**סיכום רשתות תקשורת בנקודות**

* 5 השכבות לפי מודל TCP/IP ולפי מודל OSI
* מבנים שונים של רשתות
  + Hub במבנה Star בשכבה הפיזית.
* Headers ושם המידע שנשלח מכל שכבה
* Port בשכבת התעבורה
* כתובת IP בשכבת הרשת
* כתובת Mac בשכבת הערוץ
  + Switch בשכבת הערוץ הנעזר בכתובת הmac
    1. אלג', TTL, switch מודרני, התקפות, שקוף(לא כותבים את הmac שלו).
* תיאור תהליך השליחה. לשים❤: dest IP של מחשב היעד, mac של הראוטר. הראוטר רואה בשכבת הערוץ שהחבילה מיועדת אליו ולכן מעלה לשכבת הרשת, שם הוא רואה שהdest IP≠משלו ולכן מוריד חזרה לערוץ שמשנה את הdest Mac לתחנה הבאה ברשת בדרך ליעד. switchים מנתבים את החבילה למחשב המתאים בתוך הרשת.
* שכבת התעבורה
  + פרוטוקול UDP
    1. UDP Segment: src port, dest port, length, checksum, application data
    2. UDP Socket
  + פרוטוקול TCP
    1. MSS, MTU, Path MTU Discovery(PMTUD)
    2. TCP Segment: src port, dest port, header length, seq num, ack num, flags(ACK,SYN,RST,FIN) checksum, receive window(rwnd), application data
    3. TCP Socket
    4. Send buffer, Receive buffer, Seq number, Ack number
    5. חישוב Timeout לפי RTT. Store&Forward.
    6. Pipelining בניגוד לStop&Wait
       - Send Window:
         * can send: |send windows| - (LBS-LBA)
         * חשוב לדאוג שטווח הseq numbers יהיה גדול מגודל החלון.
       - התמודדות עם אובדן חבילות:
         * Go Back N (GBN)
         * Selective Repeat
         * שיטת TCP: כשמגיעה חבילה בסדר שגוי המקבל שומר בבאפר, אבל מחזיר ACK מצטבר של החבילה האחרונה שכן הייתה לפי הסדר.

תרחישי ACK מצטבר(בצד המקבל)

Fast Retransmit(בצד השולח)

* + 1. תהליך חיבור(SYN- 3 way handshake)
    2. תהליך ניתוק(FIN)
    3. Flow control- בקרת זרימה(המקבל לא עומד בעומס- לא קורא מספיק מהר מהבאפר)
       - המקבל מדווח לשולח את הreceive window(rwnd) שלו בheader, והשולח מתאים את הsend window שלו(זה אומר לו כמה בתים מותר לו לשלוח החל מהseq num הזה).
    4. Congestion control- בקרת עומס(הראוטרים בדרך עמוסים- התורים[הבאפרים] שלהם התמלאו)
       - Slow Start:
         * cwnd= כמה סגמנטים(MSSים) אפשר לשלוח לפני קבלת ACK, מתחיל מ1MSS.
         * בקבלת ACK: cwnd=old\_cwnd+1MSS, ונזיז את send window ב1. dupACKcount=0.  
           יוצא שבכל RTT, cwnd מכפיל את עצמו כי חוזרים כמה ACKים בזמן הזה
         * Slow Start Threshold(SSThresh): סף שהחל ממנו נעבור לCongestion Avoidance.
       - Congestion Avoidance:
         * Timeout- SSThresh=old\_cwnd/2, cwnd=1MSS, dupACKcount=0, Slow Start
         * 3 Duplicated ACKS

TCP Tahoe: כמו Timeout.

TCP Reno: SSThresh=cwnd=old\_cwnd/2, Congestion Avoidance

* + - * + בקבלת ACK: cwnd=old\_cwnd+MSS\*(MSS/cwnd), dupACKcount=0.
      * Fast Retransmit/Recovery(בקבלת 3 ACKים כפולים)
        + SSThresh=old\_cwnd/2, cwnd= old\_cwnd/2 +3 ולשדר מיד את החבילה החסרה.
        + בקבלת ACK כפול: cwnd=old\_cwnd+1MSS
        + בקבלת ACK תקין: cwnd=SSThresh, dupACKcount=0. חזרה לCongestion Avoidance.
* שכבת האפליקציה
  + פרוטוקול DHCP תחת UDP לקבלת IP ולמידת הרשת עבור מחשב(host) המתחבר אליה לראשונה.
    1. Discover(broadcast: dest IP 255.255…, dest mac FF.FF…),Offer,Request,Ack
    2. תחת UDP ולכן יש transaction ID שצריך להיות זהה בין Discover-Offer ובין Request-Ack.
  + פרוטוקול DNS תחת UDP למיפוי בין כתובות אינטרנט(=שמות=דומיינים) לכתובות הIP שלהן. פורט 53.
    1. היררכיית DNS- שרתי השמות (Root, Top Level Domains-TLDs, Authoratives)
    2. Zone file- קובץ מיפויים של כל שרת DNS(לא אצל resolver)
       - האצלת סמכויות(נתינת בעלות על דומיין כלשהו)
    3. DNS Resolver
       - Local DNS Resolver
       - סוגי שאילתות(רקורסיבית/איטרטיבית), cache.
    4. סוגי רשומות DNS: A,AAAA,NS,CNAME,MX,PTR,TXT
    5. מבנה שאילתת ותשובת DNS
    6. תרשים זרימה של בקשה. glue.
  + פרוטוקול HTTP תחת TCP המאפשר לדפדפנים להוריד מידע משרתי אינטרנט ולהציגו. פורט 80.
    1. מבנה כתובת URL
    2. גרסאות HTTP: Non-persistant(1.0), Persistant(1.1), Pipelining(disabled)
       - השפעה על גודל החלון
    3. בקשת HTTP ותשובה
       - קודי סטטוס
       - Cookies(שרת שולח set cookie:…, קליינט שולח cookie:…).
       - GET vs POST
       - שיטת cache:
         * last-modified בheader התשובה
         * if-modified-since בheader הבקשה(השרת יחזיר 304 אם לא השתנה, 200 ואת הקובץ אחרת).
  + Proxy
    1. מטרות:
       - cache
       - הפעלת לוגיקה(ניטור- חסימה של חבילות יוצאות/נכנסות כלשהן)
    2. סוגים:
       - פרוקסי רגיל- הדפדפן(תוכנת הקליינט) הוגדרה ידנית לעבוד מול הפרוקסי.
       - פרוקסי שקוף- כמו הרגיל, רק שאין הגדרה ידנית אצל הלקוח, זה שקוף לו.
       - פרוקסי הפוך- משרת שרתים. לדוג' cache עבור השרת להפחתת העומס.
  + אתגרים בגלישה באינטרנט: שרתים רחוקים- זמן=כסף, הורדה חוזרת של אותם משאבים, נקודת כשל מרכזית וזמינות תחת עומס. פתרונות:
    1. Replication(שכפול שרתים).   
       בעיות: נצילות- איך נחלק את העומס ביניהם? איך ליידע את הלקוחות לאן לפנות?
    2. DNS Redirection: שרת הDNS יפנה כל אחד מהלקוחות שפונים אליו לשרת אחר בהתאם לעומס.
    3. Load Balancing: רכיב המעביר את הבקשה לשרת המתאים(לפי השיקולים שלו- אותם לקוחות לאותם שרתים, לשרת הכי פחות עמוס, הכי פחות משתמשים...) ומחזיר בעצמו את התשובה ללקוח.
    4. CDN: שרתי קצה של פרוקסי שקוף
       - יתרונות:
         * למידע סטטי:

מחזיר תשובות לקליינטים מהcache שלו- מה שמפחית עומס על השרת הראשי ומזרז את קבלת התשובה ללקוח.

* + - * + למידע דינמי(ולסטטי):

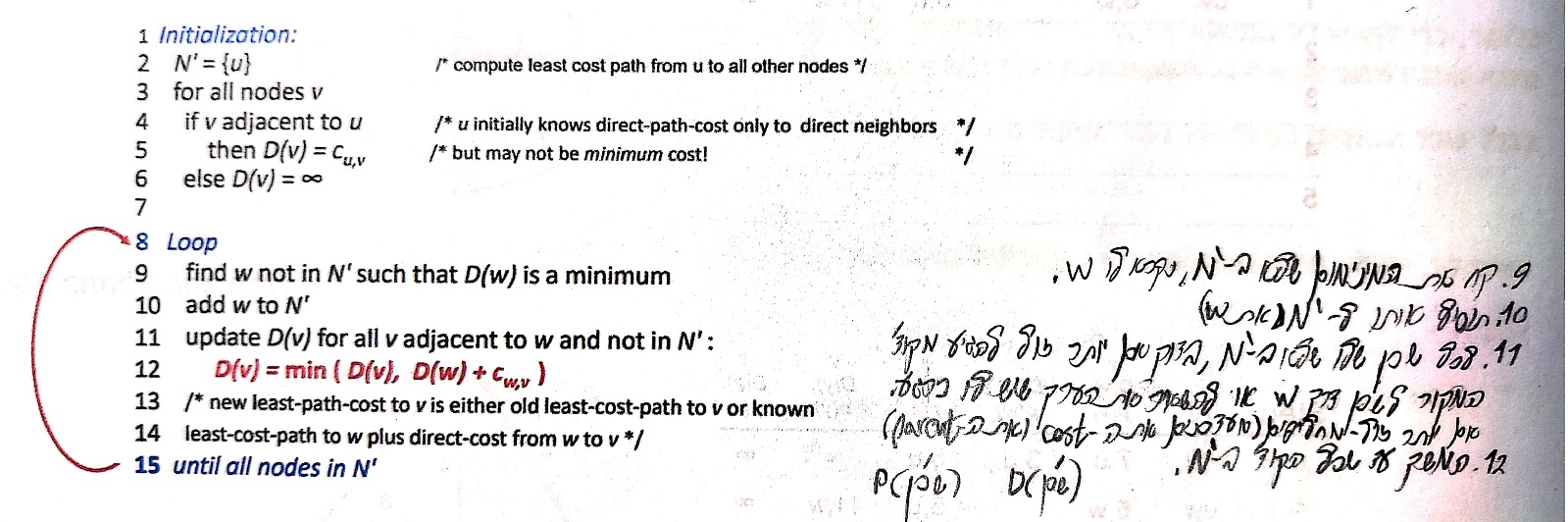
קרובים לספק האינטרנט. רוחב פס גבוה. יכולים לעשות אופט' לחיבור הTCP.

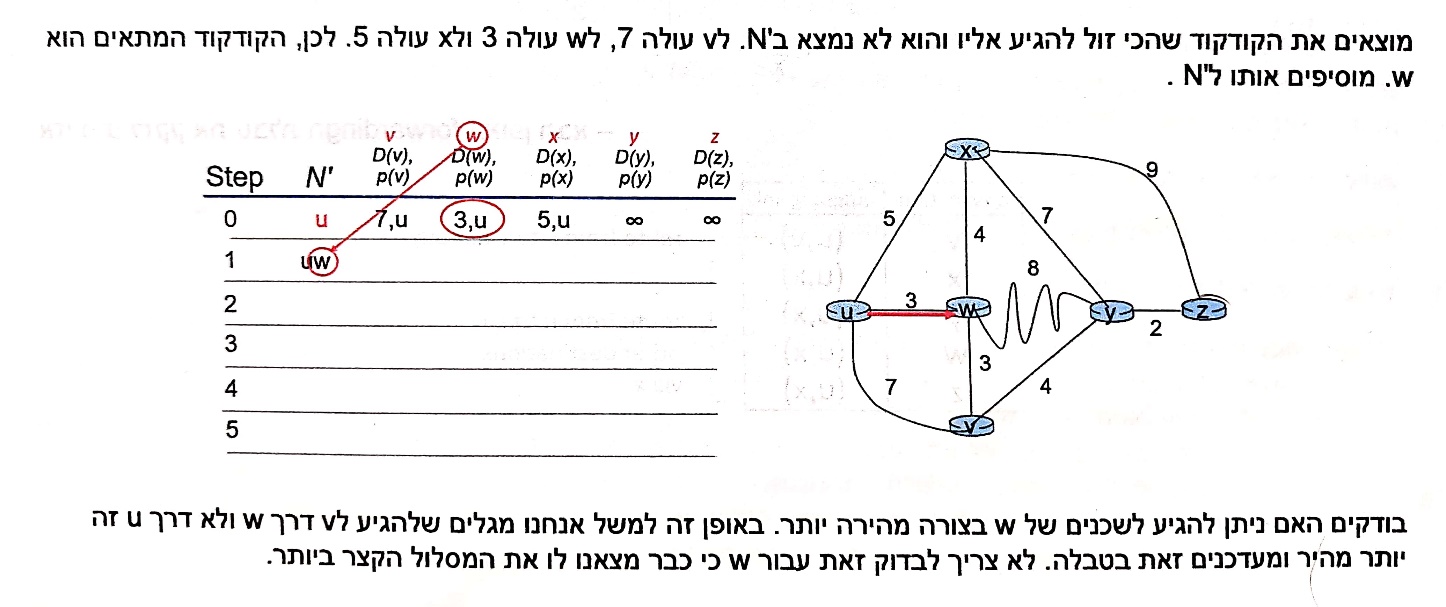
אם שרת CDN אחד קורס או עמוס אפשר להפנות לשרת CDN אחר ועדיין הלקוח יקבל תשובה.

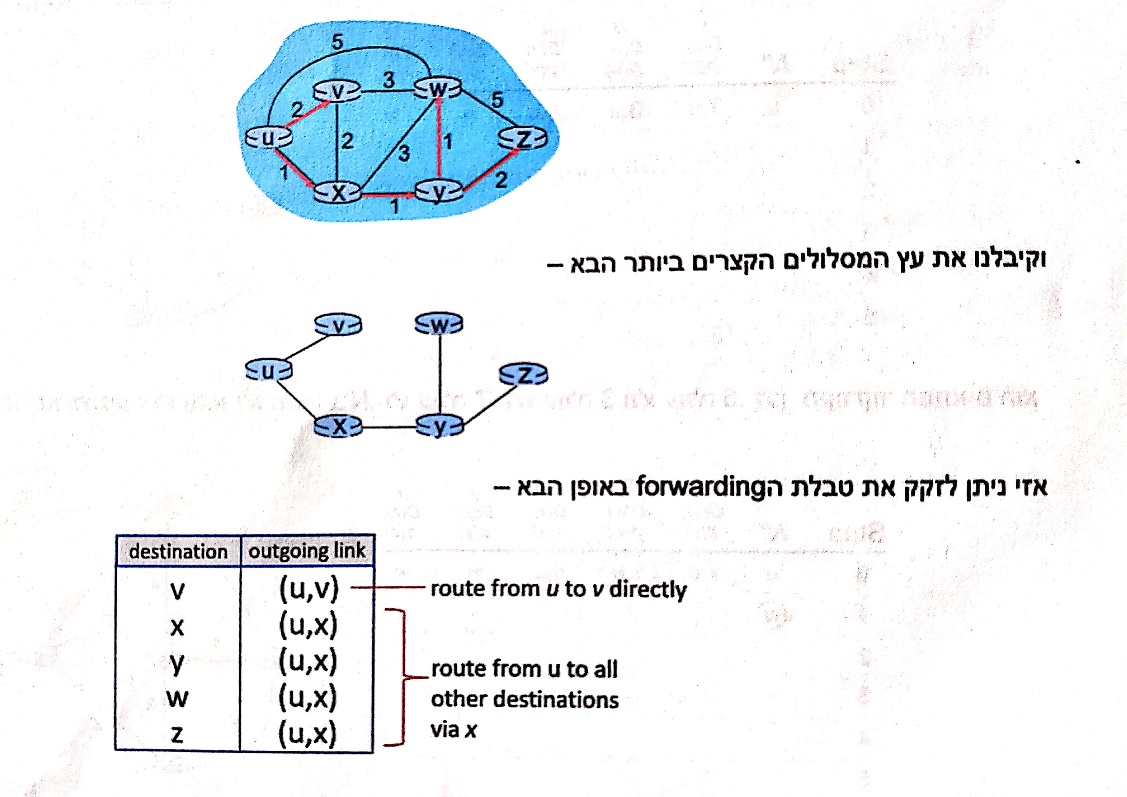
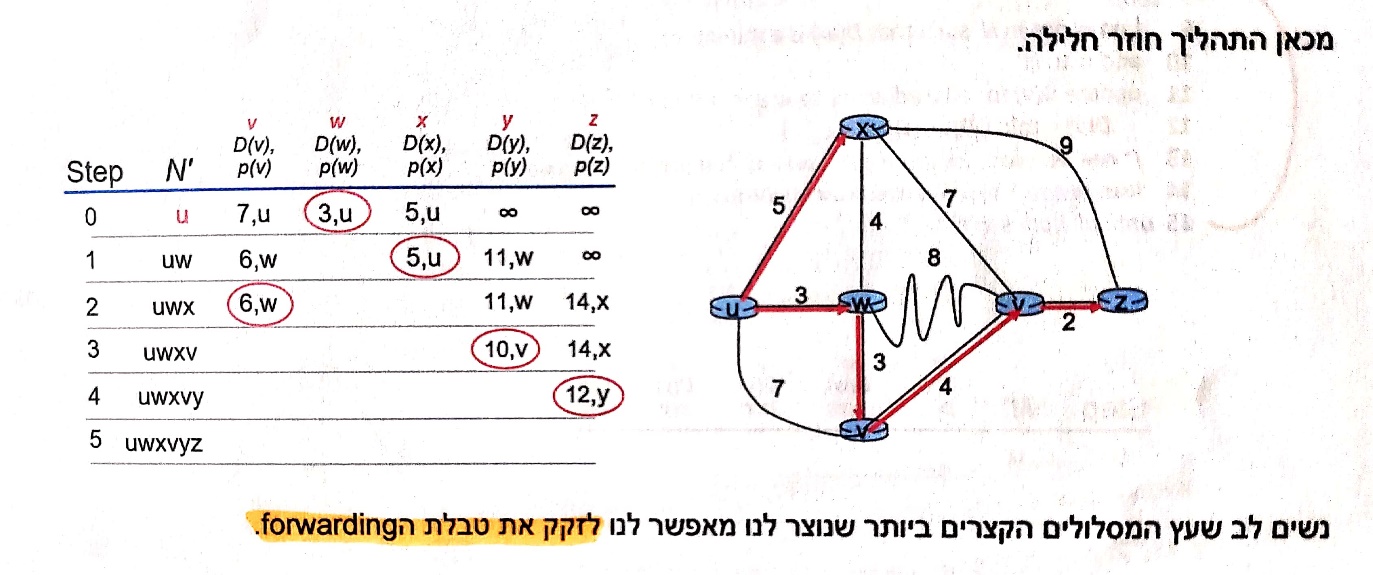
דוג': פיזור פרואקטיבי- שרת המקור מעדכן אקטיבית כשיש שינוי. אחרת אפשר להניח שהכל מעודכן ולהחזיר מידית תשובה מהcache.

* + פרוטוקול SMTP לשליחת מיילים. פורט 25.
    1. גילוי הכתובת ע"י DNS(רשומות MX).
    2. הקמת חיבור TCP
    3. שליחת המייל(SMTP בשכבת האפליקציה):
       - שלב המעטפה
       - שלב ההודעה, "." לסיום
    4. סגירת TCP
  + פרוטוקול POP3 לקבל מיילים(משיכה של מיילים מהmail server ע"י הuser agent)
    - שלב authorization: user, pass…
    - שלב transaction: list, retr x, dele x, quit.
  + פרוטוקול IMAP: משוכלל יותר- מאפשר עבודה עם מספר תיבות דואר. כל ההודעות נשמרות אצל השרת. בניגוד לPOP3 בכל Session כל משתמש נשמר המצב שלו(ההודעות והתיקיות).
* שכבת הרשת
  + Forwarding: העברת חבילה מכניסה אחת(ממשק) של הראוטר אל היציאה המתאימה.
    - Data/Forwarding Plane: מישור ההחלטות המקומיות של הראוטר.
    - Forwarding Table המיוצר ע"י אלגוריתם הrouting הנבחר.
  + Routing: ההחלטה על המסלול שחבילה צריכה לעבור מקצה לקצה.
    - Control Plane: מישור הבקרה- הלוגיקה של רמת הרשת בכללותה. החלטה על המסלול...
  + פרוטוקול IP
    - כתובת IP
      * IPv4 32-bit, IPv6 128-bit
      * Class-Based Addressing
      * Classless Inter-Domain Routing and Subnetting (CIDR)
        + Subnet Mask- כמות הביטים המשמשים לזיהוי הרשת.
        + חלוקה לתתי רשתות(Route Aggregation)
        + מציאת מזהה הרשת
        + שימוש בCIDR בforwarding table של כל ראוטר!

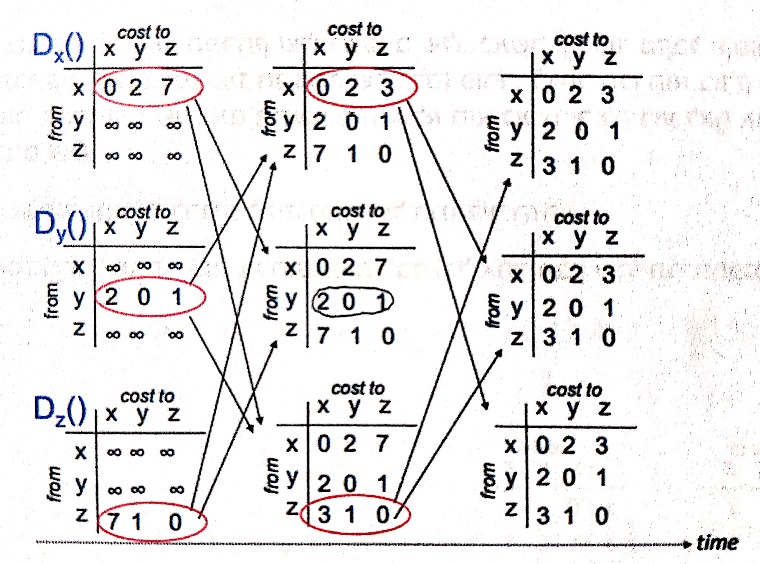
Longest Prefix Match: הראוטר יחפש בטבלה את מזהה תת הרשת הספציפי ביותר המתאים לdest IP של חבילה שמתקבלת, וישלח את החבילה לממשק(לערוץ) הממופה לרשת הזו בטבלה.

* + - * כתובות שמורות: ראשונה מזהה הרשת, אחרונה כתובת broadcast לכל המחשבים ברשת.
      * טווחים שמורים. כתובות מיוחדות.
    - IP Segment/Datagram
* תערובת שכבת רשת+תעבורה
  + פרוטוקול ICMP להודעות בקרה של האינטרנט מעל שכבת הרשת ומתחת לשכבת התעבורה!
    - Ping(echo request, echo reply)
    - Traceroute- שליחת 3 הודעות בכל פעם, תחת UDP לפורט שסביר שלא בשימוש, ותחת IP עם TTL=1,2,3…. אם הוא מקבל port unreachable הוא יודע שהוא הגיע ליעד ומדפיס את המסלול.
  + Network Address Translation- NAT: שירות המוצע ע"י הראוטר בשכבת התעבורה עבור שכבת הרשת לחסכון בכתובות IPv4.
    - הראוטר משתמש בכתובת IP פומבית אחת לייצוג כל המחשבים ברשת
      * כשחבילה נשלחת ממחשב פנימי ברשת(עם IP פנימי) החוצה(דרך הראוטר כי הוא רשום כdest mac) הוא מחליף את הsrc IP שלה להיות הכתובת הפומבית שהוקצתה לו, ואת הsrc Port למס' כלשהו המזהה את המחשב השולח בטבלת NAT Translation Table.
      * חבילה שנשלחת לרשת תישלח לכתובת הפומבית שהוקצתה לראוטר (כי העולם חושב שזו הכתובת של המחשב ברשת, האינטרנט לא יודע על ההמרה שבוצעה). הראוטר יחפש את ההמרה המתאימה בטבלה בין הIP והפורט שאליו החבילה נשלחה(WAN) לבין הIP והפורט האמיתיים של המחשב הפנימי ברשת(LAN).
    - סוגי NAT:
      * Full Cone NAT: כל מחשב מחוץ לרשת יכול לשלוח הודעה למחשב הספציפי ברשת ע"י כתיבת הdest IP וdest Port המייצגים אותו בטבלת הNAT.
      * Restricted Cone NAT: כנ"ל אבל יתקבלו רק הודעות ממחשבים שהמחשב הפנימי שלח להם הודעה בעבר.
      * Port Restricted Cone NAT: כנ"ל אבל יתקבלו רק הודעות מאותו פורט שהמחשב הפנימי שלח אליו הודעה בעבר במחשבים אלה.
      * Symmetric NAT: כנ"ל אבל מוסיף מיפוי שונה בטבלה כשמחשב פנימי שולח הודעה לdest IP או dest Port שונה.
    - NAT Traversal problem
      * ארכיטקטורות תקשורת
        + לקוח-שרת: שרת מאזין לפורט וממתין לשרת לקוחות, שהם בדר"כ יוזמי הקשר.
        + Peer2Peer: לקוח מדבר ישירות עם לקוח, והם מתחלפים בתפקיד השרת והלקוח.
        + Hybrid: משולב- Peer2Peer אבל שרתים יחברו ביניהם(יגידו מי פעיל, איפה נמצאים הקבצים המבוקשים ומה כתובת הIP של הpeer הרלוונטי).
      * הבעיה: לקוח לא יכול לפנות ישירות ללקוח כי הוא לא יודע את כתובת הIP שלו, וגם אם הוא היה יודע – ללקוח יש לפעמים restricted NAT.
      * פתרונות:
        + Port Forwarding- נוסיף מיפוי סטטי (שורה בטבלת הNAT) בין IP ופורט מסויים באינטרנט שנרצה שיכנס לבין IP ופורט מסויים ברשת הפנימית.
        + כל התקשורת ביניהם תתבצע דרך מתווך.
        + Hole Punching
  + Firewall- יכול להיות ממומש בשכבת הרשת או התעבורה או האפליקציה, על מחשב או על ראוטר. מסנן חבילות לפי חוקים מסויימים(לדוג' לפי src IP או dest IP או חבילות עם פרוטוקול מסויים).
* המשך שכבת הרשת
  + בניית הforwarding table אצל כל ראוטר
    - פרוטוקולים ממשפחת link-state(דורש ידע על כל העולם- איזה ראוטרים יש, איזה קשתות יש ביניהם, מה עלות הקשתות האלה...) עבור רשתות יותר סטטיות.
      * פרוטוקול Open Shortest Path First (OSPF) ממשפחת link-state
        + בתוך כלAS(Autonomous system) , כל קודקוד(נתב) יפרסם את הנתיבים לכל הנתבים שהוא מכיר לכל שאר הנתבים בAS. זה נקרא link-state broadcast.
        + כל נתב יריץ **אלג' דייקסטרה** כדי לחשב את טבלת הforwarding שלו(ימצא את המסלול הקצר ביותר בינו[הוא קוד' המקור בגרף שעליו הוא מריץ] לבין כל קודקוד אחר בגרף).
        + נחזור שוב ל1. נמשיך כך ואחרי k איטרציות נדע את העלות המינימלית של המסלול לk יעדים.





* + - * בעיה: Route Oscillations- תנודות ברשת. יכול להיווצר מצב של מסלול מאוד עמוס ובמקביל מסלול ריק לגמרי. זה קורה בגלל שכל ראוטר מקבל החלטות בלי להתחשב בהחלטות שהראוטרים האחרים יעשו אלא רק על סמך נתוני העבר.
        + פתרון ספציפי: חלוקה שונה של המידע בין המסלולים
        + פתרון כללי: כל פעם נעצור נתב כלשהו שיחשב מחדש את הטבלה שלו ובכך יתחשב בהחלטות של האחרים עד כה ולא רק במידע הראשוני.
    - פרוטוקולים ממשפחת Distance-Vector(דורש שכל קודקוד ידע את החיבורים הפיזיים שלהם לשכנים שלהם ומה העלות להגיע לכל שכן, וכמה[לא איך] עולה לשכנים להגיע לכל קודקוד ברשת) עבור רשתות יותר דינמיות
      * האלגוריתם:
        + כל כמה זמן נתב יפרסם את הdistance vector שלו לשכניו.
        + כל נתב המקבל distance vector יחשב את הdistance vector שלו מחדש ע"י **אלג' בלמן-פורד**(המסלול הזול ביותר להגיע מכל קודקוד לכל קודקוד הוא המינ' מבין העלות להגיע לשכן + העלות מהשכן ליעד, לכל שכן).

אם הוקטור שלו השתנה- מפרסם את הdistance vector החדש שלו לשכניו.

* + - * חדשות טובות מתפשטות מהר.
      * חדשות רעות מתפשטות לאט.
        + Count to infinity
        + פתרון: Poison Reverse- אם X מגיע לZ דרך Y, הוא ישקר ויספר לY שהוא מגיע לZ בעלות ∞.
      * פרוטוקול RIP: פרוטוקול ניתוב המתבסס על ספירת צעדים כדי להחליט על המסלול.
        + כל 30 שניות כל אחד מהקודקודים שולח את ווקטור המרחקים שלו לשכניו גם אם הוא לא השתנה כדי שהשכנים ידעו שהקודקוד עדיין קיים.
        + אם עברו 180 שניות מהפעם האחרונה שהשכנים שמעו מהקודקוד הם מניחים שהקודקוד כבר לא קיים ואי אפשר להעביר אליו חבילות. כתוצאה מכך הם יצטרכו לחשב מחדש את ווקטור המרחקים שלהם.
        + אין משקלים לקשתות, כל קשת היא במשקל 1.  
          בRIP לא נאפשר יותר מ15 קפיצות ולכן בpoison reverse ערך של ∞ הוא 16.
  + פרוטוקול BGP להגעה מAS אחד לשני. פרוטוקול ממשפחת path-vector(כל קודקוד אומר לשכניו את וקטור המסלולים שלו[לא המרחקים]- איזה ASים החבילה תעבור בדרך ליעד מסויים)
    - נבחר מסלול אחד ויחיד לפי מדיניות כלשהי.
    - אם מקבלים פרסום מלקוח שלנו מעבירים אותו הלאה לכולם (ללקוחות, לספקים, לעמיתים) אבל אם מקבלים פרסום מעמית או מספק שלנו מעבירים אותו הלאה רק ללקוחות.
    - סדר עדיפויות בבחירת המסלולים:

1. מסלולים דרך לקוחות כי הם הכי משתלמים כלכלית (הלקוחות משלמים לנו).
2. אחרת, מעדיפים מסלולים דרך עמיתים כי לפחות זה לא עולה לנו כסף.
3. אחרת, משתמשים במסלולים דרך ספקים שעולה לנו כסף.
4. מסלולים שהAS path שלהם קצר יותר

* שכבת הערוץ
  + פרוטוקול Address Resolution Protocol - ARP למציאת כתובת MAC בהינתן IP.
    - לכל מחשב יש טבלת ARP הממפה בין כתובת IP לMAC עם TTL כלשהו
    - כשמחשב רוצה כתובת MAC של מחשב שלא נמצא בטבלת הARP שלו:

1. המחשב המחפש שולח הודעת ARP בbroadcast(IP של מחשב היעד וMAC של FF.FF…).
2. ההודעה מגיעה לswitch כלשהו שמעביר אותה לכל מי שברשת.
3. מי שאינו מחשב היעד יראה שההודעה אינה מיועדת אליו כי הdest IP שונה מהIP שלו ולכן יזרוק אותה. אבל מחשב היעד יראה שמדברים אליו ויחזיר תשובת ARP (ביוניקאסט) למחפש.
4. כשהמחפש יקבל את התשובה מהיעד הוא יוסיף מיפוי בין כתובת הIP של היעד לבין כתובת הMAC שממנה נשלחה ההודעה(כתובת הMAC של היעד) לטבלת הARP שלו, יחד עם TTL כלשהו שכשהוא יעבור הוא יצטרך לחפש את היעד מחדש באותו התהליך.
   * Shared wire.
     + פרוטוקולים של חלוקת הערוץ בין המחשבים:
       - פרוטוקול Time Division Multiple Access – TDMA. חלוקה לslotים של זמן. לכל מחשב מוקצה slot בסבב גם אם הוא לא צריך לשדר.
       - פרוטוקול Frequency Division Multiple Access – FDMA. דומה אבל לכל מחשב יוקצה תדר(לא slot של זמן).
     + פרוטוקולים של גישה אקראית לערוץ:
       - פרוטוקול Slotted ALOHA.
         * הנחות: גודל frameים זהה, שידור frame רק בתחילת slot, יש סנכרון, אפשר לזהות אם הייתה התנגשות.
         * מחשב שרוצה לשדר משדר. אם הייתה התנגשות הוא ינסה לשדר שוב בslot הקרוב בהסתברות p.
         * יעילות עבור p אופטימלי 37%.
       - פרוטוקול Pure/Unslotted ALOHA
         * דומה לנ"ל, אבל ללא slots. כל אחד יכול להתחיל לשדר מתי שהוא רוצה.
         * יעילות עבור p אופטימלי 18%.
       - פרוטוקול Carrier Sense Multiple Access – CSMA
         * מקשיבים לפני שמתחילים לשדר. אם מישהו אחר משדר- מחכים.
       - פרוטוקול CSMA/CD, כלומר CSMA עם Collision Detection
         * מקשיבים לפני שמתחילים לשדר אבל גם במהלך ההתפשטות של השידור. אם מזהים התנגשות בזמן השידור- מפסיקים לשדר.
         * אלגוריתם Ethernet CSMA/CD, כלומר CSMA/CD בשכבת הערוץ:

כנ"ל, כמה זמן נחכה במקרה של התנגשות? בהתנגשות ה-m נטיל מטבע בין 0 ל2m-1, ואת התוצאה נכפיל ב bit times512 וזה הזמן שהוא יחכה.

* חישובי זמנים
  + השהיית שידור
  + השהיית התפשטות
  + השהיית תור
  + השהיית עיבוד
  + Pipelining במקום Stop&Wait(אבל עדיין עושים Store&Forward)
    - הערוץ האיטי ביותר מכתיב את הקצב: סך ההשהיות לחבילה הראשונה עד שהיא מגיעה לערוץ האיטי ביותר + סך ההשהיות באיטי ביותר + סך ההשהיות לחבילה האחרונה עד להגעה ליעד.
  + תחיליות- Headers
  + פרגמנטציה: FragFlag, offset
  + Silly Window Syndrome ופתרונותיו