תקשורת – סיכום:

RFC (request for comment) – תיעוד תקנים של פרוטוקולים בתחום התקשורת.

Protocol – מגדיר את הצורה והסדר של העברת מסרים בין לפחות 2 כלי תקשורת וכן את הפעולות שיש לנקוט בהתאם.

גישה והתחברות לאינטרנט:
ADSL (digital subscriber line):

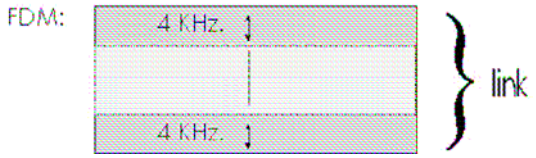
- יש צורך במרכזיה קרובה לבית.
- המהירות קבועה תמיד.

כבלים:

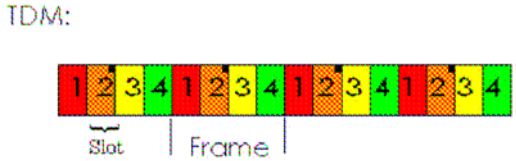
- יש חלוקה ברוחב הפס.
- אם אף אחד לא צורך את הקו, יכולה להתקבל מהירות גבוהה מ-ADSL.
- בעיה בביטחון של הקו. (אבטחת המידע).

גישות להעברת המידע:

- Circuit switching – כמו בטלפון, מוקם קו בין שתי נקודות ליצירת הקשר.
  כאן יש 2 גישות להעברת המידע:
  1. FDM (Frequency-division multiplexing) – חלק מרוחב הפס מוקצה להעברת המידע.



  2. TDM (Time-division multiplexing) – רוחב הפס מתחלק במספר המשתמשים כך שבכל פעימה כל רוחב הפס מוקצה עבור משתמש אחד.



החיסרון בגישה זו טמון בכך שכאשר המשתמש לא רוצה להעביר מידע, עדיין מוקצה עבורו רוחב פס מסוים. (idle resources).

- Packet switching – שבירת המידע שרוצים להעביר לחתיכות קטנות ושליחתן ברשת.
  החבילות (Packets) נעות דרך צינורות התקשורת (links) ודרך ה-packet switches (שהם לרוב routers וגם link-layer switches). החבילות נשלחות דרך כל קו (Link) בקצב המקסימלי שלו. מרבית ה-packet switches ממתינים עד לקבלת כל החבילה ורק אז מתחילים להעביר את הביט הראשון של החבילה להמשך דרכה.
  בנוסף לעיכוב הנ"ל,כשחבילה מגיעה ל-switch והוא עסוק, היא נכנסת לתור (output buffer). אם ה-buffer מתמלא, כלומר יש עומס של חבילות (congestion) החבילה אובדת!.
  לסיכום, גישת ה- packet switching פחות מתאימה לשירותים שהם real-time כגון שירותי טלפוניה ושיחות וידאו מאחר ויש בה עיכובים ואיבוד מידע (בזמן עומס).
  מצד שני, גישה זו מספקת דרך טובה יותר לחלוק ברוחב הפס, וכן, היא פשוטה, יעילה וגוזלת פחות משאבים מהשיטה הראשונה. בנוסף, בניגוד לגישה הקודמת, כאן יש הקצאת משאבים לפי דרישה ולכן מודל זה מנצל טוב יותר את מבנה הרשת. (זה נקרא statistical multiplexing).

כיצד החבילות נעות ברשת ומגיעות ליעדן – Virtual circuit network:

- בכל חבילה יש תג מזהה שמציין את המסלול שעליה לעבור.
- מסלול החבילה נקבע מראש.
- לכל נתב (router) יש טבלת קידום (forwarding table, יפורט בהמשך) שמכווינה את החבילה.

מס' מילים על ה-Physical media:
Link – כבל המחבר בין 2 מחשבים.
יש מספר סוגים של מדיות פיסיקליות המקשרות בין מחשבים, נתבים וכו': כבל קואקסיאלי, כבל נחושת, סיב אופטי (בטוח יותר כי לא ניתן להאזין לו באמצע, אך הרבה יותר יקר) ועוד.

<u>עומסים:</u>
קיימים סוגים שונים של עומסים הנובעים מסיבות מגוונות:
כפי שצוין לעיל, כאשר מגיעה חבילה חדשה לנתב A, היא נבחנת ע"י הנתב ואם מתגלה כתקינה (כלומר כתובת היעד שמכוונת אליו תקינה) אז היא נכנסת לתור של הנתב רק אם זה לא עסוק כרגע בהעברת חבילה אחרת ורק אם יש מקום בתור החבילות שלו.
א. Processing delay – הזמן הלוקח לבחון את ה-header של החבילה ולקבוע את ייעודה. לעיכוב זה מתווסף גם הזמן הלוקח לאמת האם נפלו טעויות בחבילה בעת השליחה.

ב. Queuing delay – הזמן הלוקח לחבילה לצאת מהתור ולהגיע לקו עצמו לפני שליחתה. (עיכוב זה תלוי במספר החבילות שכבר נמצאות בתור). אם התור ריק, ה-Queuing delay = 0.

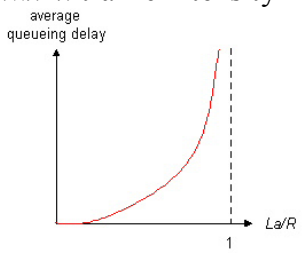
ג. Transmission delay – הזמן הלוקח להוריד את החבילה לקו.
דוגמא: עבור חבילה בגודל L=10bits וקצב הורדה של נתב A לנתב B השווה ל-R=10Mbps נקבל כי ערך ה-Transmission delay הוא: L/R = (10b)/(10000b/s) = 0.001s.

ד. Propagation delay – הזמן שלוקח לחבילה לעבור על הקו. עיכוב זה תלוי בסוג המדיה הפיסיקלית המאפיינת את הקו. במילים אחרות, עיכוב זה מחושב ע"י המנה בין המרחק בין שני הנתבים ומהירות ההעברה (~ מהירות האור), כלומר d/s (d-המרחק, s-מהירות הקו).

ה. Nodal delay – נובע מסעיפים א-ד ומחושב באופן הבא:

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

d_{proc} - ה-Processing delay – לרוב מספר מיקרושניות או פחות.
d_{queue} - ה-Queuing delay – תלוי בעומס (congestion) בתור.
יש להרחיב מעט על רכיב זה: בניגוד ל-3 הרכיבים האחרים במשוואה, רכיב זה אינו קבוע עבור כל חבילה שכן הוא תלוי במצב התור הנוכחי עבור כל קו (Link). לצורך חישוב עיכוב זה יש להגדיר תחילה את a – הקצב הממוצע של הגעת חבילה לתור. (הגעה מחזורית או הגעה אקראית).
נגדיר בנוסף את L – גודל חבילה (bits) ואת R – קצב השידור של חבילה (bits/sec) ובכך קיבלנו ערך חדש : traffic intensity = La/R המהווה מרכיב חשוב בחישוב ה-Queuing delay.



כעת, אם La/R ~ 0 ה-Queuing delay קרוב לאפס באופן ממוצע.
אם La/R→1 ה-Queuing delay גדל באופן משמעותי.
אם La/R > 1 - ה-Queuing delay הוא אינסופי, כלומר חבילות אובדות!.
(ראה http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/queuing/queuing.html).

d_{trans} - ה-Transmission delay – L/R – מקבל חשיבות יתרה עבור קווים בעלי מהירות נמוכה.

d_{prop} - ה-Propagation delay – מספר מיקרושניות ועד מאות מילישניות עבור קו בין שני מחשבים בקמפוס מסוים למול שני נתבים המחוברים דרך לוויין גיאוסטציונרי.

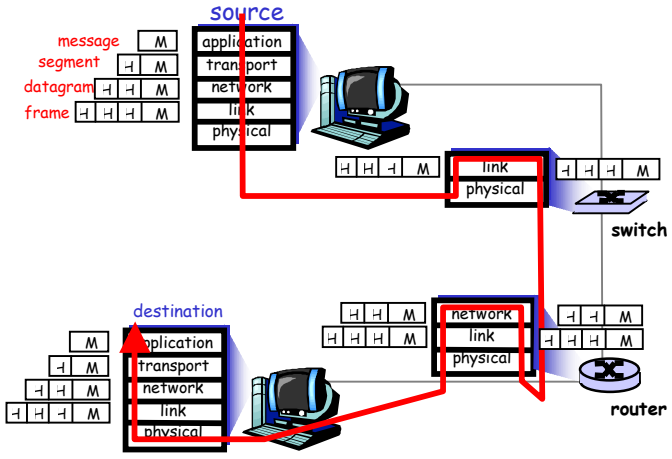
TraceRoute – תכנית המאפשרת לאמוד את הזמן שלוקח לשלוח חבילה / מס' חבילות מתחנת קצה אחת לשנייה על כל השלבים בדרך.
איך זה פועל: עבור N-1 תחנות בדרך בין המקור ליעד, תחנת המקור שולחת N חבילות. כל חבילה i המגיעה לנתב i מוחזרת על-ידי הנתב. כאשר החבילה חוזרת לתחנת המקור, מחושב הזמן הכולל לסיבוב. התהליך מבוצע 3 פעמים לקבלת ממוצע.
TraceRoute מאפשרת לברר היכן נוצר עומס מסוים בתהליך שליחת חבילה.

מודל השכבות – Protocol layering:

זהו המודל הרווח ביותר כיום בתחום התקשורת. המודל (top-down) מביא לכך שכל שכבה אחראית על המידע שנמצא בה, עוטפת ומפרקת אותו לפי הצורך וכן, מעבירה אותו לשכבה הבאה.

1. Application Layer – מופצת בעיקר בתחנות קצה ומכילה תוכנות להעברת מידע כגון Emule,Telnet וכו' המבוססות על פרוטוקולים שונים כגון HTTP,SMTP,FTP,DNS ועוד. החבילה ברמה זו מכונה לעתים בשם message.
2. Transport Layer – אחראית על סידור המידע והעברתו. שכבה זו בעצם מעבירה את המידע בין תחנות קצה של אפליקציות. מכילה את הפרוטוקולים TCP ו-UDP. חבילה ברמה זו מכונה לעתים בשם סגמנט – segment.
3. Network Layer – עוסקת בעיקר בניתוב החבילות. מחליטה אילו הודעות ייזרקו, לאן כל הודעה תלך וכו'. רמת ה-Network של תחנת מקור מקבלת חבילה מרמת ה-Transport שלה ומעבירה אותה (לפי הכתובת המצוינת בחבילה) לרמת ה-Network של תחנת היעד. חבילה ברמת זו מכונה datagram. ברמה זו מצוי פרוטוקול ה-IP המגדיר את השדות ב-datagram וגם את דרך ההתייחסות של נתבים ותחנות קצה לשדות אלה. בנוסף, רמה זו מכילה מספר פרוטוקולים הקובעים את המסלול שאותו ייקחו ה-datagrams בין תחנות המקור והיעד.
4. Link Layer – מטפלת בקישורים בין הנתבים. במילים אחרות, רמת ה-Network של קדקוד המקור מעבירה את ה-datagram לרמת ה-Link שלה. זו האחרונה מעבירה את החבילה (frame) לקדקוד הבא במסלול. בשלב זה, רמת ה-Link בקדקוד היעד מעבירה את החבילה לרמת ה-Network שלה.
פרוטוקולים ידועים ברמה זו הם Ethernet, PPP, WiFi ועוד. כאמור, בדרך בין המקור ליעד, חבילות עוברות דרך מספר לינקים ועל כן ייתכן שימוש במספר פרוטוקולים שונים במסלול. חבילה ברמה זו מכונה frame.
5. Physical Layer – הרמה הנמוכה ביותר, החומרה, אפיון הגלים וההעברה הפיסית של הביטים. שיטת הבצל – כאשר אפליקציה רוצה לשלחו חבילה, היא מוסיפה לה מזהה כלשהו וכך קורה גם בכל אחת מהרמות עד שהחבילה מגיעה לקדקוד הראשון ביעד (דרך ה-Physical Layer) ובשלב זה מתחילים ל"קלף" את התוספות שנוספו לחבילה ובכך להבין לאן לייעד אותה בדיוק.
6.

מסלול העברת חבילה – סכמה:



The Application Layer:

עקרונות בסיסיים וארכיטקטורה:

- מודל ה-Client-server: במודל זה תמיד קיים host הפועל תמיד (שרת) ומטפל בבקשות של לקוחות.
  שרת: תמיד מחובר, בעל כתובת IP קבועה, לעתים יש צורך בחוות שרתים לתמיכה בכמות בקשות גדולה.
  השרת ממתין לבקשה מהלקוח ולרוב אינו יכול ליזום תקשורת באופן עצמאי.
  לקוח: מתקשר עם השרת, יכול להיות מחובר לפרקים, יכול להיות עם כתובת IP משתנה, לא מתקשרים ישירות זה עם זה.
- מודל ה-P2P: מודל זה כמעט ואינו תלוי בשרת המחובר תמיד. המודל מסתמך על קשר ישיר בין זוגות של hosts שלא מחויבים להיות מחוברים באופן רציף ונקראים peers. (Bittorrent,Skype,Gnutella וכו').
  חסרון של מודל זה הוא הקושי במעקב אחר הלקוחות המשתנים כל הזמן.
- בנוסף, קיים מודל של הכלאה בין שני המודלים הקודמים – Napster. השרת מכיל אינדקסים של כל הקבצים אך ההורדה נעשית באמצעות P2P.

Process – תכנית שרצה אצל ה-host. באותו מחשב, שני תהליכים מתקשרים דרך מערכת ההפעלה. בין מחשבים שונים, תהליכים מתקשרים דרך העברת מסרים.
כדי שהלקוח יוכל לקבל מידע מהשרת ולהפך חייבים לפתוח ביניהם "צינור" הנקרא Socket. במילים אחרות, בתוך host מסוים, ה-Socket הוא הממשק המתווך בין רמת ה-Application ורמת ה-Transport.
לרמת ה-Application יש השפעה על רמת ה-Transport בבחירת הפרוטוקול וכן ביכולת המועטה לתת מספר פרמטרים שנקבעו ברמה זו.
כאשר פותחים Socket יש צורך להחליט באיזה פרוטוקול משתמשים.
IP Address – כתובת של 32 ביט המאפשרת לזהות מחשב ברשת. כדי לזהות תהליך בתוך מחשב מסוים יש צורך לתת לו גם Port number (16 ביט ==> יש אפשרות למקסימום 2^16=65536 תהליכים במחשב מסוים).

כל אפליקציה דורשת שירות שונה מרמת ה-Transport.
Data loss – יש אפליקציות שמאפשרות איבוד מידע מועט (אפליקציות של אודיו) ומצד שני יש כאלה שלא מאפשרות שום איבוד מידע (העברת קבצים למשל).
Timing – מספר אפליקציות מחייבות כי העברת המידע יהיה מיידי (טלפוניה, משחקים ברשת).
Bandwidth – יש אפליקציות הדורשות רוחב פס גבוה לצורך העברת המידע (מולטימדיה) ומצד שני יש אפליקציות גמישות יותר המשתמשות באיזה רוחב פס שהן מקבלות. (למשל eMail).

אפליקציות שונות משתמשות בפרוטוקולים שונים ברמת הTransport:

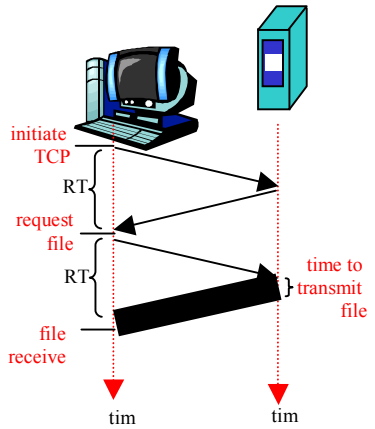
- TCP – דורש הקמת קשר בין השרת והלקוח. מספק תעבורה אמינה, הודעות לא נשלחות פעמים (Flow control), השולח נעצר אם יש עומס (Congestion control). פרוטוקול זה לא מספק Timing (כלומר החבילות לא בהכרח מגיעות עם שליחתן) וכן, לא מספק רוחב פס מינימלי כלשהו.
- UDP – העברת מידע לא אמינה, לא דורש הקמת קשר, אין flow control, אין congestion control טוב בעיקר לאפליקציות וידאו ולטלפוניה (VOIP).

מה מגדיר פרוטוקול ברמה זו?
- סוגי ההודעות שיכולות לעבור (למשל בקשה, תשובה וכו').
- תחביר ההודעה, אילו שדות היא מכילה.
- משמעות השדות וכיצד יש לפרשם.
- חוקים לקביעת זמן השליחה של הודעה או כיצד יש להגיב עליה.

HTTP – HyperText Transfer Protocol : (לרוב ב-port 80).
בנוי במודל של client-server כלומר הלקוח מתחבר ושולח בקשה והשרת שולח אובייקטים כתשובה.
מודל זה משתמש בפרוטוקול ה-TCP באופן הבא: הלקוח מבקש ליצור TCP connection עם השרת (port 80).
השרת מקבל את הבקשה. הודעות HTTP עוברות בין הלקוח והשרת ובסיום התהליך ה-TCP נסגר.

RTT – Round Trip Time – הזמן העובר משליחת הבקשה ע"י הלקוח ועד שהוא מקבל בחזרה תשובה מהשרת.
כאשר הלקוח מבקש אובייקט מסוים, עוברים 2RTT + הזמן שלוקח להוריד את הקובץ (אם הוא גדול).

Non-persistent and persistent connections:
- Non-persistent – אובייקט אחד לכל היותר מועבר דרך ה-TCP Connection. (HTTP 1.0).
  - 2RTT לכל אובייקט.
  - מערכת ההפעלה מקצה משאבים לכל TCP Connection.
  - מותר לפתוח מספר TCP Connections במקביל.
- Persistent – ניתן להעביר מספר אובייקטים דרך TCP Connection אחד. (HTTP 1.1).
  - השרת משאיר את החיבור אף לאחר שההודעה מועברת.
  - הודעות ה-HTTP הבאות נשלחות דרך אותו TCP Connection.
  - אם יש PipeLining :
    - הלקוח שולח בקשה מבלי לחכות שהבקשה הקודמת מטופלת.
    - 1RTT החל מהאובייקט השני.
  - אם אין PipeLining:
    - הלקוח שולח בקשה חדשה רק לאחר שקיבל תשובה עבור הקודמת.
    - 1RTT לכל אובייקט.



ב-HTTP יש שני סוגי הודעות: Request ו-Response
Request: השורה הראשונה נקראת request line ויכולה להכיל למשל את הפקודות GET,POST,HEAD

לאחר מכן מופיעות 4 שורות של ה-header המגדירות את המיקום של האובייקט של הבקשה (למרות שיש כבר TCP Connection בין המוקדים עדיין יש צורך בשורה הראשונה לצרכי caching). בנוסף יש שורות נוספות המגדירות את המשך הפעולה, הדפדפן שמבצע את הבקשה והשפה המבוקשת.
דוגמא:

GET /somedir/page.html HTTP/1.1 – זוהי פעולה של בקשת דף אינטרנט
Host: www.someschool.edu – כאן נמסר מיקומו של הדף
User-agent: Mozilla/4.0 – איזה דפדפן ביצע את הבקשה
Connection: close – מה יש לעשות לאחר הורדת הדף
Accept-language:fr – מהי השפה המועדפת להורדת הדף

Response: מסביר את עצמו ☺

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...

Cookies: אתרים רבים משתמשים בהן. ישנם 4 מרכיבים הכרחיים לפעילותן:

1. Cookie header line – הנמצאת בראש ה-HTTP Response.
2. Cookie header line – הנמצאת בראש ה-HTTP Request.
3. Cookie file – הנמצא אצל ה-host ומתופעל ע"י הדפדפן שלו.
4. מאגר נתונים הנמצא אצל השרת.

איך זה פועל - נניח שאיציק מתחבר בפעם הראשונה לאתר של eBay ורוצה לקנות נעלי בית של קיפי. ברגע שהוא שולח את בקשת ה-HTTP הראשונה לאתר, זה האחרון מייצר קובץ עם unique ID ומכניס אותו למאגר הנתונים של האתר. בנוסף, האתר מכניס למחשב של איציק Cookie ובו כל המידע על הרכישות האחרונות שלו, העדפותיו, פרטיו וכו'. כך, בפעם הבאה שאיציק ייכנס לאתר ויגיש בקשה, באמצעות מאגר המידע האתר ידע איפה למצוא את ה-Cookie, ישלוף אותה ויפיק את המידע הרלבנטי.

ראוי לציין את הבעיתיות בשיטה זו. כאמור, ה-Cookie יושבת רק במחשב שביצע את הפניה הראשונה לאתר כך שאם איציק ייכנס בעוד שבוע לאותו אתר, אך דרך המחשב הנייד שלו, האתר לא יידע עליו את הפרטים ולא יוכל לחלץ את ה-Cookie מהמחשב השני.

Web Caches (proxy server):
המטרה – לענות על בקשות הלקוח מבלי להטריד בכלל את השרת הראשי.
תחילה, על המשתמש להגדיר את הדפדפן שלו לפעול דרך ה-cache.
בשלב הבא, הדפדפן מעביר כל בקשה של הלקוח ל-cache. אם האובייקט שם, הוא מוחזר ללקוח באופן מיידי, אחרת, ה-cache פונה לשרת הראשי, מקבל ממנו את האובייקט ומעבירו ללקוח. בעצם, ה-cache מתפקד הן כלקוח והן כשרת.
אז מה זה נותן בכלל?

א. מקטין את הזמן לקבלת תשובה עבור בקשה של לקוח מסוים.
ב. מקטין את התעבורה היוצאת/נכנסת ממוסד, חברה, ארגון מסוים.
ג. מקטין את התעבורה הכוללת באינטרנט ובכך משפר את התפוקה והביצועים.

בנוסף, ב-HTTP ישנו מנגנון המגדיר ל-cache מתי להביא את הדף ומתי לא בכל הקשור לעדכון. מנגנון זה מתבטא דרך השדה – If-modified-since <date>. במילים אחרות, ה-Cache שולח בקשה לשרת ומוודא האם האובייקט השתנה מאז התאריך שנקבע. אם התשובה שלילית, השרת לא מחזיר שום דבר וכך ה-cache יודע להעביר את אותו אובייקט מבוקש ללקוח. אם האובייקט כן השתנה, השרת מחזיר את האובייקט המעודכן ל-Cache וזה מעבירו ללקוח.

דוגמא במצגת בעמ' 40.

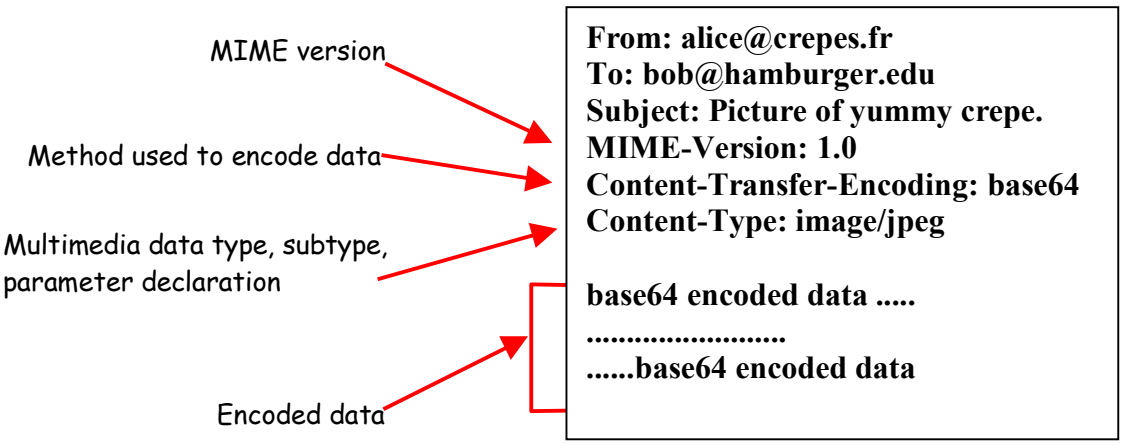
FTP – File Transfer Protocol: זהו פרוטוקול להעברת קבצים המבוסס על מודל ה-client-server. (port 21).
פרוטוקול זה מערב יצירת שני TCP Connections במקביל. הראשון (control connection) מאותחל ע"י הלקוח ונוצר לצורך העברת המידע הנחוץ לצורך העברת הקובץ (כגון שם התיקייה, סיסמאות וכו') בעוד שהשני (the data connection) מאותחל ע"י השרת עצמו ונוצר לשם העברת הקובץ עצמו. **כאן יש חריגה מהמקובל שכן <u>השרת</u> הוא זה שיוצר את הקשר עם הלקוח לצורך העברת הקובץ והוא גם זה שסוגר אותו**. יש לציין כי כאשר מעוניינים בהעברה של יותר מקובץ אחד, החיבור הראשון נשאר בעינו אך עבור כל קובץ יש לאתחל את החיבור השני (The data connection). מסיבה זו, בזמן התהליך על השרת לשמור מידע על המצב (The state) של הלקוח כדי לקשר אותו עם ה-control connection. (יש לשמור את שמות התיקיות, הסיסמאות, דרכי ההזדהות ועוד).

Electronic Mail:
מורכבים מ-3 חלקים עיקריים:
א. User agents – בעזרתם ניתן לקרוא את המייל (כגון outlook, pine וכו').
ב. Mail servers – השרת שבו לכל לקוח יש Mailbox. כאן יש תור של מיילים יוצאים ונכנסים.
ג. SMTP – הפרוטוקול שבעזרתו נשלחים המיילים מן השרת של השולח לשרת המקבל.

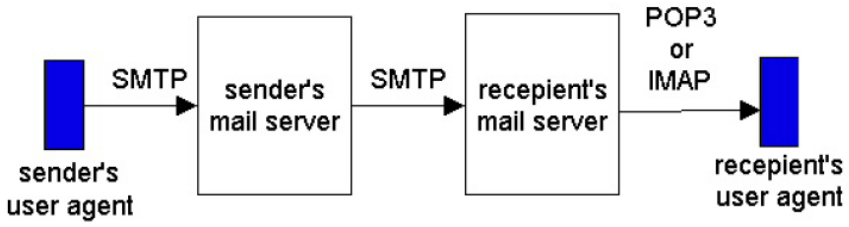
SMTP – Simple Mail Transfer Protocol: משמש להעברה אמינה של מיילים משרת ללקוח. (לרוב port 25).
המיילים עוברים בצורה ישירה בין השרת של שולח המייל לזה של המקבל.
3 שלבים להעברת המייל:
א. Handshaking – הכרה בין 2 השרתים.
ב. Transfer – העברת המייל.
ג. Closure – סגירת הצינור.
ההודעות נשלחות בצורה של 7-bit ascii.
דוגמה לשליחת מייל מאליס לבוב:
1. אליס משתמשת ב-user agent שלה לכתיבת מייל ל-bob@cs.huji.ac.il.
2. ה-user agents של אליס מעביר את המייל לשרת שלה, שם הוא נכנס לתור.
3. ה-SMTP של השרת של אליס (נחשב כ-client) פותח חיבור TCP לשרת של בוב.
4. המייל מועבר מהשרת של אליס (הנחשב כ-client) לשרת של בוב דרך חיבור ה-TCP.
5. שרת המייל של בוב שם את המייל ב-mailbox שלו.
6. בוב נעזר ב-user agent שלו כדי לקרוא את המייל.

<u>השוואה בין SMTP ו-HTTP:</u>
☒ HTTP – פרוטוקול שמושכים ממנו. (מישהו בונה דף אינטרנט ובעזרת HTTP קוראים אותו אלינו. באופן יותר פרטני, חיבור ה-TCP נפתח ע"י המכונה שמעוניינת **לקבל** את הקובץ).
SMTP – פרוטוקול שדוחפים בעזרתו. (השרת של השולח "דוחף" את הקובץ לשרת המקבל. באופן יותר פרטני, חיבור ה-TCP נפתח ע"י המכונה שמעוניינת **להעביר** את הקובץ).
☒ קידוד:
SMTP – מחייב שה-header וגוף ההודעה יהיו ב-7-bit ascii (כלומר הודעה בצרפתית למשל חייבת להיות מקודדת ל-7-bit ascii) ואילו ב-HTTP אין את ההגבלה הזו.
☒ גם SMTP וגם HTTP משתמשים בחיבורים שהם מסוג persistent.
☐ כדי לסמן את סוף ההודעה ב-SMTP חייב להופיע CRLF.CRLF (\n . \n).
☒ ב-HTTP כל אובייקט (תמונה) נשלח בהודעת תגובה משלו בעוד שב-SMTP כל התמונות והטקסטים נשלחים בהודעה אחת.

<u>MIME – Multipurpose Internet Mail Extensions:</u>
אלו הן שורות נוספות ב-header של המייל שמטרתן שליחת דברים השונים מטקסט ascii רגיל (תמונות, סרטים, טקסטים בשפות השונות מאנגלית וכו').
למשל:



:Mail Access Protocols



כפי שצוין לעיל, SMTP "דוחף" את המידע / המייל המבוקש ועל כן נוצרת בעיה מסוימת בהגעת המייל ל- user agent של הלקוח המקבל (צד ימין בכחול). כאן באים לידי ביטוי פרוטוקולי MAP כגון POP3, IMAP ו-HTTP. בגדול, המטרה של פרוטוקולים אלה היא להעביר את המיילים משרת הדואר אל המחשב הפרטי של משתמש כלשהו.

:POP3 - Post Office Protocol

פרוטוקול גישה לאינטרנט פשוט יחסית ועקב כך מוגבל במידת מה. אופן הפעולה קצר, תמציתי ובנוי מ-2 חלקים עיקריים – שלב אימות (authorization phase) ושלב ההעברה (transaction phase). יש גם שלב של עדכון (Update) אבל לא נדבר עליו.

פירוט: בשלב הראשון, POP3 מתחיל כאשר ה-user agent (הלקוח) פותח חיבור **TCP** לשרת המייל דרך port 110. כאן ה-user agent שולח את שם המשתמש והסיסמא בכדי להזדהות. לאחר קבלת אישור מהשרת, מתחיל השלב השני שבו ה-user agent מתחיל לקבל את ההודעות. בסיומו של התהליך, הלקוח שולח פקודת quit המביאה לסיום התהליך. כעת, שרת המייל מוחק את ההודעות שסומנו קודם לכן למחיקה. לרוב, פרוטוקלי POP3 משתמשים במנגנון הנקרא Download-And-Delete שמאפשר לקרוא מייל פעם אחת בלבד (כי הוא נמחק מיד לאחר הקריאה). בנוסף, קיים מנגנון הנקרא Download-And-Keep המאפשר לקרוא מייל יותר מפעם אחת ובכך מאפשר בעצם לקרוא מייל ממספר מחשבים שונים (מחשב בעבודה, מחשב ביתי, נייד וכו').

:IMAP – Internet Mail Access Protocol

מורכב יותר ומתוחכם יותר מ-POP3 ועל כן המימוש שלו מסובך בהרבה. מאפשר לבצע מספר מניפולציות על מיילים בשרת עצמו. ההבדל העיקרי בין IMAP ו-POP3 הוא שבראשון ניתן ליצור תיקיות ולשמור מיילים בשרת עצמו ובכך לקבל נגישות אליהם ממחשבים שונים.

:DNS – Domain Name System

ה-DNS היא מערכת לתיאום והמרת שמות של שרתים / ו-hosts. לצורך כך, לכל host יש 2 שמות, השם המילולי וכתובת ה-IP. למה בכלל צריך את כתובת ה-IP? מאחר ולנתבים קשה להתמודד עם אותיות. כדי לתאם בין שני השמות, יש צורך במערכת אשר תתרגם את השמות המילוליים לכתובות IP. זוהי המשימה העיקרית של DNS.

פירוט:

1. DNS הוא מסד נתונים הממומש ע"י היררכית שרתי DNS.
2. DNS הוא פרוטוקול בשכבת האפליקציה המאפשר ל-hosts לתקשר עם מסד נתונים זה.

כתובת ה-IP הינה בת 32 ביט ואליה שולחים את החבילות.

מהם השירותים שמספק ה-DNS:

א. תרגום בין hostname לבין כתובת IP.
ב. מתן שמות בדויים ל-hosts או לשרתי מייל.
ג. ישנם אתרים ענקיים (למשל cnn.com) המשכפלים את עצמם על מספר שרתים. DNS מאפשר להוריד עומס מהשרת הראשי ע"י הפניית הבקשות לשרתים המשוכפלים.

המבנה של מערכת ה-DNS מסובך להפליא. נשאלת השאלה למה לא למרכז את כלל השמות למפה אחת שתעשה את כל העבודה? במצב זה, לקוחות פשוט יפנו את בקשותיהם לשרת ה-DNS היחיד וזה יענה להם. למרות הפשטות שיש במבנה זה, הוא לא מתאים לתשתית האינטרנט הקיימת כיום והסיבות לכך הן:
א. נקודת קריסה אחת.
ב. עומס אדיר. (מיליארדי בקשות)
ג. מרחק. אם שרת ה-DNS ימוקם בניו-יורק, בקשות מאוסטרליה יאלצו לעבור מרחק אדיר.
ד. תחזוקה. יהיה מאוד קשה להוסיף ערכים חדשים למערכת.

לכן, כדי לתחזק את מערכת ה-DNS ולהתמודד עם הסוגיות לעיל, בנו מסד נתונים היררכי המושתת על מספר רב של שרתים המפוזרים בכל רחבי העולם. החלוקה נעשית באופן הבא:
- ☒ Root DNS servers – 13 בעולם, רובם בצפון אמריקה.
- ☒ TLD – Top Level Domain servers – שרתים האחראים על שמות כגון com,org,net,edu,gov,il,uk וכו'.
- ☐ Authoritative DNS servers – אלו הם שרתי ה-DNS של ארגונים, המספקים שמות לצורך מיפוי כתובות IP. (google, msn וכו').

בנוסף על 3 הנ"ל, קיים גם שרת DNS הנקרא local DNS server. שרת זה לא שייך להיררכיה שהוצגה לעיל אך קשור אליה בצורה מלאה. כל ספק אינטרנט (אוניברסיטה או חברה כלשהי) מחזיק שרת שכזה. כאשר host מתחבר לספק, זה האחרון מקצה לו כתובת IP של אחד מה-local DNS servers שלו (לרוב, זה הקרוב אליו מבחינה פיזית). כאשר ה-host שולח בקשה, היא מופנית קודם ל-local DNS אשר משמש כמתווך (proxy) ומעביר אותה בעצמו לכל אחד משלבי היררכית שרתי ה-DNS שהוצגה לעיל לפי הסדר.
יש 2 גישות שבהם פועלים ה-Local DNS servers:
א. Recursive query – גישה שבה ה-host מעביר את הבקשה שלו ל-loacl DNS server והוא מעביר את הבקשה שלו לכל אחת משלבי ההיררכיה.
ב. Iterative query – גישה שבה כל שרת מחזיר את השרת הבא שיש לפנות אליו ("אני לא יודע, אבל תשאל את השרת הזה").

DNS Caching:
הרעיון העיקרי שעומד מאחורי תפיסה זו הוא שכאשר שרת DNS מקבל תשובה על בקשה שהגיש, הוא שומר אותה גם בזיכרון האישי שלו. מאחר ושמות ה-hosts וכתובות ה-IP משתנים, יש זמן שלאחריו הזיכרון של שרתי ה-DNS מתאפס (לרוב יומיים).

DNS – Records – בהודעת DNS, יש רשומות. הן נראות כך:
(name, value, type, ttl) – כאשר ה-ttl הוא הזמן לסיום תוקפה של רשומה זו.
ערך ה-type הוא זה שמכתיב את שני הערכים הראשונים (name, value) לפי השיטה הבאה:
type = A :
- ☒ Name = hostname
- ☒ Value = IP address

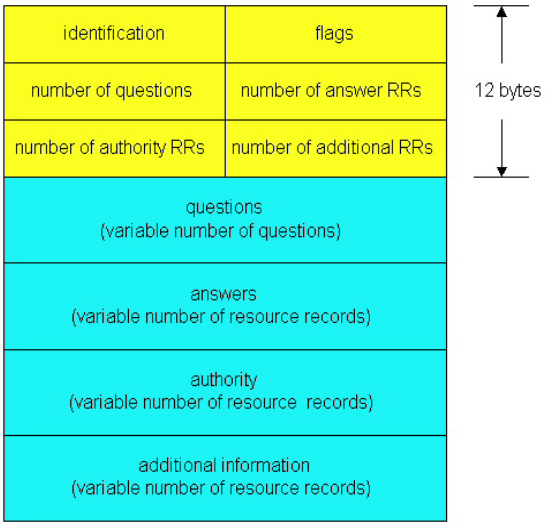
למשל: (relay1.bar.foo.com, 145.43.33.554, A)
type = NS :
- ☒ Name = domain
- ☐ Value = IP address of authoritative name server for this domain

למשל: (foo.com, dns.foo.com, NS)
type = CNAME :
- ☒ Name = alias name for some canonical name
- ☒ Value = canonical name

למשל : (foo.com, relay1.bar.foo.com, CNAME)
type = MX:
- Name = alias name
- Value = canonical name of a mail server

למשל : (foo.com, mail.bar,foo.com, MX)

DNS protocol, messages:
ישנם סוגים שונים של הודעות: שאילתא (query) והודעות תגובה (reply).
תחילה יש 16 ביט של הזדהות (identification) – למשל כדי להבדיל בין בקשות שונות ממשתמשים שונים.
בנוסף, יש את שדה ה-flags – סוג הבקשה (שאלה / תגובה), סוג הרקורסיה המבוקשת.
לסיום, יש 8 שדות המכילים מידע על השאילתא, התגובות, רשומות אחרות, ומידע נוסף על שמות, כינויים וכו'.



דוגמא להכנסת רשומות לתוך DNS:
נניח ויש לנו חברה חדשה העונה לשם Network Utopia. הדבר הראשון שנרצה לעשות, זה לרשום את החברה תחת הדומיין networkUtopia.com ב-Registrar. זה האחרון הוא איגוד מסחרי המוודא את הייחודיות של שם הדומיין ומכניס אותו לתוך מאגר הנתונים של ה-DNS. כאשר מבקשים מה-Registrar לעשות זאת, יש לספק לו גם את השמות וכתובות ה-IP הראשיים והמשניים של שרתי ה-DNS שלנו.
פירוט:
נניח והשמות וכתובות ה-IP (הראשיים והמשניים) שלנו הם:
Dns1.networkUtopia.com, dns2.networkUtopia.com, 212.212.212.1, 212.212.212.2 אזי עבור כל אחת מהרמות (הראשית והמשנית) ה-Registrar יוודא שבשרתי ה-TLD יוכנסו רשומות מדויקות עבור החברה שלנו הכוללות את הערכים NS ו-A כפי שראינו בעמוד קודם.
כלומר:
עבור השרת הראשי יוכנסו הרשומות:

(networkUtopia.com, dns1.networkUtopia.com, NS)
(dns1.networkUtopia.com, 212.212.212.1, A)

P2P – file sharing:

מה זה בעצם?
נניח ואיציק שאשו רוצה להוריד שיר. הוא מתחבר לתוכנה (ברמת האפליקציה!) בה הוא מעוניין להשתמש ומחפש את השיר (נניח, "כוס קפה עם סיגריה" של קוקו מאילת). דרך התוכנה, איציק בוחר להוריד את השיר מאורטל.
השיר עובר מהמחשב של אורטל למחשב של איציק. הקטע המעניין פה הוא שתוך כדי ההורדה, משתמשים אחרים יכולים כבר להוריד את השיר מאיציק מה שאומר, שב-P2P, הלקוח הוא גם שרת!!.

ל-P2P יש מספר גישות עיקריות:
1. The centralized directory approach:
למשל: Napster – החברה המפורסמת הראשונה בתחום שיתוף הקבצים:

כאשר peer מתחבר, הוא מדווח לשרת הראשי את כתובת ה-IP שלו ואת רשימת הקבצים המצויים ברשותו. כעת, איציק יוכל להוריד את השיר המבוקש מ-peer מסוים, אם השיר קיים ברשימת השירים שלו.
אמנם העברת הקבצים היא מבוזרת אך המידע על מיקום הקבצים הוא ריכוזי ביותר ← מספר בעיות:

א. קריסה של השרת המרכזי עלולה להוביל לקריסת המערכת כולה.
ב. מכיוון שיש שרת יחיד יש עליו עומס רב מה שמוביל לביצועים פחות טובים.
ג. זכויות יוצרים. (אין פיקוח על הקבצים המשותפים).

2. The Query flooding approach:
למשל: Gnutella – תכנה פשוטה יחסית אך בעלת יתרונות רבים.
בגישה זו אין שרת מרכזי יחיד שעליו יושב כל המידע בדבר הקבצים. בנוסף, הפרוטוקול הוא פומבי ולקוחות יכולים אף לשנותו.
אז איך זה עובד – הכל מתבסס על גרפים !
נגדיר כי ישנה קשת בין peer A ובין peer B אם ישנו TCP connection פעיל ביניהם. ככה נוצרת רשת של אנשים המעוניינים לשתף קבצים. (חשוב להבין כי המצאות של קשת בין 2 peers איננה מחייבת קיום פיזי של link בין שני ה-peers אלא איזשהו מסלול של links ביניהם). לרוב, ל-peer מסוים יהיו כ-10 שכנים.
אז השיטה היא כזו, נניח שאיציק מחפש שיר. הוא מכניס את השאילתא שלו וזו עוברת לכל השכנים של איציק. כל שכן מעביר את השאילתא לכל שכניו ובנוסף בודק אם השיר נמצא ברשימת השירים שהוא משתף. אם כן, הוא מחזיר תשובה על כך דרך חיבורי ה-TCP שכבר קיימים לאיציק. כך מתקבלת הזרימה המהירה של השאילתא. מצד שני, יש פה בעייתיות הנובעת מכך שהשאילתא עוברת לכל הקדקודים ברשת. כמו ב-Napster גם כאן, העברת **הקובץ** נעשית בעזרת פרוטוקול ה-HTTP.
Gnutella message header:

Descriptor Header

	Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length
Byte offset	0 – 15	16	17	18	19 – 22

Descriptor ID *A 16-byte string uniquely identifying the descriptor on the network*

Payload Descriptor
0x00 = Ping
0x01 = Pong
0x40 = Push
0x80 = Query
0x81 = QueryHit

בנוסף: TTL – כמות המעברים שהחבילה מבצעת. כל קדקוד מפחית את ה-TTL ב-1.
HOPS – מספר הפעמים שהחבילה התקדמה בגרף.
לאורך תנועת החבילה, השדות (TTL ו-HOPS) חייבים לקיים את המשוואה-TTL(0)=TTL(i)+HOPS(i)
השדה האחרון – ה-Payload Length מייצג את אורך המידע שנמצא לאחר ה-header. כלומר ה-header הבא נמצא בדיוק Payload Length בתים מסוף ה-header הנוכחי.
פירוט על סוגי ההודעות ברשת Gnutella:
Ping:
זוהי הודעה שמאפשרת למשתמש ברשת לאתר שכנים חדשים / חיבורי TCP חדשים.
Pong:
הודעה מעט יותר מורכבת. נשלחת כתגובה על הודעת Ping וכוללת מספר שדות ומידע על המשתמש.

Pong (0x01)

	Port	IP Address	Number of Files Shared	Number of Kilobytes Shared
Byte offset	0 – 1	2 – 5	6 – 9	10 – 13

תחילה מספר ה-port שאליו ניתן יהיה להתחבר.
כתובת ה-IP של ה-host המגיב. שני השדות הבאים ברורים!

Query:

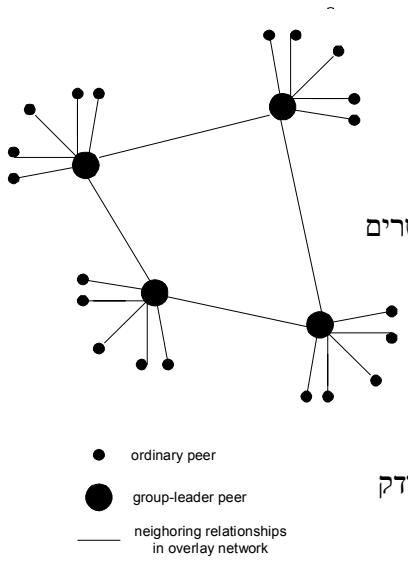
Query (0x80)

Byte offset	0	1	2	...
	Minimum Speed		Search criteria	

השדה הראשון מגדיר מהירות מינימלית שבה חייב לעמוד משתמש מסוים כדי לענות.
הודעה זו נשלחת ע"פ חיבור TCP. כל peer מעביר את ההודעה הזו הלאה. הודעה ה-Query Hit נשלחת בחזרה באותו מסלול, רק בסדר הפוך.

Query Hit message:
לקרוא שקופיות מספר 88-89

לאחר הגעת ה-hit בחזרה לשואל השאילתא, ניתן לשלוח את הקובץ.



Gnutella V.2: גרסא מחודשת של הרשת.

כל peer הוא או superNode או מקושר ל-superNode דרך חיבור TCP.
בנוסף, ה-superNodes מקושרים זה לזה גם באמצעות חיבור TCP.
השיפור פה הוא שה-superNode יודע את כל הקבצים של ה-peers המקושרים אליו ובכך יש חסכון גדול בזמן.

אז איך זה עובד?
כאשר peer מתחבר, הוא מעדכן את ה-superNode המקושר אליו בכל המידע על הקבצים אותם הו א משתף.
החיפוש נעשה בצורה הבאה:
לקוח שולח את השאילתא שלו ל-superNode המקושר אליו. זה האחרון בודק אם הרשומה נמצאת אצלו ומעביר את השאילתא ל-superNodes הבאים.
לסיום, הלקוח בוחר את הקובץ אותו הוא מעוניין להוריד.

:eMule
לקרוא מצגת בשקופיות 94-97.

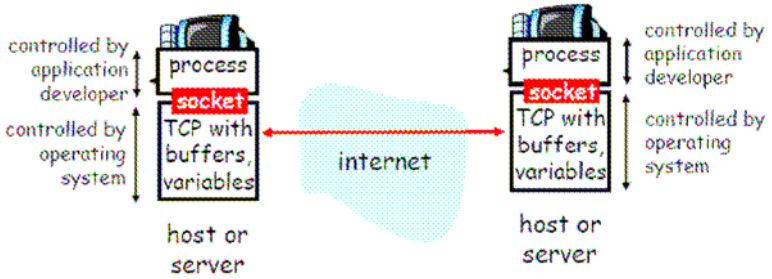
:BitTorrent
שיטת ה-P2P הפופולארית ביותר כיום לשיתוף והעברת קבצים. Torrent הוא אוסף כלל ה-peers המשתפים קובץ מסוים ברשת. לרוב, הקבצים מורדים בחתיכות (Chunks) בגודל של 256kb.
איך זה פועל?
כאשר peer מסוים מצטרף להורדת הקובץ, הוא לא מחזיק ברשותו שום חתיכה של הקובץ. עם הזמן, הוא מקבל חתיכות נוספות של הקובץ ומשתף אותן תוך כדי הורדתן. לכל torrent יש קדקוד מסוים שנקרא tracker. כל peer שמצטרף ל-torrent מתקשר תחילה עם ה-tracker ומודיע לו על התחברותו. הוא ממשיך לדווח לו על מצבו בכל תקופת זמן קצובה מראש. באופן זה, ה-tracker מנהל מעכב שותף אחר כל ה-peers הקשורים ל–torrent המדובר ומשתפים את הקובץ.
כעת, כאשר איציק מצטרף ל-torrent, ה-tracker שולח לו כתובות IP של תת קבוצה של peers (peer list) מתוך אלה שמקושרים אליו (נניח 50). באופן זה, איציק מסוגל ליצור 50 חיבורי TCP עם כל ה-peers שנשלחו אליו.
צריך לזכור כי תת הקבוצה הזו היא דינאמית מאוד שכן peers יכולים גם לעזוב את הקבוצה (נניח אם הם סיימו להוריד את הקובץ) וגם להצטרף אליה. בכל רגע נתון, איציק צריך לבקש מה-peers המקושרים אליו את החבילות / חתיכות החסרות לו להשלמת הקובץ.

השאלה הנשאלת היא כזו; אילו חבילות על איציק לבקש תחילה וממי?
התשובה היא שעל איציק לבקש תחילה מה-peers המקושרים אליו את החבילות הכי נדירות ברשת (כלומר אלו שנמצאות אצל הכי מעט peers). הסיבה לכך היא ששיתוף מהיר של החבילות הנדירות יוביל לכך שהן תהפוכנה ללא נדירות ברשת ובאופן זה כל החבילות תעבורנה לכל ה-peers.
בנוסף, כדי לקבוע עם אילו peers איציק צריך לתקשר, יהיה עליו לנהל מעכב אחר ה-peers שמספקים לו את קצב ההורדה הגבוה ביותר. המערכת פועלת כך שבכל 10 שניות, איציק מחשב מחדש את קצבי ההורדה ומשנה בהתאם את 4 ה-peers שמהם הוא מוריד. בנוסף, אחת ל-30 שניות, איציק בוחר peer נוסף באופן רנדומלי (נניח אורטל) ומתחיל לשלוח אליה חבילות. באופן זה, איציק יכול להפוך לאחד מ-4 ה-peers שאורטל בוחרת להיות אלה שהיא מורידה מהם ובכך תתחיל גם לשלוח אליו חבילות בתמורה. לסיכום, אחת ל-30 שניות, איציק יבחר בצורה רנדומלית שותף העברות חדש ויצור איתו קישור. אם שני ה-peers מרוצים מקצבי ההעברה הם ישימו אחד את השני ברשימת 4 ה-peers המוצלחים להורדה וימשיכו לשתף את הקובץ עד לסיום ההעברה או עד שאחד ה-peers ימצא שותף טוב יותר.

הערה: ברוב גישות ה-P2P קיימת התופעה של free-riding כלומר peers השותפים לרשת העברת הקבצים רק מכיוון אחד – הורדת קבצים. ברשת BitTorrent גישה זו לא מתאפשרת שכן כדי שאיציק יוריד קבצים מאורטל בקצב משביע רצון, עליו גם להעלות בחזרה בקצב שכזה.

Socket programming:

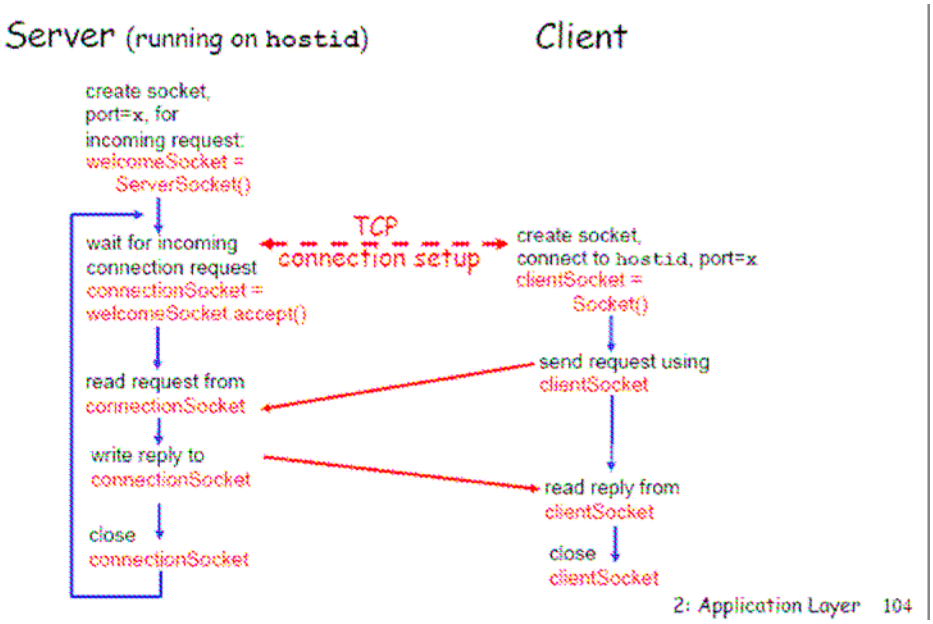
בתת-פרק זה נראה כיצד לבנות אפליקציות של לקוח / שרת המתקשרות דרך sockets.
Socket הוא מעין "דלת" / "שער" הנשלט ע"י מערכת ההפעלה ומאפשר לאפליקציה מסוימת לשלוח או לקבל מידע מאפליקציה אחרת.
כעקרון, יש 2 סוגים של הודעות שניתן להעביר דרך sockets: datagrams **Reliable** / **Unreliable**.
נתחיל בדיון על העברה **אמינה** של קבצים – כלומר Socket programming using TCP:
הסכימה בגדול נראית כך:



כיצד זה פועל? (נקודות חשובות)

- ☒ הלקוח הוא זה שיוצר קשר עם השרת (לכן השרת חייב לרוץ לפני יצירת הקשר).
- ☒ השרת חייב ליצור socket המאפשר קבלת לקוחות.
- ☒ הלקוח מתחבר אל השרת באופן הבא:
  - ❖ יוצר client local TCP socket. בשלב זה, חיבור ה-TCP של הלקוח מתקשר לחיבור ה-TCP של השרת.
  - ❖ מגדיר כתובת IP ואת מספר ה-port שיתחבר אל השרת.
- ☒ בעת החיבור של הלקוח, חיבור ה-TCP של השרת יוצר socket המאפשר לו לתקשר עם הלקוח. שיטה זו מאפשרת לשרת לתקשר עם מספר לקוחות במקביל.

<u>דוגמה לתהליך:</u>



לברר אם צריך להוסיף את הקוד ב-JAVA עבור הלקוח והשרת ב-TCP (שקופיות 105-108).

נעבור כעת לדבר על Socket programming using UDP:

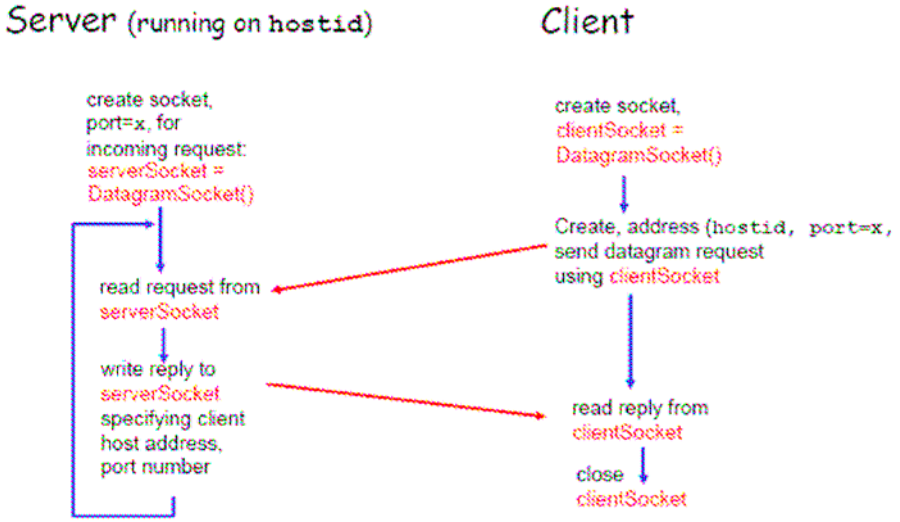
אז מה זה בכלל ה-UDP הזה?

זה אומר שבעקרון אין חיבור בין הלקוח ובין השרת כלומר ה-UDP הוא פרוטוקול לא אמין להעברת קבוצות של בתים (לרוב נקראות Datagrams) בין הלקוח והשרת.

מאפיינים עיקריים:

- ☒ אין לחיצת ידיים – אימות בין הלקוח והשרת.
- ☒ השולח מצרף לכל הודעה את כתובת ה-IP וה-port של היעד אליו הוא שולח.
- ☒ המקבל חייב לחלץ את המידע הזה מן ההודעה.
- ☐ **<u>חשוב</u>**: ב-UDP המידע שעובר יכול להגיע בצורה לא מסודרת ואף להאבד!

<u>אופן התהליך:</u>



לברר אם צריך להוסיף את הקוד ב-JAVA עבור הלקוח והשרת ב-UDP (שקופיות אחרונות במצגת).

Transport Layer

רמת הטרנספורט מספקת קישור לוגי בין תהליכים שרצים על מחשבים שונים. הפרוש של קישור לוגי הוא, שאומנם שהמחשבים לא באמת מחוברים פיסית זה לזה, מבחינת האפליקציה שרצות זה כאילו הם היו מחוברים פיסית.
האפליקציות משתמשות בקישור הלוגי כדי לשלוח הודעות זו לזו ללא צורך לדאוג מהפרטים של מעבר ההודעה בין מחשבים מרוחקים.
פרוטוקולים של רמת הטרנספורט ממומשים בתחנות הקצה **ולא** בראוטרים בדרך.
מחשב מסויים יכול להציע יותר מפרוטוקול טרנספורט אחד. לדוגמא באינטרנט משתמשים גם ב TCP וגם ב UDP .
אנלוגיה: (לא כ"כ חשוב)
נגיד וקימים 2 בתים מרוחקים זה מזה בכל בית יש 12 ילדים שכותבים מכתבים לכל 12 הילדים של הבית השני כל שבוע. כל מכתב במעטפה נפרדת. בכל בית ילד אחד (עליזה בבית אחד ובוב בשני) אחראי על לאסוף את כל המכתבים מכל שאר ילדי הבית, ולמסור אותם לדואר. בנוסף עליזה ובוב אחראים על איסוף המכתבים מהדואר וחלוקה שלהם בין ילדי הבית. באנלוגיה: בתים(מחשבים או hosts), ילדים(תהליכים), הדואר(network layer protocol) , עליזה ובוב(transport layer protocol).

Multiplexing and Demultiplexing

הרעיון הוא שכאשר מגיעה הודעה למחשב מסויים הוא צריך לדעת לאזה תהליך לקשר את ההודעה הזו. כמובן שכתובן IP אינה מספיקה. לכן לכל סגמנט(הודעה) מוסיפים שדה שמסמן לאיזה תהליך הוא מיועד. המשימה של שליחת ההודעה לתהליך המתאים נקראת demultiplexing ואילו המשימה של יצירת הסגמנט עם השדה שמסמן את היעד נקראת multiplexing . הפרוטוקולים UDP ו TCP מבצעים משימות אלו ע"י הוספת 2 דברים לheader : מספר פורט היעד, ומספר פורט המוצא. (the source port number field and the destination port number field).

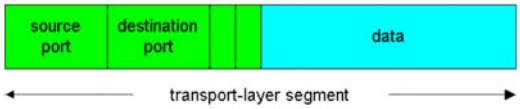
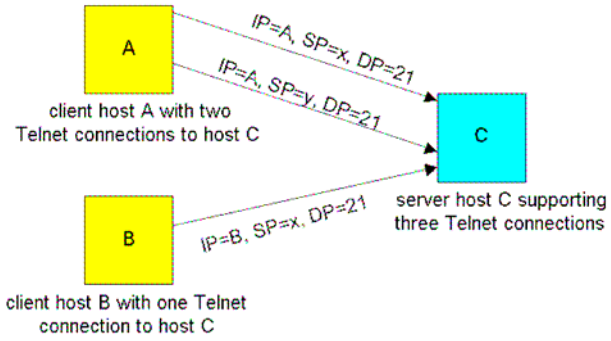


Figure 3.2-1: Source and destination port number fields in a transport layer segment.

שדות אלה **בשילוב עם כתובת ה IP** מזהים באופן יחודי למי מיועדת ההודעה. (שאלה לקורא - למה צריך גם את כתובת ה IP ?תשובה בתרשים)



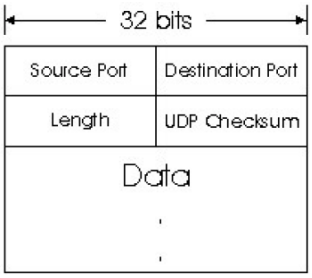
UDP

UDP עושה הכי מעט שפרוטוקול טרנספורט יכול לעשות, מלבד mux/demux הוא מוסיף מנגנן קל של בדיקת טעויות. זהו פרוטוקול לא אמין שיתכן בו איבוד הודעות, קבלת הודעות לא בסדר הנכון ובנוסף אין פתיחת קשר לכן יתכן שליחה לשרת מת. נציין כי DNS הוא דוגמא לשימוש ב UDP כי השאלה היא קצרה ויש אפשרות לשלוח ל secondary nameserver אם לא התקבלה תשובה. אפשר לממש מגנן אמין גם ב UDP אך ממש זה צריך להיות חלק מהאפליקציה. (כמו בתרגיל 3)

הסיבות **כך** להשתמש ב UDP הם : צורך פחות משאבים, אפשר לשלוח יותר בו זמנית, יותר מהיר. ובפירוט:

- No connection establishment: לכן UDP לא מבזבז זמן ליצור קישור. DNS היה הרבה יותר איטי אם הוא היה יוצר קישור כל פעם.
- No connection state: בניגוד ל TCP ששומר הרבה מידע נוסף (יפורט בהמשך). UDP הרבה יותר קל על תחנות הקצה.
- Small segment header overhead: גודל header של TCP הוא 20 בייט ואילו של UDP הוא 8 בייט.
- Unregulated send rate: המהירות שבה UDP שולח את המידע קשורה רק למהירות בה האפליקציה מייצרת את המידע ורוחב הפס. אין שם בקרה על המהירות.בניגוד ל TCP.

מבנה סגמנט הUDP



ב header יש רק 4 שדות שכל אחד הוא בגודל 2 בייט.

Checksum:

משמש לבדיקת טעויות. השולח מחלק את ההודעה לסגמנטים של 16 ביט. מבצע XOR בין כולם על התוצאה עושה NOT וזה הchecksum . ואילו המקבל מבצע את אותה הפעולה וכולל גם את ה checksum בXOR, הוא מצפה לקבל 1111111111111111 אם לא היתה שגיאה.

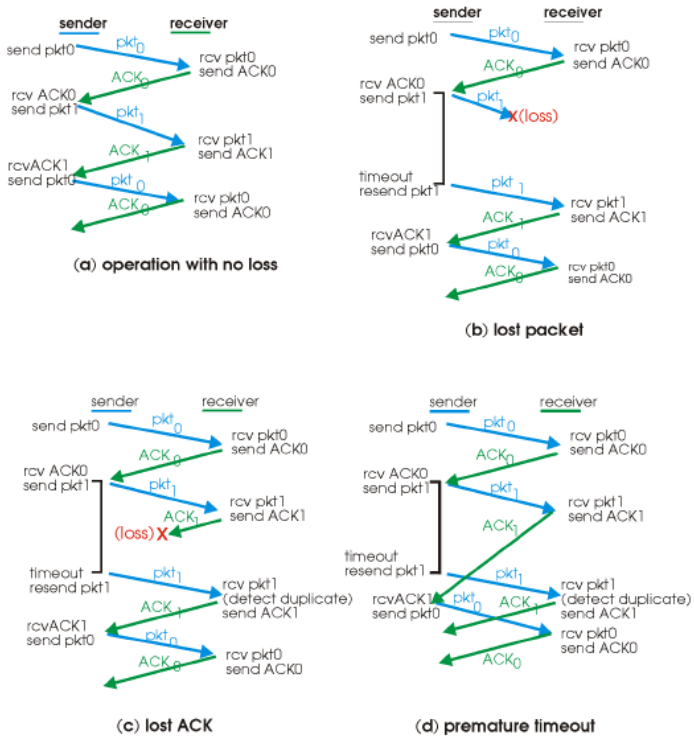
```
                1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
                1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
                -----------------------------------
                0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
                -----------------------------------
checksum        1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
```

חשוב לציין כי מגנן זה לא בטוח לגמרי כי אם היתה שגיאה גם בהודעה וגם שגיאה מפצה בchecksum אז המקבל לא יבחין בזה.

מנגנון העברת מידע בטוח מעל ערוץ לא בטוח

מנגנון נאיבי Stop & Wait :

בשליחת ההודעות יתכנו שיבושים ואובדן מידע לכן צריך לסמן לשולח(ACK) שההודעה הגיעה במלואה. כמובן שגם אובדן של ה ACK הוא אפשרי. השולח, במידה ולא הגיע ACK , לא יודע אם החבילה אבדה, הACK אבד או אם החבילה או הACK הגיעו משובשים לכן בכל מקרה הפתרון הוא לשלוח שוב. נציין כי לכל חבילה יש מספר סידורי(0 או 1) ולACK יש גם מספר סידורי שמציין על איזה חבילה הוא מדווח. בנוסף מוגדר זמן timeout שהוא הזמן המקסימלי להמתנה עד שליחה חוזרת של החבילה. (על הקורא להבין היטב את השרטטים הבאים)



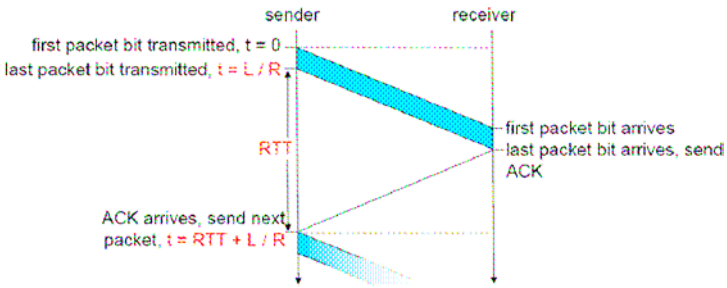
חישוב נצילות :

- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- U_{sender}: utilization – fraction of time sender busy sending
- 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

זהו פרוטוקול שליחה בעל נצילות נמוכה (פחות מ 1%) כי בכל פעם נשלחת הודעה בודדת נראה כעת 2 פרוטוקולים ליעול שמתעסקים עם יותר מחבילה אחת באותו זמן:

Go-Back-N (GBN)

בפרוטוקול זה לשולח יש את האפשרות לשלוח מספר חבילות בו זמנית מבלי להמתין ל ACK. אך השולח מוגבל בכמות החבילות שנשלחו מבלי לקבל ACK עבורן.

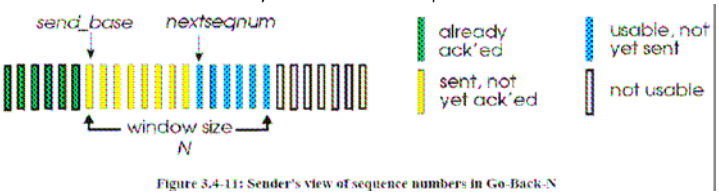


Figure 3.4-11: Sender's view of sequence numbers in Go-Back-N

טווח ההודעות שנשלחו מבלי לקבל ACK עבורן נקרא "חלון" בגודל N. החלון מתקדם כל פעם שמתקבל ACK .
כמובן שכעת המספר הסידורי של ההודעות יהיה יותר מ 0 או 1 , אך הוא מוגבל בטווח [0-2^k] כאשר k הוא מספר הביטים בשדה שמסמן את מספר החבילה.
השולח צריך להגיב ל 3 מצבים :

1. בקשת שליחה – אם החלון לא מלא, נוספת חבילה לשליחה. אם החלון מלא השולח מודיע כי אין לו מקום כעת לחבילה זה.(ניתן גם לאגור חבילות בbuffer ולהוסיף אותן לחלון כאשר יתפנה מקום, בנוסף השולח יכול לסמן לאפליקציה עם דגל מתי יש לו מקום בחלון).
2. קבלת ACK – כאשר מגיע ACK עבור חבילה n זה מסמן לשולח כי **כל** החבילות עד n הגיעו תקינות למקבל. זה נקרא **ACK מצטבר**.
3. timeout – במקרה זה על השולח לשלוח בשנית את **כל** החבילות שנשלחו כבר אך לא התקבל ACK עבורן. (מכאן מגיע שם הפרוטוקול).

תפקידו של המקבל בפרוטוקול זה הוא פשוט, כאשר מגיעה החבילה n והיא בסדר הנכון הוא שולח ACK עבור חבילה n . בכל מקרה אחר המקבל מתעלם מהחבילה שהגיעה ושולח בשנית ACK עבור החבילה האחרונה שהגיעה תקינה ובסדר הנכון. מכאן שהמידע היחיד שהמקבל צריך לשמור הוא המספר הסידורי של החבילה הבאה. ואיל השולח צריך לנהל את טווח החלון (מצביעים לתחתית וראש החלון) ומצביע נוסף שמסמן את החבילה הבאה להישלח.

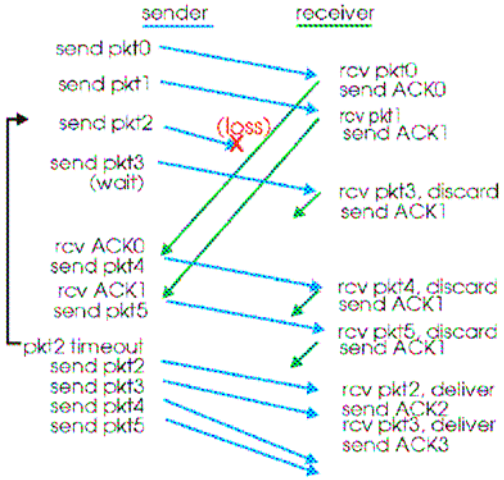


Figure 3.4-14: Go-Back-N in operation

Selective Repeat (SR)

ישנם מקרים מסוימים שבהם הפרוטוקול GBN לא יהיה יעיל, לדוגמא כאשר החלון הוא גדול, שגיאה באחת ההודעות תגרום לשליחה נוספת ומיותרת של כל ההודעות בחלון. פרוטוקול SR נמנע נמנע משליחת הודעות מיותרת, בכך שהשולח ישלח בשנית רק הודעות שהתרחשה שגיאה בשליחתן (אבדו או התקבלו שגויות) או בשליחת ה ACK עבורן.
גם ב SR נגדיר חלון הודעות בגודל N שיגביל את כמות ההודעות שנשלחו מבלי לקבל ACK עבורן. המקבל ישלח ACK עבור הודעות שהתקבלו תקינות גם אם הם לא בסדר הנכון.

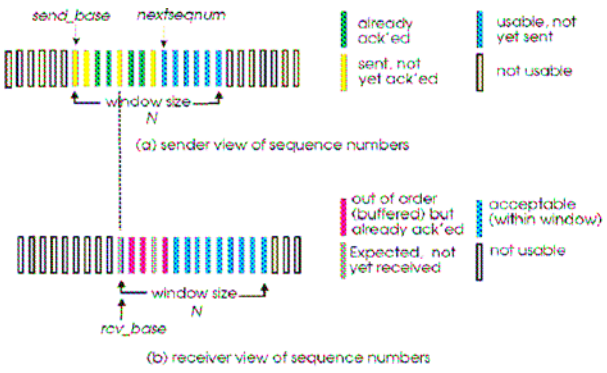


Figure 3.4-15: SR sender and receiver views of sequence number space

השולח צריך להגיב ל 3 מצבים :

1. בקשת שליחה – אם החלון לא מלא נוספת חבילה לשליחה. אם החלון מלא החבילה נשמרת או שהשולח מודיע כי אין לו מקום כעת לחבילה זה.
2. קבלת ACK לחבילה n – השולח מסמן אצלו שחבילה n התקבלה. אם n הוא המצביע לתחתית החלון של השולח עליו לקדם את החלון עד אשר הוא נתקל בחבילה שנשלחה ולא התקבל ACK עבורה. לאחר הקידום ניתן לשלוח את כל החבילות שכעת הן בתוך החלון ועדין לא נשלחו.
3. timeout – לכל חבילה מוגדר timer משלה מאחר שכאשר יתרחש timeout רק חבילה אחת תשלח בשנית.

המקבל צריך להגיב ל 3 מצבים :

1. הגיעה חבילה n, כאשר n הוא בתוך החלון של המקבל – מוחזר ACK עבור חבילה n. אם החבילה לא הגיעה כבר בעבר היא נשמרת. אם n הוא המצביע לתחתית החלון, אז חבילה זו וכל החבילות העוקבות לה שכבר נשמרו מועברות לרמה העליונה יותר, ובנוסף על המקבל לקדם את החלון כמספר החבילות שהועברו לרמה העליונה יותר.
2. הגיעה חבילה n, כאשר n הוא לפני החלון של המקבל – מוחזר ACK עבור חבילה n.למרות שכבר הוחזר ACK כזה.
3. אחרת – מתעלם מהחבילה.

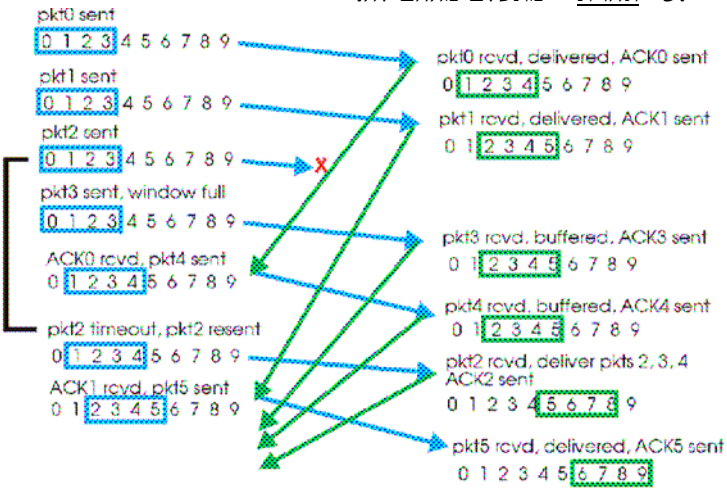


Figure 3.4-18: SR Operation

חשוב לציין כי חלון השולח וחלון המקבל לא תמיד יהיו חופפים.

בעיה: כפי שניתן לראות בתרשים למטה יתכן מצב בו המקבל לא יודע אם החבילה 0 שהוא קיבל היא חבילה חדשה או שליחה חוזרת. פתרון : גודל החלון צריך להיות לכל היותר חצי מגודל מרחב המספרים הסידוריים של החבילות.

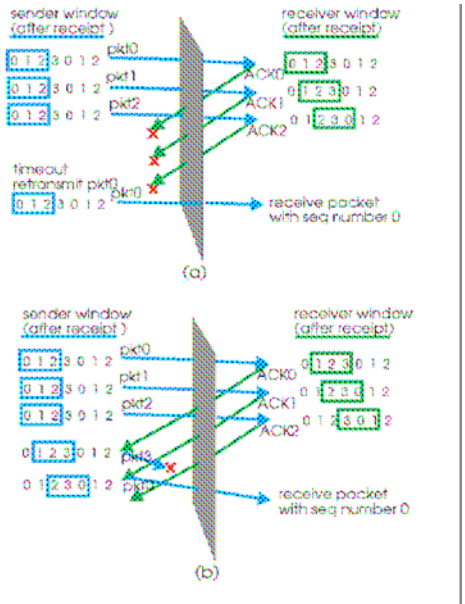


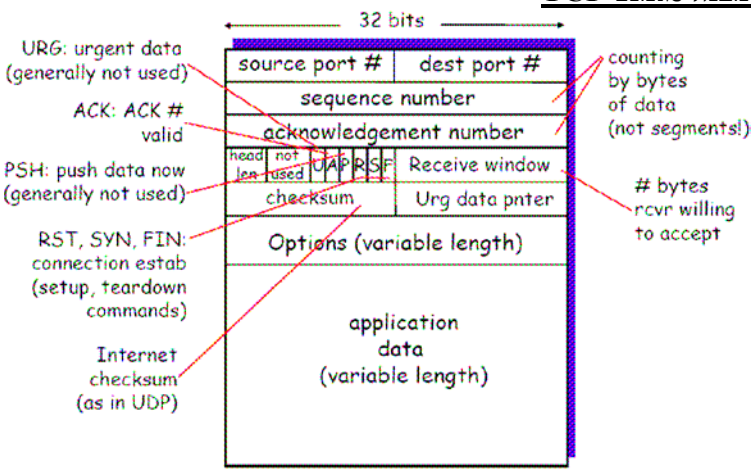
Figure 3.4-19: SR receiver dilemma with too large windows: a new packet or a retransmission?

TCP

פרוטוקול TCP הינו פרוטוקול מורכב, המספק המספק העברת נתונים בטוחה מעל ערוץ לא בטוח(IP). תכונות של TCP :

- העברת הנתונים הבטוחה כוללת : המידע מגיע תקין, ללא רווחים, ללא כפילויות ובסדר הנכון.
- בנוסף TCP תומך ב MUX/DEMUX ובבדיקת שגיאות בדומה לUDP .
- לא כמו UDP פרוטוקול TCP הוא "מכוון קישור" (connection-oriented) כלומר לפני שליחת מידע יש "לחיצת ידיים" בין 2 הצדדים.
- TCP הוא "דו צדדי לחלוטין" (full duplex) כלומר ניתן לשלוח מידע ב 2 הכיוונים. המקבל יכול גם לשלוח ולהפך.
- מנגנון Flow Control – האטת קצב השליחה כדי לא "להציף" מקבל איטי.
- מנגנון Congestion Control – האטת קצב השליחה כדי לא להעמיס על הרשת.

מבנה סגמנט TCP



- נשים לב שיש 2 שדות של מספרי פורטים – source port ו dest port הסיבה היא שהתקשורת היא דו כיוונית. (full duplex).
- השדות sequence num ו ack num ישמשו למימוש העברת נתונים בטוחה.
- Rcvr win size ישמש למימוש מנגנון flow control.
- Header length – אורך הheader שהוסיף ה TCP. (בד"כ 20 בייט)
- יש מספר "דגלים" שמסמנים איזה סוג חבילה זאת.(urg,ack,push,rst,syn,fin)
- Checksum
- Urg pointer – לא בשימוש. מצביע למידע החשוב, כדי שהמקבל יכול קודם לקרוא אותו.

הקמת הקשר

בפרוטוקול TCP יש צורך לוודא שהשרת חי לפני שמתחילים לשלוח אליו נתונים . משתמשים ב"לחיצת ידיים משולשת" (three way handshake). השלבים בלחיצת הידים הם:

1. SYN - הלקוח שולח חבילת TCP ראשונית לשרת. בחבילה זו מודלק הביט SYN והיא **לא** מכילה מידע מהאפליקציה.
2. SYN-ACK - כאשר השרת מקבל SYN הוא מאתחל משתנים ובאפרים עבור הקישור, בנוסף השרת שולח חבילה שמאשרת את קבלת ה SYN, בחבילה זו מודלק הביט SYN ויש ACK עבורו. היא **לא** מכילה מידע מהאפליקציה.
3. ACK מהלקוח לשרת - הלקוח גם מאתחל משתנים ובאפרים עבור הקישור ואז שולח חבילת אישור שבחבילה זו כבר יכול להיות מידע מהאפליקציה.

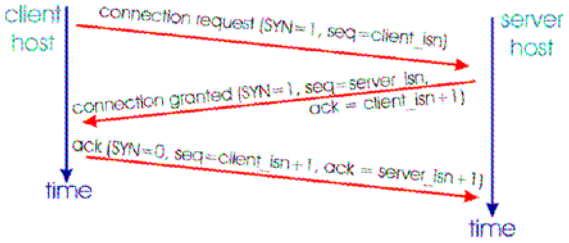
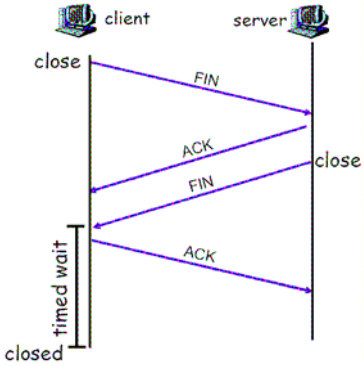


Figure 3.5-10: TCP three-way handshake: segment exchange

סגירת קשר

באופן דומה להקמת הקשר גם סגירת הקשר היא מסודרת.

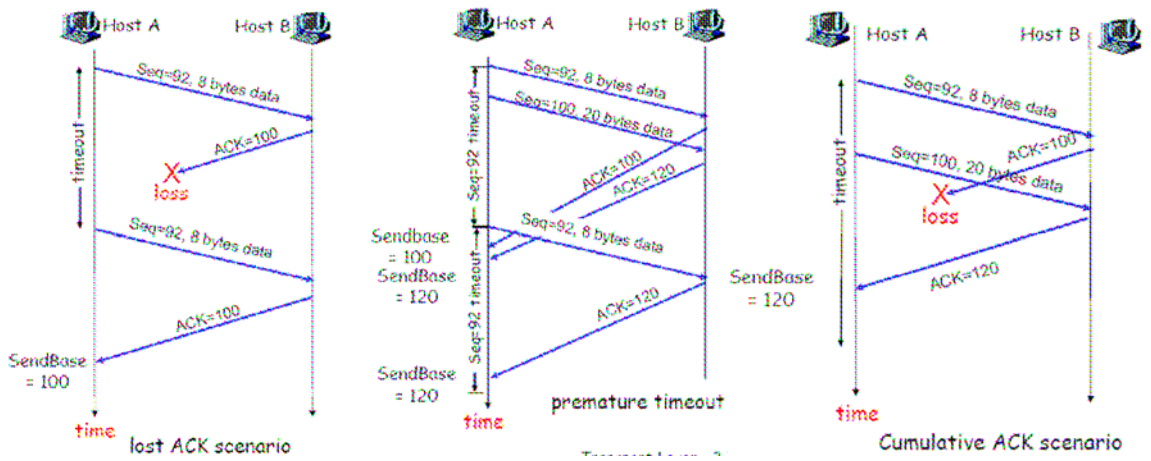
1. הלקוח שולח FIN לשרת.
2. השרת מגיב ב ACK וכאשר הוא מסיים שולח גם הוא FIN ללקוח.
3. הלקוח מגיב ב ACK ונכנס להמתנה timed out כדי לוודא שאם יש הודעות שהתעכבו הן לא יאבדו.



זה יותר מורכב מפתיחת הקשר כי יכול להיות שהשרת לא סיים לשלוח, לכן הוא גם צריך להודיע FIN שהוא מסיים.
סגירת קשר **יכולה להתחיל משני הכיוונים** בניגוד לפתיחה שהיא רק מצד הלקוח לשרת. סגירה יכולה להיות 3 הודעות ולא 4, השרת יכול לשלוח FIN ACK ביחד.

ACK

ב TCP ה ACK הוא "ACK מצטבר" כלומר, כאשר נשלח ACK עבור חבילה מסוימת, זה אומר שכל החבילות לפניה הגיעו תקינות וללא מרווחים. המספר הסידורי של החבילה הוא המספר של הביט הראשון ב data של החבילה. מספר ה ACK שישלח הוא המספר של הביט הבא שמצפים לקבל. נציג 3 תרחישים: (הקורא יבין לבד את המקרים השונים והתנהגות TCP)



לפי הפרוטוקול על המקבל לשלוח ACK עבור כל חבילה. אך יתכן וזה לא יהיה ממש יעיל(Telnet) , לכן אם מגיעה חבילה לפי הסדר אז נמתין חצי שנייה ורק אם לא מגיעה עוד חבילה לפי הסדר אז נשלח ACK אחרת נשלח ACK אחד על 2 החבילות ביחד.

Fast Retransmission : אם נקבל 3 פעמים ACK על אותה חבילה, לא נמתין שיגמר ה Timeout וישר נשלח את המידע שוב.

Round Trip Time (RTT) and TimeOut

כזכור כאשר נשלחת חבילה ע"י TCP, מאותחל timer, אם הtimer מסיים לפני שמגיע ACK על החבילה, אז החבילה נשלחת שוב. איך נקבע אורך הtimer ? ברור כי הוא צריך להיות גדול יותר מ RTT אבל לא גדול מדי. **לכן כדי לשערך את RTT**:

- נמדוד עבור **כל** חבילה שנשלח את sampleRTT שזה, הזמן שעבר מהרגע שהחבילה נשלחה עד הרגע בו מתקבל ACK עבורה.
- נחשב : EstimatedRTT = (1-x)*EstimatedRTT + x*sampleRTT
- ערך אופייני ל x הוא x = 0.125
- נשים לב שבחישוב זה נלקח ממוצע ממושקל של ערכי sampleRTT, וניתן משקל גדול יותר למדידה האחרונה.

חישוב Timeout :

- Timeout = EstimatedRTT + 4*Deviation
- Deviation = (1-x)*Deviation + x*|sampleRTT – EstimatedRTT|

נשים לב כי ה Timeout הוא הEstimatedRTT ועוד Deviation. נקבל כי Deviation יהיה גדול כאשר יש תנודות רבות בערכי sampleRTT. וקטן אם התנודות קטנות.

Flow Control

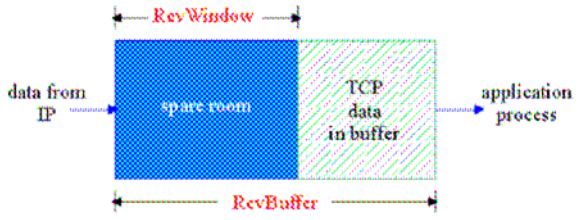


Figure 3.5-9: The receive window (RcvWindow) and the receive buffer (RcvBuffer)

נזכר כי שני השותפים בקשר TCP יכולים לקבל מידע.לכן כל אחד מקצה "באפר" לאחסן את המידע שהתקבל. יתכן שהמידע ימתין זמן רב ב"באפר" עד אשר האפליקציה תקרא אותו משם. וכתוצאה מכך השולח יכול להציף בקלות את המקבל במידע של יהיה איפה לאחסון אותו.TCP מספק מנגנון Flow Control כדי להתמודד עם בעיה זו. העיקרון הוא, שהמקבל ידווח לשולח כמה מקום נשאר לו ב"באפר" ובמידה ויש מצוקת מקום יאט השולח את קצב השליחה.

לכן השולח ישמור אצלו משתנה RevWin שמציין כמה מקום נשאר למקבל, ואילו המקבל בכל ACK ידווח לשולח מה גודל RevWin. RevWin נע בין 0 ל 64k.

מקרי קצה:

- כאשר המקבל איטי ונשאר לו מעט מקום ב"באפר" זה מכריח את השולח לשלוח בחבילות קטנות.
- כאשר 64k זה פחות מדי עבור החבילה, אז הנצילות של הקו תהייה נמוכה מאוד, כי כל פעם נשלח חבילה אחת ונחכה לACK אחד.(פתרון: יש אפשרות , בהסכמת 2 הצדדים,להגדיל את החלון פי x).

Nagle`s Algorithm: רעיון האלגוריתם הוא, אם יש המון חבילות קטנות זה מאוד לא יעיל לשלוח header עבור כל אחת, לכן נקבץ אותן ביחד ואז נשלח. (MSS = maximum segment size)

Nagle's Algorithm

- if there is new data to send
  - if the window size >= MSS and available data is >= MSS
    - send complete MSS size segment now
  - else if there is unconfirmed data still in the pipe
    - enqueue data in the buffer until an acknowledge is received
  - else send data immediately

Congestion Control

העיקרון של Congestion הוא שיותר מדי מקורות שולחים יותר מדי מידע והרשת לא יכולה להתמודד עם זה אז יש אובדן הודעות, ואז נשלחות שוב אותן הודעות ויוצרות עוד עומס וכו'.(יש בספר דוגמאות למקרים בהם יכול להיווצר עומס שמוביל להמתנה אין סופית) לכן הומצא מנגנון Congestion Control שתפקידו ליצור בקרה על כמות המידע שנשלח כדי לא ליצור עומס ברשת. מנגנון זה בעצם מאט את השולח, בדומה ל flow control אך מסיבות שונות לגמרי.
ניתן לממש Congestion Control במספר דרכים:

- End-end congestion control – במקרה זה הרשת עצמה לא מספקת פידבק על מצב העומס שלה. ועל משתמשי הקצה לאבחן עומס. TCP נוקט במימוש זה והוא מזהה עומס ברשת ע"י אבדן חבילות (timeout או ריבוי ACK-ים על אותה חבילה).
- Network-assisted congestion control – במימוש זה הראוטרים ברשת מדווחים על עומס. בצורה ישירה (ע"י שליחת הודעה choke packet לשולח) או ע"י הדלקה של ביט בחבילה שמסמן עומס, במקרה השני על המקבל של החבילה לידע את השולח שיש עומס.

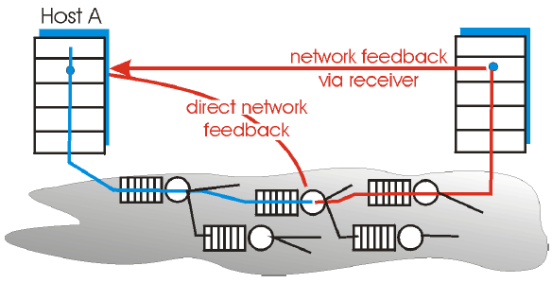


Figure 3.6-7: Two feedback pathways for network-indicated congestion information

ATM ABR (Available Bit Rate)

זה דוגמא לרשת שמממשת Network-assisted congestion control . בנוסף לחבילות ה data שמועברות ברשת מעבירים גם חבילות RM (Resource Management) כאשר חבילת RM תגיע למקבל הוא יחזיר אותה לשולח (יתכן והוא יעדכן שדות בחבילה), בנוסף הראוטרים בדרך יכולים גם הם לייצר חבילות RM ולשלוח אותם לשולח ישירות.
חבילות RM מכילות מספר שדות חשובים:

- CI (Congestion Indicated) מודלק כאשר יש עומס חמור ברשת.
- NI (No Increase) מודלק כאשר יש עומס קל ברשת.
- ER (Explicit Rate) – מסמן לשולח את המהירות בה ניתן לשלוח. שדה זה יקבע להיות הקצב של הראוטר הנמוך ביותר מכל הראוטרים בדרך.

<u>עקרון הפעולה</u>: ראוטר עמוס יכול להדליק את השדות CI, NI ולקבוע את ER. המקבל יחזיר את חבילת RM כמו שהוא קיבל אותה, פרט למקרה זה: בחבילות ה data יש שדה EFCI(Explicit Forward Congestion Indication) ראוטר עמוס ידליק את ביט EFCI , כאשר המקבל יקבל חבילת data הוא יבדוק את מצב הביט EFCI. כאשר תגיע חבילת RM למקבל הוא ידליק את הביט CI, אם בחבילת ה data האחרונה שהגיע אליו הביט EFCI היה דלוק.
כעת השולח יכול לקבוע את קצב השליחה שלו כפונקציה של NI,CI ו ER.

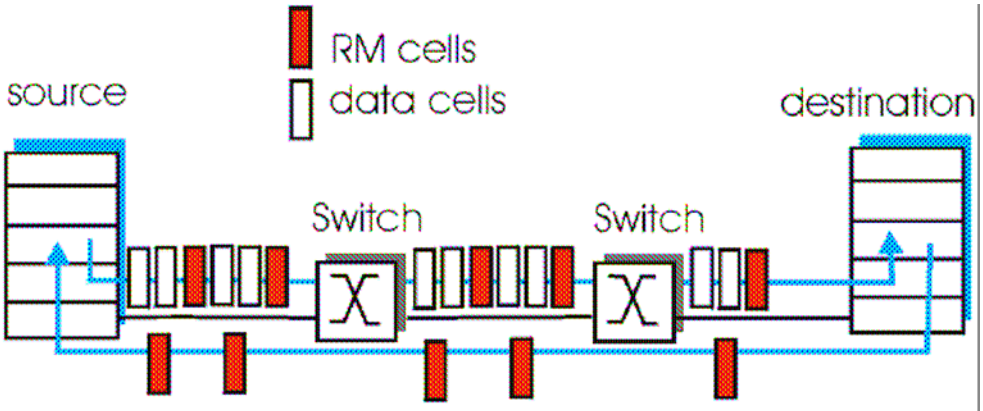
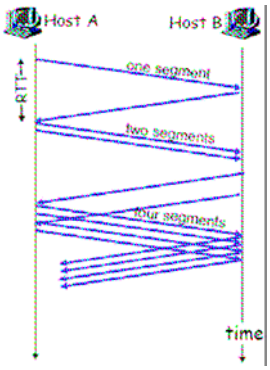


Figure 3.6-8: Congestion control framework for ATM ABR service

TCP Congestion Control

קצב השליחה ייקבע ע"י שני הצדדים -המקבל (יכולת-קבלה - flow control) והשולח (הערכת-עומס congestion - control). גודל חלון השליחה יהיה המינימום בין ה-Flow Control Window וה-**Congestion Window(CW)**. אירועי timeout ואירועים של ריבוי ACK-ים על אותה חבילה יכולים לרמז לנו על עומס ברשת. נציע מספר מימושים שונים לניהול גודל CW:

1. **AIMD** - בתחילת הקשר גודל ה CW הוא MSS אחד. אם נקבל ACK בזמן אז נגדיל את ה CW ב MSS אחד, כל עוד אין עומס נגדיל את CW בצורה לינארית. אך כאשר יש חשד לעומס(timeout) נוריד את CW לחצי מגודלו.
2. **TCP slow start** - בתחילת הקשר גודל הCW הוא MSS אחד. אם נקבל ACK בזמן אז נכפיל את ה CW, כל עוד אין עומס נגדיל את CW בצורה אקספנציאלית. אך כאשר יש חשד לעומס(timeout) נוריד את CW לחצי מגודלו.
3. שיטה יותר מעודנת - ניתן גם לקבוע סף X שיתנהג באופן הבא:
   - CW יגדל אקספוננציאלית כל עוד הוא נמוך מ X .
   - CW יגדל לינארית כל עוד הוא גבוה מ X .
   - כאשר יש 3 ACK-ים זהים רצופים X נקבע לחצי מערכו של CW וגם CW מחולק ב 2.
   - כאשר יש X timeout נקבע ל חצי מערכו של CW, ואילו CW נקבע ל MSS אחד.



TCP Fairness

TCP הוא פרוטוקול הוגן המחלק את רוחב הפס בצורה שווה בין המשתמשים בקו. חלוקה שווה זו מושגת ע"י מנגנון בקרת העומס. נניח מקרה בו 2 קישורי TCP, A ו B משתמשים באותו פס (אין הגבלה של flow control והקישורים הם בעלי אותו MSS ו RTT) במקרה זה ברור כי הקישור A שהתחיל קודם יקבל CW גדול יותר. CW של A ימשיך לגדול עד שבשלב מסויים יהיה איבוד של חבילה, וה CW יקטן בחצי, בשלב זה הCW של B ימשיך לגדול, עד אשר אחד מהם שוב יאבד חבילה והCW שלו יקטן בחצי וכו'. כך זה ימשיך ובסופו של דבר הם יתאזנו לאותו טווח של CW. אפליקציה יכולה להגדיל את החלק מרוחב הפס שהיא מקבלת ע"י פתיחת כמה קישורי TCP במקביל. (ואכן דפדפני אינטרנט רבים עושים זאת)

הספק הקו

סימונים : W - גודל מקסימלי של CW. O – גודל האוביקט שאותו מעבירים. R – קצב העברת הנתונים בקו. S הוא גודל MSS בביטים.

במקרה הפשוט בו אין הגבלה של CW . הקלינט שולח לשרת בקשה לקשר,דורש זמן של RTT אחד, לאחר אישור מהשרת הקלינט שולח בקשה לקבל אוביקט מהשרת (בקשה זו נכללת בהודעה השלישית ב three-way handshake)

ואז השרת מתחיל לשלוח את האוביקט, דורש RTT נוסף. להעברת האוביקט דרוש O/R זמן לכן סה"כ זמן לקבלת אוביקט הוא : O/R + RTT2.

<u>כעת נניח כי גודל CW הוא קבוע והוא W.</u> כאשר השרת מקבל את הבקשה מהקלינט הוא ישר שולח W סגמנטים חזרה לקלינט וממתין ל ACK. השרת ימשיך וישלח סגמנט עבור כל ACK שיתקבל. נבדיל בין 2 מקרים:

1. WS/R > RTT + S/R **כלומר כאשר השרת מקבל ACK על הסגמנט הראשון לפני שהוא סיים לשלוח את החלון הראשון**. במקרה זה השרת ימשיך לשלוח את כל הסגמנטים ברציפות. נקבל זמן של O/R + RTT2 לשליחת האוביקט. (בתרשים W=4)
2. WS/R < RTT + S/R **כלומר השרת סיים להחביר את כל החלון הראשון לפני שהוא קיבל ACK על הסגמנט הראשון**. לכן לאחר שליחה של W סגמנטים על השרת להמתין עד אשר יתקבל ACK. לכן הפעם הזמן מורכב מ RTT2 ועוד הזמן שלוקח להעביר את כל האוביקט O/R ועוד הזמן שהשרת ימתין. נסמן ב K את מספר החלונות שיש באוביקט בגודל O. ונקבל כי הזמן שדרוש להעברת אוביקט הוא 2RTT + O/R + (K-1)[S/R + RTT - WS/R]. (מבולבלים? גם אנחנו...) (בתרשים W=2)

** יש בספר פירוט על המקרה בו CW הוא דינמי ולא קבוע. לדעתי זה מעבר לרמה שלמדנו, אז ויתרתי על זה. התלמיד השקדן ישלים את זה לבד. ☺

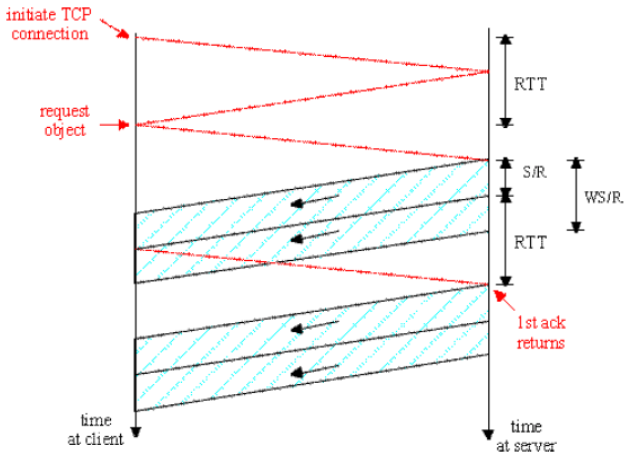


Figure 3.7-6: the case that WS/R < RTT + S/R

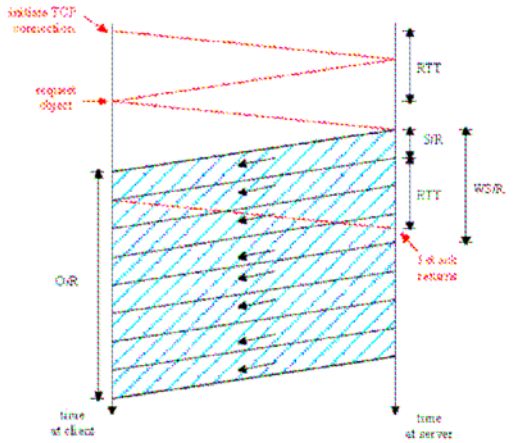


Figure 3.7-5: the case that WS/R > RTT + S/R

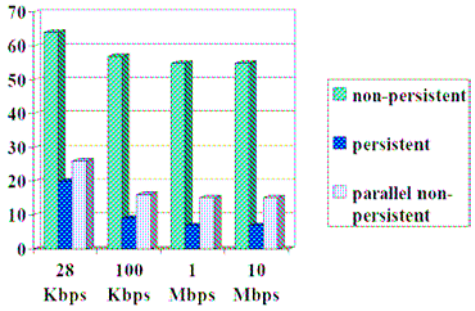
HTTP and TCP

פרוטוקול HTTP משתמש בקישור TCP. נסמן: O – גודל הדף אותו הקלינט מבקש מהשרת. הדף מכיל M תמונות שכל אחת היא בגודל O.

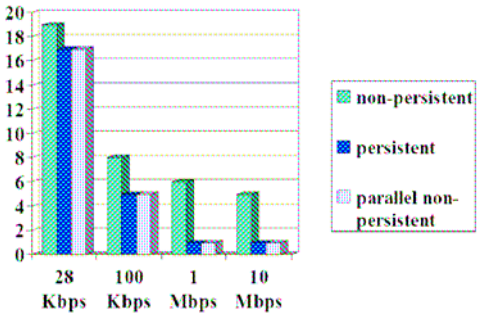
- Non-persistent HTTP – בקשה עבור כל תמונה מתבצעת בקישור TCP נפרד. לכן יש M+1 קישורי TCP. הזמן שדרוש הוא: (M+1)O/R + (M+1)2RTT + sum of idle times.
- Persistent HTTP – באותו קישור HTTP מבקשים את כל התמונות. ולא פותחים כל פעם קישור חדש. לכן דרושים RTT2 כדי לקבל את הדף בנוסף RTT1 כדי לקבל את כל M התמונות. הזמן שדרוש הוא : (M+1)O/R + 3RTT + sum of idle times.
- Non-Persistent HTTP with X parallel connections – כעת נפתחים במקביל X קישורי TCP. אז נפתח קישור אחד עבור הדף, ועוד M/X קישורים עבור התמונות. וכעת נקבל כי הזמן הדרוש הוא : (M+1)O/R + (M/X + 1)2RTT + sum of idle times.

כמובן כאשר RTT הוא גדול, אז Non-persistent פחות יעיל.כאשר R הוא קטן אין הבדל משמעותי.

RTT =1 sec, O = 5 Kbytes, M=10 and X=5



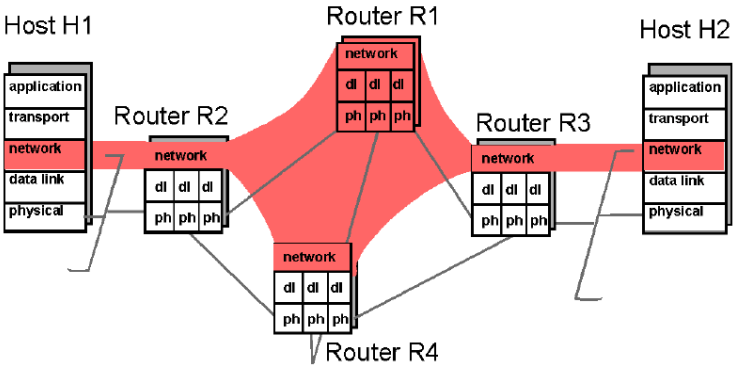
RTT = 100 msec, O = 5 Kbytes, M=10 and X=5



Network Layer

Introduction and Network Service Models - 4.1

שכבת הרשת מספקת שירותי תקשורת בין תחנות הקצה (hosts). שכבה זו מעבירה סגמנטים של שכבת ה- transport מהתחנה השולחת לתחנה המקבלת.
הציור הבא מתאר איך התהליך קורה: בתחנת הקצה השולחת, סגמנט שכבת ה-transport מועבר לשכבת הרשת, אז שכבה זו מעבירה את הסגמנט ליעד שלו. בתחנת הקצה המקבלת, שכבת הרשת מעבירה את הסגמנט שהגיע לשכבת ה-transport.
בפרק זה נלמד איך בדיוק שכבת הרשת מעבירה את החבילות (הסגמנט משכבת ה- transport) מתחנת קצה אחת לשנייה.
נשים לב שהחבילה עוברת בין מספר ראוטרים שתפקידם העיקרי הוא לנתב את החבילות מהלינקים הנכנסים לראוטר ללינקים היוצאים ממנו (switching).



שכבת הרשת פועלת לפי שלושה עקרונות חשובים:

1. Path Determination: שכבת הרשת קובעת את הנתיב או הדרך של החבילות כאשר הן זורמות מהשולח למקבל. האלגוריתמים שעושים זאת נקראים: routing algorithms (אלגוריתמי ניתוב). אלגוריתמים אלה קובעים דרך אילו ראוטרים ובאיזה סדר החבילה תעבור. אנחנו נלמד על 2 שיטות לקביעת המעבר:
   - link state routing
   - distance vector routing

   נראה שהסיבוכיות של האלגוריתמים האלו גדלה ככל שיש יותר ראוטרים ברשת.
2. Switching: כאשר חבילה מגיעה כקלט לראוטר, הראוטר מנתב אותה ללינק הפלט הנכון. אלגוריתמי הניתוב מייצרים טבלה לפיה הראוטר מחליט לאן החבילה תנותב.
3. Call Setup. בדומה לפרוטוקול TCP, שבו היה צורך בשלוש לחיצות ידיים לפני שמידע באמת התחיל לזרום מהשולח למקבל, גם כאן יש מקרים שבהם מתבצעת לחיצת ידיים בין המקור ליעד על מנת להגדיר את המצב לפני שמידע עובר. נציין שברשת האינטרנט אין את זה.

Network Service Model

מודל המגדיר איך יתבצע המעבר בין 2 נקודות קצה: מה קורה כשיש מספר חבילות? האם סדר שליחתן יהיה זהה לסדר הגעתן ליעד? האם הזמן בין שליחת חבילה אחת לאחרת יהיה זהה לזמן בין הגעת חבילה אחת לאחרת? האם המערכת תספק איזשהו משוב על הגעת / אי הגעת החבילות? וכו' וכו'....
נלמד על שני מודלים: Virtual Circuit ו-Datagram.

:Virtual circuits (VCs) (network layer connection oriented service)

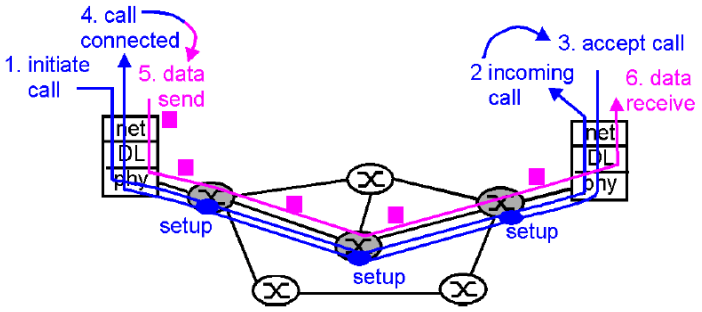
בפרק 1 ראינו מה זה מעגל וירטואלי: לפני שמתחילה העברת החבילה בין שתי תחנות קצה נפתח צינור ובסיום ההעברה הוא נסגר ומשתחררים המשאבים. הראוטרים בצינור צריכים לשמור מידע לגבי כל התחנות בצינור.
נזכיר בקצרה איך זה עובד:
מעגל וירטואלי כולל מסלול בין המקור ליעד, לכל לינק במסלול יש מספר VC ובטבלאות הניתוב בראוטרים ישנן רשומות לכל מספר VC. לחבילות שמועברות ב-VC יש מספר VC והוא משתנה בכל לינק (את המספר החדש החבילה מקבלת לפי טבלת הניתוב).

שימוש במעגל וירטואלי כולל 3 שלבים עיקריים:

1. VC setup. במהלך שלב ההגדרות, השולח מתחבר לשכבת הרשת , מציין את כתובת המקבל ומחכה לרשת שתגדיר את המעגל הווירטואלי. הרשת קובעת כיצד יהיה המעבר בין השולח למקבל כלומר קובעת את סדרת הלינקים והראוטרים שדרכה תעבור החבילה. (שלב זה כולל עדכון טבלאות, ולפעמים גם שמירת משאבים כמו רוחב פס.)
2. Data transfer. ברגע שנוצר המעגל הווירטואלי, המידע יכול לעבור.
3. Virtual circuit teardown.שלב זה מתחיל כשהשולח או המקבל מודיעים לרשת שברצונם לנתק את המעגל הווירטואלי. אז שכבת הרשת מודיעה על כך לתחנת הקצה השנייה ומעדכנת את הטבלאות כך שהמעגל כבר לא קיים מבחינת הרשת.

קיים הבדל קטן אך חשוב בין ה- VC setup שבשכבת הרשת לבין ה- connection setup שבשכבת ה-transport (שלושת לחיצות הידיים): הגדרת התקשורת בשכבת ה-transport מערבת רק את שתי תחנות הקצה. שתי התחנות מסכימות לקיים תקשורת ביניהן וביחד מגדירות את הפרמטרים של התקשורת ברמת ה-transport לפני שהמידע ממש מתחיל לזרום (הפרמטרים הם מספר רצף התחלתי, גודל חלון ה-flow control וכו'). לראוטרים שברשת אין קשר והם לא מעורבים כלל.
לעומת זאת, במעגל הווירטואלי שבשכבת הרשת, הראוטרים מעורבים בהגדרת המעגל, וכל ראוטר יודע את כל הפרטים הדרושים לגבי כל מעגל שעובר דרכו.

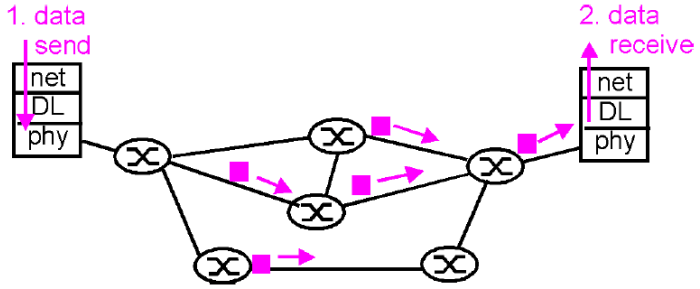
הציור הבא מתאר כיצד מועברת חבילה תוך שימוש ב-VC.
הקווים הכחולים – VC setup
הקווים הורודים – העברת המידע



Signaling massages –הודעות שנשלחות ע"י תחנות הקצה לרשת להתחלת או סיום VC והודעות שעוברות בין הראוטרים כדי להגדיר את ה-VC (כמו למשל לשנות טבלאות ניתוב).
Signaling protocols – הפרוטוקולים המשמשים לשינוי הודעות אלו. פרוטוקולים שבודקים את כל המסלול ומוודאים שיש לו את המאפיינים הדרושים להקמת הקשר.

<u>Datagram network layer</u>: (network layer connectionless service)
בכל פעם שתחנת קצה רוצה לשלוח חבילה, היא מחתימה את החבילה עם כתובת היעד שלה (כתובת של תחנת הקצה המקבלת), ואז מכניסה את החבילה לרשת. זה נעשה ללא הגדרת VC. הראוטרים לא שומרים מידע על המעגלים הוירטואליים (כי הרי אין) אלא מנתבים חבילות לעבר היעד שלהן על ידי בדיקת הכתובת שמוטבעת עליהן ובעזרת טבלאות הניתוב שלהם.
מכיוון שטבלאות הניתוב יכולות להשתנות כל הזמן, סדרה של חבילות שנשלחת מתחנת קצה אחת לשנייה יכולה לעבור בדרכים שונות ברשת, דבר שיגרום בסופו של דבר לשינוי סדר הגעת החבילות.
<u>האינטרנט משתמש במודל הזה.</u>

הציור הבא מתאר כיצד מועבר המידע תוך שימוש ב- datagram.
נשים לב שלכל חבילה יכול להיות מסלול שונה.



הערה: רשת של ראוטרים יכולה להציע רק אחד משני המודלים הללו. רשת האינטרנט למשל משתמשת רק בשירות ה-datagram.

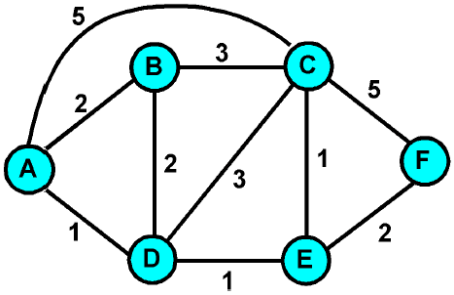
השוואה בין שני המודלים:

Network Architecture	Service Model	Bandwidth Guarantee	No Loss Guarantee	Ordering	Timing	Congestion indication
Internet	Best Effort	None	None	Any order possible	Not maintained	None
ATM	CBR	Guaranteed constant rate	Yes	In order	maintained	congestion will not occur
ATM	VBR	Guaranteed rate	Yes	In order	maintained	congestion will not occur
ATM	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided
ATM	UBR	None	None	In order	Not maintained	None

האינטרנט כיום מספק רק מודל שירות אחד: datagram שידוע גם בכינויו: best effort service. בשירות כזה אין הבטחה לרוחב פס מסוים, הזמן בין שליחת חבילה אחת לשנייה אינו נשמר, אין התחייבות לגבי סדר קבלת החבילות (כלומר סדר קבלתן לא בהכרח הוא סדר שליחתן) ואין התחייבות על שליחת החבילות בכלל.
לפי הגדרות אלה, גם רשת שבכלל לא מעבירה חבילות יכולה להיקרא best effort. נראה בהמשך שבכל זאת יש יתרונות לרשת כזו.
ל-ATM יש מספר מודלים לשירות שמבוססים על VC. ברובם מובטח רוחב פס קבוע, החבילות מגיעות באותו סדר שנשלחו, יש שמירה על הזמן ובד"כ יש התחייבות לאי איבוד של חבילות.
לא ניכנס לכל אחד מהמודלים (למרות שזה מאוד מפורט בספר).
באינטרנט, הרשת הפנימית היא יחסית פשוטה ואילו בקצוות המורכבות גדלה. תחנות הקצה הן "חכמות" (מחשבים) ולכן יכולות להסתגל לכל מיני תנאים, להתאושש משגיאות.
לעומת זאת, ב-ATM, תחנות הקצה "טיפשות" (למשל טלפון) ואילו כל המערכת הפנימית מאוד מסובכת.

Routing Principles – 4.2

על מנת להעביר חבילות מתחנת קצה אחת לשנייה, שכבת הרשת צריכה לקבוע את המסלול שלפיו החבילות יעברו.
קביעת המסלול נעשית ע"י פרוטוקול הניתוב (routing protocol).
בליבו של כל פרוטוקול ניתוב יש את האלגוריתם שקובע את המסלול של החבילה. מטרת האלגוריתם היא פשוטה:
בהינתן סידרה של ראוטרים עם לינקים שמחברים ביניהם, האלגוריתם מוצא מעבר "טוב" מהמקור ליעד. מעבר "טוב" הוא כזה שיש לו את המחיר הנמוך ביותר (least cost). למעשה, מה שקורה ב"עולם האמיתי" זה שיש כל מיני אילוצים כמו חברות שלא מוכנות לעבור דרך ראוטרים של חברות מסוימות אחרות ואז המעבר הטוב ביותר הוא לאו דווקא זה שיש לו את המחיר הנמוך ביותר.
הציור הבא מתאר מודל מופשט של רשת.
קודקודים בגרף מייצגים ראוטרים (הנקודות בהן מתקבלות החלטות הניתוב של החבילות), והצלעות בגרף מייצגות את הקשר הפיסי בין הראוטרים. לכל קשר (לינק) יש מחיר של שליחת החבילה דרכו. המחיר נקבע למשל לפי העיכוב שנוצר דרכו (לינק עם עיכוב גדול יקבל מחיר גבוה), או לפי האורך הפיסי שלו (לינק טרנס יבשתי יקבל מחיר גבוה יותר מאשר לינק תוך יבשתי) .



לפני שניכנס לאלגוריתמים עצמם, נאפיין תחילה את סדרת הקשרים במסלול עם המחיר הנמוך ביותר:

- הקשר הראשון במסלול מחובר למקור.
- הקשר האחרון במסלול מחובר ליעד.
- לכל i, הקשר ה-i והקשר ה- i-1 מחוברים לאותו קודקוד.
- עבור המסלול עם המחיר הנמוך ביותר, הסכום של מחירי הקשרים במסלול הוא המינימאלי מבין כל המסלולים האפשריים בין המקור ליעד. אם יש מספר מסלולים עם אותו מחיר, המסלול בעל המחיר הנמוך ביותר יהיה גם הקצר ביותר.

ניתן לסווג את אלגוריתמי הניתוב לשניים:

- אלגוריתם ריכוזי (global routing algorithm) – מחשב את המסלול עם המחיר הנמוך ביותר בין קודקוד מקור לקודקוד יעד תוך שימוש בידע גלובלי על המערכת. כלומר האלגוריתם מקבל כקלט את הקישוריות בין כל הקודקודים ואת המחיר של כל קשר. במקרה זה, האלגוריתם צריך איכשהו להשיג את כל האינפורמציה הזו לפני שהוא מתחיל לחשב את המסלול.
  אלגוריתמים גלובאליים אלה נקראים גם : link state algorithms מאחר שהאלגוריתם צריך לדעת את המצב (המחיר) של כל קשר ברשת.
- אלגוריתם מבוזר (decentralized routing algorithm) – חישוב המסלול עם המחיר הנמוך ביותר נעשה בצורה איטרטיבית. לאף קודקוד אין מידע על מחירי הקשרים ברשת, אלא כל קודקוד יודע את הכיוון אליו הוא צריך לשלוח חבילה על מנת שתגיע ליעד שלה במסלול עם המחיר הנמוך ביותר ואת המחיר ממנו ליעד במסלול זה.
  כל קודקוד מחליף אינפורמציה עם שכניו.
  אלגוריתמים אלו נקראים גם: distance vector algorithm .

ניתן לסווג את אלגוריתמי הניתוב גם באופן הבא:

- סטאטיים – באלגוריתמים אלו הניתובים משתנים בקצב איטי מאוד, בד"כ כתוצאה מהתערבות בני אדם (למשל הוספת טבלאות).
- דינאמיים – באלגוריתמים אלו הניתובים משתנים כתלות בעומס הרשת.

<u>Dijkstra - a link state routing algorithm</u>

כאמור, באלגוריתם זה אנו מניחים שכל קודקוד מכיר את כל מפת הרשת. למעשה זה בעייתי להעביר את כל המידע הזה לכל קודקוד ברשת לפני שבכלל האלגוריתם מתחיל לרוץ. לפי שיטה שנקראת: link state broadcast כל קודקוד יקבל את המידע הזה ללא האתחול הנ"ל, אלא ידע בהתחלה רק את האינפורמציה על השכנים שמחוברים אליו ישירות.
בהמשך הוא ילמד על הטופולוגיה של שאר הרשת ע"י קבלת link state broadcast משאר הקודקודים. כלומר, כל קודקוד "משדר" (broadcast) לכל הקודקודים ברשת את הזהות והמחיר של כל הקשרים המחוברים אליו ישירות.
בצורה זו לכל קודקוד יש תמונה שלמה של כל הרשת.

אלגוריתם דיקסטרה מחשב את המסלולים עם המחיר הנמוך ביותר מקודקוד מקור (A) לכל אחד מהקודקודים האחרים ברשת. האלגוריתם הוא איטרטיבי ומקיים שאחרי האיטרציה ה- k ית המסלולים עם המחיר הנמוך ביותר ל-k קודקודי יעד ידועים.

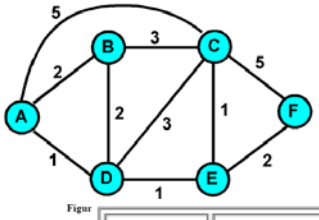
סימונים :

c(x,y) – מחיר הקשר בין צומת x לצומת y . אם x ו-y לא מחוברים, c(x,y) = inf.
נניח (לשם פשטות) ש- c(x,y) = c(y,x)
D(v) – המחיר המינימאלי של מעבר מקודקוד המקור לקודקוד v.
P(v) – הצומת הקודם במסלול ל- v.
N – קבוצת הקודקודים עם המחיר הנמוך ביותר שידוע.

<u>האלגוריתם</u>:
נשים לב שהאלגוריתם כולל שלב של אתחול ואחריו לולאה שעוברת על כל קודקוד ברשת.

```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infty

 Loop
   find w not in N such that D(w) is a minimum
   add w to N
   update D(v) for all v adjacent to w and not in N:
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    shortest path cost to w plus cost from w to v */
 until all nodes in N
```



Figur

ריצה לדוגמא:

step	N	D(B),p(B)	D(C),P(C)	D(D),P(D)	D(E),P(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infty	infty
1	AD	2,A	4,D		2,D	infty
2	ADE	2,A	3,E			4,E
3	ADEB		3E			4E
4	ADEBC					4E
5	ADEBCF					

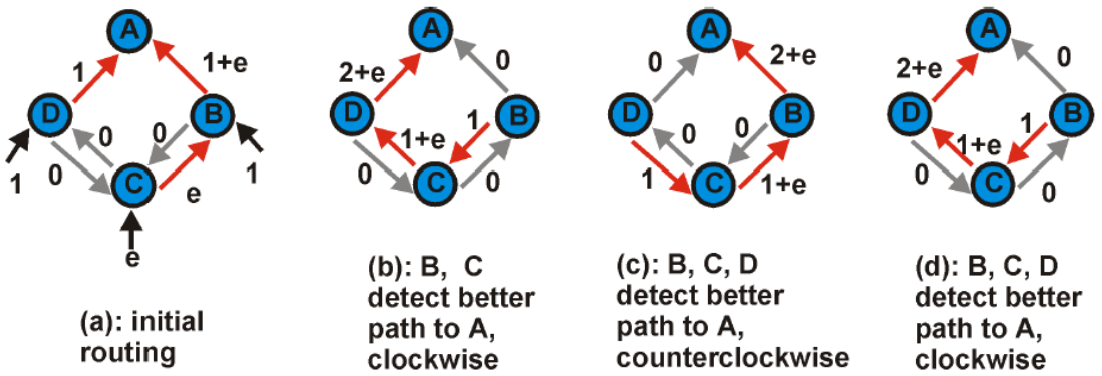
- אתחול (שורה ראשונה): המסלול עם המחיר הנמוך ביותר בין A לשכנים שלו שווה למחיר הקשר בין A לשכנים.
- איטרציה ראשונה: מסתכלים על הקודקודים שעדיין לא הוכנסו ל-N ומחפשים את הקודקוד עם המחיר הנמוך ביותר שהתקבל באיטרציה הקודמת. במקרה הזה קודקוד D. שורה 12 באלגוריתם מתבצעת ומביאה את התוצאות שבשורה השנייה בטבלה. המחיר לקודקוד B לא משתנה, המחיר לקודקוד C משתנה וכך הלאה...
- איטרציה שנייה: קודקודים B ו- E הם אלו עם המסלול בעל המחיר הנמוך ביותר (2) , לכן נבחר אחד מהם ונכניס ל-N. בשורה 12 הקודקודים שעדיין לא ב- N מתעדכנים ומקבלים את הערכים החדשים שבשורה 3.
- וכך הלאה....

בסופו של דבר נקבל לכל קודקוד את הקודקוד ממנו הגיע במסלול הקצר ביותר מהמקור. כך אפשר לשחזר את המסלול כולו.

<u>זמן ריצה:</u>

באיטרציה הראשונה, צריך למצוא מבין כל n הקודקודים שלא ב-N את זה שיש לו את המחיר הנמוך ביותר. באיטרציה השנייה צריך למצוא מבין n-1 הקודקודים שלא ב-N את זה שיש לו את המחיר הנמוך ביותר וכך הלאה...
מספר האיטרציות הוא כמספר הקודקודים ולכן במקרה הגרוע ביותר, זמן הריצה הוא $O(n^2)$. (ליתר דיוק: $(n(n+1)/2$
אם נמיין את הקודקודים בעזרת תור עדיפויות נקבל זמן ריצה של $O(n\log n)$.

נסתכל על המקרה הבא: (Oscillation)



במקרה זה, הניתוב נקבע עפ"י העומס על הצלע. אם העומס משתנה כל הזמן נקבל ניתוב לא יציב.
מחיר הקשר שווה לזמן העיכוב הצפוי של קשר זה.
מחיר הקשר לא חייב להיות סימטרי כלומר c(A,B) יהיה שווה ל- c(B,A) רק אם העומס בשני הכיוונים זהה.
בדוגמא, ציור (a) מראה את הניתוב ההתחלתי בין הקודקודים ואת המחיר של הקשרים.

באיטרציה הראשונה של האלגוריתם, קודקוד C קובע שהמסלול עם כיוון השעון לקודקוד A שווה 1 ואילו המסלול נגד כיוון השעון ל-A שווה ל- 1+e. לכן C בוחר את המסלול עם כיוון השעון. באותה מידה, B בוחר את המסלול ל-A עם כיוון השעון.
התוצאה היא, שבמסלול עם כיוון השעון ל-A נוצר עומס (ציור b).
באיטרציה הבאה, קודקודים B, C ו-D בוחרים במסלול ל-A נגד כיוון השעון (המסלול במקרה זה שווה ל-0).
באיטרציה הבאה, הקודקודים האלו שוב ישנו את כיוון המסלול ל-A, הפעם עם כיוון השעון.
(מבולבלים? גם אני...).

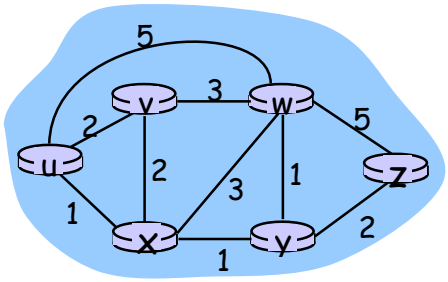
איך נמנע מקרים כאלו?

- הפיתרון הקל: להחליט שמחיר קשר לא ייקבע לפי העומס שבו.
- לוודא שכל הראוטרים לא מריצים את האלגוריתם באותו זמן (איך זה מונע?)

הערה: בדיקסטרה משקלי הצלעות אי שליליים.

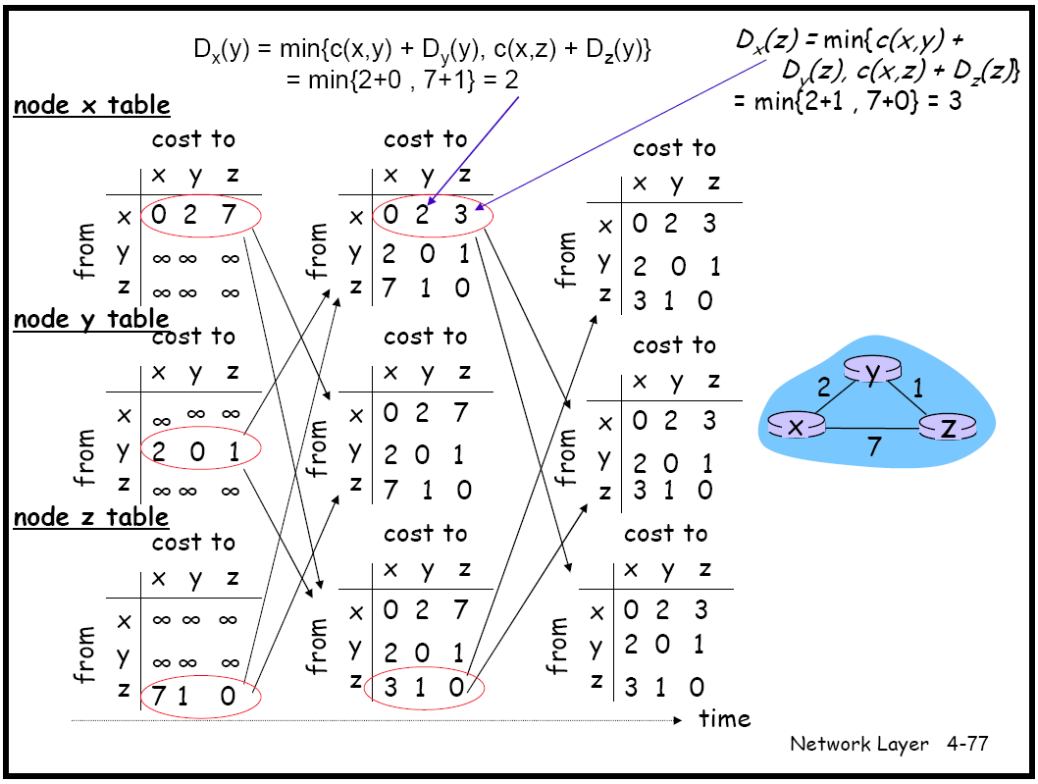
<u>Bellman-Ford - Distance vector routing algorithm</u>
אלגוריתם דינאמי:
לכל קודקוד יש טבלת מרחקים (distance table): למשל D_x היא טבלת המרחקים של קודקוד x.
נגדיר:
$D_x(y) = \min\{c(x,v)+D_v(y)\}$, המחיר של המסלול בעל המחיר הנמוך ביותר מ-x ל-y.
כאשר המינימום נלקח עבור כל השכנים של x (כלומר v הוא שכן של x).
$D_x(y)$ נקרא: Distance vector.

לדוגמא:



לפי המשוואה למעלה:
$D_u(z) = \min\{c(u,v)+D_v(z) , c(u,x)+D_x(z), c(u,w)+D_w(z)\} = \min\{2+5,1+3,5+3\} = 4$

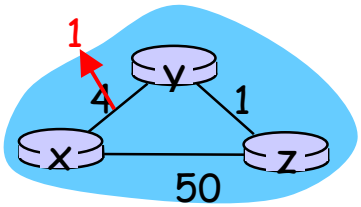
באלגוריתם זה, לכל קודקוד יש אינפורמציה על השכנים הישירים שלו ואינפורמציה שהוא מקבל משכנים אלו (זהו מידע יחסית מועט).
הרעיון של האלגוריתם:
כל קודקוד מקבל מהשכנים את המסלולים הקצרים ביותר שלהם ומעדכן את המסלול שלו בהתאם. כלומר אם המסלול לקודקוד היעד דרך שכן V משפר, הקודקוד משנה את המסלול כך שיעבור דרך V, אם זה לא משפר הוא לא משנה כלום.
הערה: באלגוריתם זה ניתן שיהיו צלעות עם משקלים שליליים, כל עוד אין מעגל שלילי (אחרת נסתובב בו לנצח).
דוגמא לריצה:



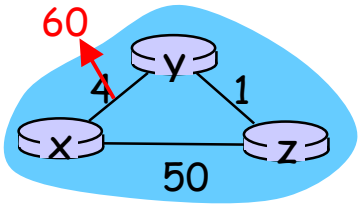
cost Link

changes and Link Failure

כאשר משקל הצלע משתנה, אם השינוי הוא לטובה, כל קודקוד מיד מעדכן את הטבלה שלו בהתאם ומשפר את המסלול שלו אם ניתן.
אבל, אם השינוי הוא לרעה, רק הקודקודים שמחוברים לצלע שבה התרחש השינוי יודעים עליו וזה יכול ליצור בעיה.
נסתכל על הדוגמאות הבאות:



שינוי לטובה- משקל הצלע בין x ל- y השתנה מ-4 ל-1.
בזמן t_0 , קודקוד y שם לב שיש שינוי, מעדכן את וקטור המרחק שלו ומודיע על כך לשכניו.
בזמן t_1, קודקוד z מקבל את העדכון מ-y ומעדכן את הטבלה שלו. קודקוד z מחשב מסלול חדש ל-x ושולח לשכנים שלו את וקטור המרחק המעודכן.
בזמן t_2, קודקוד y מקבל את העדכון של z ורואה שהוא לא צריך לעדכן את הטבלה שלו, לכן לא שולח שום הודעה ל-z.
("good news travels fast" – סה"כ שתי איטרציות).



שינוי לרעה- משקל הצלע בין x ל- y השתנה מ-4 ל-60.
בזמן t_0, קודקוד y שם לב לשינוי, מחשב את המסלול עם המחיר הנמוך ביותר ל-x.
קודקוד y יודע שמסלול ישיר ל-x עולה 60 ושקודקוד z אמר לו בפעם האחרונה ש-z יכול להגיע ל-x במחיר של 5. לכן, כדי להגיע ל-x, קודקוד y יעבור דרך z. קודקוד y לא יודע שבעצם המסלול של x הוא zyx ובעצם המחיר שלו הוא לא 5).
יש כאן לולאה בניתוב- המסלול של y ל-x הוא y-z-y-x.
בזמן t_1, y מודיע על המסלול החדש שמצא ל-z.
z מקבל את המחיר החדש ל-x דרך y (שהוא 6), z יודע שהוא יכול להגיע ל-y במחיר של 1 ולכן מחשב מסלול חדש ל-x במחיר של 7.
כעת, y רואה שהמחיר ממנו ל-x גדל ולכן מודיע על כך ל-z בזמן t_2.
כך זה ממשיך, כשבכל איטרציה מחיר המסלול מתעדכן ב-1 עד שהוא מגיע ל-51 וההודעה תעבור דרך z ישירות ל-x.

פיתרון לבעיה- poisoned reverse:

במצב שבו z מנותב דרך y כדי להגיע ליעד x, z יודיע ל-y שהמרחק שלו ל-x הוא אינסופי. z ימשיך לשקר ל-y כל עוד הוא מנותב ל-x דרכו. מאחר ש-y מאמין שאין מסלול מ-z ל-x , הו לעולם לא ינסה לנתב את המסלול דרכו.

פיתרון זה פותר רק מסלולים באורך 2!

<u>השוואה בין שני האלגוריתמים:</u>

- Massage Complexity –
  <u>LS</u>: כל קדקוד צריך לדעת את המחיר של כל צלע ברשת. זה דורש שליחה של O(nE) הודעות, כאשר n = מספר הקודקודים ברשת, E = מספר הצלעות.
  כמו כן, בכל פעם שמחיר של צלע משתנה, המחיר החדש חייב להישלח <u>לכל</u> הקודקודים.
  <u>DV</u>: בכל איטרציה יש חילופי הודעות בין שכנים ישירים.
  הזמן שלוקח לאלגוריתם להתכנס תלוי בהרבה פקטורים (כלומר הוא משתנה).
  כאשר מחיר של צלע משתנה, האלגוריתם מפיץ את השינוי רק אם המחיר החדש של הצלע גורר שינוי במסלול עם המחיר הקצר ביותר עבור אחד מהקודקודים שמחוברים לאותה צלע.
- Speed of Convergence –
  <u>LS</u>: זמן הריצה הוא $O(n^2)$, מספר הודעות שנשלחות הוא O(nE).
  לאלגוריתם יש פוטנציאל ל-oscillations.
  <u>DV</u>: יכול להתכנס בצורה איטית מאוד, יכולות להיות לולאות במסלול, יש לו בעיה של count-to-infinity.
- Robustness – מה קורה אם ראוטר לא מתפקד בצורה נכונה?
  <u>LS</u>: כל קודקוד יכול להעביר את הטעות רק ללינק אחד שמחובר אליו (ולא ליותר).
  כמו כן קודקוד יכול להשחית או לגרום להורדה של חבילות שהוא מקבל כחלק מה- link state brosdcast.
  אבל, כל קודקוד מבצע את החישובים שלו ומייצר לעצמו את טבלת הניתוב ולכן חישובי המסלול הם באיזשהו אופן מופרדים וזה נותן דרגה מסוימת של חוסן.
  <u>DV</u>: קודקוד יכול לעביר טעות לכל היעדים, כלומר הוא מעביר את הטעות לכל השכנים שלו, אלה מעבירים לכל השכנים שלהם וכך לאה. בסופו של דבר טעות שנעשית בקודקוד אחד תגיע לכל הרשת.

הערה: בשני האלגוריתמים משתמשים באינטרנט.

(בספר מתוארים עוד כמה אלגוריתמים שעליהם לא למדנו.)

<u>Hierarchical Routing 4.3</u>

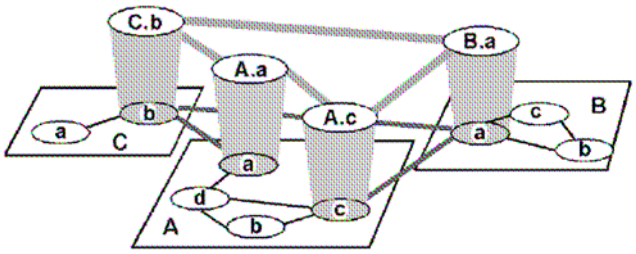
בחלק הקודם ראינו את הרשת כאוסף של ראוטרים המחוברים ביניהם ומבצעים את אותו אלגוריתם ניתוב המחשב מסלולים בכל הרשת. למעשה, תיאור כזה הוא לא נכון ואף בלתי אפשרי מהסיבות הבאות:

- Scale: ככל שמספר הראוטרים גדל, האינפורמציה שכל אחד מכיל והחישובים שכל אחד עושה גדלים.
  האינטרנט היום מכיל מיליונים של ראוטרים ויותר מ-50 מיליון נקודות קצה. שמירת אינפורמציה על כל הרשת דורשת הרבה מאוד זיכרון. (אלגוריתם DV לעולם לא יתכנס...).
- Administrative autonomy: באופן אידיאלי, ארגון אמור להריץ את הרשת שלו איך שירצה, כל עוד הוא יוכל לתקשר עם רשתות אחרות. בפועל, החברות צריכות להתחשב אחת בשנייה ויש כל מיני אילוצים.

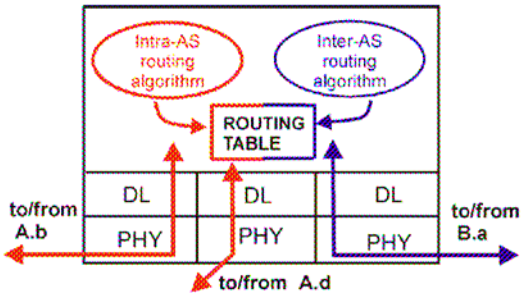
את שתי הבעיות הללו ניתן לפתור ע"י קיבוץ הראוטרים ל- "regions" או "autonomous systems" (ASs).
ראוטרים באותה קבוצת AS מריצים את אותו אלגוריתם ניתוב ויש להם אינפורמציה מלאה אחד על השני. אלגוריתם הניתוב שרץ בתוך קבוצת ראוטרים נקרא: intra autonomous system routing protocol.
הקשר בין הקבוצות השונות נעשה דרך מספר ראוטרים מקשרים. ראוטר כזה אחראי להעביר חבילות אל מחוץ לקבוצה שבא נמצא והוא נקרא: gateway router. שערים אלה צריכים לדעת איך לנתב את החבילות ביניהם. אלגוריתם הניתוב שבו משתמשים השערים על מנת לנתב בין הקבוצות השונות נקרא: inter autonomous system routing protocol.
לסיכום:

- בכל AS כל הראוטרים מריצים את אותו intra autonomous system routing protocol.
- ראוטרים מיוחדים שנקראים gateways routers והם נמצאים בקבוצות שונות מריצים inter autonomous system routing protocol.
- בעיית ה- Scale נפתרה כי ראוטרים בתוך כל קבוצה צריכים לשמור אינפורמציה רק לגבי הקבוצה שלהם.
- בעיית ה- administrative authority נפתרה מאחר שכל ארגון יכול להריץ כל פרוטוקול שהוא רוצה בתוך ה- AS כל עוד ראוטר השער יכול להריץ את הפרוטוקול שלו.

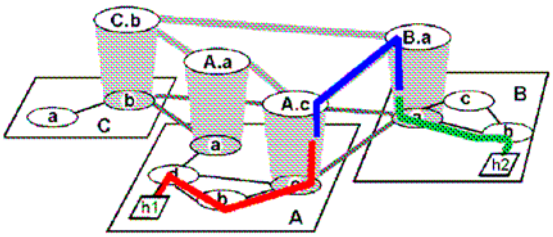
דוגמא:
בתרשים ניתן לראות 3 קבוצות AS: A, B ו-C.
לקבוצה A יש 4 ראוטרים: Aa, Ab, Ac, Ad והם מריצים את פרוטוקול הניתוב שלהם.
כל אחד מארבעת הראוטרים יודע הכול על השני.
באותה מידה לקבוצה B יש 3 ראוטרים ולקבוצה C יש 2 ראוטרים.
ה- gateway routers הם: A.a, A.c, B.a, C.b והם מריצים איזשהו אלגוריתם ניתוב ביניהם (שפועל על המערכת האפורה).



הציור הבאה מראה שראוטר A.c שהוא gateway router צריך להריץ intra-AS routing protocol עם השכנים (A.b ו-A.d) וגם inter-AS protocol עם ראוטר השער B.a.



נסכל על התרחיש הבא:
נניח שתחנת קצה h1 המתחברת לראוטר A.d צריכה לנתב חבילה לתחנת קצה h2 שנמצאת בקבוצה B.
נניח שטבלת הניתוב של A.d מראה שראוטר A.c אחראי לניתוב החבילות של A.d מחוץ לקבוצה A. מה יקרה?
החבילה מנותבת בהתחלה מ- A.d ל-A.c תוך שימוש בפרוטוקול הניתוב הפנימי של קבוצה A. נזכור שראוטר A.d לא יודע כלום על מבנה קבוצות B ו-C וגם לא יודע כלום על הטופולוגיה שמחברת בין הקבוצות.
ראוטר A.c מקבל את החבילה ורואה שהיעד שלה הוא מחוץ לקבוצה A. טבלת הניתוב של A לפרוטוקול הניתוב בין הקבוצות, מראה שיעד של החבילה הוא לקבוצה B והניתוב שלה הוא מ-A.c ל-B.a. לכן, A.c יודע לשלוח את החבילה ל-B.a. כשהחבילה מתקבלת ב- B.a ת פרוטוקול הניתוב הפנימי של קבוצה B רואה שיעד החבילה הוא בתוך B ולכן מעביר אותה אליו. אז החבילה מנותבת ליעד הסופי h2.



מסלול אדום- A's intra-AS protocol
מסלול כחול- inter-AS routing protocol
מסלול ירוק- B's intra-AS protocol
על אלגוריתמי הניתוב בין הקבוצות נלמד בחלק 4.5.

Internet Protocol -4.4

תזכורת מחלק 4.1: רמת הרשת באינטרנט לא מספקת שירות VC אלא שירות connectionless datagram. כאשר שכבת הרשת, בתחנת הקצה השולחת מקבלת סגמנט משכבת ה- transport, היא עוטפת את הסגמנט ב- IP datagram, כותבת את כתובת היעד של תחנת הקצה האחרת (ועוד כמה שדות) ב- datagram, ומפילה את ה- datagram לתוך הרשת. (בדומה לתהליך שבו אדם כותב מכתב, מכניס אותו לתוך מעטפה, כותב את כתובת היעד על המעטפה וזורק את המכתב לתיבת דואר). גם ברשת האינטרנט וגם בשירות הדואר לא יוצרים קשר עם היעד לפני שמתחיל תהליך העברת המכתב/ חבילה.
רשת האינטרנט בכלל לא מתחייבת שה- datagram תגיע בזמן מסוים, לא מתחייבת שסדרה של datagrams תגיע באותו סדר שבו נשלחה ולמעשה, היא בכלל לא מתחייבת שה- datagram בכלל תגיע.
לשכבת רשת שמשתמשת בשירות datagram יש שני מרכיבים עיקריים:

- פרוטוקול הרשת שמגדיר את מרחב הכתובות, השדות שב- datagram ואיך תחנות הקצה והראוטרים פועלים לפי שדות אלה.
  פרוטוקול הרשת באינטרנט נקרא Internet Protocol או IP Protocol.
  היום משתמשים בשתי גרסאות של פרוטוקול IP . בפרק הזה נלמד את הגרסה הנפוצה יותר IPv4 (גרסה 4).
  בהמשך נלמד על גרסה 6 שצפויה להחליף את גרסה 4 בשנים הקרובות.
- מרכיב האחראי לקביעת את המסלול ליעד ועליו נלמד עכשיו.

IP Addressing

לתחנת קצה יש קשר (link) אחד לתוך הרשת. כאשר IP בתחנת הקצה רוצה לשלוח , הוא מעביר את ה- datagram ללינק שלה. הגבול בין תחנת הקצה ללינק נקרא interface.

לראוטר יש לינק אחד או יותר שמחוברים אליו. כאשר ראוטר מעביר הלאה datagram הוא מעביר דרך לינק אחד בלבד. הגבול בין הראוטר לכל אחד מהלינקים שמחוברים אליו גם נקרא interface. מכאן שלראוטר יש כמה ממשקים, אחד עם כל לינק.

מכיוון שכל ממשק יכול לשלוח או לקבל IP datagram, נדרש שלכל ממשק תהייה כתובת IP.

אורכה של כתובת IP הוא 32 ביטים והיא כתובת ב- "dot-decimal notation" כלומר כל בית (byte) של הכתובת כתוב בצורה הדצימלית שלו והוא מופרד בנקודה.

לדוגמא, נניח שכתובת IP היא 193.32.216.9. המספר 193 מיוצג ע"י 8 הביטים הראשונים של הכתובת, המספר 32 מיוצג ע"י 8 הביטים הבאים וכך הלאה.

הכתובת הבינארית תהייה: 11000001 00100000 11011000 00001001

מכיוון שכל כתובת IP מורכבת מ-32 ביטים, מספר הכתובות האפשרי הוא 2^{32}.

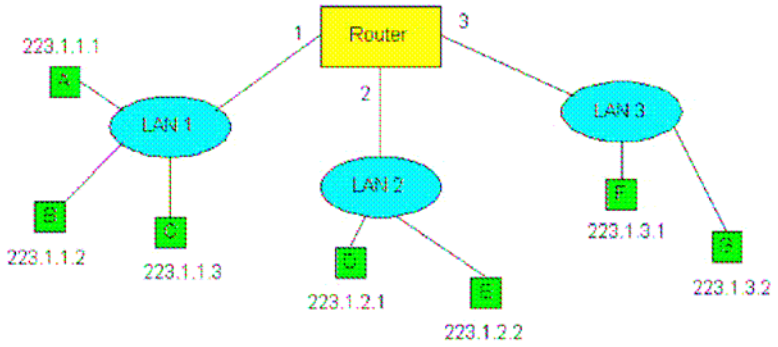
כתובות IP הן ייחודיות בעולם.

בציור הבא ניתן לראות דוגמא למרחב כתובות IP וממשקים. הראוטר מחבר שלושה LANs (אזור רשת מקומי). לפי הג'רגון של ה-IP, כל LAN נקרא IP network או network.

נשים לב שלראוטר יש שלושה ממשקים (1, 2 ו-3) ולכל אחד מהם יש כתובת IP (לא נמצא בציור), לכל תחנת קצה יש גם כתובת IP.

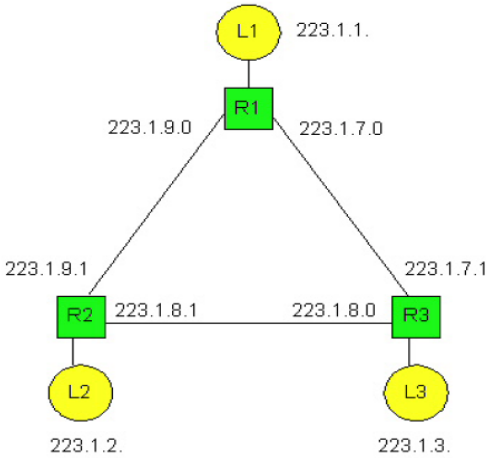
לכל כתובת יש שני חלקים:

- שלושת הבתים הראשונים – מגדירים את הרשת. Network mask
- שאר הבתים- מגדירים תחנת קצה ספציפית. Host mask



בדוגמא,
לכל הממשקים שמחוברים ל-LAN1 (כולל הממשק של הראוטר) יש כתובת IP מהצורה 223.1.1.xxx,
לכל הממשקים שמחוברים ל-LAN2 יש כתובת IP מהצורה 223.1.2.xxx, ולכל אלה שמחוברים ל-LAN3 יש כתובת IP מהצורה 223.1.3.xxx.

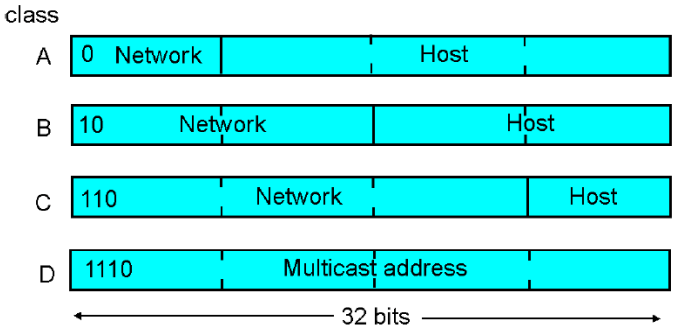
ה-IP שמגדיר את הרשת אינו מוגבל ל-LAN.



נסתכל על הדוגמא הבאה:
בציור רואים כמה LANs שמחוברים לשלושה ראוטרים.
כל הממשקים שמחוברים ל-LAN1, כולל הממשק של R1, בעלי כתובת מהצורה 223.1.1.xxx .
כמו כן, כל הממשקים שמחוברים ל-LAN2 הם מהצורה 223.1.2.xxx ואלה שמחוברים ל-LAN3 הם מהצורה 223.1.3.xxx.
נשים לב שלכל LAN יש את הרשת שלו (IP network) אבל יש עוד 3 רשתות נוספות:
רשת עבור הממשקים שמחברים בין R1 לR2, רשת עבור הממשקים שמחברים בין R2 לR3 וכך הלאה.

במצגת, המרצה התייחס ל-LAN כאל subnet.

ישנם ארבעה פורמטים אפשריים לכתובת IP:



באופן כללי כל ממשק שייך לרשת. כאמור, החלק הראשון (network part) מראה את הרשת אליה שייך הממשק, והחלק השני (host part) מראה את הממשק הספציפי בתוך אותה רשת.
בתוך מחלקה A, 8 הביטים הראשונים מגדירים את הרשת ו-24 הביטים הבאים מגדירים את הממשק הייחודית. מכאן, שבמחלקה A יכולות להיות לנו 2^7 רשתות (הביט הראשון מתוך ה-8 הוא 0), ו- 2^{24} ממשקים ייחודיים.
במחלקה B מרחב הרשתות הוא 2^{14} ומרחב הממשקים הוא 2^{16} בתוך כל רשת.
מחלקה C משתמשת ב- 21 ביטים לרשת וב-8 לממשק.
מחלקה D שמורה לכתובות שמשדרות בו זמנית (נראה בהמשך).

Assigning Addresses

ניתן לקבל כתובת IP בשתי דרכים:

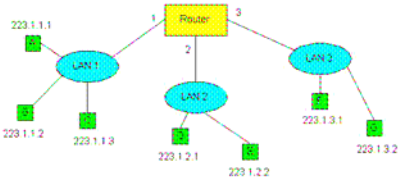
- Manual configuration : קבלה ידנית של הכתובת ע"י מערכת האדמיניסטרציה.
- DHCP – Dynamic Host Configuration Protocol: כשמחשב מתחבר לרשת השרת מקצה לו כתובת זמנית.

את החלק של הרשת בכתובת מקבלים מספק השירות שיש לו מרחב כתובות שקנה. ספק שירות יכול לאחד את תתי הרשתות שמתחתיו לכתובת אחת וכל פנייה ללקוח תנותב ישירות מהספק. שיטה זו טובה בכך שהיא מפחיתה את מספר הכתובות שהראוטרים צריכים לזכור.
נניח שאחד מהארגונים שנמצא מאחורי כתובת של ספק שירות מסוים רוצה לעבור לשרת אחר והוא רוצה לשמור על מרחב הכתובות שלו. על מנת לאפשר זאת הראוטר של ספק השירות הישן מוסיף כלל בנוסף לכללים הקיימים, שכל חבילה שנשלחת לכתובת הספציפית הישנה של הארגון תנותב לספק החדש.
ספקי השירות מקבלים את מרחבי הכתובות שלהם מארגון שנקרא ICANN
(Internet Corporation for Assigned Names and Numbers) .

The big picture: Transporting a Datagram from Source to Destination

באופן כללי, שדות המפתח של IP datagram הם:

נקודת הקצה שממנה נשלח המידע ממלאת את ה- source address בכתובת IP שלה ואת ה- destination address בכתובת ה-IP של תחנת הקצה אליה מיועד המידע. (כל כתובת 32 ביט).
שדה ה- Data ממולא ע"י סגמנטים של TCP או UDP.
לגבי שאר השדות, נדון עליהם בהמשך.
לאחר שיש datagram מוכן, איך שכבת הרשת מעבירה אותו מהמקור ליעד? נסתכל בדוגמא שראינו קודם.



<u>העברת datagram מ-A ל-B</u> נעשית בצורה הבאה:
IP מחלץ את החלק של הרשת מהכתובת של A (233.1.1) ועובר על טבלת הניתוב שלו.
בטבלה, number of hops to destination זה מספר הרשתות שצריך לעבור (כולל הרשת בא נמצא היעד). במקרה שלנו, מספר ה-hops הוא 1 שזה אומר שהיעד נמצא באותה רשת. A מעביר את ה- datagram לפרוטוקול ה- link layer ומראה לו שהיעד נמצא באותו LAN. פרוטוקול ה-link layer אחראי להעביר את ה- datagram ל-B.
<u>העברת datagram מ-A ל-E</u>:
כתובת ה- IP של היעד היא: 223.1.2.2 ולפי הטבלה של A מספר ה-hops ליעד הוא 2 כלומר היעד נמצא ברשת אחרת. כמו כן, בטבלה כתוב לאיזה ראוטר יש לשלוח את ה- datagram על מנת שהיא תגיע ליעד (223.1.1.4).
IP מעביר את ה- datagram ל- link layer ואומר לו שצריך לשלוח את ה- datagram לכתובת 223.1.1.4.

ה-link layer מעביר את ה- datagram לממשק 1 של הראוטר.
עכשיו תפקידו של הראוטר להעביר את ה- datagram ליעד. הראוטר מחלץ את החלק של הרשת בכתובת של היעד (223.1.2) וסורק את טבלת הניתוב שלו. על פי הטבלה צריך להעביר את ה- datagram דרך ממשק 2. לאחר שה-datagram בממשק הנכון, היא מועברת לפרוטוקול ה- link layer עם הודעה שהיעד נמצא באותו LAN. הפרוטוקול אחראי להעביר את ה- datagram מממשק 2 של הראוטר ל-E.

טבלת הניתוב של A:

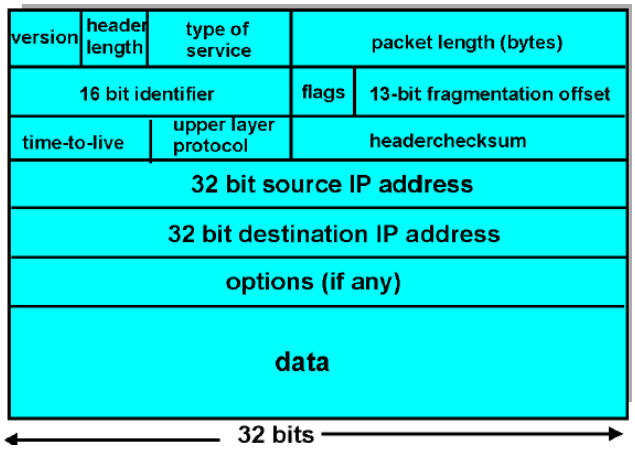
destination network	next router	number of hops to destination
223.1.1.	-	1
223.1.2.	223.1.1.4	2
223.1.3.	223.1.1.4	2

טבלת הניתוב של הראוטר:

destination network	next router	number of hops to destination	interface
223.1.1.	-	1	1
223.1.2.	-	1	2
223.1.3.	-	1	3

<u>Datagram Format</u>

- Version (4 ביט) – הגרסה של פרוטוקול ה- IP. לפי מספר הגרסה, הראוטר יידע איך לפרש את שאר השדות ב-datagram. (גרסאות שונות משתמשות בפורמטים שונים של datagrams). הפורמט שמופיע כאן הוא עבור גרסה 4.
- Header Length (4 ביט) – מכיוון שה- datagram יכולה להכיל מספר אופציות, צריך להכניס את אורך ה-header, כלומר היכן ב- datagram המידע באמת מתחיל. רב ה- datagrams לא מכילות אופציות ולכן אורך ה-header הוא בד"כ 20 בתים.
- TOS (Types of Service) – נתינת עדיפויות ל- datagrams למצב של עומס למשל. על פי המרצה, אין סטנדרט שמחייב ראוטרים לנהוג לפי עדיפויות. לפי הגדרת הרשת, הראוטרים יבצעו זאת או לא ולכן זה לא עובר בין רשתות.
- Datagram length (16 ביט)- האורך של כל ה- datagram כולל ה- header. נמדד בבתים. מכיוון שאורך השדה הזה הוא 16 ביט , האורך המקסימאלי של datagram הוא 65,535 בתים. בד"כ האורך נע בין 576 בתים ל-1500 בתים.
מידע זה דרוש כדי לדעת כמה באפרים להקצות.
- Identifier, Flags, Fragmentation Offset: שלושת השדות קשורים לפרגמנטציה. נדון על כך בהמשך.
- Time-to-live: מגביל את זמן ה- datagram ברשת. יכול להיווצר מצב שבו ה- datagram נכנסת ללולאה ואם לא יוגבל זמן החיים הוא יישאר תקוע בלולאה אינסופית. השדה קטן ב-1 בכל פעם שה- datagram מגיעה לראוטר. אם ראוטר יקבל datagram עם TTL=0 הוא יזרוק אותה.
- Protocol: השדה הזה מתמלא רק כשה- datagram מגיעה ליעד הסופי. זהו מספר שמראה לפרוטוקול ה-transport layer שבנקודת היעד, לאן ה-data של ה-datagram יעבור. למשל 6 אומר שה- data יעבור ל- TCP ו-17 אומר שה- data יעבור ל- UDP.
נשים לב שלמספר הזה יש תפקיד דומה למספר ה- port שבסגמנט ה-transport layer .
מספר הפרוטוקול הוא "הדבק" בין רמת הרשת לרמת ה-transport ומספר הפורט הוא "הדבק" בין רמת ה-transport לאמת האפליקציה.
- Header Checksum : עוזר לראוטר להבחין בשגיאות כשהוא מקבל את ה-IP datagram.
- Source and Destination IP address: כל אחת מהכתובות היא באורך של 32 ביט. הצורך בכתובת היעד ברורה. לגבי כתובת המקור, היא משמשת בתחנת הקצה לכוון את ה- application data בצינור הנכון.
- Options: בדרך כלל משתמשים רק ל-debugging.
- Data (payload): השדה החשוב ביותר. ברב המקרים מכיל את סגמנט ה- transport layer (TCP או UDP) שאמור לעבור ליעד. אולם הוא יכול להכיל עוד סוגים של data כמו הודעות ICMP (נלמד בהמשך).



כמה בסה"כ עובר? 20 בתים של ה- header ואם מדובר על סגמנט TCP אז זה עוד 20 בתים. סה"כ 40 בתים.

<u>IP Fragmentation and Reassembly</u>

כל רשת מגדירה את כמות ה-data המקסימאלית שחבילה יכולה לשאת:
MTU(maximum transfer unit).
מכיוון שכל IP-datagram עטופה בחבילה ב- link layer עבור מעבר מראוטר אחד לשני, ה- MTU של פרוטוקול ה- link layer מגדיר גבול נוקשה על האורך של ה- IP datagram.
הבעיה נוצרת כאשר בין כל שני ראורטרים יש פרוטוקול link-layer שונה ולכל אחד מהם יש MTU שונה. למה יש בעיה? ניקח לדוגמא ראוטר שמחוברים אליו כמה קשרים וכל אחד מריץ פרוטוקול link layer שונה עם MTU שונה.
אם הראוטר צריך להעביר datagram דרך קשר שבו מוגדר MTU שקטן מאורך ה- datagram, נוצרת בעיה.
הפיתרון: פרגמנטציה, כלומר פיצול ה- datagram לכמה datagrams קטנות שכל אחת נקראת fragment.
פרגמנטים צריכים להתאחד שוב לפני שהם מגיעים ל-transport layer ביעד, כלומר גם TCP וגם UDP מקבלים משכבת הרשת סגמנטים שלמים ולא מפוצלים.
אז מי יאחד את הפרגמנטים?

- אפשרות אחת היא שהראוטרים יעשו זאת (במעבר לקשר עם MTU גדול יותר). הבעיה היא שזה מוסיף עבודה רבה לכל ראוטר (וגם ככה הוא מאוד עסוק). כמו כן זה לא כל כך יעיל כי יכול להיווצר מצב שבו datagram תפוצל ותאוחד כמה פעמים.
- אפשרות שנייה היא שתחנות הקצה יאחדו את הפרגמנטים. (שומר גם על עיקרון ה- end-to-end של האינטרנט).

איחוד ע"י תחנת קצה:
תחנת הקצה צריכה לדעת אם היא בכלל צריכה לאחד את ה- datagrams שהיא מקבלת. אם כן, אז מתי הגיעה ה- datagram הראשונה ומתי האחרונה.
לשם כך, מוגדרים 3 שדות ב- header של ה- datagram:

- Identification: כשנוצרת datagram, המקור השולח נותן לה מספר ID (ביחד עם הכתובת של המקור והכתובת של היעד). המקור השולח מגדיל את ה-ID עם כל datagram שהוא שולח. כאשר ראוטר רוצה לפצל datagram, כל פרגמנט קטן שיוצא מקבל את כתובת המקור, כתובת היעד ומספר ה-ID של ה- datagram המקורי. כשיעד מקבל את סדרת ה- datagrams מאותו מקור, הוא יכול לבדוק את ה-ID ולהחליט אילו datagrams הם בעצם פרגמנטים של datagram גדולה.
- Flags: מכיוון שIP הוא שירות לא אמין, פרגמנטים יכולים להיאבד בדרך. על מנת שהיעד יידע בוודאות שקיבל את הפרגמנט האחרון של ה- datagram המקורית, לפרגמנט האחרון יש flag של ביט אחד ששווה ל-0 ולשאר הפרגמנטים ה- flag שווה ל-1.
- Offset: משמש כדי להראות היכן הפרגמנט נמצא ב- datagram המקורית.

דוגמא:
datagram בגודל של 4000 בתים הגיעה לראוטר והיא צריכה להמשיך ללינק עם MTU של 1500 בתים. זה אומר ש- 3980 הבתים של ה- data צריכים להתחלק לשלושה פרגמנטים שכל אחד מהם הוא datagram. נניח שה- ID של ה- datagram המקורית היה 777, הפרגמנטים ייראו כך:

	פרגמנט 1	פרגמנט 2	פרגמנט 3
אורך	1480 בתים	1480 בתים	1020 בתים
ID	777	777	777
offset	0	1480	2960
flag	1 (כלומר יש עוד)	1 (כלומר יש עוד)	0

דוגמא למצב קשה:
ראוטר מפצל datagram בגודל 10k לפרגמנטים בגודל 1k. בעקבות עומס, הגיעו רק 9 פרגמנטים (אחד הלך לאיבוד). תחנת הקצה המקבלת לא תישלח ack על ה-datagram כי היא לא קיבלה את כולה. התחנה השולחת תישלח שוב את ה-datagram המקורית, כלומר בזבוז של 9k .

ICMP – Internet Control Message Protocol

משמש תחנות קצה וראוטרים כדי להעביר אינפורמציה ברמת הרשת. בד"כ דיווח על שגיאות (כמו אי יכולת להגיע לפרוטוקול, פורט, רשת, תחנת קצה). משמש גם ב- ping, צד אחד שולח echo request והצד השני עונה echo reply. מדובר בשכבת רשת שהיא "מעל" IP: הודעות ICMP מועברות בתוך IP datagram. כלומר upper layer protocol. להודעות ICMP יש שדות של type ו-code והן כוללות את 8 הבתים הראשונים של חבילת IP שיצרה את ההודעה (כך השולח יכול לדעת איזו חבילה גרמה לשגיאה).
להלן טבלה שמכילה כמה הודעות ICMP:

Type	Code	description
0	0	echo reply (ping)
3	0	dest network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	Source quench (congestion control-not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Traceout – אפליקציה שבודקת מהו המסלול בין שני מחשבים כלשהם (מפורט בפרק 1).
Traceout משמש גם הודעות ICMP. על מנת לקבוע שמות וכתובות של ראוטרים בין המקור ליעד, traceroute במקור שולח סידרה של datagrans ליעד. ל- datagram הראשונה יש TTL של 1, לשניה יש TTL של 2 וכך הלאה. המקור מודד זמן (בעזרת timer) לכל אחד מה- datagrams.
כאשר ה- datagram ה- i מגיעה לראוטר ה- i, ראוטר זה רואה שה-TTL של ה- datagram בדיוק נגמר. לפי החוקים של פרוטוקול IP, הראוטר זורק את ה- datagram ושולח הודעת אזהרה ICMP למקור. הודעה זו כוללת את השם והכתובת של הראוטר. כאשר הודעת ה- ICMP אשר מקבילה ל-datagram ה-i מגיעה למקור, המקור יכול לחשב את ה- RTT מה-timer ואת השם והכתובת מהודעת ה- ICMP.

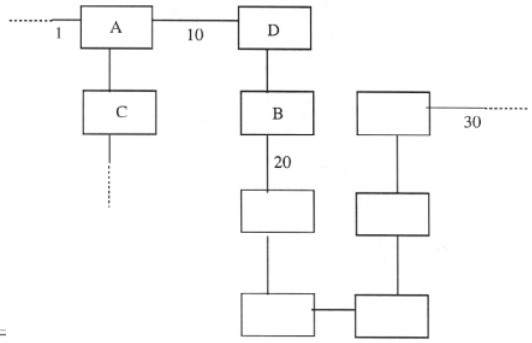
Routing in the Internet -4.5

כאמור, האינטרנט מכיל הרבה ASs (interconnected autonomous systems). כל AS מכיל הרבה רשתות (IP network). כל AS הוא עצמאי ומריץ איזה פרוטוקול ניתוב שירצה.

פרוטוקלי ניתוב בתוך AS (Intra-AS routing protocols נקראים גם interior gateway (protocols. ישנם שלושה פרוטוקולים נפוצים: RIP, OSPF ו-IGRP.

RIP (Routing Informative Protocol) - אחד הפרוטוקולים הראשונים לניתוב ב-AS.
אלגוריתם וקטור מרחק (כמו בלמן פורד).
משקל כל לינק הוא 1 ואורך המסלול המקסימאלי הוא 15 (מעל 15 המרחק נחשב כאינסופי). מכאן ש-RIP מוגבל רק ל-AS עם קוטר של פחות מ-15 לינקים.
נזכיר שבפרוטוקולים של וקטור מרחק, ראוטרים שכנים מחליפים אינפורמציה אחד עם השני.
ב-RIP, טבלאות הניתוב מתחלפות בין שכנים כל 30 שניות. זה נעשה דרך response message (נקרא גם advertisement).
כל advertisement מכילה רשימה של עד 25 יעדים ב- AS.

דוגמא:
בציור הבא, כל מלבן מייצג ראוטר (B,A...) והקווים מייצגים רשתות (1,10,20....).



destination network	next router	number of hops to destination
1	A	2
20	B	2
30	B	7
10	--	1
....	....	....

הטבלה מייצגת את טבלת הניתוב עבור ראוטר D.
עמודה ראשונה – יעד כלשהו ברשת.
עמודה שנייה – הראוטר הבא לאורך המסלול הקצר ביותר ליעד.
עמודה שלישית – מספר ה- hops כלומר מספר הרשתות שצריך לעבור כולל הרשת שבא נמצא היעד.
לפי הטבלה, על מנת לשלוח datagram מראוטר D לרשת 1, צריך קודם לשלוח את ה-datagram לראוטר השכן A.
לפי הטבלה, רשת 1 רחוקה ב- 2 hops ורשת 30 נמצאת 7 hops מראוטר B

destination network	next router	number of hops to destination
30	C	4
1	--	1
10	--	1
....	....	....

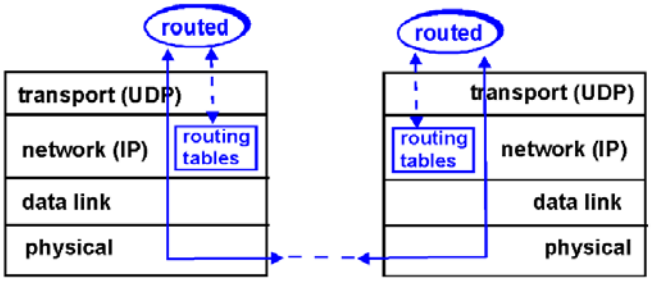
לאחר 30 שניות, ראוטר D מקבל מראוטר A את ה- advertisement הבא:
נשים לב שה- advertisement הוא פשוט טבלת הניתוב של A. לפי הטבלה, לרשת 30 יש רק 4 hops מ-A.

destination network	next router	number of hops to destination
1	A	2
20	B	2
30	A	5
....	....	....

לאחר שראוטר D מקבל את ה- advertisement, מאחד אותו עם הטבלה הישנה שלו.
ראוטר D מבין שיש עכשיו מסלול דרך ראוטר A לרשת 30 שהוא קצר יותר מהמסלול דרך ראוטר B. לכן, ראוטר D מעדכן את הטבלה שלו

אם ראוטר לא שומע שום דבר מהשכן שלו למשך 180 שניות, השכן לא יהיה ניתן להשגה כלומר או שהלינק ביניהם מת או שהשכן מת. כשמקרה כזה קורה, טבלת הניתוב משתנה ונשלחות advertisements לשכנים שעדיין ניתנים להשגה.

הציור הבא מתאר איך ה- RIP ממומש ב-UNIX (כלומר תחנת עבודה של unix משמשת כראוטר).
תהליך שנקרא routed (מבטאים route dee) מריץ את פרוטוקול RIP (מחזיק את טבלאות הניתוב ומחליף הודעות בתהליכי routed שרצים בראוטרים השכנים)
מכיוון שRIP ממומש כתהליך ברמת האפליקציה, הוא יכול לשלוח ולקבל הודעות דרך צינור סטנדרטי ולהשתמש בפרוטוקול transport, מכאן ש-RIP הוא פרוטוקול רמת האפליקציה שרץ על UDP.

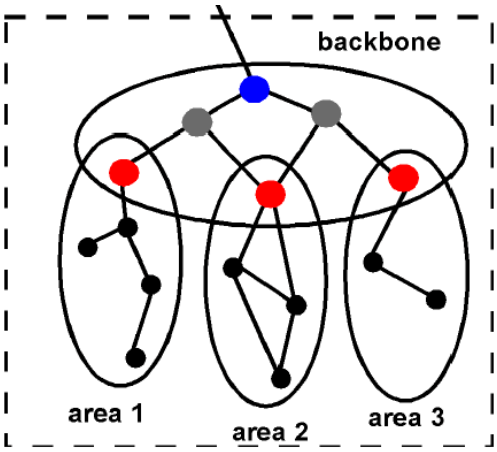


<u>OSPF (Open Shortest Path First)</u>
ה- open אומר שפרוטוקול הניתוב נגיש לכולם.
הפרוטוקול הוא link state והוא מבוסס על dijkstra. בפרוטוקול זה, ראוטר יודע את הטופולוגיה הכוללת של המפה (גרף מכוון) של כל ה- AS. הראוטר מריץ dijkstra מקומי כדי למצוא את העץ הקצר ביותר לכל הרשתות כאשר הוא עצמו השרש. טבלת הניתוב של הראוטר נבנית מהעץ.
נשווה את ה- advertisement שנשלח ע"י RIP לעומת זה הנשלח ע"י OSPF.
עם OSPF, הראוטר שולח אינפומציה באופן מחזורי <u>לכל</u> הראוטרים האחרים שב-AS (לא רק לראוטרים השכנים שלו).
לאינפומציה זו יש entry לכל אחד מהשכנים ושם יש מידע על המרחק (link state) מהראוטר לשכן.
עם RIP, האינפורמציה שנשלחת ע"י ראוטר כוללת אינפורמציה על כל הרשתות ב- AS, למרות שהאינפורמציה הזו נשלחת רק לשכנים.
במובן מסוים, ניתן להגיד שטכניקת ה- advertising של ה- RIP ושל ה- OSPF דומות.
היתרונות ב- OSPF:

- Security: כל החלפות המידע בין הראוטרים (כלומר link state updates) מקבלות אישור. זה אומר שרק ראוטרים אמינים משתתפים בפרוטוקול וזה מונע מפורצים רשעים להחדיר מידע לא נכון לטבלאות הראוטרים.
- Multiple same cost paths: כאשר יש מספר מסלולים בעלי אותו מחיר לאותו יעד, הפרוטוקול מאפשר לכולם להיות בשימוש. כך הראוטרים יכולים לעשות איזון עומסים ע"י שליחה פעם ממסלול אחד ופעם ממסלול שני.
- Different cost metrics for different TOS traffic: הפרוטוקול מאפשר שלכל לינק יהיו מספר מחירים עבור שירותים שונים (TOS = type of service). למשל, מחירו של לינק לוויני בעל רוחב פס גדול יכול להיות נמוך כשאין עומס משמעותי, ולהיות גבוה כשיש עומס.
  כלומר, OSPF רואה טופולוגיות שונות לרשת עבור רמות שונות של עומס ולכן יכול לחשב מסלולים שונים לכל מצב של עומס.
- Integrated support for unicast and multicast routing (בהמשך נבין מה ההבדל בין השניים).

<u>OSFP היררכי:</u>
ניתן לחלק את הראוטרים לאזורים, כשכל אזור מריץ OSPF משלו וכל ראוטר באזור משדר את ה- link state שלו לכל הראוטרים שבאזור שלו. הפרטים הפנימיים של כל אזור אינם ידועים לראוטרים מחוץ לאזור. ניתוב בתוך אזור מערב רק את הראוטרים בתוך אותו אזור.
בתוך כל אזור יש כמה ראוטרים שנקראים area border router והם אחראיים לנתב חבילות מחוץ לאזור. אזור אחד נחשב כאזור השלד (backbone) ותפקידו העיקרי הוא לנתב את התנועה בין האזורים האחרים ב- AS. אזור השלד מכיל את כל ה- border routers ב- AS. באלגוריתם ניתוב בין אזורים ב-AS, החבילה מנותבת בהתחלה ל-border touter, אז מועברת בתוך אזור השלד ל-border router שנמצא באזור בו נמצא היעד ומשם מועברת ליעד הסופי.

דוגמא להיררכיה:



Internal routers (שחורים) מריצים רק אלגוריתמים בתוך ה- AS.
Area border routers (אדומים) שייכים גם לאזור וגם לשלד.
Backbone routers (אפורים) מריצים אלגוריתם ניתוב בתוך השלד.
הראוטרים הפנימיים מקבלים מידע על ראוטרים מאזורים שונים מאינפורמציה ששודרה לאזור ע"י ראוטרי השלד.
Boundery routers (כחול) מחליפים מידע עם ראוטרים ששיכים ל- AS אחר

IGRP: Internal Gateway Routing Protocol – לא למדנו

פרוטוקל ניתוב בין Ass (Inter AS routing)
<u>BGP : Border Gateway Protocol</u> – זהו פרוטוקול path vector (דומה ל-distance vector) שבו הראוטר מפיץ אינפורמציה על מסלול, כמו רצף ה-ASs שצריך לעבור כדי להגיע ליעד (בניגוד ל-distance vector שבו הראוטר הפיץ אינפורמציה על מחיר כמו מספר ה- hops עד ליעד).
נשים לב שאינפורמציה זו מכילה את שמות כל ה- ASs במסלול ליעד, אבל לא מכילה את המחיר של המסלול. כמו כן, BGP לא מציין איך מסלול מסוים אמור להיבחר מבין כל המסלולים האפשריים. ההחלטה נשארת לכל אדמיניסטרטור של דומיין. כל דומיין יכול לבחור את המסלולים שלו לפי איזה קריטריון שירצה ולא צריך לדווח על כך לדומיינים אחרים.
כלומר, BGP מספק את המנגנון להעברת המידע בין ה-ASs, אבל משאיר את ההחלטה על איך ייבחר המסלול לאדמיניסטרטור של הרשת.

איך BGP עובד?
זוג ראוטרים (BGP peers) מחליפים מידע דרך תקשורת TCP: BGP sessions. המסלול בין הראוטרים הוא מסלול לוגי ולא לפי החיבור הפיסי ביניהם (כלומר שני ראוטרים שכנים במסלול לאו דווקא קשורים פיסית אחד לשני).
כאמור, לכל AS יש AS number. בכל זמן נתון, AS מספר x, יודע או לא יודע את מסלול ה- ASs שמוביל ל-AS מספר z. לדוגמא, נניח ש- x הכין רשימה בטבלת ה- BGP שלו שנראית כך : $xy_1y_2y_3z$. זה אומר ש-x יודע שהוא יכול לשלוח datagram ל-z דרך y_1, y_2 ו- y_3 . כאשר x שולח עדכון לשכנים שלו, x שולח בין השאר את כל המסלול ממנו ל-z. אם למשל w הוא שכן של x, לאחר האינפורמציה שקיבל מ-x הוא יוכל להרכיב מסלול חדש ולהכניס אותו לטבלת ה- BGP ($wxy_1y_2y_3z$). w יכול להחליט גם לא להכניס את המסלול החדש מהסיבות הבאות: w כבר נמצא במסלול (נניח ש- $y_3=w$), ל-w כבר יש מסלול ל-z בטבלה שלו והוא טוב יותר, ל-w יש מדיניות שלפיה הוא לא מעביר datagram דרך למשל y_2.

ישנן 4 סוגי הודעות ב- BGP:
- OPEN – פתיחת תקשורת TCP בין שני peers.
- UPDATE – משמשת להעברת אינפורמציה על מסלול.
- NOTIFICATION – להודיע ל-peer שיש שגיאה בהודעה הקודמת. משמש גם לסגירת קשר.
- KEEPALIVE – ה-peer מחזיר הודעה זו לאחר שפעולת פתיחת ה- TCP הצליחה

הרבה פעמים, ל-AS יש כמה ראוטרים המשמשים שערים ליצירת תקשורת עם AS-ים אחרים. למרות ש- EGP הוא פרוטוקול ניתוב שבין AS-ים, הוא יכול לשמש גם בתוך ה-AS עצמו כצינור לעדכונים בין ראוטרי השערים ששיכים לאותו AS.
בתוך AS, הפרוטוקול נקרא Internal BGP (IBGP), ומחוץ ל-AS הוא נקרא External BGP (EBGP).

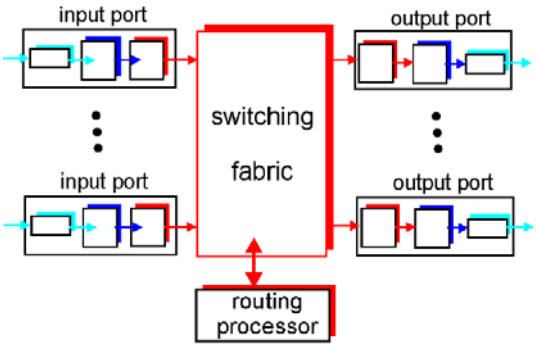
<u>מדוע יש פרוטוקולים שונים לניתוב בתוך ובין ASs?</u>
- מדיניות (Policy):
בין ASs, אילוצים של מדיניות הם דומיננטים. כל AS שולט באיך התנועה תנותב, מי עובר דרכו ומי ייכנס אליו.

בתוך AS , מכיוון שהשליטה היא ע"י אותו AS , המדיניות משחקת תפקיד קטן בבחירת פרוטוקול הניתוב.

- Scale:
בין ASs , היכולת של אלגוריתם הניתוב לטפל בכמות גדולה מאוד של רשתות היא קריטית.
בתוך AS, יכולת זה היא פחות משמעותית מכיוון שאם זה נעשה גדול מידי, תמיד אפשר לפצל לשני ASs.
- Performance:
מכיוון שניתוב בין ASs מושפע מאוד ממדיניות, איכות הראוטרים פחות חשובה. לעומת זאת, בתוך AS איכות הראוטרים היא חשובה מאוד.

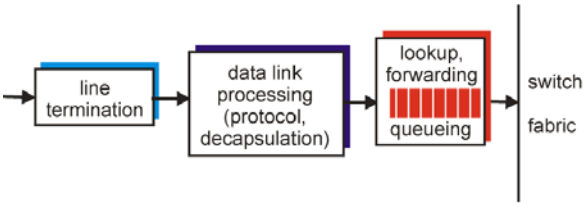
What's inside a router? 4.6

החלק החשוב ביותר ברמת הרשת (בלעדיו היא לא הייתה קיימת) הוא העברת ה- datagrams מהמקור השולח ליעד המקבל. מרכיב חשוב ביותר בתהליך הזה הוא העברת datagram מהלינק שנכנס לראוטר ללינק שיוצא מהראוטר. בחלק הבא נלמד איך זה נעשה.



לראוטר יש 4 מרכיבים עיקריים:

- input ports
- switching fabric
- output ports
- routing processor



Input ports:

מבצעים מספר פעולות:

- ברמה הפיסית, סיום של לינק פיסי לראוטר. (תכלת)
- ברמת ה-link, פעולה הדדית שמתבצעת עם הצד השני של הלינק. (?) (כחות)
- פעולת החיפוש והקידום של datagram (אדום)

ישנם סוגים של ראוטרים שבהם כבר בשלב זה, הראוטר מחליט מהו הפורט שאליו ה- datagram תמשיך לאחר שתעבור דרך ה- switching fabric. הבחירה של ה- output port נעשית תוך שימוש באינפורמציה שבטבלת הניתוב.
למרות שטבלת הניתוב מחושבת ע"י הפרוססור, העתק שלה מאוחסן בכל אחד מה- input port והוא גם מתעדכן כשצריך ע"י הפרוססור. כלומר עם העתקים כאלו, ההחלטה לאן לנתב את ה- datagram נעשית בצורה מקומית בכל input port, ללא עירוב של הפרוססור. ניתוב לא מרוכז כזה, מונע פקקי תנועה.
בראוטרים שבהם יש מקום מוגבל ב- input port, החבילה מועברת לפרוססור, שם הוא מסתכל על טבלת הניתוב ומחליט מה יהיה ה- port הבא.

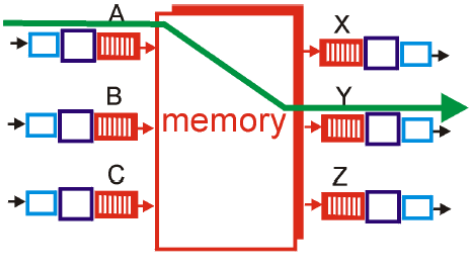
בהינתן טבלת ניתוב קיימת, ה- lookup מתבצע באופן הבא: חיפוש של כתובת היעד של ה- datagram בעמודה של היעד בטבלה. אם כתובת היעד לא נמצאה בטבלה, ישנו מסלול דיפולטיבי.
למעשה, זה לא כל כך פשוט.
הפקטור החשוב ביותר שמסבך את הכול הם הראוטרים של השלד שצריכים לבצע את העבודה שלהם במהירות גבוהה, כך שיוכלו לבצע מיליוני חיפושים בטבלה בשנייה. לכן, ה- input port יצרה להיות מסוגל להתקדם ב- line speed כלומר חיפוש יכול להיעשות בזמן קצר יותר מאשר בזמן שלוקח לקבל חבילה ל- input port. במקרה זה, תהליך ה- Input של קבלת חבילה יכול להסתיים לפני שהפעולה הבאה מסתיימת. כדי להבין את הביצועים שנדרשים לחיפוש מסתכל בדוגמא הבאה: נניח שלינק מסוג OC48 רץ ב- 2.5 Gbps ונניח שהחבילות באורך 256 בתים. אנו מקבלים שמהירות החיפוש היא בערך מיליון חיפושים בשנייה.

(את החלק הבא אני חושבת שלא למדנו)
כיום, חיפוש ליניארי בטבלת ניתוב גדולה הוא בלתי אפשרי. דרך לייעל את החיפוש היא לאחסן את טבלת הניתוב כעץ כשכל שלב בעץ מקביל לביט אחד בכתובת היעד. כדי לחפש כתובת, מתחילים מהשורש. אם הביט הראשון הוא 0, אז ממשיכים לבן השמאלי, אם הביט הוא 1 ממשיכים לתת העץ הימני.
בצורה זו, החיפוש נעשה ב-N צעדים כאשר N זה מספר הביטים שבכתובת.
אבל, עם כתובות באורך של 32 ביט, זה עדיין לא מספיק מהיר. היום מפתחים טכניקות נוספות לשיפור הביצועים.
ברגע שהוחלט מה יהיה ה- output port, החבילה יכולה לעבור ל- switching fabric. בהמשך נראה שחבילה יכולה להיחסם לזמן מסוים ולא תוכל להיכנס ל-switching fabric . זה קורה כשיש חבילות מ-input ports אחרים שכרגע משתמשים ב-fabric . חבילה חסומה נכנסת לתור ב- input port והיא תיכנס ל-fabric בשלב מאוחר יותר. (בהמשך נבחן את הנושא הזה לעומק).

Switching Fabrics

בתהליך זה ה- datagram עוברת מה- input port ל- output port.
יש מספר דרכים לעשות את ה-switching:

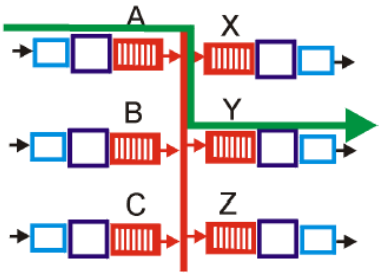
Switching via memory: הדרך הפשוטה ביותר.



- לאחר שה-input port מאותת לפרוסור, החבילה מועתקת ממנו לזיכרון הפרוסור.
- הפרוסור מחלץ את כתובת היעד מה-header, מחפש את ה-output port המתאים בטבלת הניתוב ומעתיק את החבילה ל- output port.

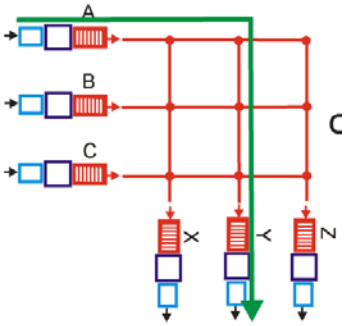
נשים לב שאם רוחב הפס של הזיכרון הוא כזה ש- B חבילות לשנייה יכולות להיכתב אליו או להיקרא ממנו, אז הקצב שחבילות יכולות לעבור מה-input ports ל-output ports הוא פחות מ-B/2.
הרבה ראוטרים מודרניים משתמשים בדרך זו אך ההבדל ביניהם לבין ראוטרים ישנים הוא בכך שהחיפוש של כתובת היעד ואחסון החבילה במקום הנכון בזיכרון נעשה ע"י פרוסורים שנמצאים על input line cards.

Switching via a bus:



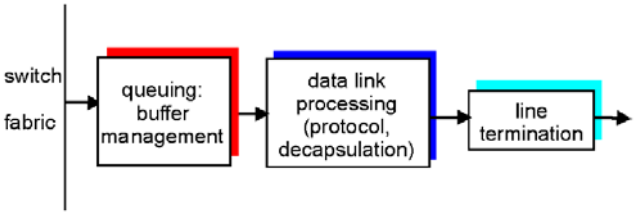
ה- input ports מעבירים datagram ישירות ל-output ports על פני אפיק משותף, ללא התערבות של הפרוסור.
מאחר שהאפיק הוא משותף, רק חבילה אחת יכולה לעבור בו בכל יחידת זמן. חבילה שמגיעה והאפיק לא פנוי צריכה לחכות בתור.
מכיוון שכל חבילה צריכה לעבור באפיק, רוחב הפס של ה- switching מוגבל למהירות האפיק.
בהינתן רוחב פס גדול יותר מ- ג'יגה ביט לשניה, ה- switching דרך האפיק מספק ראוטרים שפועלים ברשתות access ו-enterprise .

Switching via an interconnection network:



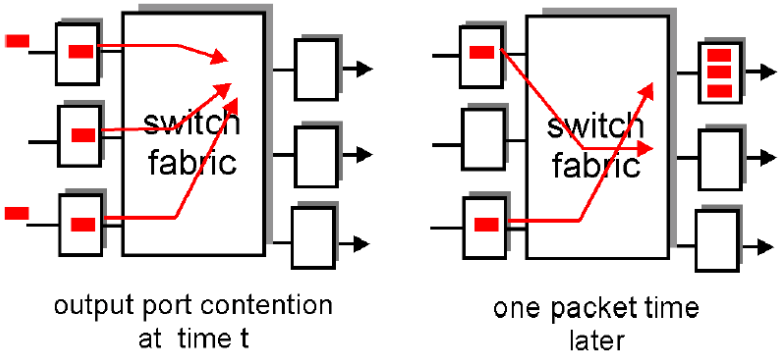
אחת הדרכים להתמודד עם הגבלת רוחב הפס של אפיק משותף היא להשתמש בעוד רשת ביניים.
crossbar הוא רשת ביניים המורכבת מ- 2N אפיקים שמחברים N input ports עם N output ports.
חבילה המגיעה ל-input port נעה לאורך האפיק המאוזן שמחובר אליו עד שהיא פוגשת אפיק אנכי שמוביל אותה ל-output port הרצוי. רק אם האפיק האנכי לא פנוי החבילה מחכה בתור ב- input port.

Output ports



חבילה שנשמרה בזיכרון ה- output port נלקחת ומועברת לעבר הלינק היוצא מהראוטר.
תהליך ה- data link וה- line termination הם כמו אלו שהיו ב- input port.
יש את ה- Buffering למקרה שהחבילות מגיעות מה- fabric בקצב מהיר יותר מקצב ההעברה.
Routing processor
מריץ את פרוטוקל הניתוב, מחזיק את טבלאות הניתוב ומעדכן אותן.

מתי נוצר תור?
התורות שנוצרים ב- input port וב-output port גורמים לעיתים לאיבוד חבילות. זה המקום העיקרי שבו האיבוד יכול להתרחש במעבר החבילות מהמקור ליעד.
התור נוצר כתלות בעומס (כמה חבילות יש) ובמהירות היחסית בין ה- switching fabric ל-line speed (מהירות קו).
Output port Queuing:
נניח שמהירות הקו של ה- input ושל ה-output זהות ונניח שיש N input ports ו-N output ports . אם המהירות של ה- switching fabric היא לפחות פי N ממהירות הקו של ה-input, אז לא ייווצר תור ב- input ports. מדוע? גם במקרה הגרוע שבו כל N הקווים הנכנסים מקבלים חבילות, הסוויץ' יוכל להעביר N חבילות מה-input port ל- output port בזמן שלוקח לכל אחד מ-N ה- input ports (בו זמנית) לקבל חבילה אחת.
מה קורה ב- output ports? במקרה הגרוע ביותר, כל החבילות שהגיעו לכל אחד מה- input ports יגיעו לאותו output port ואז בזמן שלוקח לקבל (או לשלוח) חבילה אחת, N חבילות יגיעו ל-output port אחד. מאחר ש- output port יכול להעביר רק חבילה אחת ביחידת זמן, N החבילות שיגיעו יאלצו לחכות בתור עד שיועברו החוצה. אז יכולות להגיע עוד N חבילות בזמן שלוקח להעביר רק חבילה אחת מה-N שהגיעו קודם. כך זה יכול להמשיך ובסופו של דבר, הבאפרים יכולים לגדול עד למצב שבו הזיכרון ב- output port מלא ואז חבילות נזרקות.
הציור הבא מתאר את התור הנוצר ב- output port. בזמן t, חבילה הגיעה לכל אחד מה-input ports. כל אחד מהם רוצה להעביר את החבילה לאותו output port. בהנחה שמהירות הקו דומה וה-switch פועל במהירות הגדולה פי 3 ממהירות הקו, לאחר יחידת זמן אחת, כל שלושת החבילות שהועברו ל-output port מחכות בתור כדי להישלח. לאחר עוד יחידת זמן, אחת מהחבילות האלו תישלח ובדוגמא יש עוד 2 חבילות חדשות שמגיעות לראוטר (אחת מהם אמורה להישלח לאותו output port).



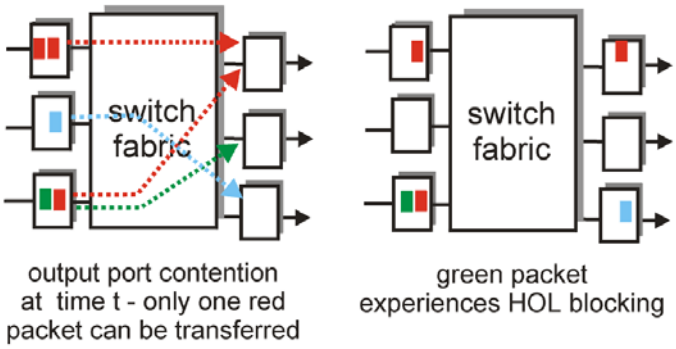
ניהול התור יכול להיות FIFO או לפי WFQ (weighted fair queuing) .

Input port Queuing:
אם ה- switch fabric לא מהיר מידי (קרוב מאוד למהירות הקו) כדי להעביר את כל החבילות המגיעות ללא עיכוב, אז יווצר תור ב- input ports.
נניח שיש לנו crossbar switching fabric שמקיים את התכונות הבאות:

- כל המהירויות של הלינקים דומות.
- חבילה אחת יכולה לעבור מכל input port ל-output port כלשהו באותו זמן שלוקח לקבל את החבילה ב- input link.
- ניהול התור הוא FIFO

במצב זה, מספר חבילות יכולות לעבור במקביל, כל עוד ה- output ports שלהן שונה. אם יש שתי חבילות שנמצאות ראשונות (בשני תורים שונים) ורוצות להגיע לאותו תור ב- Output, אז חבילה אחת תיחסם ותחכה. הרי ה- switching fabric יכול להעביר רק חבילה אחת ל-output port ביחידת זמן אחת.
בציור הבא יש דוגמא שבא 2 חבילות (אדומות) נמצאות בראשם של שני תורים ב- input והן מיועדות לעבור לאותו output port. נניח שהסוויץ' החליט להעביר את החבילה שנמצאת בתור העליון. החבילה בתור התחתון נאלצת לחכות

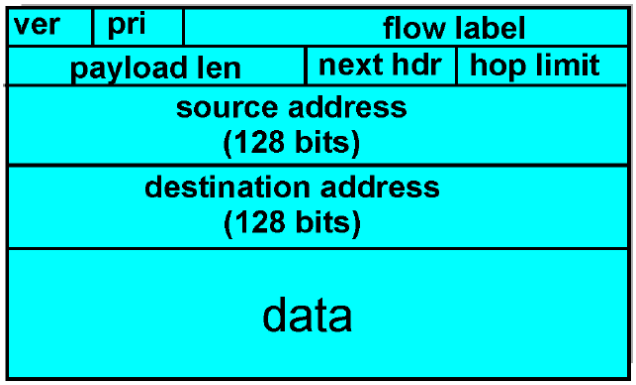
ובנוסף גם החבילה הירוקה שנמצאת מאחוריה למרות שאין תחרות על ה- output port האמצעי אליו החבילה הירוקה רוצה להגיע. לתופעה זו קוראים head of the line (HOL) blocking



IPv6 4.7

המוטיבציה הראשונית לפתח גרסה חדשה הייתה בשל העובדה שמרחב הכתובות בנות 32 ביטים הולך ונגמר.
את רב השינויים בין הגרסאות ניתן לראות בפורמט החבילה:

- הגדלת מרחב הכתובות: אורך הכתובת גדל מ-32 ל-128 ביט (לכל גרגר חול בעולם יכולה להיות כתובת IP!!).
- Header קצר יותר שמאפשר מעבר מהיר יותר של חבילות.
- Flow labeling ועדיפויות



פירוט:

- version- כמובן שכאן הגרסה היא 6.
- Priority – שדה זה דומה לשדה TOS שבגרסה 4. מספרים בין 0 ל-7 משמשים למתן עדיפות כשיש תנועה צפופה, מספרים בין 8 ל-15 משמשים למתו עדיפות כשיש תנועה לא צפופה.
- Flow label – שדה זה מיועד לאפיין "flow" של חבילות.
- Payload length – מספר הבתים בחבילה מעבר לאורך הקבוע שהוא 40 בתים (אורך ה- header).
- Next header – מזהה הפרוטוקול בשכבה העליונה (TCP או UDP). כמו השדה protocol בגרסה 4.
- Hop limit- כמו TTL.
- כתובת המקור והיעד. מבנה כתובת נראה כך:

שדות שהורדו:

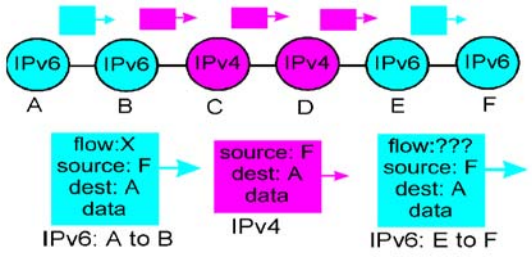
- fragmentation / Reassembly. גרסה 6 לא מאפשרת. אם מגיעה לראוטר חבילה גדולה יותר מהלינק היוצא, הראוטר פשוט זורק את החבילה ושולח הודעת שגיאה לשולח. במקרה זה השולח יכול לשלוח שוב את המידע בחבילות קטנות יותר.
- Checksum. מכיוון שגם שכבת ה-transport וגם שכבת ה- data link עושות checksum, מיותר לעשות זאת גם בשכבת הרשת.

- Options. שדה האופציות לא קיים ובמקום הוא אחד מהאפשרויות בפוינטרים next header. כלומר כשם ש-TCP או UDP יכולים להיות הפרוטוקולים הבאים, כך גם שדה האופציות

ICMPv6: גרסה חדשה של ICMP.
type חדש: "Packet too big"
code חדש: "unrecognized IPv6 options".
לגרסה החדשה יש פונקציות של multicast group management (נלמד בהמשך).

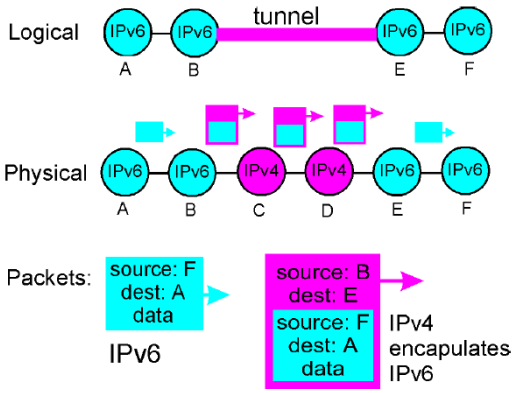
Transition from IPv4 to IPv6
איך כל האינטרנט שמבוסס על IPv4 יוחלף ל-IPv6?
הבעיה היא שמערכות עם הגרסה החדשה יכולות לתקשר עם מערכות מהגרסה הישנה, אולם ההפך זה בלתי אפשרי.
ישנם מספר פתרונות לשדרוג:
פיתרון ראשון: "flag day"- בתאריך מסויים, על כל המערכות להיות בעלות הגרסה החדשה. מניסיון עבר, זו דרך בלתי אפשרית.
שני הפיתרונות הבאים הם אפשריים ויכולים להתבצע ביחד או כל אחד לחוד:

- dual stack: למערכות IPv6 יהייה מימוש מלא גם ל-IPv4. מערכות אלה יקראו IPv6/IPv4 והן יוכלו לשלוח ולקבל חבילות גם ממערכות IPv6 וגם ממערכות IPv4. למערכות אלה יהיו כל הכתובות של IPv6 ושל IPv4.
כמו כן, מערכות אלה צריכות להבחין מאיזו גרסה המערכת שמתקשרת איתם, זה אפשרי ע"י שימוש ב- DNS (פרק 2).



נסתכל בציור הבא:
נניח שקודקוד A מגרסה 6 רוצה לשלוח חבילה לקודקוד E שגם הוא מגרסה 6. קודוקדים A ו-B יכולים להחליף חבילות לפי IPv6 אולם קודקוד B צריך ליצור חבילה לפי IPv4 ולשלוח אותה ל-C. כמה מהשדות בחבילה נשארים אותו הדבר (למשל ה- data וכתובת ה-IP), אולם יש כמה שדות לא חופפים והאינפורמציה שבהם תאבד. בסופו של דבר, החבילה שקודקוד E יקבל לא תכיל את כל השדות שהיו בחבילה שנשלחה מ-A.

- Tunneling: שיטה זו פותרת את הבעיה. איך? נניח ששני קודקודים IPv6 רוצים להעביר ביניהם חבילות IPv6 ויש ביניהם מספר קודקודים IPv4 (לקודקודים אלו קוראים tunnel-מנהרה). לפי שיטת ה- tunneling, קודקוד B, שזה קודקוד IPv6 שנמצא בצד אחד של המנהרה, לוקח את כל החבילה שלו ושם אותה בשדה data של חבילה IPv4. כתובת היעד של חבילה זו היא הכתובת של קודקוד IPv6 שנמצא בצד השני של המנהרה (קודקוד E). אז החבילה עוברת בין קודקודים IPv4 מבלי שידעו בכלל שהם מכילים את כל תוכן החבילה מגרסה 6. כשהחבילה מגיעה לקודקוד E, הוא מחלץ את תוכן החבילה שאותה צריך להמשיך להעביר ושולח אותה כחבילה IPv6.



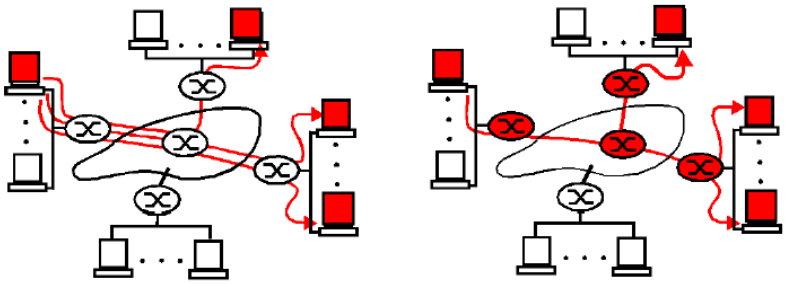
Multicast Routing 4.8

פרוטוקולים שבהם יש רק שולח אחד ומקבל אחד נקראים unicast protocols. (אלה הפרוטוקולים שלמדנו עד כה).
Multicast – שליחת חבילה משולח אחד לקבוצה של מקבלים עם פעולת שידור יחידה.
שתי צורות מימוש :

- השולח משתמש בכמה unicast transport connections, אחד לכל מקבל. המידע שמועבר משכבת האפליקציה לשכבת ה-transport משוכפל אצל השולח ועובר דרך כל אחד מה-connections. לפי שיטת מימוש זו, שכבת הרשת לא צריכה לתמוך בשום דבר מיוחד.

בציור השמאלי ניתן לראות איך שיטה זו עובדת (הראוטרים צבועים בלבן כדי להדגיש שהם לא מעורבים במימוש).

- השולח שולח חבילה אחת והיא משוכפלת בראוטר רק כשהיא צריכה לעבור לכמה לינקים כדי להגיע לכמה מקבלים. בציור הימני ניתן לראות איך שיטה זו עובדת.
בשיטה זו משתמשים באינטרנט ובא נתמקד.



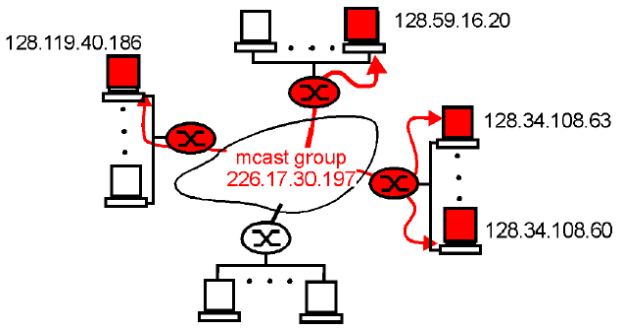
multicast via unicast network multicast

ישנן שתי בעיות עיקריות עם השיטה:

- איך לזהות את המקבלים
- איך לכתב את החבילה כך שתגיע למקבלים הללו.

בשיטה הראשונה אין את הבעיות הללו, מכיוון שהשולח משכפל את החבילה, נותן כתובת שונה לכל אחת.
לפי שיטת ה- muticast, חבילה שמשוכפלת מקבלת את כתובתה לפי address indirection. בצורה זו, לקבוצת המקבלים יש identifier וכל עותק של ה- datagram שמיועד לקבוצה בעלת ה- identifier, מועבר לכל המקבלים השייכים לקבוצה. באינטרנט, ה-identifier שמייצג קבוצה של מקבלים שייך למחלקה D (כפי שראינו בחלק 4.4).
קבוצת המקבלים נקראת multicast group.

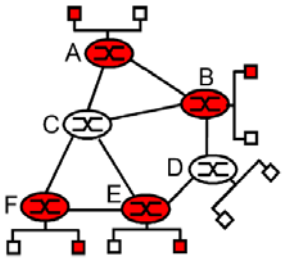
בציור הבא, 4 תחנות הקצה האדומות שייכות ל- multicast group של כתובת 226.17.30.197 והן תקבלנה את כל החבילות שמכותבות לכתובת זו.

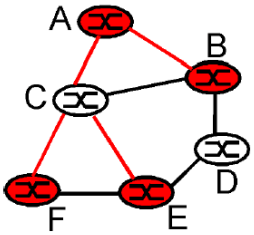


<u>פרוטוקול IGMP</u> – פועל בין תחנות הקצה לראוטרים המחוברים אליהן ישירות. מאפשר לתחנת הקצה להודיע לראוטר שאפליקציה שרצה שצלה רוצה להצטרף לאיזושהי multicast group.
(בספר כתוב על פרוטוקול זה הרבה אך בכיתה לא למדנו עליו)

נעבור כעת ל- <u>multicast routing problem</u>:

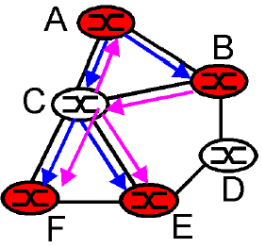
מטרת ה- multicast routing היא למצוא עץ שמחבר בין כל הראוטרים שמחוברים לתחנות קצה ששייכות ל-multicast group.
דוגמא:
תחנות קצה שהצטרפו ל- multicast group והראוטרים שמחוברים אליהן צבועים באדום. רק הראוטרים שצבועים באדום צריכים לקבל את ה- multicast traffic.





2 גישות לבניית העץ:

- Group shared tree: עץ אחד עבור ניתוב יחיד לכל ה- multicast group. בדוגמא, העץ מחבר בין הראוטרים A, B, C, E ו- F. חבילות יזרמו רק לאורך הלינקים האדומים שהם דו כיווניים.
כלומר, אותו עץ משמש את כל חברי הקבוצה

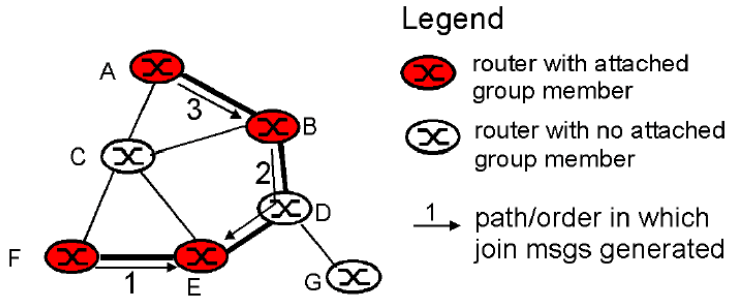


- Source- based trees: לכל שולח ב- multicast group נבנה עץ ניתוב. אם בקבוצה יש N תחנות קצה, נקבל N עצים שונים. בדוגמא מתוארים 2 עצים, עבור A ועבור B.
כלומר, עץ שונה לכל שולח למקבלים שלו.

שימוש ב- group shared tree:

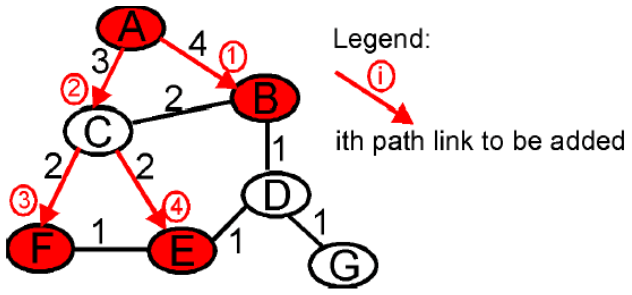
- Steiner tree - מציאת עץ מינימאלי שמחבר בין כל הראוטרים שמחברים תחנות קצה ששייכות ל- multicast group. עץ מינימאלי הוא עץ שסכום מחירי הצלעות בו הוא הקטן ביותר.
אף אחד מהאלגוריתמים באינטרנט לא אימץ את השיטה הזו. למה? הסיבוכיות גבוהה, בגישה זו צריך את כל האינפורמציה על כל הלינקים ברשת. כמו כן, בכל פעם שמחירו של לינק משתנה, האלגוריתם צריך לרוץ מחדש.
יש עוד סיבות שלא ניכנס אליהן.

- center based trees- ראוטר אחד מאופיין כקודקוד המרכזי של העץ. ראוטרים משדרים הודעות "join" לקודקוד המרכזי. הודעה כזו מועברת תוך שימוש ב- unicast routing לעבר הקודקוד המרכזי עד שהן מגיעות או לראוטר שכבר שייך ל-nulticast group או לקודקוד המרכזי. המסלול שבו עברה ההודעה מגדיר ענף בעץ בין הראוטר ששלח את ההודעה למרכז.
הציור הבא מתאר את בניית העץ:
נניח שראוטר E נבחר להיות הקודקוד המרכזי.
קודקוד F מצטרף ראשון ל-multicast group ומעביר הודעת join ל-E. הלינק EF יוצר בעצם את העץ ההתחלתי.
בשלב הבא, קודקוד B מצטרף לעץ ע"י שליחת הודעה ל-E. נניח שמסלול ההודעה עובר מ-B ל-E דרך D. נוצר עוד ענף בעץ- BDE.
אחרון חביב, קודקוד A מצטרף ל-multicast group ע"י שליחת הודעה ל-E ונניח שהמסלול שלה עובר דרך B.
מאחר ש-B כבר הצטרף לעץ, ישר ייווצר הענף AB.



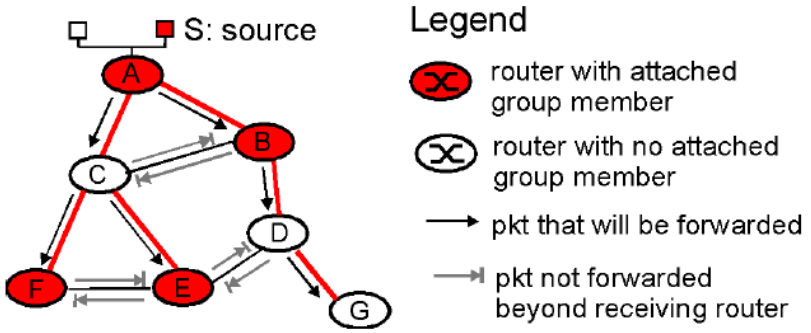
שימוש ב- source based tree:

- shortest path trees- כמו dijkstra.
כאמור, אלגוריתם זה דורש שכל ראוטר ידע את המצב של כל לינק ברשת על מנת שיוכל לחשב את המסלול הקצר ביותר ממנו לכל היעדים.
בדוגמא הבאה מצויר עץ המתאר את המסלולים הקצרים ביותר מ-A לכל שאר הקודקודים שב- multicast group.



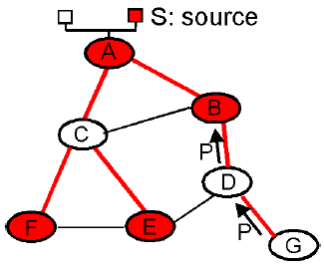
- reverse path forwarding- הרעיון: כאשר ראוטר מקבל multicast packet עם כתובת המקור, הוא מעביר את החבילה לכל הלינקים היוצאים ממנו חוץ מזה שהחבילה הגיעה ממנו. הוא עושה זאת רק אם החבילה הגיעה מלינק שנמצא על המסלול הקצר ביותר ממנו למקור. אחרת, הראוטר לא עושה כלום עם החבילה. חבילה כזאת יכולה להיזרק מכיוון שהראוטר יודע שהוא או יקבל או כבר קיבל בעתק של החבילה בלינק שנמצא על המסלול הקצר ביותר ממנו למקור.

נשים לב שהראוטר לא צריך לדעת את כל המסלול ממנו למקור אלא רק את ה- hop הבא במסלול.
בציור הבא ניתן לראות כיצד נבנה העץ.
נניח שהלינקים האדומים מייצגים את המסלולים הקצרים מהמקבלים למקור A.
ראוטר A שולח חבילה S לראוטרים C ו-B. ראוטר B מעביר את S ל-C ול-D (הוא מעביר מכיוון ש-A נמצא על המסלול הקצר ממנו ל-A).
ראוטר B ייתעלם מ-S אם יקבל אותה מכל ראוטר אחר.
נניח כעת שראוטר C ייקבל את S ישירות מ-A וגם מ-B. מאחר ש-B לא נמצא במסלול הקצר ביותר מ-C ל-A, C ייתעלם מהחבילה שהוא ייקבל מ-B.



מה יקרה בראוטר D? הוא יעביר חבילות לראוטר G למרות שאין לו בכלל תחנות קצה ששייכות ל-multicast group. אם ל-D היו אלפי ראוטרים כמו G, אז אלפי ראוטרים היו מקבלים חבילות שהם בכלל לא רוצים.
הפיתרון לקבלת חבילות לא רצויות נקרא pruning (גיזום).
ראוטר שקיבל חבילה ואין לא תחנות קצה ששייכות לקבוצה ישלח prune message לראוטר ממנו הגיעה החבילה. אם ראוטר קיבל prune message הוא יעביר אותה הלאה, במעלה הזרם.

דוגמא ל-pruning:
נשים לב ש-pruning דורש שהראוטר יחזיק יותר מידע: אילו ראוטרים תלויים בו במורד הזרם.
כמו כן, אם ראוטר שלח prune message במעלה הזרם, מה יקרה אם בשלב מאוחר יותר הוא יצטרך להצטרף ל- multicast group? צריך לשמור את הצלע שנמחקה מהעץ ברגע שנשלחה הודעת הגיזום. דרך אחת היא להוסיף הודעת graft שתאפשר לראוטר להחזיר את הצלע לעץ. אפשרות נוספת היא לתת לענפים שנגזמו time-out ואז להחזיר אותם שוב לעץ.

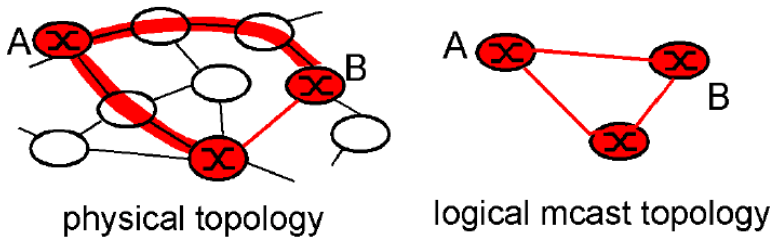


Multicast routing in the internet

DVMRP- distance vector multicast routing protocol
הפרוטוקול מממש source-based trees עם reverse path forwarding כולל pruning ו-grafting . הפרוטוקול משתמש באלגוריתם distance vector שמאפשר לכל ראוטר לחשב את הלינקים היוצאים ממנו שהם על המסלול הקצר ביותר ממנו לכל מקור אפשרי.
בנוסף, לשם ה- pruning, הפרוטוקול שומר לכל ראוטר רשימה של הראוטרים במורד הזרם ממנו.

בפרוטוקול זה הודעת הגיזום כוללת זמן חיים של הגיזום (ברירת מחדל שעתיים) כלומר כמה זמן ענף גזום יישאר גזום לפני שהוא יחזור לעץ בצורה אוטומטית.
הודעת graft נשלחת במעלה הזרם כדי לאלץ חיבור של ענף גזום.

איך האינטרנט יכול להשתמש באלגוריתם כזה הרי הבעיה היא שרק חלק קטן מהראוטרים יכולים לעשות multicast.
אם ראוטר שיכול לעשות את זה מוקף בשכנים ללא היכולת הזו, הוא לא יוכל בכלל להשתמש בפונקציונאליות שלו..
הפיתרון הוא tunneling (כפי שראינו ב- IPv6)- ניתן ליצור רשת וירטואלית של ראוטרים המסוגלים לעשות multicast והיא תהייה מעל הרשת הרגילה שמכילה גם ראוטרים שיכולים לעשות זאת וגם ראוטרים שלא יכולים.



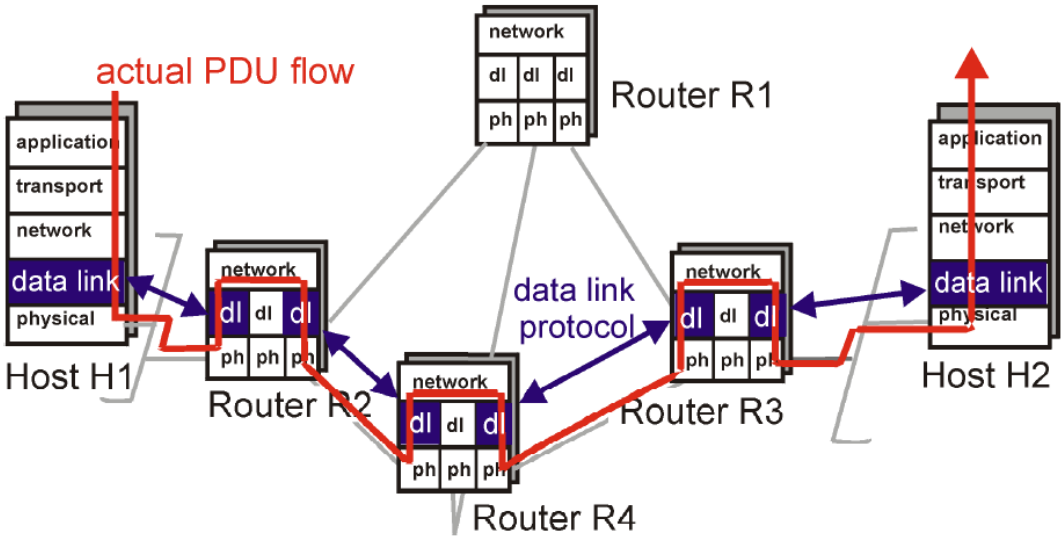
(פרוטוקולים שלא למדנו: MOSPF, CBT)

<u>PIM: Protocol Independent Multicast</u>
פרוטוקול שלא תלוי באף אלגוריתם ניתוב שהוא unicast. לפרוטוקול שני מודים:

- dense mode: חברי ה- multicast group צפופים אחד לשני, לכן רוב הראוטרים שבסביבה צריכים להיות מסוגלים להעביר multicast datagrams .
  כאן משתמשים בטכניקת ה- reverse path forwarding.
- sparse mode: מספר הראוטרים שיש להם קבוצה מחוברת הוא קטן ביחס למספר הכולל של הראוטרים.
  כאן משתמשים בגישת ה- center based.

The Data Link Layer

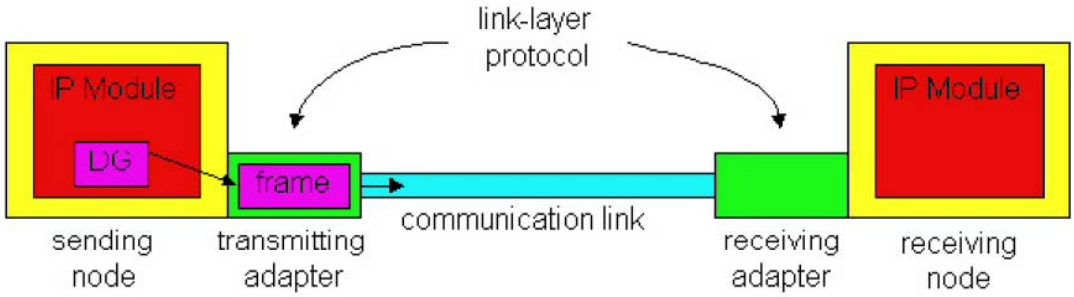
בשכבה זאת אנו נשתמש בביטויים של nodes במקום להגדיר את ה host ואת ה routers כשני טיפוסים שונים.שכבה זה תפקידה הוא לקשר בין כל שני nodes. בנוסף נגדיר את כל הדרכים אשר מחברות בין שני nodes כlink.



טוב אז אחרי
הקדמה קטנה והבנה על המושגים נגדיר את התפקיד ה"רשמי" של רמה זאת.רמת ה data link מטרתה היא להעביר חבילות בין שני routers דרך ה links, רמת ה network אחראית על כל הניתוב ורמה זו יותר ספציפית-חלק קטן מהניתוב.
בין תפקידי הפרוטוקול בשכבה זאת הם כמובן לשלוח ולקבל frames כולל זיהוי שגיאות,העברה חוזרת,שליטה על קצב העברת הנתונים ומניעת הצפת המקבל ומונעת התנגשויות ז"א ששני מקורות לא ידברו ביחד ויפריעו אחד לשני לתקשר.

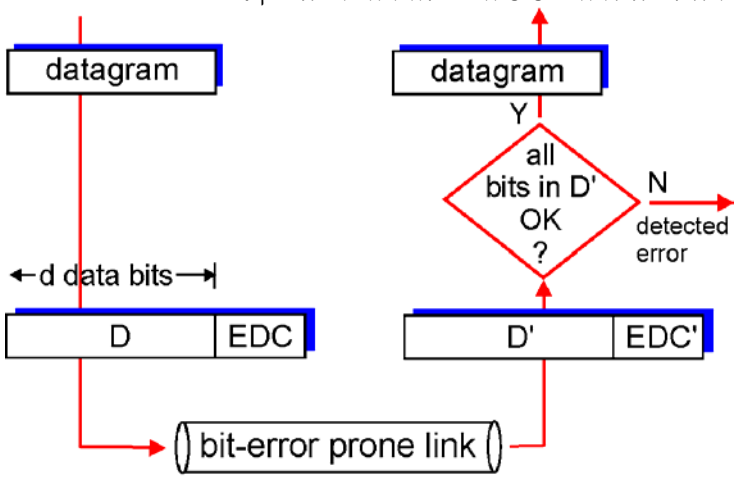
אחת התכונות של הרמה הזאת היא שחבילה יכולה להיות מטופלת הכמה פרוטוקולים שונים באותו מסלול.לדוגמא,חבילה יכולה להיות מטופלת בעזרת פרוטוקול ה Ethernet בlink הראשון ובפרוטוקול PPP בlink האחרון.
Framing and link Access-ברמה הפיזית נשלח רצף של ביטים בניהם נמצא ה header וה data של אותה חבילה.
בעזרת ה framing נוכל לזהות מתי נגמר הheader ומתי מתחיל המידע.

כתובת MAC-מרחב הכתובות שאנו נשתמש יהיו כתובות ה MAC של המחשב. כתובת MAC זוהי כתובת פיזית שנמצאת על החומרה עצמה(כרטיס רשת למשל) כך שכל כרטיס שמיוצר מקבל כתובת שייחודית לו כדי לשלוח broadcast לכל הnodes נשתמש בכתובת FF-FF-FF-FF-FF-FF MAC
Full-duplex-2 הצדדים יכולים לשדר בו זמנית
Half-duplex- רק צד אחד יכול לדבר כל פעם.
כרטיס רשת(adapter/NIC) – תפקידו של כרטיס הרשת זה לקבל מהnode השולח את החבילה ולעטוף אותה ב frame ולהעביר אותה הלאה בקו המתאים מצד השני של התקשרות כרטיס הרשת המקבל מקלף מתוך הframe את החבילה שמיועדת לרמת התקשרות(network) ומעביר אותה הלאה.



Error Detection and Correction Techniques

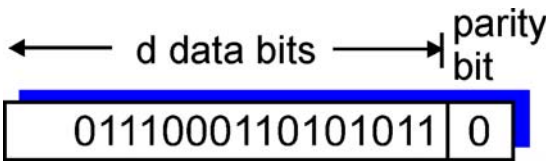
נתחיל בהגדרות בסיסיות שילוו אותנו בהמשך!



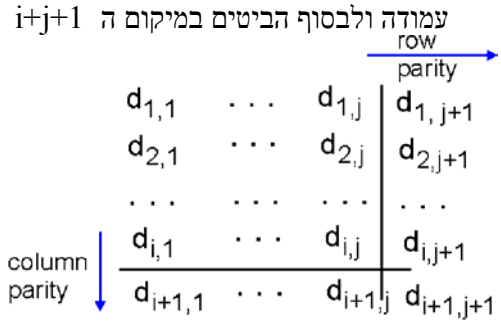
EDC-זיהוי שגיאות ותיקון הביטים
D-מידע אשר אנחנו רוצים "להגן" מפני שגיאות בביטים.
המידע יכול לכלול לא רק את החבילה עצמה אלא גם את הheader שמכיל את כתובות היעד של שכבת הlink.
גם D וגם EDC נשלחים לתחנה המקבלת תחת מסגרת של רמת הlink.
נשים לב ש'D וגם 'EDC שמגיעים לתחנת היעד עלולים להיות שונים מן המקוריים בגלל שגיאות שנוצרו בביטים בזמן השליחה.תפקידה של תחנת היעד היא להגיד אם 'D שהגיע אליה הוא D המקורי שנשלח.

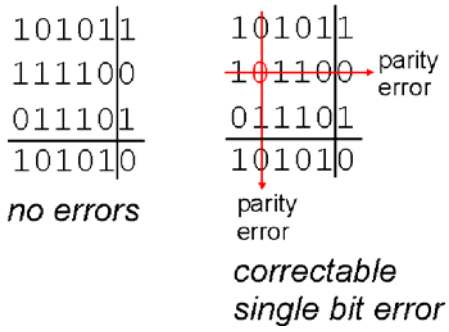
Parity Checks

- **Parity bit**-השיטה הפשוטה ביותר לזיהוי שגיאות.
  - o נניח שאנחנו שולחים מידע כלשהו ,D, והוא בעל d bits.לאלגוריתם זה 2 וארציות:
    - הוארציה הזוגית-אם משתמשים בוארציה זו לפני שנשלח נוסיף בסוף ההודעה ביט(0\1) ככה שבסה"כ ב1+d הביטים יהיה מספר אי זוגי של '1'
    - הוריאציה האי זוגית-אותו דבר רק להשלים למספר אי זוגי של '1'



- **two-dimensional parity** - מחלקים את D ל i שורות ול j עמודות.מחשבים סכום של כל שורה ושל כל עמודה ולבסוף הביטים במיקום ה i+j+1 משמשים לבדיקת שגיאות.





Checksumming Method

בשיטה זאת השולח מתייחס אל הbytes של ההודעה כinteger 16 bit הבדיקה מתבצעת ע"י חיבור של חלקי התוכן.השולח מכניס את הערך הסופי לתוך השדה המתאים לchecksum בUDP.
המקבל מחשב גם הוא את סכום של כל החלקים בודק אם ההערך שיצא לו תואם לערך שקיבל אם כן השגיאה(אם הייתה)לא התגלתה אם לא שגיאה התגלתה!

Cyclic redundancy check(CRC)

שיטה לזיהוי שגיאות מאד נפוצה ברשתות כיום.
נניח שקיים מידע,D, בעל d bit שהשולח מעוניין לשלוח ליעד מסוים.
השולח והמקבל חייבים קודם כל להסכים על איזשהי תבנית בעלת r+1 bit שנקראת generator ונסמנה בG.הדרישה היחידה על G היא שהביט הMSB(הכי שמאלי) יהיה 1.
כעת השולח בוחר צרוף אחר בעל r bit שנסמנו בR.ומצרף אותם לD כך שהתוצאה תהיה תבנית של d+r bit.אך ישנה דרישה על R שהיא ש<D,R> יתחלק במדויק בG כאשר מחלקים בעזרת modulo 2. כעת המקבל מחלק את d+r הביטים בG.אם התוצאה איננה אפס המקבל יודע ששגיאה ארעתה בדרך, אחרת הוא מוכן לקבל את ההודעה.
ישנה דוגמא בשקף מספר 14!!

Multiple Access Protocols and LANs

נפריד בין שני דרכים לתקשר בין point to point: links ו broadcast links.
Point to point כולל שולח אחד ומקבל יחיד הוא משתמש בפרוטוקלים של:PPP,HDLC לדוגמא.
הbroadcast יכול להכיל מספר רב של שולחים ומקבלים של ההודעה.אשר כולם מחוברים לאותו ערוץbroadcast .
כעת נעסוק בבעיה המרכזית ברמה זאת איך לאפשר לכל מי שרוצה לדבר ברשת לדבר בלי להפריע למשהו אחר אשר משותף לאותה הרשת.
למרות שטכנית node עצמו לא יכול לתקשר עם הרשת אלא הכרטיס רשת שלו מתקשר אנו נתייחס לפה לכך שכל node מתקשר ישירות עם הרשת(מטעמי נוחות☺).הרשת בנויה ככה שמאות ואלפי nodes מחוברים אליה בוזמנית,ובגלל שכל אחד מהם יכול לשלוח חבילה יכול לקרות מצב שבו שני nodes ירצו לשלוח חבילה באותו זמן.כאשר מצב זה קורה מתקיימת **התנגשות** בין שתי החבילות דבר שפוגע במידע של החבילה וגורם לה להיות לא שימושית לכל אחד מהצדדים.

ולכן בגלל שיש אלפי מקרים כאלו החליטו לחשוב על פרוטוקלים אשר ימנעו זאת עד היום יש אלפים כאלו אך ניתן לחלק אותם לשלוש קבוצות עיקריות

1)חלוקה לזמנים-לחלק את הזמן בין התחנות כך שלכל תחנה יהיה זמן משלה לשדר,חלוקה לתדרים

2)Random Access-כל תחנה משדרת מתי שהיא רוצה אם היא לא מצליחה מנסה שוב.

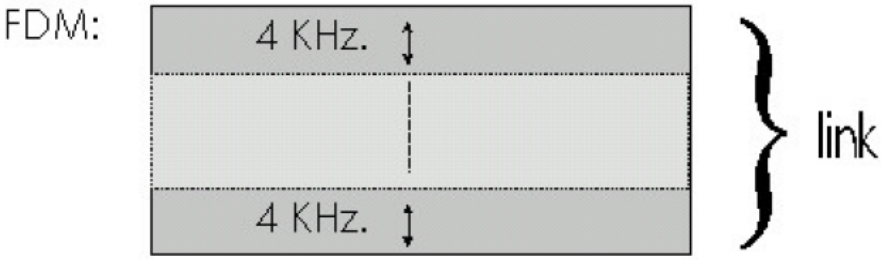
3)חלוקה לתורות-כל חבילה שרוצה לשדר מבקשת רשות ונכנסת לתור מסוים.

חלוקה לתדרים-FDM

נניח שהרשת שלנו תומכת בN nodes ורוחב הפס הוא R bps אז כל אחד מן המשדרים יקבל N\R רוחב פס.

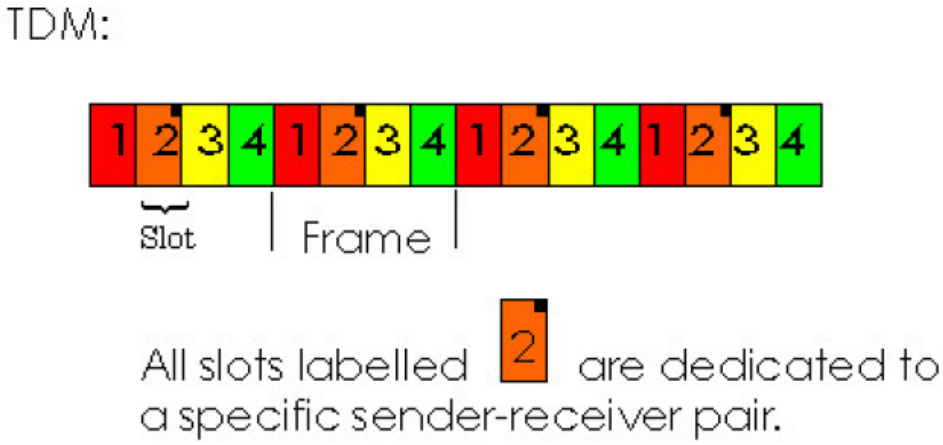
יתרונות-זול,וקל למימוש

חסרונות-במקרה ותחנה אחת לא צריכה לשדר בכל הרוחב פס שהוקצה לה היא מבזבזת רוחב פס.



חלוקה לזמנים-TDM

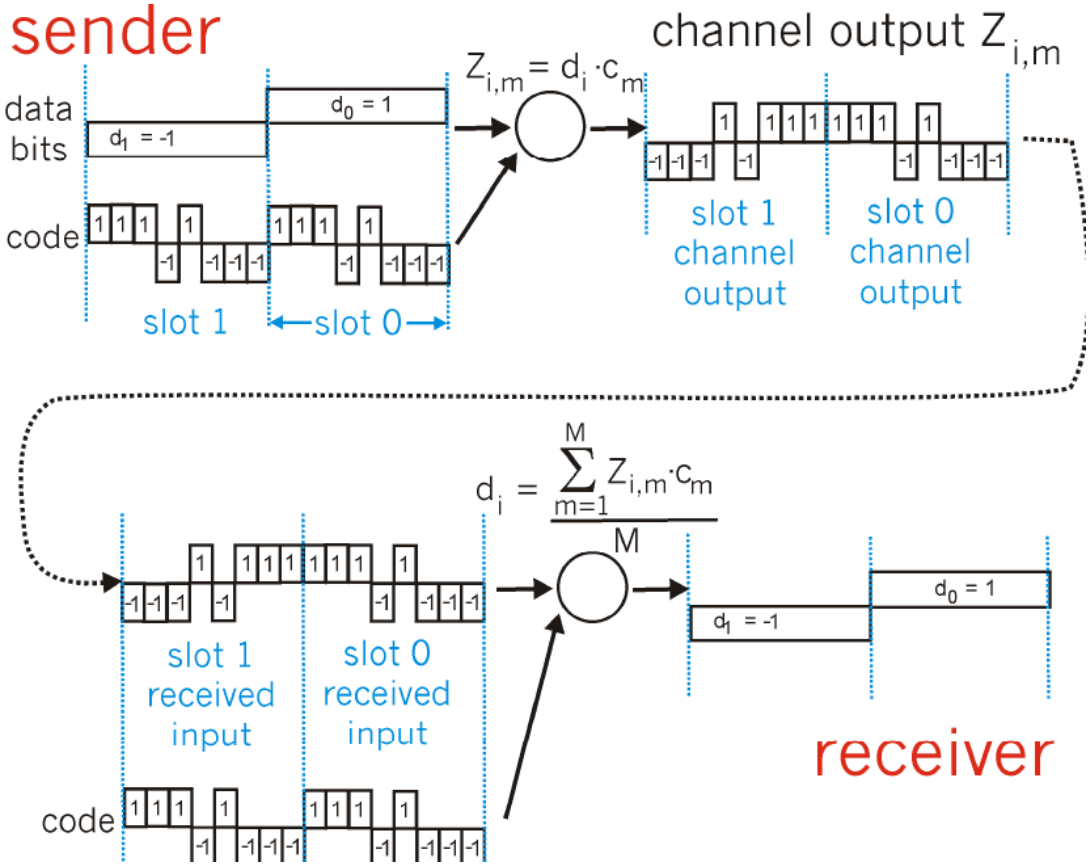
על אותו עקרון של החלוקה של תדרים.כל תחנה תקבל זמן מסוים בו היא יכולה לשדר כמובן שגם פה יש חסרון גדול מאד שאם תחנה מסוימת לא רוצה לשדר אז היא מבזבזת את התור שלה שתחנה אחרת שכן רצתה לשדר יכלה.



<u>CDMA-Code Division Multiple Access</u>

בניגוד לFDM ולTDM פרוטוקול זה לא מחלק בין הזמנים או תדרים של כל תחנה להפך הוא נותן לכולם לשדר באותו זמן ועל אותו תדר.הוא מונע התנגשויות וערבוב של המידע בכך שהוא מקצה לכל תחנה קוד משלה שבעזרתו היא מצפינה את הקוד שלה וכך מפריד בין הקוד שלה לקוד של אחרים.הוא מצפין את הקוד בעזרת מכפלה ע"י סיגנל שמשתנה

במהירות יותר גבוה מאשר זרימת המידע.



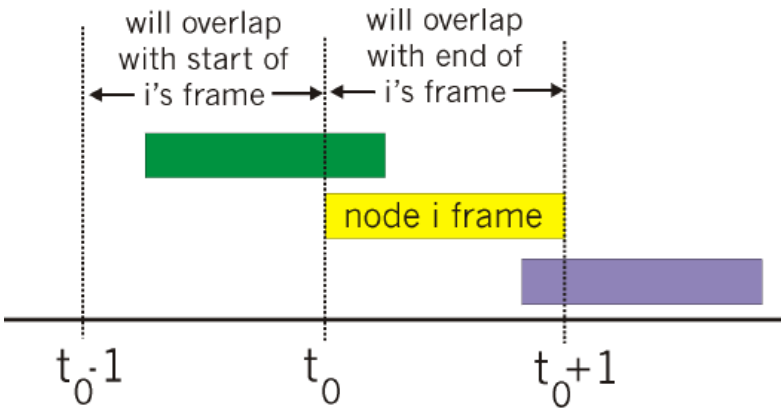
באיור מצורף תרשים מדויק של ההצפנה ושל הפענוח.
נניח שהזמן שלוקח לביט אחד להגיע מוגדר כיחידת זמן אחת.

Random Access

בניגוד לחלוקה לכל תחנה מאפיין משלה כמו בשיטות הקודמות בשיטה זאת כל תחנה תשחל מתי שתרצה ואם תשמע שקרת התנגשות היא תשלח אחרי זמן מסוים.

-ALOHA

פרוטוקול מאד פשוט כל תחנה משדרת ומאזינה לקו אם היא שמעה התנגשות היא מחכה זמן אקראי כלשהו ושולחת שוב ככה עד שהיא בטוח שהחבילה הגיעה ללא התנגשויות



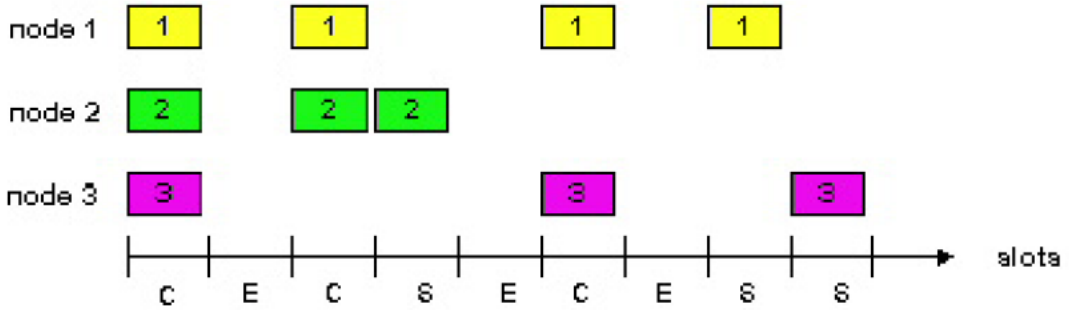
הנצילות פה יחסית מאד נמוכה כשיש הרבה תחנות שרוצות לשדר כי כל אחת מפריעה לתשדורת של השניה.
בעקבות פרוטוקול זה הגיעה שכלול שנקרא **slotted ALOHA**
אך קודם נרחיב על סוגי התנגשויות קיימת התנגשות מלאה-שתי התחנות התחילו לשדר בו זמנית וכתוצאה מכך נוצרה התנגשות,או התנגשות חלקית-למשל תחנה התחילה לשדר כשתחנה אחרת כמעט סיימה ואז יכול להיווצר מצב של התנגשות רק של ביט אחד אך למרות זאת נצטרך לשלוח את כל ההודעה מחדש.
אנחנו נגדיר כל מיני מושגים שתקפים לפרוטוקול שבעזרתם נוכל לחשב את הנצילות שלו אנו נניח ש:

- כל frame מכיל בדיוק L bits
- הזמן מחולק לקטעים(slots) אשר כל קטע בגודל שלוקח להעביר frame אחד
- כל תחנה מתחילה לשדר רק בתחילת כל קטע זמן(slot).
- כל התחנות מסונכרנות בזמנים.
- אם שני frames או יותר התנגשו בקטע זמן מסוים אזי כל התחנות זיהו את ההתנגשות לפני סוף קטע הזמן.

פרוטוקול התהליך:

- ברגע שלחתנה מסוימת יש איזשהי חבילה שהיא רוצה לשלוח היא ממתינה עד שיתחיל קטע זמן חדש ומעבירה את כולה במרווח זמן של הקטע.
- אם אין התנגשות סיימנו.
- אם התחנה זיהתה התנגשות התחנה מעבירה שוב את החבילה בהסתברות p בכל אחד מתחילת הקטעים הבאים עד אשר היא מצליחה.

באמירה "בהסתברות p" התכוונתי לזה שהיא בתחילת כל קטע זמן מחליטה אם לשלוח את החבילה או לא.
למשל בתחילת כל קטע זמן החבילה מטילה מטבע בהסתברות p להצלחה אם לשלוח עכשיו את החבילה או לא!

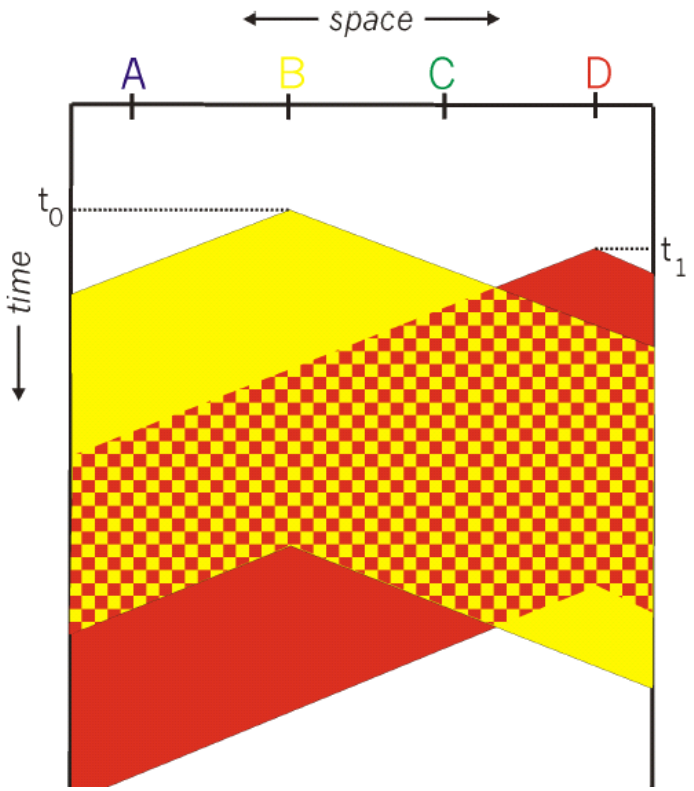


פרוטוקול זה שיפר משמעותית את הנצילות פרוטוקול זה פותר גם את בעיית השרת בכך שאין צורך שיהיה מישהו שינהל את הגישה ואת החלוקה לזמנים\תדרים\צפנים!
יעילות הפרוטוקול-כמה מה קטעי זמן באמת שימשו להעברתdata
נניח שיש N תחנות וכל אחת משדרת חבילה ב slot מסוים בהסתברות P .
הסיכוי להצלחה יחידה: p(1-p)^(N-1)
התוחלת:N*P*(1-P)^(N-1)
מה שממקסם את הביטוי זה כאשר P=1\e ואז יש כ 37% נצילות. אם נוריד את החלוקה לקטעים ונשתמש בALOHA רגיל הנצילות תקטן ל18 אחוז בלבד!

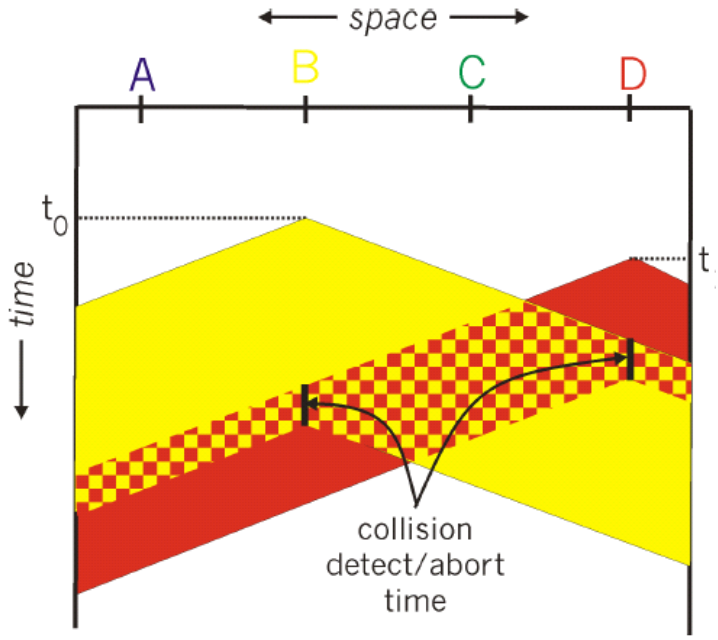
CSMA-Carrier Sense Multiple Access

פרוטוקול זה הוא מאד נפוץ ומשתמשים בו ברשת ה wireless ובethernet.
הרעיון של הפרוטוקול הוא שהתחנה עצמה "מנומסת"ולכן קודם כל מקשיבה לקו רואה שהי לא מפריעה עכשיו לאף אחד לדבר ואז שולחת בשלבים הבאים:

- **להקשיב לפני שמדברים**-אם באותו רגע הקו תפוס ע"י תחה אחרת שמדברת נחכה עד שתסיים ונשלח רק כשהקו פנוי
- **במקרה של התנגשות**(במקרה ומדובר בפרוטוקול בעל זיהוי התנגשיות-CSMA/CD)-במקרה ומישהו התחיל לדבר באות זמן איתנו או במקרה לא זיהנו שיחה כי היא הייתה עדיין רחוקה ,יכולה להתרחש התנגשות.לכן מהרגע שאנחנו נתחיל לשלוח את החבילה התחנה תקשיב לקו ובמקרה של זיהוי התנגשות, נעוצר מיד את השליחה.



בשרטוט שמצרף ניתן להבחין ששני התחנות לא ממשות את CSMA/CD אלא את CSMA הפשוט משום שגם אחרי שזוהתה התנגשות התחנות ממשיכות לשלוח ולא הפסיקו מיד עם הזיהוי.



בשרטוט זה נבחין שהתחנות הפעם כן ממשות את CSMA/CD משום שברגע שזוהתה ההתנגשות התחנות הפסיקו מיד לשדר וכך מנעו נזק נוסף ויעילו את הפרוטוקול.

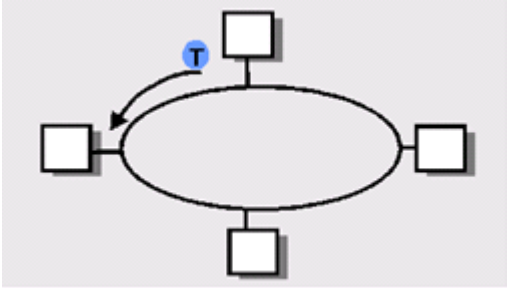
Taking-Turns Protocols

המהות של פרוטוקול זה שכל תחנה משדרת בתורה אבל בלי חלוקה לזמנים או לתדרים כדי להגדיל את הנצילות,כדי שלא יצא מצב שמקצים לתחנה מסוימת רוחב פס או זמן מצוים לשדר והיא איננה מנצלת אותו.למרות שיש הרבה פרוטוקולים אשר משתמשים בשיטה זאת כדי למנוע התנגשויות אנו נתמקד בשניים מרכזיים.

Polling protocol –בפרוטוקול זה נדרוש שאחת מן התחנות(nodes) יתוכנן כתחנה שולטת(master node),כאשר תחנה זאת מזמינה כל node ש"משרת" אותה בתורו לשדר.

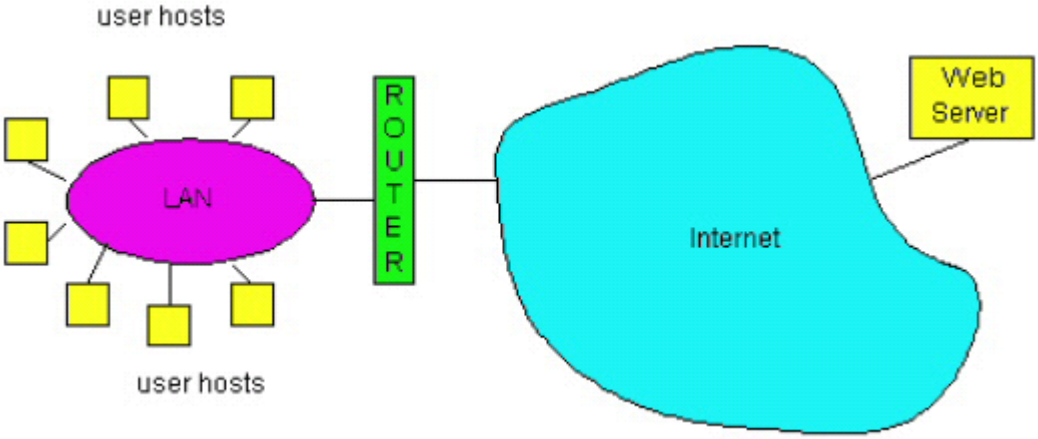
טכנית התחנה הראשית הולכת לתחנה הראשונה בסדר ואומרת לה שהיא יכולה לשדר מספר חבילות מוגבל ואז ממשיך לתחנה השניה והשלישית וכן הלאה....

Token-passing protocol בפרוטוקול זה אין אף תחנה ראשית ששולטת.
בפרוטוקול זה כל תחנה נותנת את הרשות לשדר לתחנה אחרת ,והסדר נקבע מראש.ישנה חבילה קטנה שכל מי שמקבל אותה יכולה או לשלוח את כל מה שהיא רוצה לשדר ללא הגבלה על מספר החבילות,או להעביר מיד לתחנה הבאה בסדר אם אין לה מה לשדר.

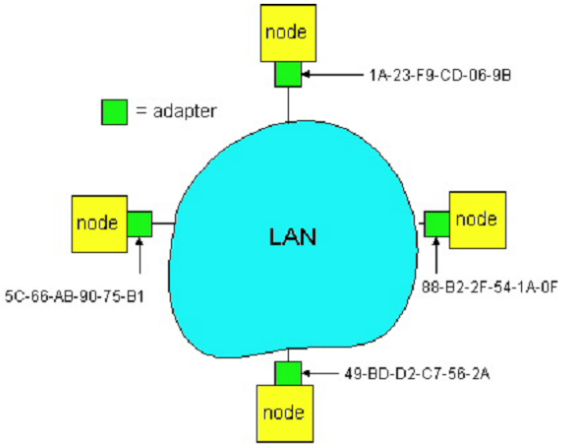


החסרונות של שיטה זאת הם כמובן שאם יש חבילה שרוצה לדר משו ענק היא תתפוס להרבה זמן לכל שאר התחנות את הקו.

Local Area Network(LAN)

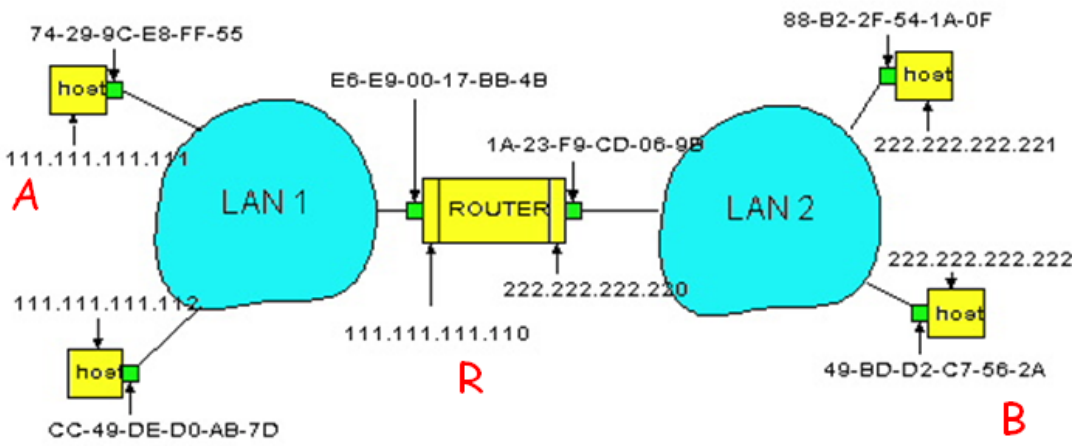


רשת LAN מחברת מספר מחשבים ברשת אחת לדוגמא האוניברסיטה.כך שאם יש משתמש שמתחבר לאחד המחשבים לא כל מחשב מתחבר בנפרד אל האינטרנט אלא רשת ה LAN מחברת אותם דרך router כמו שנלמד בהמשך...כאשר תחנה שמחוברת לרשת הLAN רוצה לשלוח הודעה מסוימת היא שולחת את ההודעה בbroadcast מה שאומר שכל שאר התחנות שמחוברות לרשת מקבלות את ההודעה.לכן אם תחנה מסוימת לא רוצה שכל שאר התחנות ידעו מה היא שולחת לתנה מסוימת לכל תחנה יש כתובת.
הכתובת שאליה נשלח היא כתובת מסוג MAC ,שזאת כתובת פיזיקלית,מה שאומר שכל כרטיס רשת שנוצר מקבל כתובת MAC והיא קבועה לכרטיס ומוטבעת עליו.לא יתכנו שני כרטיסים עם אותה כתובת!



רק תזכורת,להדיל מכתובת IP אשר כתובת זאת היא זמנית ואיננה קבועה,למעשה כל חיבור לרשת מחדש נותן לנו כתובת IP אחרת לכן איננה קשורה למחשב ספציפית.
פרוטוקול ARP-דומה לפרוטוקול DNS אבל הוא ממפה בין כתובות IP לכתובות MAC.
כמו שהזכרנו כבר קודם כדי לשלוח מכתובת IP לכתובת IP צריך לדעת את הכתובת הפיזית-MAC.לכן כל תחנה מחזיקה טבלת ARP.טבלה זאת מייפה בין כתובות הIP לכתובות הMAC ובנוסף היא מחזיקה TTL (TIME TO LIVE) שכאשר הוא נגמר המיפוי יורד מן הטבלה .במידה ואני רוצה לשלוח הודעה לתחנה שאין לי את כתובת הMAC שלה בטבלה אך יש לי את כתובת הIP אז אני אשלח בbroadcast (FF-FF-FF-FF-FF-FF) חבילת ARP לכל התחנות ואך ורק התחנה שמזהה את הIP שלה בחבילת הARP תשלח חזרה ישירות לחבילה שביקשה את חבילתARP עם הMAC שלה.לאחר קבלת ההודעה עם מספרהMAC נוסיף את הIP ואת כתובת הMAC התואמת לטבלה.

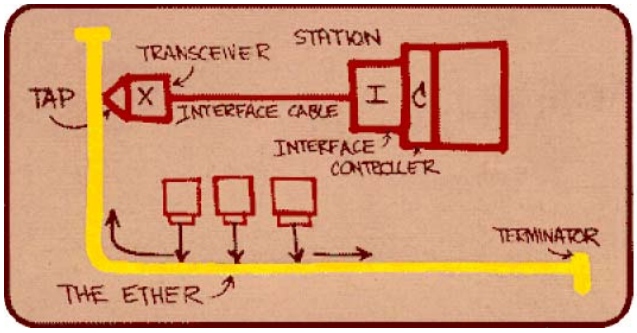
<u>שליחת חבילה מחוץ לרשת הLAN</u>



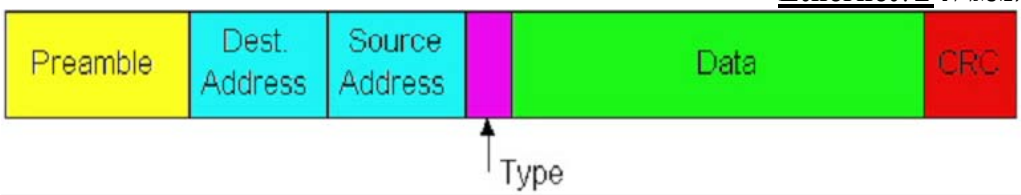
אנחנו נתעניין במצב בו תחנה A רוצה לשלוח חבילה לתחנה B.
נשים לב שA וB לא באותה רשת LAN ולכן יצטרכו לעבור דרךR gateway.
נשים לב שלראוטר R יש שני כתובות IP וזאת משום שיש לו שני כרטיסי רשת אחד לכל רשת LAN שאותה הוא צריך לבר לINTERNET.
במקרה כזה נבצע את הפעולות הבאות:
א)A יוצר מסגרת בה הוא מציין את עצמו כמקור ואת B כיעד.
ב)A בודק אם B באותה רשת LAN ע"י netmask
ג) B לא באותה רשת ולכן A בודק את כתובת IP של R(gateway) ואליו יקדם את החבילה.
ד)(נניח שטבלת ARP עדיין ריקה).מכיוון שR איננו קיים בטבלה,A שולח בקשת ARP עם הIP-R.
ה)R עונה ישירות לA עם MAC-R .
ו)A מכין חבילת Ethernet (יורחב בהמשך)אשר עוטפת את החבילה המקורית אך הפעם כתובת היעד היא MAC-R.
ז)R מקבל את החבילה ומוציא מתוכה את כתובת ה IP של B.הוא בודק את זה בטבלת הניתוב ומחליט לקדם את החבילה לLAN2.
ח)R מבקש בבקשת ARP את כתובת ה MAC של B.
ט)B מחזיר תשובה לR עם כתובת ה MAC שלו.
י)R מכין חבילת Ethernet עם MAC-B עם תוכן של חבילת הIP עם היעד של B-IP והמקור הוא IP-A.

<u>Ethernet</u>

Ethernet זה הפרוטקול שעליו לרוב מבוססת רשת ה LAN כשבדרך הוא מביס פרוטוקולים יריבים כמו FDDI ו ATM. יש המון סיבות להצלחות של פרוטוקול זה אך העיקריות מהן למשל :
א)הוא היה הראשון!
ב)מאד זול למימוש!
ג)מהירות העברת נתונים גבוה(וממשיכה לעלות)



המסגרת **של**Ethernet



נניח שהמטרה שלנו עכשיו היא להעביר חבילה מפרוטוקול IP בעזרת רשת הethernet בין שני שרתים על אותה רשת LAN.

:DATA-FIELD

חלק זה מכיל את חבילת הIP.הגודל המקסימלי של חבילה בEthernet הוא 1500 bytes. ז"א אם חבילת הIP גדולה 1500 bytes השרת השולח יצטרך לחלק לחבילות קטנות יותר.
הגודל המינמלי הוא46 bytes ואם חבילה קטנה מדי נרפד אותה עד שנגיע ל46 bytes.

Destination/Source-Address

חלק זה מכיל את כתובות הLAN שזה כמו שהוסבר מקודם כתבות ה MAC.כתובתMAC מורכבת מ6 זוגות אותיות hexa סך הכל 6 בייטים.

Type Field (two bytes)

שדה זה מכיל מה תכלול הרמה הגבוהה היותר הnetwork layer.
Cyclic Redundancy Check (CRC) (4 bytes)
המטרה של שדה זה לזהות האם במהלך הדרך נוצרה איזשהי שגיאה בחבילה.כאשר שדה זה נוצר בסוף והוא תלוי בערכים אשר נוצרו בשאר המסגרת.

Preamble: (8 bytes)

כל חבילת Ethernet מתחילה ב8-byte כאשר 7 bytes הראשונים הם 10101010 והbyte האחרון הוא10101011 .המטרה של ה 7 bytes הראשונים הם ל"עורר" את הכרטיס רשת המקבל.ולסנכרן את השעונים בינו לבין השעון של השרת השולח.ה 2 bits האחרונים של הbyte השמיני והאחרון("11")הם נותנים סימן לכרטיס רשת"שים לב עד מעט מגיע המידע החשוב באמת".

An Unreliable Connectionless Service

נשים לב שהפרוטוקול איננו אמין משני סיבות עיקריות:

- כאשר אנחנו מתחילים לדבר ולשלוח הודעות Ethernet נניח משרת A לשרתB שני השרתים אינם מבצעים תהליך שנקרא handshake שזהו בעצם תהליך זיהוי לוודא ששרת A באמת מדבר עם שרת B.
- אחרי ששהודעה נשלחה מA לA,B איננו יכול לדעת שההודעה באמת הגיעה ואם היא הגיעה האם עברה את שלב הCRC או שצריך לשלוח שוב את ההודעה.

CSMA/CD: Ethernet's Multiple Access Protocol

האלגוריתם לינהול הגישה לקו הואCSMA/CD.
נחזור על כללי הפרוטוקול שהוזכר כבר בעבר:

- אין חלוקה לקטעי זמן.לכן כרטיס הרשת יכול להתחיל לשדר מתי שהוא רוצה.
- הכרטיס רשת לא ישדר לעולם אם הוא מרגיש שתחנה אחרת מנסה לשדר.

- ברגע של זיהוי התנגשות התחנה מבטלת מיד את השידור.
- לפני שהתחנה מנסה לשלוח שוב היא מחכה זמן רנדומלי.

וכעת נעבור על השלבים המדויקים שמבוצעים:

- התחנה מכינה מסגרת(frame) של Ethernet ודוחפת אותה לתוך ה buffer של הכרטיס רשת.
- אם הכרטיס מרגיש שהקו אינו תפוס ולא נשלחת חבילה הוא מתחיל בשליחה של הframe.
- בזמן השליחה החיישנים של הכרטיס ממשיכים להאזין אם אין התנגשות.אם הוא סיים לשלוח בלי להרגיש התנגשות או שכרטיס רשת אחר התחיל לשדר סיימנו.
- אחרת הוא מפסיק את השליחה ובמקום זה שולח48 bit של אות שמזהיר את כל שאר התחנות שהייתה התנגשות.
- לאחר זיהוי ההתנגשות נכנס הכרטיס להפסקה אקספונציאלית אשר אומרת שאחרי ההתנגשות הmית נחכה 2^m-1

יעילות:

יש שני דברים שמשפיעים על יעילות:

- אורך הקו
- רוחב הפס

נסמן

Tprop-המרחק המקסימלי בין 2 תחנות ברשת הLAN.

Ttrans-הזמן שלוקח להעביר חבילה בגודל מקסימלי

$$\text{efficiency} = \frac{1}{1 + 5t_{prop} / t_{trans}}$$

ככל שרוחב הפס גדל-היעילות עולה

ככל שאורך הקו גדל-היעילות יורדת.

Ethernet Technologies

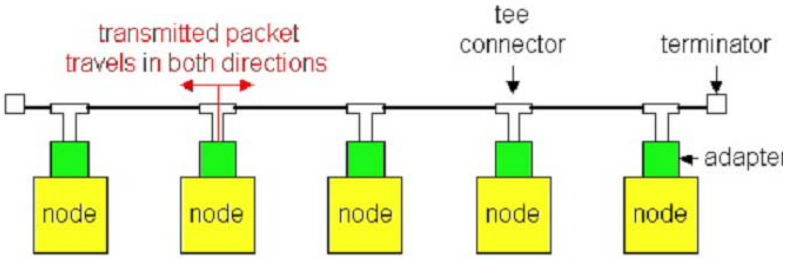
הטכנולוגיה הנפוצה ביותר של ה ethernet היא 10Base2.

אשר משתמש בשיטת הbus ומעבירה מידע במהירות של 10Mbps.

10BaseT,100BaseT:בטכנולוגיות אלו נשתמש בשיטת הכוכב בעזרת רכיב שנקראhub והן מגיעות למהיריות של 10 Mbps ו100 Mbps בהתאמה.

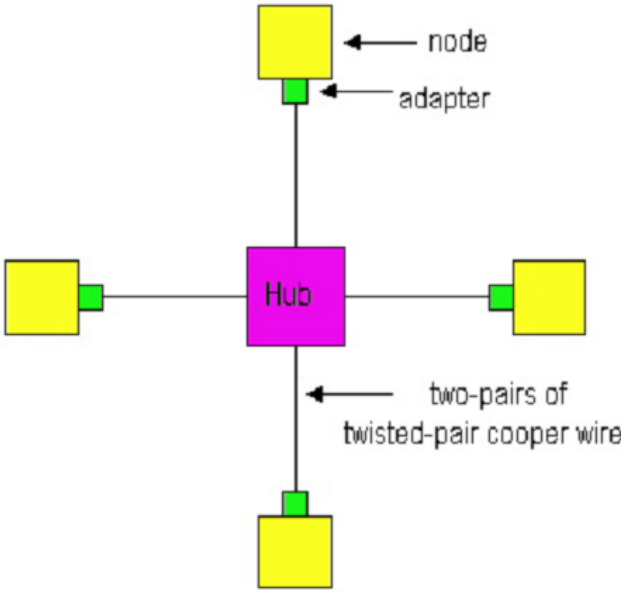
10Base2 Ethernet

כנאמר לפני זוהי טכנולוגיה מאד נפוצה.ה10 בשם עומד עבור המהירות של 10Mbps וה2 מסמן200meter שזהו המרחק המקסימלי בין שני תחנות ברשת.



10BaseT and 100BaseT

אנחנו נתאר את שני הטכנולוגיות ביחד כי הן זהות.וההבדל היחידי בנהן הוא שבטכנולוגית10BaseT המהירות היא 10Mbps ולעומת זאת ב 100BaseT המהירות היא100Mbps.שתהן משתמשות בשיטת הכוכב(star topology)



כאשר המרחק המקסימלי בין תחנה לhub עומד על 100 מטר
כעת נסביר מעט על המכשיר החדש hub:

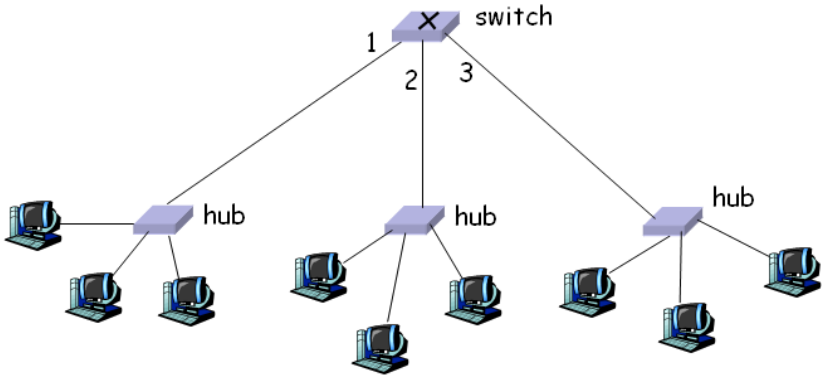
HUB

זהו רכיב מרמת החומרה שהתפקיד שלו די פשוט-הוא חוזר על דברים.
כאשר הוא מקבלי מאחד ה adapters איזשהו אות הוא חוזר על האות לכיוון כל שאר adapters .
תכונות:

- בניגוד למשל לrouters אין תורים.
- אינו מטפל או מזהה התנגשויות שקרו בקו.
- חוזר על האות באותו תדר.
- ברגע שהוא מקבל אות מlink מסוים הוא שולח את אותו אות לכל שארהlinks.
- הוא מאפשר תכונות של רשת(אם אחת התחנות מושבתת הרשת עדיין יכולה להתקיים וכל שאר התחנות יכולות לתקשר בניגוד ל10Base2)

SWITCH

הswitch בניגדו לhub נמצא בdata link layer.
כאשר חבילה מגיעה אל הswitch הוא לא רק מעתיק אותה אל עבר כל שאר הרכיבים.אלא להפך,הוא בוחן את החבילה ובודק את היעד של החבילה ומשתדל לשלוח אותה אך ורק למקום המיועד לו.



Switch מטפל בבעיות אשר נוצרו בhub למשל הוא מסוגל לטפל ולזהות התנגשויות(משתמש בפרוטוקול CSMA/CD) ובנוסף הוא מסוגל להכיר בכמה סוגי טכנולוגיות LAN ולהעביר בכל הסוגים אשר נלמדו.

Forwarding and Filtering

סינון זאת היכולת להחליט אם להעביר את החבילה לאיזשהו מחלקה של מחשבים או שפשוט לא לטפל בה.
העברה(forwarding) זאת היכולת להחליט לאן החבילה תכוון.
שני המושגים האלו ממומשים ע"י טבלת הswitch.
הטבלה של כל תחנה ברשת הLAN מורכבת מ:

1. כתבות הLAN של מחשב מסוים
2. המחלקה(interface) אליה אותו המחשב שייך.
3. הזמן שבו המחשב הוכנס לתוך הטבלה.

לדוגמא:

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
...	...	...

Self-Learning

אחת התכונות החשובות של הswitch היא שהוא לומד לבד ובונה את הטבלה שלו תוך כדי ריצה.זאת נעשה ללא כל התערבות של מנהל הרשת או של הפרוטוקול במילים אחרון הswitch פשוט לומד לבד!כעת נדגים את השלבים בהם זה נעשה:

- הטבלה של הswitch מאותחלת להיות ריקה.
- כאשר חבילה מגיעה מאחד הinterfaces אך היעד שמצוין במסגרת איננו בטבלה,אז הswitch מעביר עותקים של החבילה לכל היציאות שלו חוץ מן הlink ממנו הגיעה החבילה.(בכל אחד מהשליחות לתוך רשת הLAN הוא משתמש כדי לגשת לרשת בפרוטוקול CSMA/CD)
- על כל מסגרת שמגיעה הswitch מכניס לתוך הטבלה שלו:

1. את כתובת הLAN שנמצאת תחת השדה source address בתוך המסגרת של החבילה.
2. את המחלקה(interface) ממנה הגיעה החבילה
3. את הזמן שהגיעה.

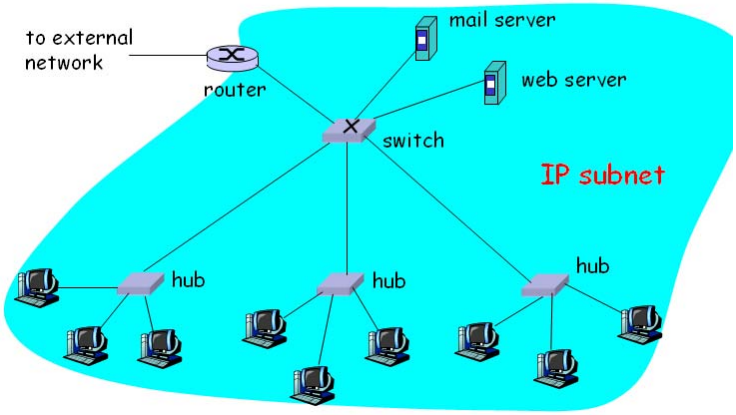
בדרך זאת יכול הswitch לקלוט את כל אחד מכתובות הLAN של החבילות ששלח אליו חבילה.אם כל חבילה ברשת הLAN בסופו של דבר ישלח חבילה דרך הswitch אז הטבלה תהיה מלאה ונוכל לנווט במדויק כל חבילה.

- כאשר חבילה מגיעה וכתובת היעד אשר במסגרת כן מופיע בטבלה ניתן לשלוח את החבילה למחלקה שבה נמצאת הכתובת ישירות.
- הswitch מוחק מן הטבלה תחנות אם לא קיבל מהם ,זמן מסוים,שום חבילה.

***<u>הערה:</u>ניתן לראות דוגמא מצוינת בשקפים של ההרצאה שקף מספר 70.**

Switch הם רכיבים ממשפחת הplug&play משום שאין צורך לקנפג או לסנכרן אותו לפני השימוש כי לפי מה שהוזכר עכשיו הרכיב לומד לבד!!
יתרונות שימוש בswitch:

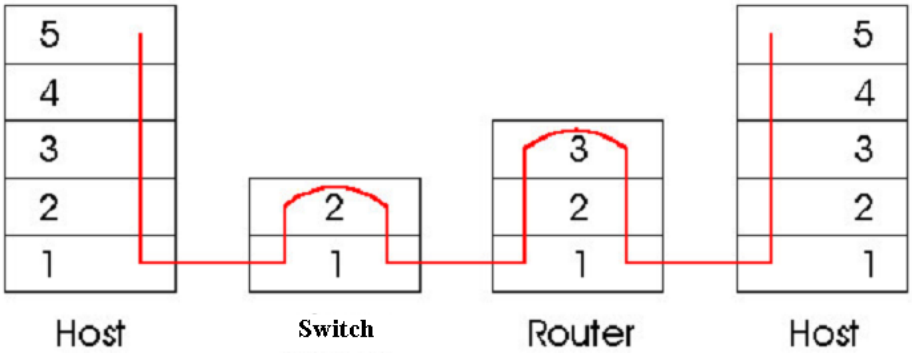
- לא מעביר broadcast לרשתות אחרות.
- אפשר גם לחבר לswitch תחנות באופן ישיר ולאו דווקא דרך מחלקות שונות.
- הוא יכול להעביר חבילות בו זמנית ולמנוע התנגשויות.
- ישנו שכלול בswitch מרגע קבלת החבילה הוא מזהה את היעד ע"פ המסגרת ומתחיל לשלוח לפני שקיבל את כל החבילה.זה נקרא cut through



דוגמא למערכת הכוללת של switch routers ו-hub

Switches versus Routers

כמו שהוזכר בסיכום על שכבת הnetwork, routers גם מאכסנים ומעבירים חבילות ומנווטים אותם ליעדם בעזרת כתובת ה IP.למרות שswitch גם הוא רכיב שמאכסן ומעביר חבילות הוא לגמרי שונה מן הrouter בזה שהחבילות מגיעות ליעדם בעזרת כתובת הLAN.כמובן גם שהrouters נמצאים בשכבה מס' 3 זוהי שכבת ה network ולעומת זאת הswitch נמצא בשכבה מס' 2 data link layer.



אז מתי נשתמש תאכלס בכל אחד מן הרכיבים:

כאשר מדובר ברשתות קטנו של כמה מאות שרתים יש מעט קטעים של LAN. לכן switch יהיו מספיקים עבור רשתות קטנות כאלו משום שהם יהיו מסוגלים לנווט בלי קינפוג מראש של כתובות ה IP.אבל ברשתות גדולות בעלות אלפי שרתים מכילות routers (בנוסף לswitch).משום שהrouters מבטיחים ניווט יותר טוב של ה"תנועה" של החבילות,שליטה על הצפות של שליחת broadcast,וניווט יותר חכם בין שרתי הרשת.

אני רק אדגיש ואוסיף שrouters קובעים את המסלולים שלהם ע"י אלגורתמי ניווט מוגדרים מראש!לעומת זאת ה switch עצמו לומד לבד תוך כדי ריצה!

מצורפת טבלה עם סיכום על ההבדלים העיקריים:

	Hubs	Routers	Switches
Traffic isolation	**No**	**Yes**	**Yes**
Plug & play	**Yes**	**No**	**Yes**
Optimal routing	**No**	**Yes**	**No**
Cut through	**Yes**	**No**	**Yes**

PPP: the point-to-point protocol

ONE SENDER ,ONE RECIVER, ONE LINK!!!

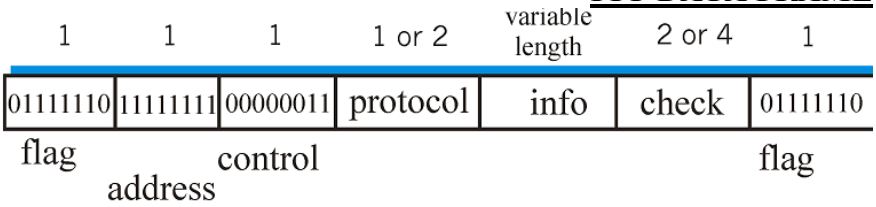
כפי שמשתמע משמו של פרוטוקול זה הוא מורכב מקו בין נקוד לנקודה.ישנו צינור משותף בעל שני קצוות ודרכו משדרים בין תחנה לתחנה.בפרוטוקוך זה משתמשים למשל בחיבורי אינטרנט שהם dialup connection.

דרישות עיצוב

מסגרת של חבילה-השולח ברמת הdata link בעזרת פרוטוקול PPP חייב להיות מסוגל לעטוף(למסגר) את החבילה מרמת הnetwork כך שמי שמקבל את החבילה יהיה מסוגל להבחין איפה נגמרת העטיפה ומתחילה ההודעה עצמה מרמת הnetwork.

העברה-פרוטקול PPP חייב להעביר את כל הbits שמועברים מהרמה מעל כולל ה header...
תמיכה של מספר פרוטוקולים ברמת הnetwork-פרוטוקול הPPP חייב לתמוך בכמה סוגים שונים של פרוטוקולים מהרמה שמעליו שרצים על אותו רכיב פיזי מרמת הlink באותו זמן.כמו שפרוטוקול ה IP חייב לתמוך בכמה פרוטוקולים מהרמה שמעליו(transport layer) כמו למשל UDP וTCP .
תמיכה במספר דרכים של קישורים-בנוסף לזה שפרוטוקול PPP צריך להיות מסוגל להתמודד עם מספר פרוטוקולים שרצים ברמה מעליו הוא גם חייב לדעת לטפל במספר דרכים של קישור למשל:להעביר כל פעם ביט אחד בכיוון נתון או להעביר מספר ביטים(ברבים),להעביר ביטים בצורה מסונכרנת(בעזרת שעון) או לא מסונכרנים,במהירות נמוכה או במהירות גבוהה,בצורה אלקטרונית או אופטית....
זיהוי שגיאות-מי שמקבל את החבילה בעזרת פרוטוקול PPP חייב להיות מסוגל לזהות שגיאות במסגרת שהתקבלה.
Connection-liveness פרוטוקול PPP חייב להיות מסוגל לזהות אם הפרוטוקול עדיין"חי" ולא קרס...
הדברות ברמת הnetwotk- PPPחייב להיות מנגנון, בשביל פורטוקלי התקשורת ברמת הnetwork,שילמד כל אחד מהאחרים את כתובת ברמת הnetwork.

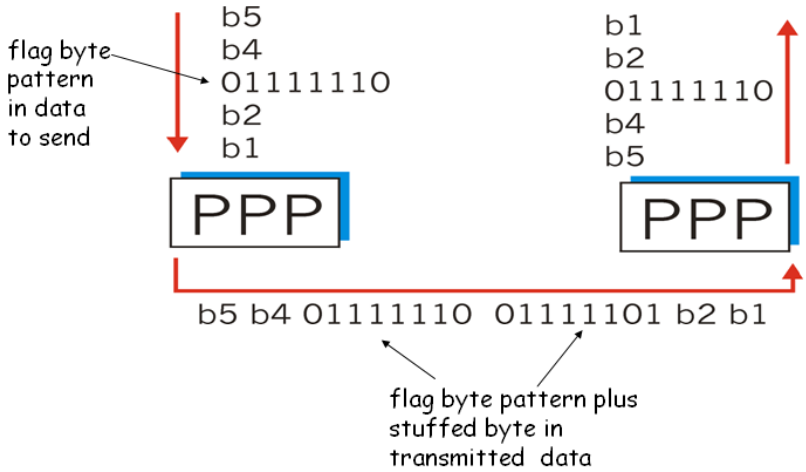
PPP DATA FRAME



Flag-כל מסגרת של PPP מתחילה ונגמרת בbyte1 בעל ערך של 01111110
Address field-הערך היחידי האפשרי לשדה זה הוא 11111111
Control field-הערך היחידי האפשרי לשדה זה הוא 00000011.משום ששני השדות האלו קבועים אין באמת צורך לשלוח אותם וכך לחסוך שני bytes מהמסגרת של PPP.
Protocol-שדה זה אומר למקבל של חבילת הPPP את הפרוטוקול לשכבה העליונה שאותה עוטף הPPP(המידע שנמצא בתוך המסגרת של הPPP משתמש בפרוטוקול זה)
Information-שדה זה מכיל את המידע.שכאמור נשלח ע"י הפרוטוקול שצוין בשדה הקודם.
Checksum-בשדה זה משתמשים כדי לזהות שגיאות בbitים במסגרת המועברת.

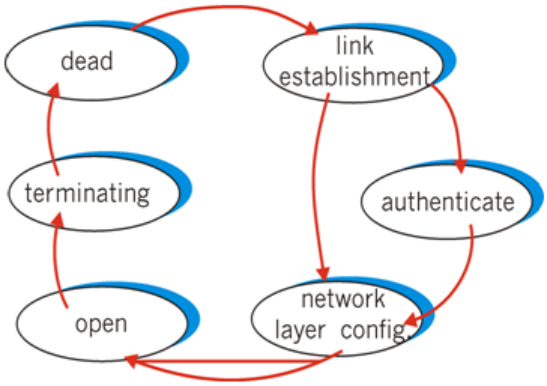
Byte Stuffing

בזמן הגדרת הframe נתקלנו בשדות הflag אשר מסמנים את הסוף וההתחלה של המסגרת.אך שדות אלו יוצרים גם בעיה מה יקרה אם במהלך המסגרת יופיע 0111110 נוסף?
איך נוכל להיות בטוחים שזהו flag ולא מידע מהחבילה?
הפתרון שפרוטוקול PPP משתמש בו נקרא byte stuffing.
הפרוטוקול מגדיר byte מיוחד שמוגדר כ:01111101.אם במהלך המסגרת,חוץ מזהשדה של הflags,מופיע 01111110 אז מיד אחריו הוא דוחף את הbyte המיוחד ככה מי שמקבל את המסגרת ומזהה 01111110 יבדוק אם מיד אחריו מופיע הbyte אם כן סימן שזה חלק מהמידע ואינו מסמן את סוף ההודעה!



PPP Link Control Protocol (LCP) and network control protocols

לפני שמתחילים להעביר מידע בין שתי התחנות על הקו של PPP חייבים לבצע:
כל אחד מן תחנות הקצה מבצע פעולות אשר מתוארות בעזרת מכונת המצבים הבאה:



קו הPPP תמיד יתחיל ויסיים במצב הdead.מתי שמתקיים שהרמה הפיזית מוכנה לשדר הוא יעבור למצב: link establishment.

במצב זה קצה אחד של הקו שולח את הקונפיגורציה שהוא רוצה שיהיה לקו בעזרת מסגרת של הLCP שנקראת configure-request(מסגרת ששיכת לפרוטוקולPPP שמכילה את הקונפיגרציות שצד אחד דורש).תחנת הקצה השניה יכולה לענות לה או ב:

- Configure-ack-הכל בסדר והיא מסכימה למה שהתחנה הראשונה הציעה.
- Configure –nak-קיבלתי את כל האופציות הבנתי אבל אני לא מסכימה.
- Configure-reject-האופציות שהציעו לא מוכרות לי או לא מקובלות עלי

מהרגע שבו הגיעו להסכמה ונניח שבוצע גם אימות(authentication) אם יש צורך,שני הצדדים יחליפו בינהם חבילות שיתאמו את רמת הnetwork.לאחר הסכמה על הnetwork אפשר להתחיל לשלוח חבילות עם המידע עצמו ברמת ה network על גבי קו הPPP.

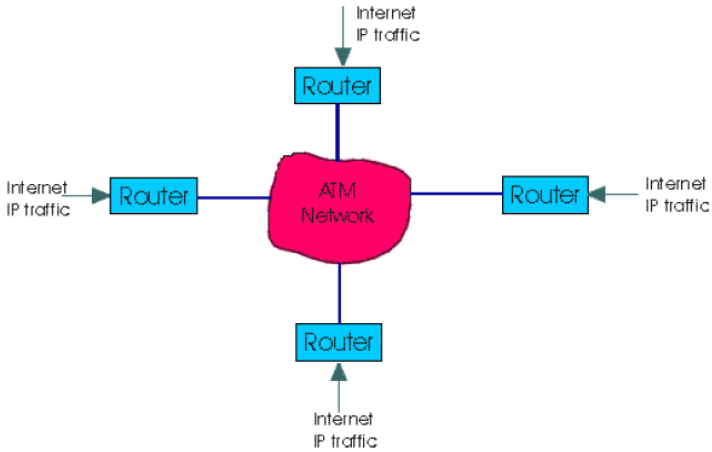
לסיכום!!!!!!!!

PPP זה פרוטוקול ברמת הdata link layer שמקשרת בין שתי תחנות ברמת הlink.אחת בכך צד של הlink מחליפים מסגרות של PPP אשר מכילות חבילות ברמת הnetwork.

ATM

המטרה הראשונית של פרוטוקול ATM הייתה ליצור מעבר בין שתי נקודות קצה של קבצי קול וידאו ונתונים.
בשרטוט הבא ניתן לראות את השלד של ATM בעל 4 נקודות כניסה\יציאה בשביל תנועה של Internet IP.נשים לב שכל כניסה\יציאה מהווה router.לרוב בשלדים של ATM ישנם virtual channel/circuit
(ובעברית:וירטואליזציה של רשתות)מה שאומר שבין הכניסות והיציאות של הATM ישנם רשת\חוג שעליה ניתן להעביר את ההודעות באופן קבוע וסטטי בניגוד למשל לrouter שאצלו המסלול נקבע באופן דינמי אשר תלוי במצב הנוכחי של הרשת.

כל router או interface אשר מחוברים לרשת ATM בעלי שני סוגי כתובות הסוג הראשון תהיה כמובן כתובת IP כרגיל אך תהיה לו גם כתובת ATM.



נתאר כעת תנועה על פני השלד שמתואר למעלה.נניח שאנחנו מעוניינים להעביר חבילת IP.נגדיר את הrouter אליו נכנסת החבילה לתוך הATM כrouter כניסה ואת הrouter ממנו יוצאת החבילה מן הATM כrouter יציאה.

Router הכניסה מבצע את השלבים הבאים:

1. בודק את כתובת היעד של החבילה

2. מחליט לאיזה router הוא ינווט את החבילה
3. כדי שהחבילה תגיע לתוך router היציאה הrouter רואה את רשת הATM כמו כל פרוטוקול אחר ברמת ה link.לכן הוא מחפש את כתובת הATM שלו בעזרת פרוטוקולARP בעזרת כתובת הIP של router היציאה וכך הוא יכול לכוון אותו לכתובות הATM של routerהיציאה.

כעת האחראיות על העברת החבילה כבר אינה של שכבת הIP אלא פרוטוקול הATM בשכבת הlink אחראי על כך.ATM חייב להעביר את החבילה לכתובת שהשגנו בשלב מספר 3.לכן הוא מבצע:

- מחליט על VC(virtual circuit) שיובילו אל כתובת ה ATM.
- מחלק את החבילה לתוך תאים בצד השולח של הVC ומשחזר את התאים לתוך הודעה בצד המקבל של הVC.

אז איך עובד בדיוק פרוטוקול הATM? אז ככה לפרוטוקול יש שלוש שכבות:

1. שכבה פיזית
2. שכבת ה ATM
3. שכבת הadaptation(עבוד)

הסבר קצר על Virtual circuit.

VC transport –תאים מעוברים ע"י ה VC מהמקור ליעד.

יתרונות של שימוש בVC תחת ATM:

- בעת שימוש בVC מובטח איכות שירות לחיבור(רוחב פס,delay,וכו...)

השכבה הפיזית של ATM

השכבה הפיזית דואגת לשליחה עצמה של תא שלATM על גבי קו פיזי יחיד.

לשכבה הזאת יש 2 תתי שכבות:

- Physical Medium Dependent(PMD)
- Transmission Convergence(TC)

Sublayer	Responsibilites
Transmission Convergence(TC) Sublayer	Idle Cell Insertion Cell Delineation Transmission Frame Adaptation
Physical Medium Dependent (PMD) Sublayer	Physical Medium Bit voltages and timings Frame structure

PMD

שכבת הPDM זאת השכבה התחתונה ביותר בפרוטוקול ATM.כפי שמשתמע משמה זאת השכבה הפיזית של הlink.

השכבה כוללת:

1. SONET\SDH (synchronous Optical Network/Synchronous Digital Hierarchy)-על גבי סיב בעל מצב יחיד.SONET וSDH יש מבנה של מסגרת אשר מאפשרת סינכרון ביטים בין השולח למקבל בשני קצוות הקו.
2. T1\T3 מסגרות על פני הסיב שמסוגלות להעביר בקצב של15Mbps/45Mbps בהתאמה.
3. unstructured-תאים בודדים ללא מסגרת מותאמת.

TC

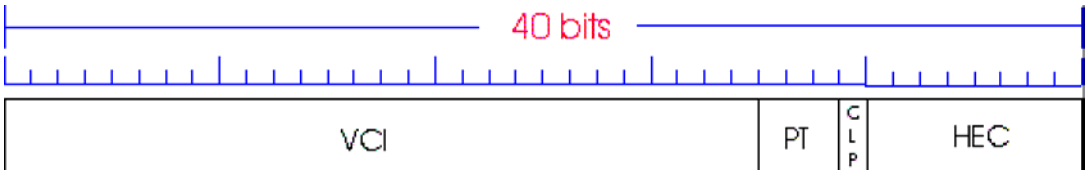
תפקידו של ה:TC

1. בצד של השולח של הקו אמור לקבל תאים של פרוטוקול ATM מרמת הATM ולהכניס את התאים לתוך הPMD
2. בצד המקבל של הקו עליו לקבץ את כל הביטים שמגיעים מתוך הPMD לתוך תאים ולהעביר את התאים לתוך שכבת הATM.

שכבת הביניים TC יושבת מעל שכבת הביניים PMD.ומתחת לשכבהATM.

שכבת הATM

כאשר פרוטוקול הIP רץ על פרוטוקול הATM (לפרטים נוספים על ריצה על ATM ניתן לראות במצגת מספר 105) תאי ATM משחקים בתפקיד של מסגרות של שכבת הlink.שכבת הATM מגדירה את מבנה של תאי ה ATM ואת תוכן השדות במבנה זה.הbytes 5 הראשונים מגדירים את ה header שאר הbytes 48 מגדירים את המידע של הATM.מבנה הheader:



נפרט על התאים:

VCI(virtual Channel Identifier)-מציין לאיזה VC לאן התא הזה משתייך.

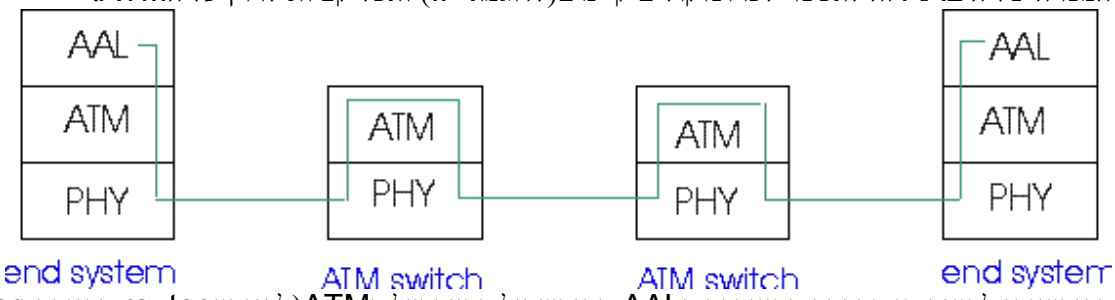
PT(Payload Type)-מציין את סוג המידע שהתא מכיל.

CLP(Cell Loss Priority)-מציין אם התא בעל עדיפות נמוכה או גבוה.

HEC(header Errir Checksum)-בדיקה על כל הheader.

ATM Adaptation Layer(AAL)

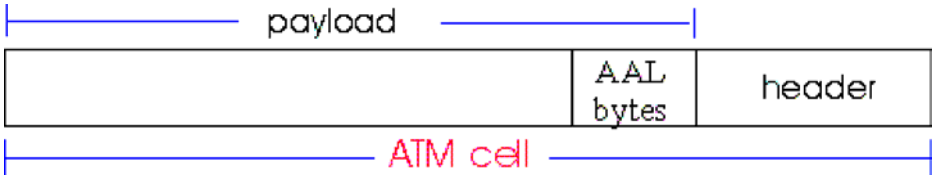
המטרה של ה AAL זה לאפשר לפרוטוקולים קיימים(לדוגמא IP) ואפליקציות לרוץ על הATM.



כפי שניתן לראות מן הסכמה המצורפת הAAL ממומש על הקצה של הATM(לדוגמא router הכניסה וrouter היציאה) ולא בswitches הפנימיים של הATM.

שכבת הAAL מתקשרת אנאלוגית לשכבת הtransport בפרוטוקול האינטרנט.

לתת השכבה AAL יש שדות משלה בheader.



שדות אלו מכילים חלק קטן מהמידע שנמצא בתא הATM.

ישנם גרסאות שונות של שכבת AAL.בחירת השכבות תלויות בשירות ש ATM אמור לספק:

AAL 1: For Constant Bit Rate (CBR) services and circuit emulation.

AAL 2: For Variable Bit Rate (VBR) services.

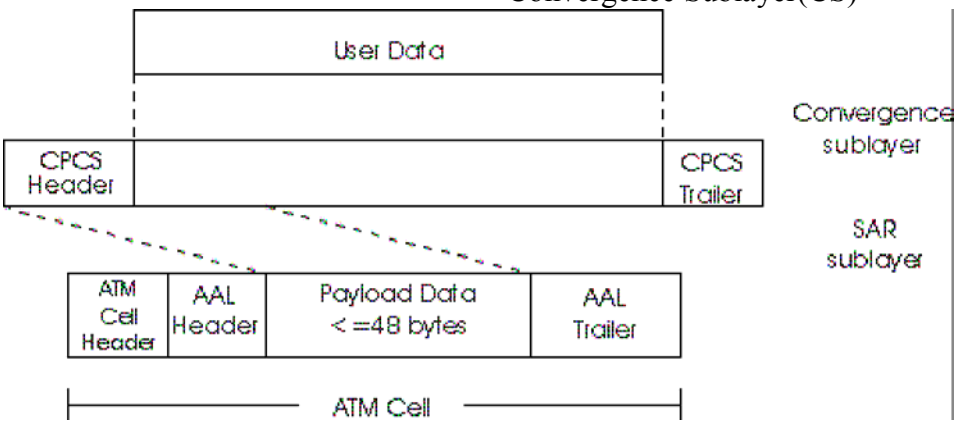
AAL 5: For data (e.g., IP datagrams)

<u>מבנה הAAL</u>

ל AAL יש שתי תתי שכבות:

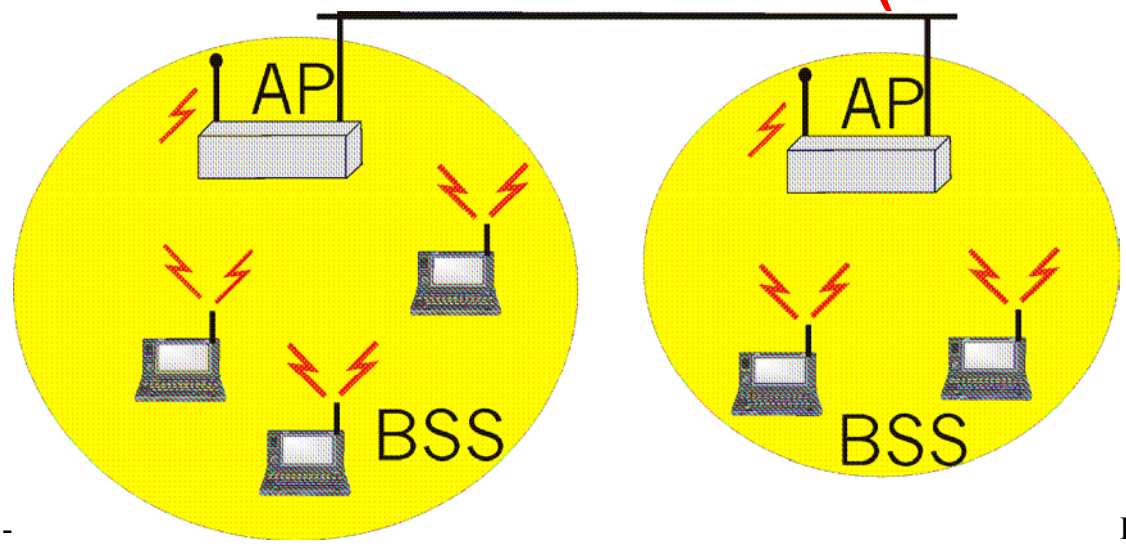
- Segmentation And Reassembly(SAR)

- Convergence Sublayer(CS)



כפי שניתן לראות מהשרטוט שכבת הSAR נמצאת מתחת לשכבת הATM,ושכבת הCS נמצאת בין האפליקציה של המשתמש לSAR.
המידע של המשתמש(למשל חבילת הIP) דבר ראשון נעטף ע"י התת-שכבה CPCS בתור PDU בתוך שכבת הCS.ה PDU יכולה להוות או בתור header CPCS או בתור trailer.
שכבת הSAR מחלקת את הCPCS-PDU ומוסיף header של שכבת הAAL.

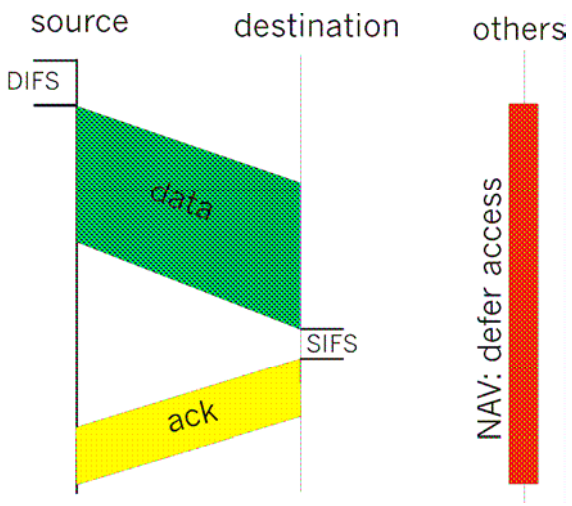
IEEE 802.11 LANs(wireless)!



BSS - basic service set-ה BSS מכיל בדרך כלל תחנה אחת או יותר של wireless ותחנת בסיס(access point-AP).

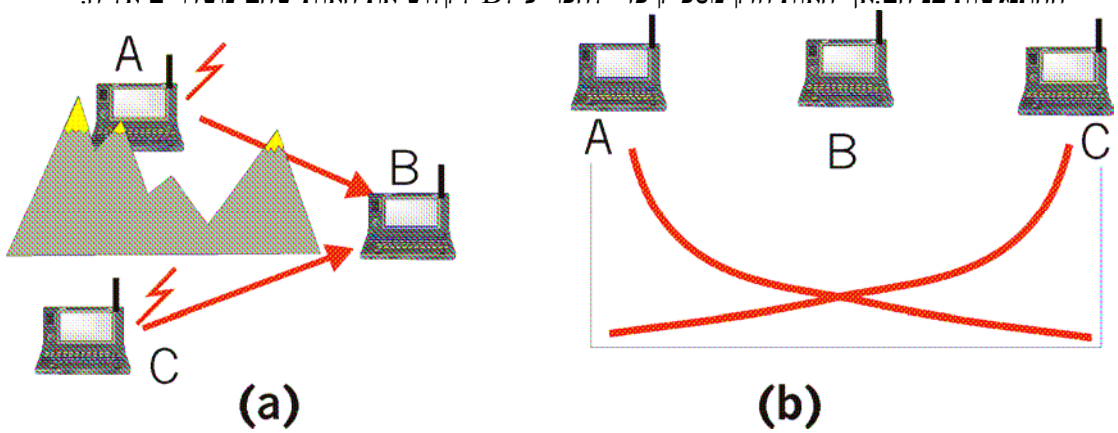
802.11 Media Access Protocols

כמו ברשת LAN החוטית גם ברשת האלחוטית חייבים לתאם את הגישה לרשת התקשורת(במקרה הזה מדובר בתדרי רדיו).גם הפעם אנחנו נשתמש בפרוטוקול CSMA/CA.
נגדיר מושגים-
Distributed Inter Frame Space-DIFS.
Short Inter Frame Spacing – SIFS.
אז כמו שלמדנו CSMA הוא פרוטוקול מנומס הוא קודם כל מקשיב ורק אם הקו פנוי הוא מתחי לשדר.אז רשמית הפרוטוקול מתנהג כך:
אם הקו פנוי לזמן בגודל DIFS אזי לתחנה מותר להתחיל לשלוח.כאשר התחנה המקבלת קיבלה בהצלחה ובשלמות את ההודעה היא מחכה זמן קצר בגודל SIFS ואז שולחת משוב על ההצלחה לשולח.



השרטוט מתאר מצב של שליחה כאשר הקו פנוי,אך מה קורה כאשר הקו איננו פנוי?במקרה כזה אנחנו נבצע תהליך דומה אשר בוצע ברשת החוטית(Ethernet).אך נשים לב שלא כמו פרוטוקול הethernet פרוטוקול הwireless איננו מטפל במקרה של התנגשויות ויש שתי סיבות לכך:

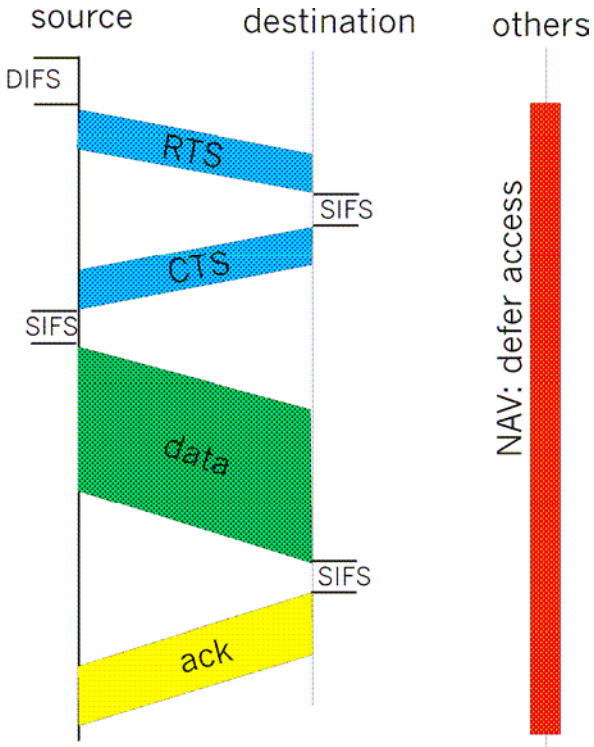
- היכולת לזהות התנגשויות דורשת את היכולת גם לשדר וגם לקלוט באותו זמן.דבר שעלול להיות יקר במערכות אלו.
- הסיבה היותר חשובה היא נגיד אני שולח את החבילה מקשיב לקו אבל איני מזהה התנגשות עדיין יכולה להיות אופציה שהייתה התנגשות אצל המקבל.למשל נגיד תחנה A שולחת לתחנה B.נניח שגם תחנהC שולחת לתחנה B.כעת נניח שיש משו פיזי שמפריע לA וC לשמוע אחד את השני(לדוגמא:הר,קיר..)למרות שהשידורים שלהם מתערבבים ביעד,B.עד דוגמא לבעיה שנובעת מחוסר זיהוי של התנגשות אצל המקבל כתוצאה מהתחלשות של חוזק האות.לדוגמא תחנות A וC ממוקמות כך שהחוזק של האות שלהם לא חזק מספיק כדי לזהות את ההתנגשות בניהם.אך האות חזק מספיק כדי להפריע לB לקלוט את האות שהם משדרים אליה.



בעיות אלו שמנעו מאתנו דרכים לזהות התנגשויות אחרי השליחה,גרמו לחשוב על פרוטוקולים שימנעו התנגשויות לגמרי כדי שלא יהיה צורך בכלל לזהות אותם.לכן לפרוטוקול CSMA צירפוavoid collision (וביחד CSMA/CA).

הפרוטוקול עובד כך:

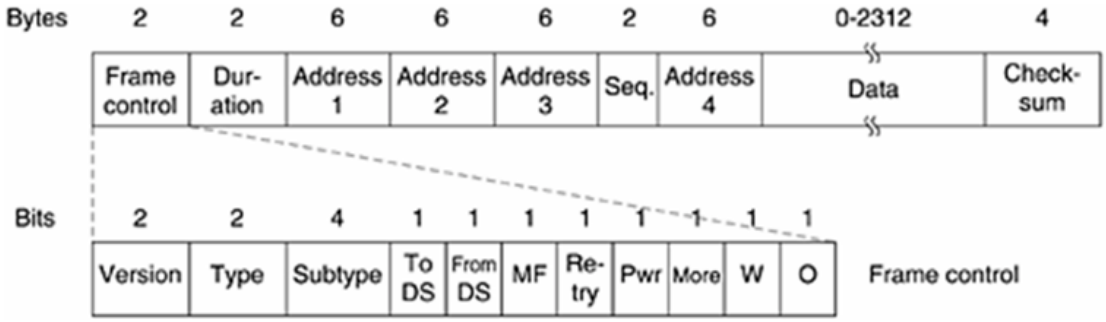
תחנה שרוצה לשדר שולחת הודעה קצרה מאד ליעד ששמה RTS(Request To Send). אם המקבל(היעד) מקבל את ההודעה ומפענח אותה ובתגובה שולח לכל התחנות CTS(Clear To Send) עם שם התחנה שקיבלה את האישור לשדר.בזמן הזה התחנה שקיבלה אישור ממשיכה לשדר את החבילה עם המידע כמו שהוסבר לפני כן וכל שאר התחנות נמצאות במצב NAV ונמנעות משליחה עד שיודיעו להם אחרת.



בפרוטוקול 802.11 יש מנגנון של Beacon-הbase station שולת ב broadcast מידע על עצמה ומזמינה תחנות להתחבר אליה.

Power management-לBS מותר להגיד לתחנה ללכת לישון (כדי לחסוך בטירה) והוא יעיר אותה אם יהיו לו הודעות בשבילו.

802.11 Frame



Data/Control/Management-Type

Sub Type-מה הסוג בתוך הtype לדוג':CTS,RTS.

To DS/From DS-מדליקים את הביטים האלו אם יוצאים לרשתות חיצוניות.

MF-more fragments-אומר אם יש עד פרגמנטים שצריכים להגיע.

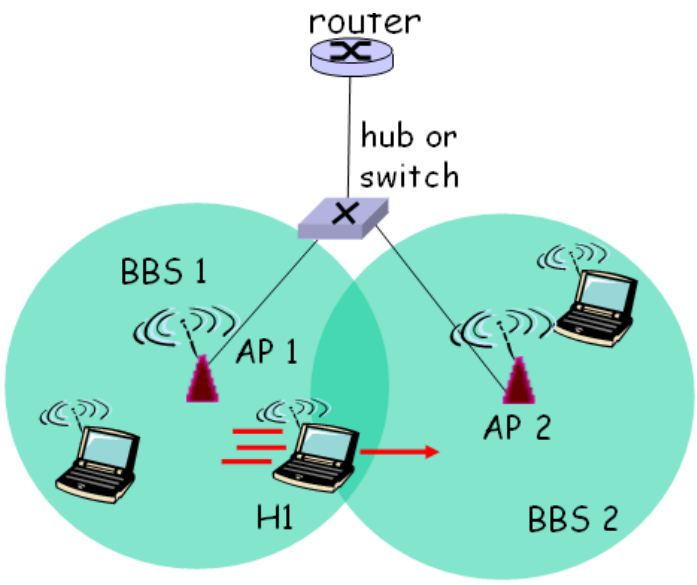
Retry-הביט דולק אם זה retransmission.

Pwr- אם הBS רוצה שהתחנה תלך לישון היא מסמנת לה בשדה זה.

W-הביט דלוק אם יש הצפנה.

O-אומר למקבל שהוא צריך לשמור על הסדר של ההודעות שהוא מקבל.

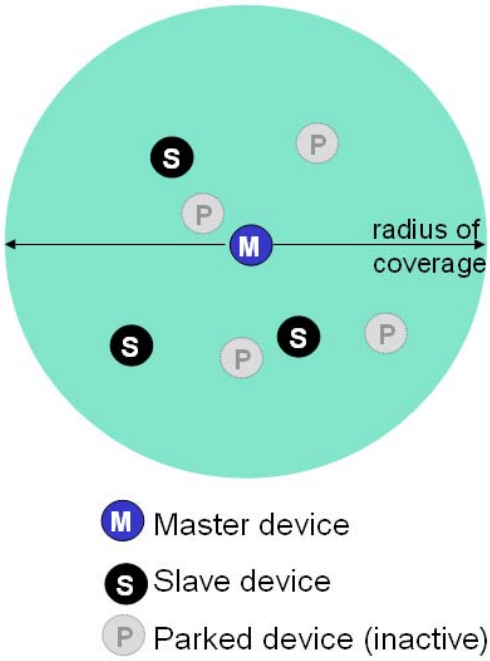
<u>התניידות בתוך אותה תת רשת.</u>



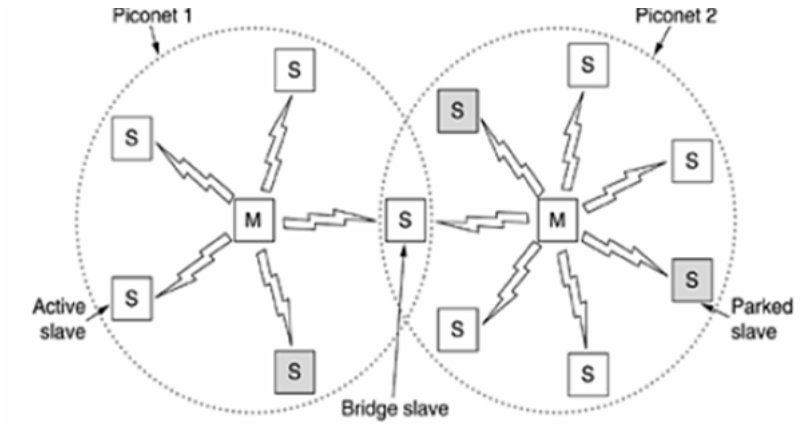
H1 מתנייד לBBS2 אך מכיוון שהוא נשאר באותו חיבור לרשת(router) כתובת הIP שלו נשארת זהה.מה שעושים זה משתמשים בswitch ובגלל התכונה שלו של למידה לבד הוא לומד ומזהה איפה נמצא הhost ובפעם הבאה שמישהו ישלח לו הודעה הוא יידע לנווט אותו לAP הנכון.

802.15 Bluetooth

זהו פרוטוקול לטווחים מאד קצרים-טווח מקסימלי של 10 מטר.
הוא נועד להחליף כבלים למשל באוזניות,עכבר,מקלדת וכו...
בד"כ איננו מסוגל לעבור מכשולים פיזים כמו קירות.
משתמשים במשדרים קטנים עם טווח קצר.
יש device שנקרא master והוא תומך ב 8 devices שנקראים slaves.



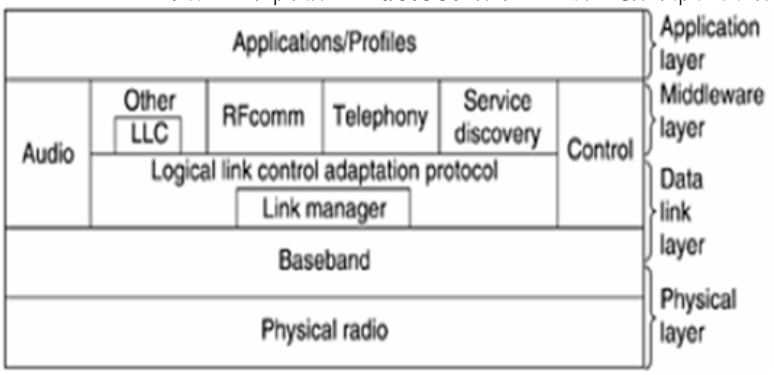
משתמשים בתדר 2.4GHZ –זהו התדר שמשתמשים בwifi ולכן יש הפרעות בשידור כשאר משתמשים בשניהם.
קצב השידור-1 מגה-ביט אבל בגלל שיש 8 devices הוא 64kb יחסית נמוך.
יש אפשרות לחבר בין שני "מערכות" כמו שהוצגו למעלה ע"י כך שאחד ה slaves משמש כגשר שמחבר בית שתי הרשתות לדוגמא:



פרוטוקול ה Bluetooth מעניק לנו שימוש להרבה אפליקציות שונות שאת כולן ניתן לראות במצגות פרק 36(פשוט חבל על המקום...:))

The Bluetooth Protocol Stack

הפרוטוקול הסטנדרטי של הBluetooth מחולק לשכבות.



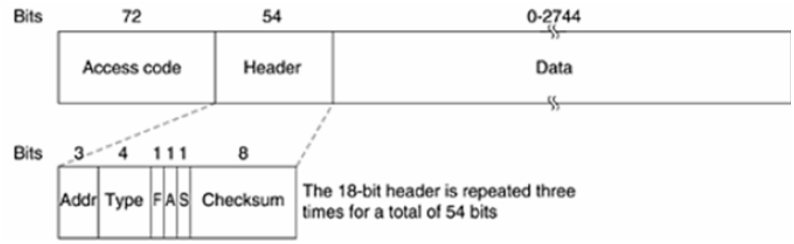
- **הרמה התחתונה ביותר היא כפי שניתן לראות הרמה הפיזית**.היא מתעסקת בשידור ובקליטה של גלי הרדיו.היא מגדירה את התדרים ואיך להשתמש בהם.השיטה בה משתמשים כדי לקבוע את התדרים היא קפיצה בין תדרים כך שבכל תדר נשאר זמן קצר וכך נמנע התנגשויות ונעלה את רמת ה"פרטיות".
- **שכבת הBaseband-** היא השכבה המקבילה לשכבת הMAC בLAN אך גם כולל בתוכה אלמנטים של השכבה הפיזית.היא מתעסקת איך תחנת הmaster שולט על מקטעי הזמן ואיך הקטעים האלו עם ההודעות מקובצים לתוך מסגרת אחת.
- **Link manager-** מטפלת בהקמת הקשר בין רכיבים,כולל האפליקציות של power management ,אימות ואיכות השירות.
- **Middleware**-שכבה זאת כוללת כמה פרוטוקולים.

למשל :

  - **RFcomm**-פרוטוקול שפועל מול המחשב לחיבור מקלדת,עכבר או מודם למכשירים אחרים.
  - **Telephony**-פרוטוקול זה זהו פרוטוקול זמן אמת אשר משמש להעברת קול הוא גם מטפל קבלת שיחות ויצירתן.
  - **Service discovery**-משמש כדי למצוא עד ישומים או רכיבים ברשת.

- **The top layer-** בשכבה זאת נמצאים האפליקציות והמאפיינים.כל אפליקציה תלויה ברכיב עליה היא מותקנת.

The Bluetooth Frame Structure



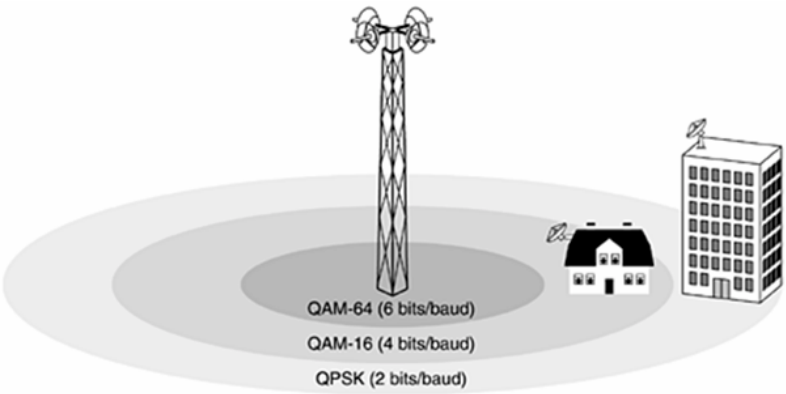
- **Address field**-מגדיר אילו מין 8 ה slaves פעילים.משום שישנם 8 משתמשים ב3 ביטים.
- **Type field**-מגדיר את סוגר הפרוטוקול את החיבור אם יש את סוג הCRC ואת אורך שנקבע למקטע הזמן בו ניתן לשדר.
- **Flags:**
  - **Flow**-נדלק ע"י ה slave אם הbuffer שלו מלא.
  - **ARQN**-הביט מהווה כACK אם הוא דלוק סימן שקיבלנו את החבילה.
  - **SEQN**-sequence number.
- **Checksum**

הערה חשובה!:כל הbit-18 של הheader נשלחים 3 פעמים כך שנשלחים בסופו של דבר 54 bit.
למה???
משום זהו מנגנון לתיקון של שגיאה אחת!הצד המקבל עובר על כל ביט בכל שלושת העותקים.אם בכל שלושת העותקים הביט זהה אז הוא מקבל אותו אם לא אז קובעים את הביט לפי הרוב(לדוגמא ב2 עותקים הביט '1' ובעותק אחד '0' נבחר את הביט להיות '1') הסיבה שעושים את זה שהקו הוא מאד רועש וחלש.

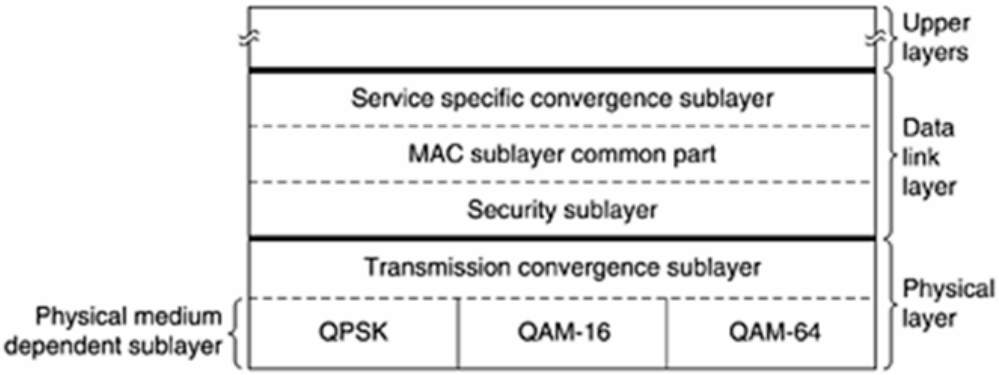
Broadband Wireless 802.16

הרעיון הוא אספקה של אינטרנט לבניינים או כל דבר נייח גדול עם הרבה מאד משתמשים.
אז למה בעצם לא להשתמש בwifi????
בעיקר משום שהם מביאים פתרונות לבעיות שונות...
ההבדלים בין פרוטוקול זה לפרוטוקול wifi

- שניהם מייצרים פס רחב של wireless.
- בעוד 802.16 משמש לשידור לבניינים לעצמים נייחים פרוטוקול 802.11 יכול לשדר גם עם רכיבים ניידים.
- בבניינים סביר להניח יש יותר ממחשב אחד סיבוך אשר לא יכול להווצר כאשר אני משדר wifi.
- בבניינים יכולת השידור והקליטה יותר טובים כי יש להם יכולת להפעיל יותר הספק מאשר פלאפון לדוגמא.
- בגלל שב 802.16 אתה יכול לשדר לעיר שלמה אז מרחק בין תחנות יכול להיות קילומטרים שלמים לכן האות הרבה יותר חזק מאשר מהפלאפון שלנו ולכן הוא יותר בטוח מבחינת פרטיות ורעש.
- QOS האיכות של הקו ושל העברת הנתונים ב802.16 הרבה יותר גבוה הוא מסוגל להעביר שיחות טלפון וידאו ועוד...
- בקיצור 802.11 נועד להיות מין Ethernet נייד בעוד 802.16 נועד פשוט מאד להיות כבל טלוויזיה אבל בwireless וכמובן נייח!



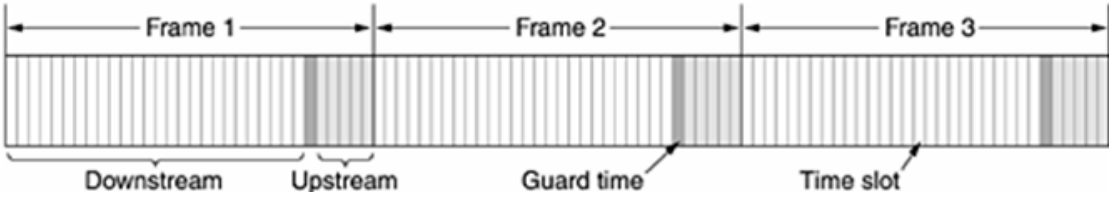
The 802.16 Protocol Stack



המחסנית של הפרוטוקול כמעט זהה לכל שאר הפרוטוקולים ממשפחת 802 אבל עם יותר תת שכבות.
ברמה הפיזית מחליטים מה עוצמת השידור לפי עוצמת הרעש וכו..
וכל השאר זהה למה שראינו עד כה.

Frames and Time Slot

הגישה לרשת נעשת גם ע"י FDM וגם ע"י TDM.



שכבת הConvergence.

Downstrem-נקבע ע"י תחנת הבסיס.

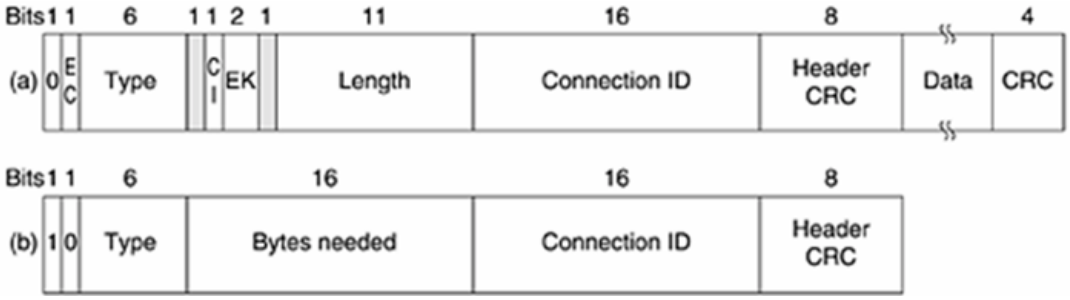
שירותי Upstream אפשריים–

- CBR(constant bit rate)-קצב קבוע מובטח זה מאד בזבזני
- Real time VBR(variable bir rate) קובעים בזמן אמת מה יהיה הקצב
- real time VBR בלי
- Best effort service.

Polling-הBS שואל מי רוצה לשדר.יש אפשרות לשאול רק את התחנות שמשדרות הרבה וכל כמה זמן לתת לתחנות שלא משדרות הרבה הזדמנות לשדר.

בפרוטוקול ה802.16 משתמשים במנגנון Hamming error connection לתיקון שגיאות.

The 802.16 Frame Structure



כפי שניתן לראות ישנם שני סוגי חבילות
החבילה הראשונה היא רגילה להעברת מידע והחבילה השנייה היא חבילה מיוחדת לבקשת bandwidth מסוים.
נעבור על שדות החבילה:

- **EC**-אם הביט דלוק זה אומר שהמידע מוצפן.
- **EK**-אומר לנו מה היא סוג ההצפה.
- **Type**-מוודא את סוג המסגרת של החבילה לרוב הוא מגדיר אם החבילה עברה חלוקה או נארזה.

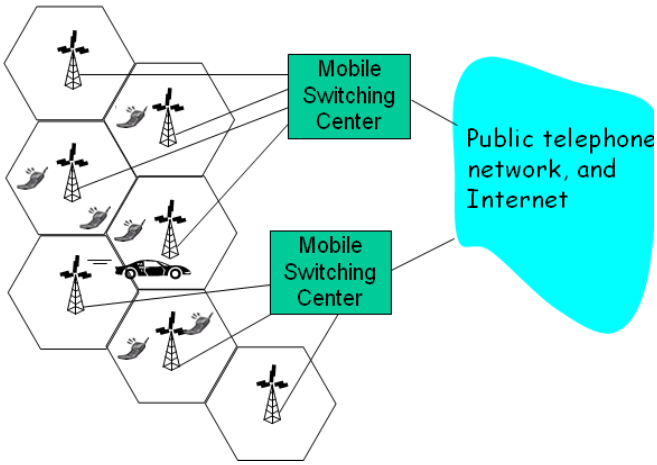
- **CI**-ביט שאומר אם יש או אין checksum.
- **Length**-הגודל המלא של החבילה כולל ה header.
- **Connection ID**-מזהה של הconnection ונותן מידע לאיזה חיבור החבילה שייכת.
- **CRC**-אנחנו מבצעים בדיקה על הheader באמצעות הפולינום $x^8 + x^2 + x + 1$

רשתות סלולריות

*הערה עד כה סיכמתי מתוך הספר של רוס או טננבאום אך פרק זה אינו מופיע בשניהם אך ורק בהרצאות לכן אני משתמש בסיכום של אסתי קומר וכדאי מאד לעקוב במקביל על המצגות של ההרצאה ממצגת 50 בפרק 6.

אנחנו נתאר את הרשת באמצעות חלוקה לתאים אשר בכל תא ישנה BS או AP (access point) שנותנות שירות.
מעל לרמה זאת ישנם את הMSC(mobile switch center) שהתפקידים שלו הם

1. לקשר בין תאים וכך ליצור רשת על גבי שטח רחב
2. מנהל את השיחות
3. ומטפל בבעיית הניידות.



ישנם שני סוגים כדי לנהל את הגישה לרשת:

1. שילוב של FDMA וTDMA
   חלוקה של טווח התדרים לכל תחנה וחלוקת הזמנים למקטעי זמן לכל תחנה
2. **CDMA** כמו שנלמד בפר שעבר ומאפשר למספר תחנות לדבר במקביל.

בארץ אנחנו משתמשים בשתי שיטות GSM(Orange Cellcom) וCDMA(פלאפון).
כך שהGSM משתמש בשילוב של הFDMA והTDMA.

<u>ניידות</u>

נגדיר כמה מושגים שבהם נשתמש בהמשך:

- **Home network**-הרשת בה המשתמש נמצא רוב הזמן(מוגדר).
- **Home agent** -האחראי על ביצוע הניידות.ז"א לתת שירות למשתמש כאשר הוא נמצא מחוץ ל home network.
- **Foreign agent** -מי שאחראי לתת שירות לאורחים ברשת שאינם קבועים שם.

- **Visited network**-הרשת החדשה בה אנחנו נמצאים.

<u>איך הניידות מתבצעת??</u>

היינו רוצים שאם משהו נמצא ברשת אחרת הrouter יידע להעביר אליו את המידע לא משנה באיזה רשת הוא נמצא.הדרך הטבעית היא שהrouters יעקבו אחריו ויהיה ממש קל להגיע אליו אך לצערנו זה בלתי אפשרי(וגם לא משתלם לחברות הסלולר!!) משום שיש ניידות רבה אז אם נבקש מהrouters לעקוב זה יגרום עומס נוראי.
ולכן הhome agent אחראי על זה.
ישנם שני דרכי גישה לטיפול בניידות דרך home agent:

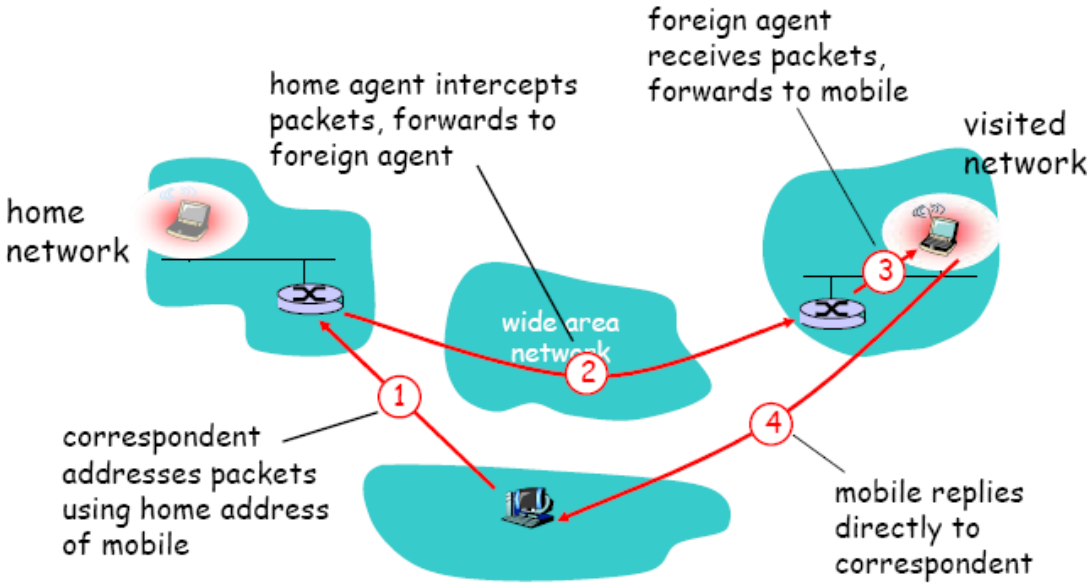
- **Indirect routing**-תקשורת ממקור ליעד נייד כלשהו עובר דרך ה home agent ואז מועבר.
- **Direct routing**-מי שיוצר את השיחה מקבל את הכתובת החדשה של הנייד ושולח ישירות אליו.

תהליך registration

זהו תהליך "ההרשמה" של תחנה אשר מתארחת ברשת זרה המטרה שלנו שבסוף התהליך הforeign agent יכיר את המכשיר והhome agent ידע איפה הוא.
התהליך מתבצע בשני שלבים:

1. המכשיר הנייד יוצר קשר עם foreign agent, ומבקש ממנו להצטרף ולקבל שירות.
2. הforeign agent יוצר קשר עם ה home agent של המכשיר ומודיע לו שהמכשיר הזה עכשיו נמצא אצלו ברשת.

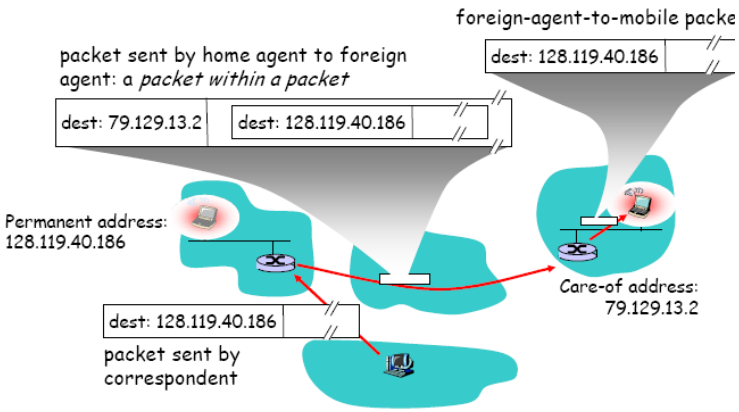
יצירת קשר דרך indirect



כפי שניתן לראות קודם כל ההודעות מעוברות על הhome agent.ורק לאחר מכן מועבר לforeign agent.זהו דבר שיכול להיות מאד לא יעיל כי לפעמים שניים יכולים להיות מאד קרובים אחד לשני אך בכל זאת התקשורת תעבור דרך הhome agent גם אם הוא הרבה יותר רחוק(חברות הסלולר בארץ משתמשות בשיטה זאת-עד כסף☺)

<u>Mobile IP</u>

משתמש בindirect routing.
השליחה מתבצעת ע"י כך שהחבילה המקורית מגיעה אל הhome agent והוא עוטף אותה בheader חדש עם היעד הנוכחי שלו.



מבנה החבילה

החלק הראשון של החבילה הוא כמו בICMP והוא משמש לdebug.
והחלק השני קשור לmobililty.ישנם ביטים שאומרים האם זה home agent/foreign agent,האם זו בקשה להירשם,באיזו שיטת קידוד עוטפים את החבילה ובאיזה כתובת הagent מטפל כרגע.

הניתוב

קודם כל מגיעים אל ה home location והוא מנתב את השיחה למיקום החדש.והswitching center מעביר תחנה מ base station אחת לשניה.

Handoff

המטרה:להעביר משתמשים מBS ישן לBS חדש.

התהליך:

1. הBS הישנה מודיעה לMSC שעומד להתרחש handoff.ומקצים מקום בתחנה החדשה
2. הMSC קובע דרך מעבר לBS החדש.
3. הBS החדש מגדיר תדר בשביל הנייד.
4. הBS החדש מודיע לBS הישן ולMSC שהוא מוכן!!
5. הBS הישן מודיע לנייד שיבצע handoff לBS החדשה.
6. הנייד מתקשר לתחנה החדשה.
7. הנייד דרך התחנה החדשה מודיע לMSC שהצליח לעבור ושומר את הנתונים
8. הMSC משחרר את הנתונים על הBS הישנה.

ולסיכום!!!!

GSM element	Comment on GSM element	Mobile IP element
Home system	Network to which the mobile user's permanent phone number belongs	Home network
Gateway Mobile Switching Center, or "home MSC". Home Location Register (HLR)	Home MSC: point of contact to obtain routable address of mobile user. HLR: database in home system containing permanent phone number, profile information, current location of mobile user, subscription information	Home agent
Visited System	Network other than home system where mobile user is currently residing	Visited network
Visited Mobile services Switching Center. Visitor Location Record (VLR)	Visited MSC: responsible for setting up calls to/from mobile nodes in cells associated with MSC. VLR: temporary database entry in visited system, containing subscription information for each visiting mobile user	Foreign agent
Mobile Station Roaming Number (MSRN), or "roaming number"	Routable address for telephone call segment between home MSC and visited MSC, visible to neither the mobile nor the correspondent.	Care-of-address