני מתקפות. בנוסף זה קל מאוד למימוש, והוגן יותר מ RR משום שמתחשב תכלס בכמה מידע עבר ברשת ולא רק כמה פקטות.

- ***GPS:*** מודל תיאורטי אידאלי שמספק הוגנות מקסימלית, אך אינו ישים בעולם האמיתי כי הוא מניח שניתן לשלוח כמות אינפיניטסימלית קטנה של מידע. הרעיון הוא שבכול רגע נתון מחלקים הקו שווה בשווה (או אולי באופן ממושקל) בין כול הלקוחות שרוצים לשדר במקביל. תמיד מחלקים את מלוא רחב הפס האפשרי. החלק שכול לקוח מקבל ברגע t הוא החלק היחסי של המשקל שלו מתוך סך כול המשקולות של לקוחות שרוצים לשדר באותו זמן כמוהו. נשים לב שזה הדרך הכי הוגנת שקיימת, וכול פרוטוקול אחר נמדד בהוגנות שלו לפי היחס שלו ל GPS. (נשים לב ש GPS זה למעשה Max-Min-Fairness אופטימלי! כול מי שלא משדר כעת לוקחים את המשאבים שלו ומחלקים שווה בשווה בין כולם)
- ***השוואה ל GPS:*** לכול פרוטוקול A נגדיר $Work_A(i,a,b)$ להיות מספר הביטים שנשלחו מתור i בזמן $[a,b]$ (בהנחה שתמיד יש לתור הזה מה לשלוח).
  - ***Absolute fairness:*** עבור פרוטוקול A זה מוגדר להיות $Max_i(|Work_A(i,a,b) - Work_{GPS}(i,a,b)|)$ ההפרש המקסימלי בין מה שלקוח קיבל לבין מה שהיה מקבל ב GPS.
  - ***Relative fairness:*** ההפרש המקסימלי בין מה ששני לקוחות שונים קיבלו בפרוטוקול במקטע זמן מסוים.
- ***WFQ***: נסיון לבצע מימוש יעיל ל GPS עד כמה שאפשר בעולם האמיתי. הרעיון הוא לבצע אימולציה של GPS ואז לטפל בפקטות לפי הסדר שבו היו מסיימות להישלח ב GPS.(כאשר חבילה מסוימת נמצאת בטיפול היא מקבלת את מלוא רוחב הפס). (נחזיק את הפקטות בתור עדיפויות לפי זמני הסיום) עולה השאלה איך עושים אימולציה?
  - אימולציה של ***FIFO***: לכול פקטה שמגיעה ניתן לה משתנה זמן T כלשהו, ולפקטה הבאה ניתן את הערך הקודם של T ועוד הגודל שלה, ואז נשדר אותן לפי הסדר.
  - אימולציה של ***RR***: לכול תור יהיה משתנה משלו, וכול פקטה שתגיע לתור מסוים מעלה באחד את המשתנה של התור, וזמן השליחה שלה יהיה הערך החדש של המשתנה. (אם התור ריק אז היא תקבל את ערך הסיבוב הנוכחי בסימלוץ+1).
  - אימולציה של ***GPS***: כול חבילה שמגיעה, זמן הסיום שנקצה לה הוא הגודל שלה(חלקי המשקל של התור הזה במקרה הממושקל) ועוד הזמן האחרון של פקטה שלפניה באותו התור, או אם התור ריק אז זמן הסיבוב הנוכחי באימולציה. נשים לב שבמקום לשנות את זמן הסיום של פקטות ישנות כול פעם שפקטות חדשות מגיעות (כי בGPS הקצב שבו פקטות נשלחות תלוי במספר הקווים המשדרים כרגע) אז פשוט נשנה את הקצב שבו "הזמן" מתקדם. הקצב שבו נקדם את מונה ה ROUND יהיה ביחס הפוך למספר החיבורים הפעילים כרגע(כלומר $Round(T+x) = Round(T) + x/B(T)$ כאשר B זה סכום משקלי הלקוחות המשדרים כרגע, וכאשר בין זמן T לזמן $T+x$ מספר הלקוחות ומשקלם לא השתנה.
  - ***WFQ***: לאחר שהצלחנו לעשות אימולציה בזמן ריצה ל ***GPS*** אז הפרוטוקול ***WFQ*** תמיד ישדר את הפקטה בעלת זמן הסיום הנמוך ביותר שידוע כרגע. (וכמובן ממשיך לסמלץ את ***GPS*** ברקע)
  - טיב של ***WFQ***: זמן הסיום של פקטות ב WFQ גדול יותר מזמן הסיום שלהן ב GPS בלכול היותר זמן שידור של פקטה יחידה. (וברוב המקים זה אפילו יהיה טוב יותר) (המקרה הגרוע קורה כאשר חבילה קטנה מגיעה באמצע טיפול בחבילה גדולה)
- ***WFFQ:*** בא לתקן הבעיה של הגונות יחסית שיש ב WFQ. נשים לב כי ב WFQ יש תורים שמרוויחים המון, בזמן שתורים אחרים מרוויחים מעט מאוד, אם בכלל. לכן ב WFFQ נדרוש כמו ב WFQ שתמיד נשרת את הפקטה שהייתה מסיימת ראשונה ב GPS אבל לא נתחיל

לטפל בפקטה לפני ש GPS היה מטפל בה. כלומר אם סיימנו לשדר פקטה מתור מסוים, לא נתחיל לשדר את הפקטה הבאה מהתור הזה, לפני שבGPS שאותו אנו מסמלצים הפקטה הזו הייתה מתחילה שידור.

- ***Multiple Buffers:*** הסיטואציה היא רכיב שצריך לנתב מידע, יש לו הרבה תורי כניסה והרבה תורי יציאה והוא צריך להעביר הפקטות מתורי הכניסה לתורי היציאה המתאימים. הרכיב הזה צריך לעבוד פי N יותר מהר מהקצב ברשת, כאשר N זה מספר התורים שנכנסים אליו.
  - ***Head Of Line Blocking:*** בעיה שיכולה להווצר במקרה זה היא שיש חבילות שמיועדות ליעד א' אבל הן בתור אחרי חבילות שמיועדות ליעד ב' כאשר יעד ב' מאוד עמוס ואילו יעד א' פנוי לחלוטין. לכן החבילות ליעד א' יצטרכו לחכות לתורן, למרות שלכאורה הדרך עבורן פנויה.
  - ***Look aHead:*** פתרון אפשרי הוא שהרכיב יסתכל לעומק התורים שלו וינסה לחלץ פקטות שתקועות ב Head Of Line Blocking. כמובן שזה דורש הרבה כוח חישוב מצד הרכיב. זה כמו לעשות באפרים וירטואלים, כך שלכול תור כניסה יהיה באפר (וירטואלי) עבור כול תור יציאה, וכול באפר יתקדם בקבצו שלו, באופן בלתי תלוי באחרים, למרות שלמעשה כולם יושבים על אותו הבאפר.
  - ***OutPut buffer:*** הפקטות יחכו בקווי היציאה ולא בקווי הכניסה, כלומר כאשר פקטות יגיעו לרכיב הן ישר ינווטו לקו היציאה המתאים להן, ואת ההמתנה יבצעו שם, זה פותר לחלוטין את HoL אבל מצריך את תור היציאה להיות מהיר לפחות כמו כול תורי הכניסה יחד כדי לעמוד בעומס.
- ***Switching:*** מדובר ביחידות שיושבות ברמת ה Data Link ותפקידן להוביל מידע שנכנס אליהן, אל עבר היעד שאליו הוא מיועד, רכיבים כאלה ישבו גם בתוך ראוטרים ברמה השלישית. המטרה העיקרית שלנו היא למקסם את הקצב שבו רכיב כזה מסוגל להוציא ממנו את החבילות שנכנסות אליו, כמו כן נרצה להביא למינימום את איבוד המידע. יש שני סוגים של פרדיגמות ל Switchin, פקטות ו Virtual connection.
  - ***Virtual Connection/Circuit Switch:*** ניתוב מבוסס על חיבור, הדורש הקמה של חיבור לפני שהשיחה יכולה להתחיל, והתחייבות על הנתיב הזה עבור השיחה. לאחר הקמת השיחה החבילות עצמן לא צריכות לדעת את היעד הסופי שלהן, רק את מזהה השיחה שלהן כאשר הן עוברות בין כול שני Switch. נשים לב שבשלב הקמת השיחה כול הרכיבים בדרך צריכים להתחייב על הקצאת רוחב פס לשיחה הזו, ואם יש רכיב בדרך שאין לו מקום אז השיחה לא תיווצר! נרצה להביא למינימום את המקרים הללו.
  - ***Packet Switching:*** המידע נשלח ליעד בחבילות מבודדות, שכול אחת יכולה להגיע ליעד שלה בנתיב אחר ברשת. זה לא דורש לייצר קישור בין הקצוות בתחילת השיחה. כמו כן זה ידרוש מהחבילות להכיל את כתובת היעד בתוכן, וזה ידרוש מSwitchs בדרך להכיל באפרים שישמרו את המידע שהגיע, כי הם לא התחייבו מראש שהם מסוגלים לטפל בזרם הנכנס, ולא נרצה שהם יזרקו את הפקטות אם הם עמוסים.

Network Layer:

תפקידה של שכבה זו הוא לנתב ולהכוון את המידע ברשת מהמקור שלו אל יעדו, על גבי רשת האינטרנט תוך מעבר בהרבה נתבים בדרך, ובין צמתים שאין ביניהם חיבור פיזי ישיר. ראינו כבר איך עובד נתב ברמה הפיזית (Switching/ Scheduling) וכעת נראה את רמת התוכנה, בה הנתבים מבצעים אלגוריתמים מתוחכמים להכוונת המידע ברשת ליעד הנכון שלו.

שכבת ה Network מתרחשת בכול רכיב ברשת.

לראוטר יש שני תפקידים עיקריים:

- Forwarding שזה ממש להעביר הפקטות שמגיעות אליו מקווי הכניסה לקווי היציאה הנכונים, את זה ראינו כבר ברמה הקודמת, כי למעשה מהבחינה הזו זה בדיוק Switch.
- Routing בניית טבלאות הניתוב שיחליטו בהינתן שפקטה מסוימת מגיעה לאן צריך לשלוח אותה.

רשת שקמה צריכה להחליט ברמה הכי בסיסית איזה סוג רשת היא, האם VC או Datagram. כיום רשתות VC שדורשות הקמת השיחה מראש הן לא פופלריות, ובפרט האינטרנט היא רשת Datagram.

- ***Virtual Circuit:*** כדי לייצר השיחה עוברת הודעת ביקורת מהשולח ועד למקבל, כך שבהודעה רשום רוחב הפס הרצוי, גודל הבאפר הדרוש ועוד מאפיינים של השיחה, כול ראוטר בדרך יכול לבחור אם לקבל ולהתחייב לשיחה או לדחות אותה מעומס. השיחה תקושר רק אם נמצא נתיב שבו כול הראוטרים בדרך אישרו הקישור. כול ראוטר צריך להיות מסוגל לזהות את השיחה הספציפית כאשר מידע יגיע אליו, כדי שידע לאן לנתב אותה. לכן בעת יצירת השיחה כול ראוטר מקצה לשיחה מזהה יחודי, ומודיע לראוטר שלפניו בתור את המזהה שלו, כך שזה שלפניו ידע תחת איזה מזהה לשלוח את החבילות. וכך המידע עובר בין הראוטרים, כך שבין כול שניים הוא מחליף מזהה שיחה. זה מאפשר לחבילות לא להצטרך להכיל מידע על כתובת יעד, ולראוטרים עצמם לא להכיל טבלאות ענק לכתובות IP, אלא רק כמות קטנה של מידע, והיא טבלה ממזהה שיחה לפורט יעד, וזו תהיה טבלת ה Routing שלו. נשים לב בנוסף כי עבור כול קישור שנוצר ועובר דרך ראוטר מסוים, הראוטר הזה צריך להקצות State שזה אומר להחזיק זיכרון ומשאבים. חשוב לשים לב שגישה זו דורשת עמידות גדולה של הרשת, כי מספיק שראוטר אחד נופל וכול השיחות שעברו דרכו מתנתקות.
- ***ATM:*** פרוטוקול רשת שנבנה באקדמיה ולא תפס בעולם האמיתי. מספק QOS מסוגים שונים, בעוד האינטרנט הוא בסך הכול Best effort. רשת ה ATM דורשת מהראוטרים בדרך להיות חכמים, בעוד האינטרנט דורש ממשתמשי הקצה לבצע את האלגורתמיקה. רשת ה ATM היא Circuit based.
- ***DataGram Network:*** לא נוצר חיבור וירטואלי/אמיתי ישיר בין השולח למקבל. הפקטות זורמות ברשת כיחידות עצמאיות שעשויות ללכת כול אחת בנתיב אחר. הראוטרים בדרך לא מחזיקים State עבור כול קישור, אבל הם מסוגלים לנתב חבילות לפי כתובת היעד שלהן.
  - ***בעיה:*** יש הרבה מאוד כתובות יעד אפשריות, ליתר דיוק יש 4 מיליארד כתובות IP אפשריות, וזה לא סביר שלכול ראוטר תהיה טבלה בגודל 4 מיליארד שתגיד לאן לנתב כול חבילה שתגיע.
  - ***פתרון: Longer prefix:*** הטבלאות של הראוטר יכילו תחיליות של כתובות IP, כך שכול כתובת שתואמת עם התחילית הזו, תשלח למקום שרשום בטבלה בשורה של התחילית הזו. כאשר יש התנגשויות, כלומר תחיליות שמכילות אחת את השנייה, אז הארוכה יותר מנצחת.

Routing:

המטרה היא בהינתן רשת של Routers שזה למעשה גרף, צריך למצוא נתיבים טובים בין הצמתים. כאשר "טיב" של מסלול יכול להימדד בעלות/מרחק/רוחב פס וכו'. כמו כן נרצה שאלגוריתם הניתוב שלנו יהיה פשוט, יביא לתוצאה אופטימלית וכמו כן יהיה עמיד מפני שגיאות.

בתכנון אלגוריתם ניתוב יש הרבה חופש בחירה:

- מי קובע את הנתיב, קצוות התקשורת או הראוטרים בדרך?
- כמה מסלולים שונים יהיו בין שני צמתים ברשת?
- האם המסלולים יכולים להשתנות תוך כדי ריצה בהתאם לתנאי הרשת?
- כול כמה זמן מעדכנים את המסלולים?
- גלובלי – כול הראוטרים יודעים את מבנה כול הרשת או מבוזר – כול ראוטר יודע רק על שכניו.

יש שתי גישות אלגוריתמיות נפוצות:

- ***Link state:*** כול ראוטר ברשת ידע את כול המידע על הרשת, כלומר ידע את כול המרחקים בין כול ראוטר ושכניו, ובהנתן המידע הזה יוכל לבצע אלגוריתם דיאקסטרה למשל למציאת מרחקים קצרים ביותר לכול צומת אחר ברשת, וככה הוא ידע את מסלול הניתוב הרצוי ממנו לכול יעד. כדי שיהיה בידי כול צומת ברשת את המידע על כול הרשת, אז כול צומת שולח לכול שכניו את המידע שיש בידיו, והם מעבירים את זה הלאה עד שהמידע מפעפע לכול הרשת.
- ***Distance Vector:*** אלגוריתם מבוזר, כול צומת צריך רק מידע מהשכנים המידיים שלו כדי לחשב את טבלת הניתוב שלו. בכול צומת יש טבלת ניתוב ששורותיה הן יעדים ברשת ועמודותיה הן שכנים מידיים של הצומת. התאים בטבלה זה העלות להגיע ליעד שעל שמו השורה דרך השכן שעל שמו העמודה. הערכים מחושבים לפי ערכי השכנים בתוספת העלות להגיע לשכן. כול צומת מודיע לשכניו את המרחקים עבורו, כול פעם שיש שינוי, ורק כאשר יש שינוי, ושכניו מחשבים מחדש את המרחקים עבורם בהתאם, ומודיעים לשכניהם אם יש צורך. מבוסס על אלגוריתם Bellman-Ford, ובדומה לו כול הערכים מתחילים באינסוף, ואם הזמן מקבלים ערך ממשי של מסלול כלשהו ליעד, ובסופו של דבר הערך יהיה אורך המסלול הזול ביותר.
  - לאחר התכנסות האלגוריתם כול ראוטר יכול לייצר טבלת ניתוב, כך שלכול יעד הוא ישלח אותו לשכן שדרכו הכי קצר להגיע ליעד הזה.
  - ***שינוי במרחקים אחרי התייצבות:*** נניח כי לאחר ההתייצבות חל שינוי בעלות של חיבור מסוים. אם השינוי הנ"ל משפיע על המרחק הקצר ביותר של צומת מסוים ליעד מסוים אז הוא מודיע את זה לכול שכניו. נשים לב כי אם אלו בשורות טובות, כלומר המחיר ירד, אז ההודעה עוברת מהר בכול הרשת. אם זה חדשות רעות, כלומר המחיר עלה, יכול להיווצר מצב של Count To Infinty שבו המחיר עלה בהרבה, אז הצומת פתאום רואה שדרך השכן שלו הוא מסוגל להגיע במחיר נמוך יותר, אבל למעשה המסלול שהשכן שלו עובר דרכו, ולכן ככה לאט לאט שני הצמתים יעלו את המחיר של החיבור עד למחיר האמיתי, ומשום שהמחיר אינו חסום, גם זמן ההתכנסות של התהליך אינו חסום.
  - ***Poisoned Reverse:*** אם צומת X משתמש בשכנו Y כדי להגיע לצומת Z אז כש X מודיע לשכניו את המרחקים שלו, ל Y הוא יגיד שהמרחק שלו מ Z הוא אינסוף. זה מונע מצב של Count to infinity על מעגלים בגודל 3, אבל זה לא פותר הבעיה עבור יותר.
  - ***BGP:*** כדי למנוע את count to infinity ניתן לשלוח ביחד עם המרחקים גם את המסלול עצמו, וזה מה שעושה פרוטוקול BGP שהוא פרוטוקול עמוד השדרה של האינטרנט.
  - ***חוזקה:*** נשים לב שאם צומת כלשהו מפרסם מידע שקרי או שגוי זה ישנה את טבלאות הניתוב אצל כולם, ויגרום להם לשנות את הנתיב שאליו הם מכוונים את החבילות.

<u>דוגמאות לאלגוריתמי ניתוב קיימים כיום:</u>

- ***RIP (Rotuing Information Protcol)*** פרוטוקול לניתוב בתוך AS, כלומר פרוטוקול Intra-AS שמבצע ***Distance Vector*** ומודיע לשכניו כול 30 שניות על מצבו. מספר הקפיצות המקסימלי הוא 15, כי מדובר ברשת פנימית ל AS. אם לאחר 180 שניות אין תגובה משכן מסוים, מכריזים עליו מת ומעדכנים את כול המסלולים ואת השכנים. כמו כן מבצעים Poision Reverse (המרחק חסום על ידי 16).
- ***OSPF (Open Shortest Path First)*** פרוטוקול פתוח שמבצע LS. הפרוטוקול מאפשר להחזיק מספר מסלולים שונים אל היעד אם הם בעלי אותו מחיר. כמו כן מאפשר להכיל הרבה מטריקות

מרחק שונות, כמו כן מספק ניתוב ל Multicast. כמו כן מאפשר ניתוב היררכי כך שבכול רמת היררכיה מבצעים LS בנפרד, וחוסכים מנתבים לדעת יותר מידע ממה שהם צריכים באמת לניתוב.

- ***IGRP (Interior Gateway Routing Protocol)*** מבצע DV תוך התחשבות במספר מטריקות מרחק שונות, עובד מעל TCP.
- ***BGP(Border Gateway Protocol)*** פרוטוקול עמוד השדרה של האינטרנט שמבצע ניתוב בין AS שונים, Inter-AS. זה מבצע DV אבל מודיע לשכניו את המסלול כולו, ולא רק את המרחק ליעד (משלל שיקולים, חלקם פוליטים/כלכלים וכו'.) נשים לב שפרוטוקול זה צריך להתמודד עם הרבה דברים שנובעים משיקולים שאינם טהורים, כמו רשת AS שלא רוצה לנתב דרך AS אחרת, או להעביר דרכה את המידע של ה AS הזו וכו'. עובד מעל TCP.

Hierarchical Routing: בעולם האמיתי האינטרנט מחולק להרבה AS שונים, ובכול AS האדמין שלה יכול לבחור באיזה פרוטוקול Routing שהוא רוצה! כמו כן זה לא סביר שלכול ראוטר בעולם תהיה טבלה לכול יעד אפשרי בעולם, יש יותר מדיי. לכן בתוך כול AS הראוטרים מריצים את אותו פרוטוקול ניתוב, שיקרא Intra-AS. ולכול AS יש ראוטר שער, Gateway שמדבר עם ראוטרי השער של AS אחרים, זה נקרא Inter-AS. ולכול ראוטר בטבלת הניתוב שלו יהיה יעדים לתוך ה AS לפי פרוטוקול הניתוב הפנימי, ויעדים מחוץ ל AS ישלחו לראוטר השער, או במידה ויש הרבה כאלה, אז לראוטר השער המתאים. אם באמצעות Inter-AS רשת AS כלשהי למדה על מיקומה של רשת AS אחרת, ודרך איזה ראוטר שער צריך לצאת אליה, היא תעביר מידע זה לכול הראוטרים בתוכה באמצעות Intra-AS.

Broadcast Routing: המטרה היא לשלוח הודעה מסוימת להרבה יעדים, מבלי לשכפל את ההודעה ולשלוח בנפרד לכול יעד. נרצה כמות מינימלית של שיכפולים של המידע, רק כאשר יש בכך צורך. (כיום באינטרנט לא ממשים אלגוריתמים מתוחכמים כאלה, ופשוט משכפלים את הפקטות ושולחים אותן ב Unicast)

- **Controlled Flooding:** ראוטר יעשה broadcast לפקטה, כלומר ישלח אותה לכול הרגלים שלו, רק אם הוא לא עשה לה כבר flooding בעבר, אבל זה דורש ממנו לזכור ID של פקטות עבר שהוא שלח.
- **ReversePathForwarding:** ראוטר יעביר את פקטת ה Broadcast רק אם היא הגיעה מרגל המהווה חלק ממסלול קצר ביותר מהראוטר למקום ההודעה.
- **Spanning Tree:** ניצור עץ פורש לרשת הראוטרים לצורך העברת הודעות Broadcast וזה ימנע שכפול מיותר של מידע, כי פקטות Broadcast ינועו רק על העץ, ולכן כול פקטה על מסלול מיועדת ליעד אחד, עד לשכפול.
  - **Source-based-Tree:** לכול שולח בקבוצת ה Multicast יהיה עץ משלו, שיבנה כעץ מסלולים קצרים ביותר, ונשתמש בשיטת ReversePathForwarding. זה בעייתי כי יכולים להווצר מסלולים לא סימטרים, כלומר בכיוון אחד התקשורת מהירה מאוד, אבל "במעלה" העץ חזרה לשולח התקשורת איטית מאוד.
  - **Group-Shared-Tree:** יהיה עץ יחיד משותף לכול הקבוצה. נרצה שהוא יהיה פורש מינימלי. זה נקרא עץ stiener וזה עץ פורש מינמלי של קבוצת צמתים חלקית, וזה NPC ולכן זה בעייה.
  - **Center-Based-Tree:** תהיה צומת מרכזית אחת, וכול צומת יצטרף לעץ על ידי שליחת הודעת UniCast לצומת הזה, ובפעם הראשונה שההודעה הזו תתקל בצומת שכבר חלק מהעץ, אז הוא יצרף את הצומת השולח לעץ.

IP Addressing: נצטרך מזהה יחודי לכול ישות ברשת האינטרנט. המזהה הזה הוא כתובת IP שהיא מספר בעל 4 בתים. כמו כן נרצה שהמזהה הזה יספק Aggregation לכתובות(כמו חלוקה למדינות, ערים וכו' שקיימת במערכת הדואר) כדי שנוכל לעשות ניווט היררכי בקלות. ליתר דיוק כול כתובת IP משויכת ל Interface ברשת האינטרנט, כאשר Interface זה קישור פיסי, לכן בד"כ ל Host ברשת יהיה אחד כזה, בעוד של Router שלו יש הרבה רגלים, יהיה מספר גדול של IP.

כתובת IP מתחלקת לשתיים, מזהה הרשת/ Subnet שהוא הביטים העליונים, ומזהה הHost בתוך ה Subnet, שזה הביטים התחתונים. Subnet זה קבוצה של מחשבים שנמצאים כולם מאחורי אותה הרגל של ה Router, וכולם חולקים אותה תחילית ב IP.

- **חלוקה ל Class:** פעם חילקו את כתובות ה IP לשלוש מחלקות:
  - **A** כתובות בהן הבית הראשון הוא לרשת וה3 האחרים למחשבים ברשת. זה לא סביר, זה רשתות בעלות 2^{24} מחשבים, זה מגוחך.
  - **B** כתובות בהן שני הבתים הראשונים הם לרשת והשניים האחרונים למחשב. מחלקה זו הייתה הנפוצה ביותר, כי היא אפשרה 2^{16} מחשבים ברשת.
  - **C** כתובות בהן 3 הבתים הראשונים הם לרשת, והאחרון למחשבים. זה בעייתי כי זה מאפשר רק 256 מחשבים ברשת.
- **CIDR:** בעקבות הגסות של החלוקה לעיל, עברו לחלוקה עדינה **Classless InterDomain Routing,** שבחלוקה זו כמות הביטים שמייצגת את הרשת היא באיזה אורך שאנו רוצים, כאשר כעת יחד עם הכתובת צריך לציין כמה מהביטים הם לרשת, וכמה למחשב עצמו. בזכות חלוקה יעילה זו ניתן לבצע ניתוב היררכי בצורה יעילה יותר ברשת, כאשר נזכר שבמקרה של הכלה בתחיליות, התחילית הארוכה ביותר מנצחת. נשים לב שהמידע של כמה מהביטים הם רשת וכמה לא אינו חשוב לתהליך הforwarding עצמו, ולכן שיטה זו לא דורשת יותר מקום ב Header של ה IP, זה רק דרוש לתהליך בניית טבלת הניתוב.

DHCP: כאשר Host חדש מתחבר לרשת, איך הוא יקבל כתובת IP? איך הוא ידע איפה יושב שרת ה DNS? וכו'. אפשרות אחת היא שהאחראי של הרשת יתן לו את המידע הזה ידנית, אבל אנחנו נרצה משהו הוא Plug & Play ולכן הפרוטוקול הזה שעובר מעל UDP על פורט 67-68 מספק הקצאה דינמית של כתובות IP. (***Dynamic Host Control Protocol***)

ברשת יהיה שרת DHCP שתפקידו הוא להקצות את כתובות ה IP מבלי ליצור התנגשויות. כאשר לקוח חדש מתחבר לרשת הוא שולח פקטת כניסה בBroadcast שזה אומר הוא שולח לכתובת 255.255.255.255 מכתובת יעד 0.0.0.0 (כיוון שעוד אין לו כתובת IP). השרת עונה ללקוח עם הצעה לכתובת IP, גם כן הודעה זו נשלחת ב Broadcast כלומר לכתובת 255.255.255.255, ולכתובת מצורף זמן חיים שלה. הלקוח עם קבלת ההצעה מודיע ב Broadcast שהוא מעתה יהיה בכתובת הזו (נשים לב שיתכן שיש הרבה שרתי DHCP ולכן תפקיד הודעה זו הוא להודיע לשרתים שהצעתם לא התקבלה שהם נדחו, ושהשרת שכן התקבל ידע לשמור כתובת זו), ולאחר מכן השרת עונה לו באישור סופי, ישירות אליו לכתובת החדשה.

ICMP: פרוטוקול שמשמש נתבים ו***Host*** לשתף בינהם מידע בשכבת ה ***Network***, לא משתמש בפרוטוקולים בשכבת ה ***Transport***. (***Internet Control Message Protocol***). למשל משמש להודעה על ***header*** לא תקין, או על אי יכולת להגיע ליעד מסוים. שימוש נפוץ נוסף הוא ***Ping*** ו ***Traceroute***.

- ***TraceRoute***: נועד כדי למפות את הנתיב ממקור מסוים ליעד מסוים, ובכלליות למפות את הרשת. מנצל את טוב ליבם של הראוטרים בדרך (שיכולים לבטל אופציה זו) בכך שהוא שולח פקטות ליעד עם ***TTL*** נמוך בכוונה, ולאחר מעט מאוד ***HOPS*** ה***TTL*** יגמר והראוטר שבו הפקטה סיימה את חייה ישלח חזרה הודעת ***ICMP*** יחד עם השם שלו.

NAT: (Network Address Translation) הבעיה היא שמספר כתובות ה IP האפשריות הולך ונגמר, לכן מה שנעשה הוא שניתן כתובת IP חיצונית יחידה עבור רשת שלמה, כלומר כול ההודעות שיצאו החוצה מהרשת הזו לתוך האינטרנט יצאו עם אותה כתובת מקור, וכול ההודעות שיכוונו למחשבים ברשת הזו, המגיעות מבחוץ, ישלחו לאותה הכתובת. מאחורי הראוטר שמספק את ה NAT לרשת, ברשת עצמה יהיו כתובות IP פנימיות לתקשורת תוך רשתית.

אז איך הראוטר ידע בהינתן פקטה שהגיעה מבחוץ לאיזה מחשב היא שייכת? כול שיחה הוא משייך לפורט, ותהיה לו טבלת המרה מפורט שיחה למחשב ברשת. כעת כול שיחה עם מחשב מחוץ לרשת הוא יבצע על פורט אחר, והפורט יזהה את המחשב המתאים ברשת הפנימית.

מספק לנו עוד הרבה יותר כתובות IP אפשריות ברשת האינטרנט. נשים לב שיש 16 ביטים לשדה הפורט, ולכן יש לנו מרחב של 60 אלף אפשריות, תחת IP חיצוני יחיד. כמו כן זה מאפשר לשנות כתובות ברשת הפנימית מבלי להצטרך לשנות את הכתובת כלפי העולם החיצוני.

כמו כן זה מספק יתרון הגנתי, כי לא ניתן לפנות ישירות למחשב שמסתתר מאחורי NAT מבחוץ לרשת הפנימית.

אולם זה מעורר בעיה, שני Hosts שמסתתרים מאחורי שני NAT שונים לא יכולים ליצור שיחה אחד עם השני, כי אף אחד מהם לא יכול ליזום שיחה עם השני כי הוא לא יודע את הכתובת המדויקת שלו.

IP Fragmentation & Reassembly: לחיבורים שונים יש גדלים מקסימלים שונים של חבילה שיכולה לנוע בהם, לכן זרם של מידע נצטרך לשבור להרבה חבילות קטנה שיחוברו חזרה למידע השלם רק בצד המקבל הסופי. נשתמש בHeader של החבילה כדי לטפל בפרגמנטציה הזו.

Transport Layer:

השכבה הראשונה שלא מתקיימת בכול צומת בדרך, אלא אך ורק בצומתי הקצה של התקשורת. שכבה זו באה לספק אימולציה לוגית של חיבור אמין וישיר בין שני הצדדים תוך שהיא משתמשת בשכבה מתחתיה.

Reliable Data Transfer:

המטרה היא להיות מסוגלים להעביר מידע באמינות מקצה לקצה, תוך שיודעים כי המידע עובר דרך תווך לא אמין המועד לטעויות. כלומר השכבה הזו מספקת עבור השכבה שמעליה, שהיא שכבת האפליקציה, פונקציות העברת מידע אמינות, תורך שהיא משתמשת בפונקציות העברת מידע לא אמינות שמספקת לה שכבת ה Network.

סוגי שגיאות:

- שגיאות בפקטה עצמה, כלומר שדות של המידע התהפכו ושינו ערכם בזמן ריצתם על פני התווך. תמיד נניח כי ניתן לזהות שקרתה שגיאה כזו.
- פקטת אבדו ברשת ולא הגיעו ליעדן הסופי
- פקטות השכפלו ברשת, ויגיעו פעמיים
- פקטות יכולות לשנות את סדרן ולהגיע לא לפי הסדר בו הן נשלחו.

הנחת יסוד: נשים לב כי אנו מניחים Liveness על הקו, כלומר שמתישהו בעתיד החבילות שלנו אכן ישלח על הקו. כלומר אם אין בקו שגיאות אז זמן ההמתנה של פקטה עד להגעה ליד הוא סופי, ואם יש שגיאות ואיבוד פקטות אז לאחר מספר סופי של ניסיונות פקטה תצליח לעבור בשלום.

RDT 1.0: תחילה נניח כי הקו מושלם, כלומר אין בו שגיאות בכלל משום סוג אז הפרוטוקול הוא שפשוט ברגע שיש מידע שולחים אותו בקו, ומובטח שהוא מתישהו יגיע ליעדו (Liveness) ולפי הסדר הנכון וללא שגיאות, לכן בצד המקבל פשוט ברגע שמקבלים חבילה מעבירים אותה כמו שהיא לרמת האפליקציה.

RDT 2.0: כעת נניח כי ביטים יכולים להתהפך בזמן ריצתם על הקו, אך עדיין נניח כי כול מה שנשלח מתקבל בצד השני לפי הסדר. (כמו כן נניח כי ניתן לזהות שקרתה שגיאה, והשגיאה לא קורת בפקטות ה ***ACK/NACK***). נוסיף את מנגנון ה ***ACK*** ומנגנון ה ***NACK***. הפרוטוקול אומר כי השולח שולח הודעה מיד עם הגעתה אליו מרמת האפליקציה, ואז עובר למצב המתנה. אם קיבל ***ACK*** הוא עובר לשלוח את הבאה בתור, אם קיבל ***NACK*** הוא ישלח שוב את הקודמת. בצד המקבל זה פשוט מאוד, בעת שהגיעה חבילה בודקים אם היא תקינה, אם כן שולחים ***ACK*** אם לא שולחים ***NACK***. נשים לב שבבירור הפרוטוקול הזה מספיק העברת מידע אמינה, כי תמיד כול המידע עד עתה הועבר בצורה טובה, ורק פקטה יחידה יכולה להיות על הקו ברגע נתון, והטיפול בה מבטיח שמתישהו נצליח להעביר אותה בצורה תקינה.

RDT 2.1: מה אם פקטות ה ***ACK/NACK*** בעצמן התקלקלו? נשים לב שלא ניתן להניח כי זה ***ACK*** ולא ניתן להניח כי זה ***NACK***, כי אם הנחנו בצורה שגויה נוביל לטעות בפרוטוקול. לכן נוסיף מספרים ייחודיים לפקטות (מספיק ביט מבדיל בינהן). כעת השולח מבצע את אותו הדבר שהשולח הקודם ביצע אולם כעת הוא מחליף לסירוגין בין מספר 1 למספר 0, כאשר הוא עובר בינהם רק לאחר קבלת ***ACK***. אם התקבל ***ACK*** פגום אז שולחים שוב את המידע האחרון ששלחנו. בצד המקבל אם התקבלה שוב חבילת מידע עם המספר הקודם אז זורקים אותה ושולחים שוב ***ACK*** על המספר הזה. כמו כן הפרוטוקול הזה תקין משום שתמיד יש רק חבילה אחת על הקו בתנועה בכול רגע נתון.

RDT 2.2: אפשר להחליף את שליחה ה ***NACK*** בשליחת ***ACK*** נוסף על הפקטה האחרונה שקיבלנו, זה יוביל לאותה התוצאה.

RDT 3.0: כעת נניח שבנוסף לשגיאות במידע, פקטות יכולות פשוט להיעלם ברשת ולא להגיע לעולם ליעד שלהן. (עדיין מניחים שהקו הוא ***FIFO*** כלומר שאם חבילות כן מגיעות, אז זה לפי הסדר). הרעיון הוא שהשולח יחכה זמן "סביר" בציפייה ל ***ACK*** ואם הזמן עבר הוא ישלח שוב את החבילה האחרונה שנשלחה. זה לא יצור בלבול אצל המקבל כי יהיו מספרים לפקטות, וזה יצריך את הצד המקבל לציין ב ***ACK*** שלו לאיזה חבילה הוא עושה ***ACK***. כמו כן נשים לב שזה יכול ליצור שכפול ברשת, אבל אנחנו יכולים להתמודד עם זה. ה ***TimeOut*** הוא רק בצד של השולח, וכעת אם מגיעות הודעות זבל אז לא עונים להם בשליחה חוזרת כמו קודם, פשוט מחכים ל ***TimeOut***, אם נענה בשליחה חוזרת אז נוכל לקצר את זמן התקשורת אבל נוכל ליצור מצב של שכפול תמידי ברשת $Sorcerr's apprentince$. בעקבות ***TimeOut*** מוקדם מדיי אחד.

הפרוטוקולים שלעיל נקראים ***STOP & WAIT*** כי הם מצריכים את השולח לחכות את כול הזמן עד שחבילה חזרה מהמקבל, ולכן לפחות ***RTT*** אחד, וזה מעכב מאוד את התקשורת ולא מנצל את כול רוחב הפס! $Rate = \frac{\frac{L}{R}}{RTT+\frac{2L}{R}}$ וזה רע.

Piplined Protocols: כדי לפתור את בעיית היעילות של ***STOP & WAIT*** נאפשר להרבה פקטות להיות כרגע ברשת ללא ***ACK*** עליהן, במקביל. השאיפה היא לשלוח כמות כזו של פקטות בלי ***ACK*** כך שה ***ACK*** הראשון יגיע בדיוק כשסיימנו לשלוח את האחרונה.

Go Back N: לפקטות כעת יהיו מספרים בין 0 ל ***K*** במקום מתוך $\{0, 1\}$ ובצד השולח יהיה חלון בגודל ***N*** כך שבתחילת החלון תהיה הפקטה הכי מאוחרת שעוד לא קיבלנו עליה ***ACK***, לאחריה יהיו ***N*** פקטות שהשולח יכול לשלוח מבלי לקבל ***ACK***. נשים לב ש***ACK*** הם מצטברים בפרוטוקול זה. כמו כן

יהיה ***Timer*** עבור הפקטה הראשונה שהיא ***Un-ACKed*** וברגע שהוא פוקע, שולחים שוב את כול הפקטות החל ממנה. בצד המקבל אין חלון! הוא תמיד מצפה לקבל את הפקטה הבאה בתור ויזרוק כול דבר אחר.

אם מספרי הפקטות לא חסומים אז ברור שהפרוטוקול נכון, כי המקבל תמיד דורש לקבל את הפקטה הבאה הנכונה בתור. אם מספר הפקטות חסום אז בהנחה שהקו הוא ***FIFO*** מספיק ***N+1*** מספרים כי החלון של השולח לא יחזור אחורה לעולם.

Selective Repeat: הרעיון הוא שהמקבל לא זורם פקטות שהגיעו לא לפי הסדר, אלא שומר אותן ומאשר קבלתן כדי למנוע שליחה מחודשת של חבילות תקינות. והשולח ישלח מחדש רק חבילות שעוד לא קיבל עליהן ***ACK***. המקבל יעביר את החבילות לרמה מעל רק כאשר הוא יקבל את כול הרצף ויוכל להעביר אותן לפי הסדר. לכן כעת יהיה חלון בגודל ***N*** גם בצידו של המקבל, וכעת בשולח יהיה ***Timer*** נפרד לכול פקטה שכבר נשלחה ועוד לא התקבל עליה ***ACK***. החלון בצד השולח לא מתקדם מעבר לפקטה הכי מוקדמת שהיא אינה ***ACK*** פלוס גודל החלון, ברגע שיגיע ***ACK*** על הפקטה הזו תחילית החלון תקפוץ ישר לפקטה הבאה שאינה ***ACKed***.

אם מספרי הפקטות לא חסומים אז ברור שזה נכון, אחרת נדרוש לפחות $2N + 1$ כדי שלא יהיו בלבולים.

פרוטוקול זה יקר יותר במשאבים בצד המקבל, בעוד הקודם דורש אפס משאבים ממנו. מבחינת סיבוכיות רשת פרוטוקול זה יעיל יותר.

גודל החלון האופטימלי הוא $RTT * C$, כי אז בהנחה שאין איבודים בכלל נקבל נצילות מלאה של הרשת, כי בזמן שלוקח לשלוח את כול החלון כבר עבר ***RTT*** אחד, ולכן ה ***ACK*** כבר הגיעו.

Multiplexing/ DeMultiplexing: נניח והרבה אפליקציות שונות רצות על אותו ה HOST וכולן מקבלות מידע מהרשת. נצטרך איכשהו להפריד את המידע המגיע ולזהות אותו עם האפליקציה עליה הוא מיועד. נעשה זאת על ידי ports. כול חיבלת מידע נשלחת מפורט מסוים אצל השולח ותגיע לפורט מסוים אצל המקבל. השאיפה היא שיחד, גם פורט הקבלה וגם פורט השליחה יזהו באופן יחיד את האפליקציה על המחשב, ושכבת ה Transport תדע לנתב את החבילות לאפליקציות המתאימות. שני הפורטים מתווספים ל Header של ההודעה, ויחד תופסים 32 ביטים.

UDP: (User Datagram Protocol)

פרוטוקול שלא מצריך יצירת קישור בין הצדיים, אך הוא רק Best Effort, אינו מבטיח הגעה של הפקטות, אינו מבטיח סדר הגעת החבילות, כול חבילה נשלחת באופן נפרד מכול האחרות, אינו מבטיח שליטה בעומס ובקצב (לטוב ולרע...). שימושי בגלל הפשטות. באה לשימוש באפליקציות שהן טולרנטיות לאובדן מידע, אבל צריכות ביצועים מהירים, או אפליקציות שלא חשוב להן אמינות המידע ורק חשוב להן פשטות התקשורת, כמו DNS וכו'.

- checksum יש ב UDP ביט Checksum לכול 16 ביטים. משמש לבדיקת שגיאות.

TCP: פרוטוקול קצה לקצה המספר העברת מידע דו-כיוונית אמינה, מספק בקרת קצב ועומס. מעביר את המידע כ"זרם" ומצריך יצירת Connection לפני שניתן להתחיל שיחה. בשני צדדי השיחה יש באפרים.

בפרוטוקל הזה ממספרים את הבתים שעוברים מצד אחד לצד השני. כול פעם שצד כלשהו שולח חבילה הוא מודיע ב SEQ שלה את מספרו של הבית הראשון בחבילה הזו מתוך זרם המידע, בעוד שבאותה החבילה הוא מצרף גם מספר ACK שמייצג את מספר הבית האחרון שהוא קיבל מהצד

השני, וזה קומולטיבי, כלומר ACK מצטברים. נשים לב ששני הצדדים צריכים בתחילת השיחה לתאם את ה SEQ ההתחלתי (לשני הצדדים)

- ***Three Way – Handshake:*** תהליך יצירת הקישור הוא א-סימטרי. הצד המעוניין ליצור הקישור שולח פקטת SYN עם מספר ה SEQ שהוא רוצה להתחיל ממנו. הצד השני יענה לו ב SYN ACK שיאשר את המספר הזה, ויצרף את מספר ה SEQ שהוא רוצה להתחיל ממנו, בשלב זה הוא גם יקצה באפרים אצלו לקישור הזה. לאחר מכן היוזם ישלח פקטת ACK על ה SEQ שהצד השני ביקש, וכעת ניתן להתחיל בשיחה החל מה SEQ שנקבע. כמו כן זה מספק הגנה מפני מתחזים, כי אם יתקשו לנחש את ה SEQ שבחר הצד השני, והודעה על כך תגיע למחשב האמיתי ולא למתחזה.
- ***סגירת קישור:*** בפרוטוקול TCP צריך להודיע על סיום הקישור כדי שהצד השני יוכל לשחרר את המשאבים אצלו. הצד המסיים שולח הודעת FIN הצד השני שולח על זה ACK, סוגר הקישור ושולח גם הוא פקטת FIN. הצד הראשון ברגע קבלת ה FIN ישלח על כך ACK ויכנס למצב המתנה (הסיבה להמתנה זה למנוע יצירת קישור חדש על אותו IP ופורט, מיד לאחר סגירת הקישור הזה , כאשר עוד יש פקטות ברשת)
- ***Reliable Data Transfer:*** מבחינה זו פרוטוקול TCP ממש בצד המקבל חלון בגודל אחד, כלומר הוא לא מאפשר Gap במידע שלו (בד"כ... המקבל יכול להחליט שכן, אבל זה נתון לשיקולו). אם הגיע מידע מתקדם מדיי הוא שולח שוב ACK על המידע הכי חדש שהוא מצפה לו. בצד השולח הוא שולח הרבה סגמנטים, ולכול אחד מדביק Timer משלו. בנוסף בצד המקבל הוא מעקב ACK כדי לנסות ולהדביק שניים רצופים ל ACK יחיד.
- ***אורך ה TimeOut:*** נרצה שזמן הTimeOut יהיה בערך כמו ה RTT. לא נרצה אותו קצר מדיי כי זה יגרום לשכפול מידע מיותר ברשת, ולא נרצה אותו ארוך מדיי כי זה יגרום לבזבוז זמן. לכן נרצה איכשהו להעריך את ה RTT של הרשת. בד"כ נדגום לכול פקטה את ה RTT שלקח להעביר אותה, ונמצא אותו ביחד עם דגימות העבר בממוצע אקספוננציאלי דועך כדי לתת יותר משקל לדגימות חדשות על פני ישנות. $E - RTT =$ $(1-\alpha) * E - RTT + \alpha * Sample$ כלומר הערך החדש זה ממוצע בין הערך הישן, לבין מה שדגמנו כרגע. זה יתן מדד שהוא פחות מושפע מתנודות רגעיות של הרשת. נרצה בנוסף גם איזשהו מרווח ביטחון שיהיה פרופורציוני לשונות של הדגימות, שגם אותה נחשב בממוצע אקספוננציאלי מצטבר.
- ***Flow Control:*** כדי למנוע את המקרה שבו השולח מציף את המקבל במידע בקצב גדול יותר ממה שהוא מסוגל לעבד, בפקטת ה TCP יש שדה שבו המקבל רושם את גודל הבאפר שהוא הקצה למידע, והשולח לא ישלח יותר מכך עד שיתוודא לו שיש עוד מקום בבאפר של המקבל.

Congestion Control: יכולה להיווצר בעיה ברשת כאשר יותר מדי מקורות שולחים יותר מדיי מידע לרשת בו זמנית, והרשת לא מסוגלת להתמודד איתו. זה יכול ליצור אובדן של פקטות מהתמלאות של באפרים וגם יכול לגרום לדיליים גדולים ברשת בעקבות תורים ארוכים. ותקלות אלה הן כמו כדור שלג שרק גורם לעומס ולדיליי לגדול, ולסתום את הרשת.

העיקוב ברשת גדל אקספוננציאלית כאשר הניצולת של הקו מתקרבת לקיבול, כי האופן שבו פקטות מגיעות הוא סטוכסטי ומצטברים תורי ענק.

העיקובים גורמים לשולחים לשלוח שוב הודעות שגורמות לעוד עיקובים וכו'. יכול גם להיווצר מצב של DeadLock אם הבאפרים סופיים.

פתרונות:

- Net-Work Assisted: הראוטרים עצמם ברשת מודיעים ללקוחות הקצה על המצב ברשת מבחינת עומסים, ומדווחים להם באיזה קצבים לשדר והאם להגביר או להנמיך הקצב. בעקרון פתרון

אידאלי כי הראוטרים מודעים הכי טוב למצב העומס ברשת, אבל זה פתרון מסורבל וקשה שדורש ראוטרים חכמים במיוחד ולכן פחות נפוץ באינטרנט. כן קיים ב ATM. זה מכביד על עבודת הראוטר ויאט אותו, אבל זה מספק תגובה מהירה ומדויקת לעומסים ברשת. כמו כן זה דורש תיאום מושלם בין כול הראוטרים.

- End to End: הרשת עצמה לא מדווחת למשתמשי הקצה על העומסים, אלא הם לומדים על העומס בעקבות אובדן פקטות ו TimeOut ומסיקים לבד על מצב הרשת ופועלים בהתאם. בכול RTT מתשמשי הקצב מקבלים פידבק מהרשת האם ניתן להגביר קצב או להנמיך קצב, והפידבק נובע מהאם היה אובדן.

מטרות:

- למקסם את הנצילות של הרשת, כלומר מספר הביטים שעוברים מצד לצד.
- לחלק המשאב בצורה הוגנת בין כול השיחות השונות. (Max-Min-Fairness). כדי לחשב את החלוקה פשוט נסתכל על כול קישור ונחלק את הקיבולת שלו חלקי מספר הקישורים שעוברים דרכו וניתן לכול אחד את החלק שלו. כעת נמחק את הקישור הזה ונמשיך לבצע זאת לכול אחד מהקישורים. (נתחיל מהקשת המינמלית ונתקדם)

Network Assisted:

- ATM ABR: דוגמא למימוש Network Assisted. אחת לכול 32 פקטות (Cells ב ATM) שולחים פקטת RM שתפקידה לנתר אחרי העומס. החבילות האלה עוברות ברשת מהשולח למקבל ובחזרה, כך שכול ראוטר בדרך משנה אותה כך שהיא תדווח על מצב הרשת הנוכחי, ועול העומס שמסוגל לעבור דרכו. השליטה נמצאת בידי הראוטרים שמחשבים הוגנות אצלם ומסוגלים לדווח לכול שיחה באיזה קצבים לשדר. (או שיש 2 ביטים שרק מציינים האם להאט או להגביר, או שיש 2 בתים שממש מציינים את מידת העומס ברשת והקצב המדויק שהשיחה צריכה לשדר).

End to End:

- **Multiplicative Update:** כשאין עומס מכפילים פי שתיים הקצב, כשיש עומס מחלקים בשתיים. זה מביא תגובה מהירה לעומס, אבל זה לא הוגן, היחסים בין השיחות שהיו בהתחלה יישארו אותו הדבר למשך כול התקשורת.
- **Additive Update:** מעדכנים באחת מעלה/מטה בהתאם להאם יש או אין עומס. זה מביא תגובה איטית מאוד לעומסים ברשת, וזה משמר הפרשים התחלתיים שהיו בין השיחות, ולכן לא יביא להגינות מושלמת.
- **Additive Increase Multiplicative Deacrese:** בעליה עולים בהופסת קבוע, בירידה יורדים בכפליות. זה מספק תגובה מהירה לעומסים ברשת וכמו כן מקרב את היחס בין השיחות ליחס המאוזן ביותר. (וקטורים מתכנסים...)

TCP Congestion Control:

יש חלון בגודל מסוים, Cwnd, כך שלא ניתן לשלוח יותר מידע מגודל החלון ב RTT אחד, ולכן גודל החלון קובע את קצב השידור. מתחילים מחלון קטן ומגדילים אותו באופן אקספוננציאלי עד לסף מסוים ואז עוברים לגידול לינארי עד שיש אירוע של Congestion (איבוד פקטה, טיימאוט...) ואז מורידים את הקצב (כול מודל קצת באופן שונה). השאיפה היא לשדר בקצב הגבוהה ביותר שניתן עד שיש עומס, ולכן צריך לדגום הרשת כדי לדעת מתי יש עומס ואז להוריד הקצב. קצב השידור יהיה $\frac{W*MSS}{RTT} Bytes/sec$ כאשר W זה גודל החלון(נספר בפקטות) בכול רגע נתון.

- **Slow Start:** שלב של גידול כפלי עד שמגיעים ל Threshold. על כול ACK מגדילים את גודל החלון באחד! וזה יצור גידול כפלי ב RTT שלם.
- **Congestion Avoidence:** גידול לינארי עד לאירוע של עומס. ברגע שקורה אירוע של עומס מקבעים את ה Threshold להיות חצי מגודל החלון הנוכחי. אם קרה TimeOut חוזרים לגודל חלון מינמלי (1).
- **Fast Retransmit:** בפרוטוקול TCP Reno בשונה מ TCP Taheo יש שלב שנקרא FR. TCP Reno מבדיל בין TimeOut לבין Dup Ack. הרעיון הוא לנסות להימנע מ TimeOut, ובמידה ורק מספר בודד של פקטות הלכו לאיבוד, אז אמורים להגיע הרבה ACK משוכפלים מצד המקבל, ולכן נזהה את זה ונשלח לו את ההודעות הללו שוב במהירות, ולא נקטין את גודל החלון יותר מדי כמו ב TCP Taheo שמחזיר לאחד את גודל החלון.
  אם הגיעו 3 ACK משוכפלים על אותה פקטה נכנס למצב הזה שבו חותכים את הסף לחצי גודל החלון, ואת גודל החלון להיות הסף פלוס קצת. ברגע שמגיע ACK חוקי חוזרים ל CA. אם קרה TimeOut עוברים ל SS.
- **TCP Vegas:** רעיון שונה, דוגמים את ה RTT ומשנים את קצב השידור לפי ההבדל בין ה RTT שדמנו לבין מה שציפינו.
- **ניתוח:** נניח מדובר ב TCP Reno אזי קצב השידור נע בין W לבין W/2 כאשר W גודל החלון המקסימלי שאליו מגיעים. קצב השידור הממוצע הוא בערך $\frac{3}{4}W * MSS/RTT$ ואובדן של פקטה קורה בשפיץ של המשולש, ולכן כול בערך $O(W^2)$ פקטות יש אובדן. לכן הסיכוי לאבד פקטה $P = O\left(\frac{1}{W^2}\right)$ או לחילופין $W = O\left(\frac{1}{\sqrt{p}}\right)$.

<u>TCP Latency Modeling:</u>

<u>NetWork Security:</u>

<u>DDOS (Distributed Denial Of Service):</u> משתמשים בצבא של מחשבים כדי לשלוח הרבה מאוד בקשות לשרת מסוים במטרה למנוע את היכולת שלו לענות לבקשות לגיטימיות מלקוחות אמיתים ובכך מונעים ממנו לספק את השירות שאליו הוא מיועד.

- **פתרון אפשרי:** BlackHoling: ראוטרים בדרך פשוט יזרקו בקשות שיגיעו לשרת המסכן, ובכך יצילו אותו מקריסה, אבל למעשה התוקף השיג את מטרתו, כי גם בקשות לגיטימיות יזרקו לפח.

<u>SYN-Flooding:</u> נזכר שב TCP לאחר שלקוח שלח SYN לשרת, השרת מקצה לו State ושומר זיכרון ומחכה ל ACK חזרה מהשולח. לכן אם ישלחו הרבה מאוד SYN לשרת מבלי תגובה, הוא יקצה המון זכרון, ובסוף יחסם.

- **פתרון אפשרי:** SYN-Cookies. לאחר קבלת ה x SYN השרת לא מקצה זכרון, אלא מחשב $f(x)$ כאשר f פונ' האש לא הפיכה, ושולח $SYN\ f(x), ACK\ x+1$ ושוכח מהלקוח הזה. אם הלקוח שקרי אז שום דבר לא יקרה כי הוא לא יענה ב ACK. לקוח אמיתי יענה ב ACK אם המספרים הנכונים, ואז השרת יוכל לחשב שוב את ההאש ולבדוק את נכונות הפקטה, ואז לפתוח קישור. כמו כן לתוך פונ' ההאש מכניסים עוד פרמטרים כמו הזמן הנוכחי וה IP של השולח וכו' כדי לחתום על אמינות התקשורת.

<u>Spoofing:</u> מצב שבו מישהו מתחזה למשתמש אחר ברשת על ידי גניבת IP/Mac/Arp/DNS וכו' כדי ליצר DDOS.

- **פתרון:** להרכיח את הלקוח לענות מה IP שאליו הוא מתחזה. למשל ב HTTP עושים בקשת ReDirect. ב TCP אפשר לשלוח הודעה שגויה שתחייב את הצד השני להגיב ב RST.

Quality Of Service:

העקרון שבו הרשת מבטיחה שירות ללקוחות שלה, ומבטיחה להם תנאים מסוימים. נשים לב שזה לא מתקיים באינטרנט כיום שהיא רשת Best Effort אבל עושים מאמצים להביא את האינטרנט למצב שיש בו QoS. כלומר נרצה להגן את שיחות מסוימות מפני שיחות אחרות כבדות יותר, כאשר לכול שיחה יש הדרישות שלה מבחינת קצב, דיליי וכו'. רלוונטי כאשר סך התקשורת שרוצה לעבור גדול מרוחב הפס.

עקרונות:

- סימון: נצטרך להיות מסוגלים לסמן פקטות לפי הסוג שאליו הן שייכות, כדי שנוכל בעתיד להעדיף חלק על פני אחרות בהתאם למדיניות הניתוב. צריך להיעשות בכניסה לרשת עצמה, כלומר בקצה.
- הגנה: לספק בידוד והגנה של סוג פקטות אחת על פני השנייה. כלומר מנגנוני שיטור שכול Class עומד בהתחייבות שלו לקצב שידור. גם כן יעשה בכניסה לרשת.
- יעילות: אחרי כול האפליה והסימון, עדיין נרצה שננצל ככול שניתן את הקיבולת של הקו.
- הזמנה מראש: כול תקשורת צריכה להזמין מראש את רוחב הפס והקצב שבו היא רוצה לשדר, והרשת תשריין לה מקום(או לא) בהתאם לזמינות.

אבני יסוד:

- ***Scheduling:*** העדפה והכוונה של תורים שנכנסים לתוך Router. ראינו כבר. כלומר הראוטר ייבחר להעדיף קו כניסה אחד על פני השני.
- ***Packet Dropping:*** אם באפרים מתמלאים צריך לבחור איך לזרוק את הפקטות. לא סתם נחכה שהוא יתמלא ונזרוק, אלא נפעל באופן אקטיבי לזריקת פקטות. נשים לב שבאפר קטן יאבד פקטות בBursts אבל ישמור על Delay נמוך, בעוד באפר גדול יעשה בדיוק ההפך. לכן נרצה להשתמש ב
  - ***Randomly Early Detection(RED):*** נרצה להודיע על עומס בבאפרים לפני שהם מתמלאים ונצטרך לזרוק הכול, כדי שלזרמים יהיה מספיק זמן להוריד את הקצב שלהם בהדרגה מבלי לאבד יותר מדיי פקטות. לכן החל מסף מסוים בגודל הממוצע של הבאפר נתחיל לזרוק פקטות באקראי, כאשר הסיכוי שבו זורקים תלוי בגודל הממוצע של הבאפר באופן לינארי בין הסף התחתון לעליון. למה ממוצע? כדי להתעלם מפרצים רגעיים. בשילוב עם TCP זה יוביל לדעיכה של הקצב באופן מתון (FR). בגדול לזרוק פקטות זה רעיון רע... זה שימושי לשיחות ארוכות טווח, לא יכיל לשיחות קצרות.

- ***Trafic Shaping:*** שליטה בקצב שבו החבילות נשלחות (ולא רק כמה נשלחות). בזמן יצירת הקישור השולח מתאם עם הרשת את אופי התקשורת, ומשתמשים בשני מנגנונים כדי לנתר על כך:
  - ***Leaky Bucket:*** אלגוריתם פשוט לשליטה שבקצב שפקטות נכנסות לרשת. יש לו באפר בגודל B, כך שאם הוא מתמלא פשוט זורקים פקטות. וכול r יחידות זמן נכנסת פקטה בודדת לרשת. ככה הוא מווסת את קצב הכניסה, ומסוגל לבלוע Bursts ולמתן אותם לקצב שהרשת תעמוד בו.(זה קורה בצד הכניסה לרשת)
  - ***Token Bucket:*** לבאפר יש Tokens וכדי להכניס פקטה מהדלי לתוך הרשת צריך לשלם Token. כאשר ה Token מתמלאים חזרה בקצב קבוע, ויש גודל מקסימלי. זה מאפשר לווסת את קצב הכניסה לרשת, תוך איפשור של Bursts אבל עד גודל מסוים, ולא מאפשר Bursts רצופים.
- **Model (σ,ρ):** מודל לדימוי שליטה בקצבי כניסה לרשת. כאשר ρ זה הקצב הממוצע ו σ זה גודל הפרץ המקסימלי. לכן לגודל אינטרוול בגודל t אפשר לשלוח $\sigma + t\rho$. עבור כמה זרמים

פשוט מחברים הפרמטרים. ראוטר צריך להבטיח כי גודל הבאפר שלו גדול מסכום הפרצים האפשרים, וכי קצב השידור שלו גדול מסכום הקצבים. כך ראוטר יוכל להבטיח *Asmission control* (הזמנת שיחה) כי הוא יבדוק עם הוספת הקו החדש תגרום לגלישה בבאפר או בקצב.

- ***WFQ* מבטיח חסם על העיקוב:** אם יש קו (σ, ρ) שעובר דרך *K* ראוטרים שעובדים ב *WFQ*, והקו הזה מרוסן כלומר מציית למודל אז $delay \leq \frac{\sigma}{g} + \sum_{i=1}^{k} \frac{L}{g(i)} + \frac{L}{r(i)}$ כאשר L הוא גודל החבילה המקסימלית, $r(i)$ קצב השידור של ראוטר *i* ו $g(i)$ זה כמה קצב קיבל השידור שלנו. כמו כן $g = \min g(i)$.
- ***QoS Routing:*** המטרה היא למצוא מסלול ניתוב טוב ביותר, רק שכעת יש אילוצים של QOS אל המסלול הזה, כגון חסם על הדילייי וחסם תחתון על רוחב הפס. בגדול זה בעייה קשה למצוא מסלול קצר ביותר תחת אילוצים. אבל אפשר לנסות למצוא מסלול תוך מקסום ומינום של ערכים מסוימים.
- ***MultiplePath Routing:*** נרצה לנצל את העובדה שיש כמה מסלולים שונים בין שני יעדים ברשת כדי להיות מסוגלים לפצל את התשדורת ולהגדיל את רוחב הפס שאנחנו מסוגלים לספק לקישור הזה.

Application Layer:

השכבה העליונה במודל שמספקת ממש את הצרכים של המשתמש. יש סוגים שונים של אפליקציות, למשלWeb/Mail/Streaming/Voice/Remote login וכו'. לכול סוג אפליקציה יש דרישות שונות מבחינת אמינות, קצב שידור, רוחב פס וכו'.

Client – server: גישה אחת למימוש אפליקציות היא גישת השרת-לקוח. בגישה זו יש קבוצה מצומצמת של מחשבים שמהווים השרתים ותפקידם לספק שירות ללקוחות. התקשורת א-סימטרית כך שהשרתים תמיד יושבים ומחכים שלקוח יזום איתם שיחה והם ישרתו אותו. השרתים הם סטטים ובעלי IP ידועים מראש, בעוד הלקוחות מחלפים, ולמעשה כול התקשורת מתבצעת בין שרת ללקוח ולקוחות לא מדברים בינם לבין עצמם. נשים לב שגישה זו מהווה בעיה כאשר מבחינת Scalabilty, כי השרתים מתישהו לא יוכלו לעמוד בעומס.

P2P: גישה שינה היא גישת ה Peer to Peer שבה אין שרתים מוקצים מראש, ולקוחות מספקים שירות אחד לשני. הלקוחות מדברים בינהם, ומספקי השירות ומקבלי השירות מתחלפים בינהם לאורך הזמן. פותר את בעיית ה Scalabilty כי ככול שיש יותר מקבלי שירות כך יש גם יותר נותני שירות.

HTTP: דוגמא פשוטה לפרוטוקול ברמת האפליקציה היא פרוטוקול HTTP שמספק שירותי Web. הפרוטוקול הוא במודל השרת-לקוח, כך ששרשת הינו דפדפן שרוצה להשיג דף אינטרנט שנמצא אצל השרת, כאשר דף אינטרנט הוא פשוט קוד HTML ותוספים כמו תמונות, שירים וכו'.

הפרוטוקול בנוי מבקשות ותגובות, כך שלקוח מבקש דף מסוים מהשרת, והשרת עונה לו. השרת מאזין לבקשות על פני פורט 80. הפרוטוקול עובד מעל TCP, והפרוטוקול הוא Statless כלומר כול בקשה מטופלת בנפרד מכול הבקשות שהיו בעבר, והשרת לא זוכר מה היה בעבר ומי ביקש מה.

משום שהפרוטוקול הוא מעל TCP צריך לפתוח קישור לשיחה. בגרסא הישנה של HTTP פתחו קישור חדש עבור כול בקשה וכול העברה של קובץ בנפרד, וזה גורם להרבה OverHead. היום משתמשים באותו הקישור להעברת הרבה קבצים ביחד.

Coockies: משום ש *HTTP* הוא *Statless*, כדי לאפשר לשרת בכול זאת לזכור מידע לגבי המשתמש נשתמש במנגנון ה *Coockies* שפשוט אומר שהשרת מקצה לכול לקוח שמתחבר איזשהו מספר

מזהה, ומודיע ללקוח מהו המספר הזה, וכעת הלקוח יפנה לשרת עם המספר הזה וזה יאפשר לשרת לזכור עליו מידע, ולקשר בין המידע ללקוח בעזרת המספר המזהה.

Web Cache: Proxy. המטרה היא לשמור דפים שהלקוח ביקש לאחרונה, כדי שאם הוא יבקש אותם שוב הם כבר יהיו בזכרון ויהיה אפשר לחסוך את הפנייה לשרת המרוחק. כעת כול הבקשות של הלקוח קודם כול עוברת דרך ה Proxy שבודק אם הדף כבר נמצא ברשותו. אם כן הוא שולח אותו ללקוח מבלי בכלל לפנות לשרת המדובר, אם לא הוא פונה לשרת, משיג את הקובץ, שומר אצלו ומעביר ללקוח. זה מפחית את הזמן שהלקוח צריך להמתין לדף, ומקטין את התעבורה על הקו יציאה מהשרת הפנימית לאינטרנט.