



תקשורת ומחשבות

מחברת סיכומים מההרצאות של ד"ר דביר עמיית זאב

תוכן עניינים

3	קדמה
5	ליבת הרשת
6	מבנה האינטרנט
6	ה摔יות
7	אבטחת הרשת
8	כתובת IP
9	מודול ה- ISO
9	מחסנית פרוטוקול האינטרנט
10	שכבה האפליקציה
10	תפקידו השכבה
10	פרוטוקולי השכבה
10	תהליכי התקשורת ((Processes communicating))
11	Socket
12	פרוטוקולים בשכבה הרשת ((An application-layer protocol))
12	דרישות האפליקציה משירות התעבורה
12	סוגי שירותים בשכבה התעבורה
13	HTTP: פרוטוקול שכבה האפליקציה של האינטרנט
14	HTTP Request Messages
14	HTTP response status code
14	HTTP Header
14	Proxy
15	גרסאות ה-HTTP
16	DNS: Domain Name System
17	Local DNS name servers
18	DNS Security
18	- תכני מולטימדיה Video Streaming and CNDs
19	DASH : Dynamic, Adaptive Streaming over HTTP
19	CDNs
20	שכבת התעבורה
20	Multiplexing and Demultiplexing
21	UDP: User Datagram Protocol

21	UDP Segment Structure
22	UDP Checksum
22	העקרונות של העברת מידע בצורה אמינה
26	TCP: Transmission Control Protocol
26	TCP Segment Structure
27	TCP Sequence numbers and ACKs
27	TCP round trip time, timeout
28	TCP ACK generations rules
28	TCP Fast Retransmit
29	TCP Flow Control
29	TCP Connection Management
31	SYN Flooding Attack
32	יצירת Socket בשפת C
42	Principles of Congestion Control
46	TCP Congestion Control
48	שכבות הרשת
49	ארכיטקטורת הנטב
49	Input port functions
50	Forwarding Table
50	Switching Fabrics
52	Input Port Queuing
52	Output Port Queuing
54	IP Datagram format
54	DHCP
55	ICANN
55	NAT
56	IPv6
58	Routing-Protocols
58	DIJKSTRA Algorithm
60	Bellman-Ford Algorithm
61	Distance Vector Algorithm
62	Autonomous Systems (AS)
62	BGP : Border Gateway Protocol
63	ICMP: Internet Control Message Protocol

הקדמה

האינטרנט (בעברית: **מִרְשַׁתָּה**) היא רשת תקשורת נתונים בעל היקף כלל עולמי. הרשת נוצרה כתוצאה מהיבורים רבים בין רשותות מחשבים - חיבורים אשר איפשרו תקשורת בין מחשבים רבים ברשותות רבות. היקף הרשת, כמוות המידע העצומה האגורה בה, והפעולות הרבה שמתבצעת הוודות לה - הפכו את האינטרנט לגורם רב משמעות וລיזרת ההשפעות הכלכלית והתרבותית. ישנים מליארדי מכשירים המוחברים לרשת:



מארח (host) - הוא ציוד אשר שלוח ומתקבל הודעות ישירות דרך הרשת – לצד זה יש כתובות MAC וכותבות IP. למשל מחשבים, מדפסות, מצלמות רשת, סורק, שירותים וכו... הם הקצוות או "המסגרת" באior והם מה שאנו מכירים ביום יום.

כתובת IP - היא מספר המשמש לזיהוי נקודות קצה, כגון מחשב, ברשות תקשורת שבנה משתמשים ב프וטוקול התקשרות IP, כגון רשת האינטרנט.

אם נכנס פנימה נראה את ה- **Packet switches** (switches) ונתבים (routers). שם נתבים (routers) ונתגים (switches).

חבילה (Packet) - היא אוסף נתונים אליו מצורפים כתובות המקור, כתובות היעד ונתוני בקרה, כך שהרשת תוכל לנתח את החבילה לעד הנכון.

נתבים (Routers) - נתבים משמשים לניטוב נתונים מרשת אחת לרשת אחרת. לדוגמה, מרשת האינטרנט, לרשת הפרטית של המשתמש הביתי.

נתגים (Switches) - הוא רכיב ברשת מחשבים המחבר בין צמתים שונים ברשת, בין אם הם מכשירי קצה (כגון מחשבים) ובין אם הם מרכיבי רשת בסיסיים.

אבל איך כל העסק הזה מתחבר ביחד? –

בעזרת **קווי תקשורת (Communication links)**, (Communication links), חוטיים ואל-חוטיים.

כמו למשל fiber, copper, wifi, או wifi, לוויינים, סלולר ורדיו. כਮון לכל אחד מהם יש קצב שידור (bit rate) שהוא אשר נמדד במידת רוחב פס (Bandwidth) ככלומר כמהו הנתונים שנitin להעביר בקו תקשורת בזמן נתון, רוחב פס נמדד בצורת - (bps - bits per second).

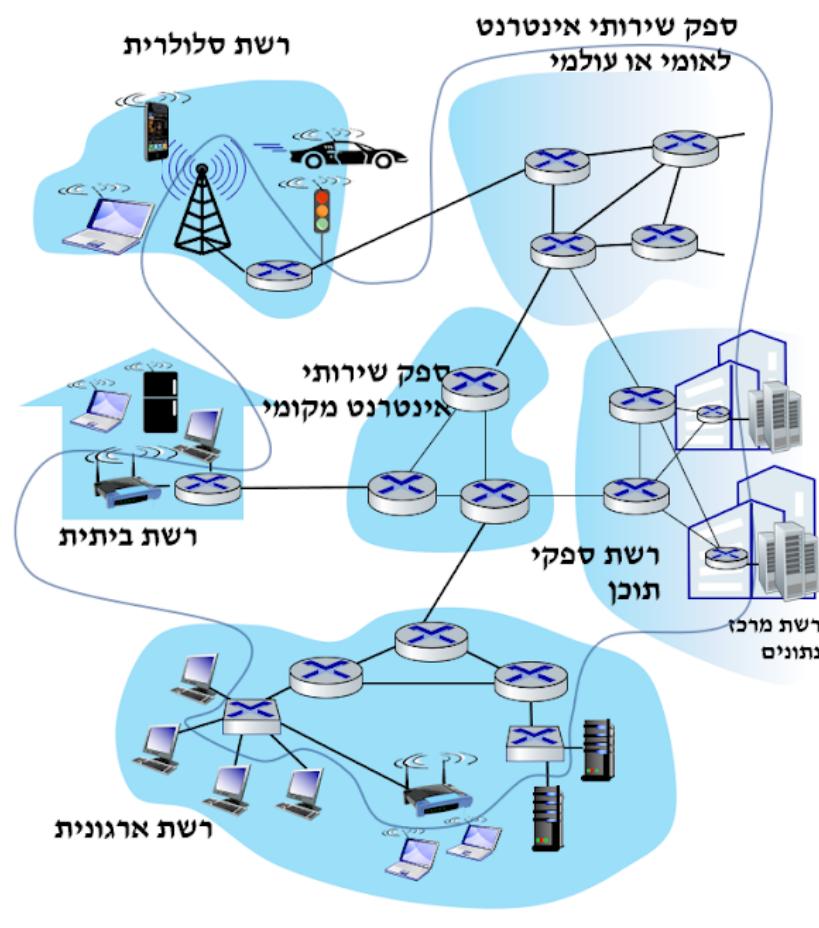
ኖזרים רשותות המיעודות לגופים שונים (Networks), כמו למשל בזק בינלאומי - היא רשת עם הלוקחות שלה, האוניברסיטה היא רשת.



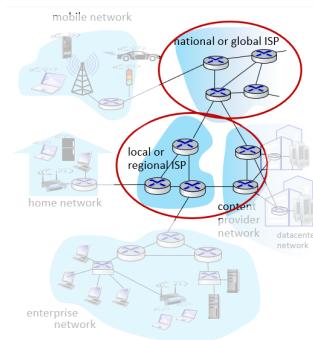
האינטרנט הוא בעצם "רשת של רשתות" הכולמת שירותי האינטרנט (ISPs) מחוברים זה לזה.

ספק האינטרנט (ISP) - הוא הגוף המאפשר למשתמש להתחבר ל프וטוקול IP/TCP המקשר בין רשת האינטרנט העולמית. כמו למשל בזק בינלאומי, פרטנר ו-HOT. ניתן להסתכל על האינטרנט בצורה של "תשתיות של אפליקציות".

הצורה שבה כל השירותים יודעים "לדבר" אחד עם השני וכל הגוף ברשות מתקשרים הוא **פרוטוקול תקשורת (Protocols)** - תפקido לייצור שפה משותפת בין גופי רשות, והוא שולט בקבלה ושליחה של הודעות. כמו למשל HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet :



ליבת הרשת



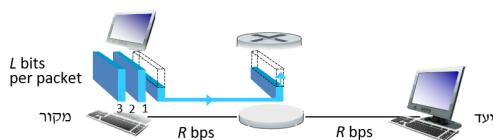
בליבת הרשת יש רשת של נתבים המחברים זה לזה. המשמש מחלק את המידע לחבלים (חבילות) שעוברות לנット והנתב מעביר את החבילות בכמויות רוחב הפס היכי גדולה שהוא יכול לאפשר שהחבילות אמוריה להגיע.

אחריות הנתב בקבלת-ה- **Packet-Switching** לאחר קבלתו מהמשמש:

תהליך זמן העברה עבור כל חבילה מתבצע כך:

$$L = 10 \text{ Kbits}, R = 100 \text{ Mbps}, \text{one-hop transmission delay} = 0.1 \text{ msec}$$

.1 - כמה זמן לוקח לנו לשדר לתוכן. L/R



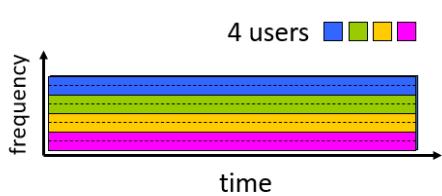
.2 - פעולת הנתב או המטג, הוא ממחה שכל החבילות הגיע אליו ורק אז הוא קיבל החלטה לאן להעביר את המידע.

.3 - הגע אל היעד, לפחות R/L .

- תור ואובדן של חבילות: אם קצב ההגעה (בקצב S bps) לקישור עולה על קצב ההעברה (bps) של הקישור, החבילות יעדמו בתור ויחכו להעברתן בקשרו היציאה, חבילות יכולות להאביד אם הזיכרון (buffer) בנתב (router) מתמלא עד אפס מקום. (כלומר אם התור מלא).
- יתרון-ה- **Packet-Switching** הוא קבלת שירות לפי דרישת הצורן וחיסירון שהוא לאאפשרלי לשמרם משאבים (אוں הבטחה לרוחב פס קבוע ולהשוויה מסוימת - כי אני לא יודע כמה משאבים יכנסו אליו).

:Circuit-Switching היא חלופה ל- **Packet-Switching** ופחות פופולרית:

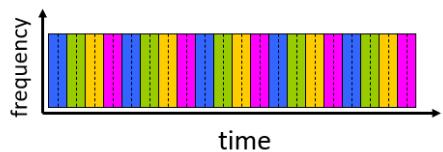
הוא יכול לשמר משאבים, ככלומר שומר הקצאות עברו משאב אחד ויחיד.



יש 2 שיטות שפועלות באמצעות :

:Frequency Division Multiplexing - FDM - חלק מרוחב הפס מוקצה להעברת מידע.

כמו שכל תחנת רדיו משדרת בתדר שלה, היא לא מפריעה לאף אחד, היתרון שככל אחד יכול לשדר מתי שהוא רוצה ואייר שהוא רוצה.

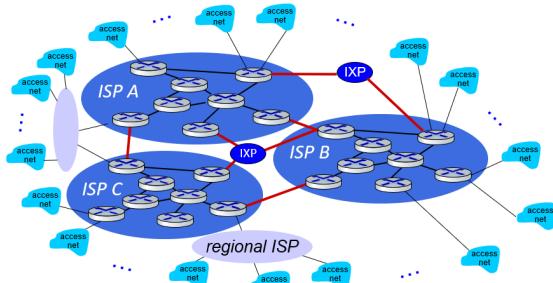
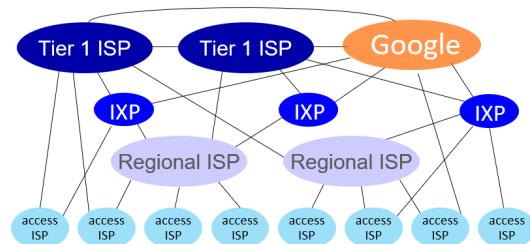


:Time Division Multiplexing - TDM - רוחב הפס מתחולק במספר המשתמשים כך שבכל פעימה כל רוחב הפס מוקצה עבור משתמש אחד.

מבנה האינטרנט

אנחנו המארחים, מתחברים לאינטרנט באמצעות ספקי שירותי אינטרנט (ISP).

- ISPs חיברים להיות מחוברים זה לזה כך שבין שני מארחים יהיה את האפשרות לשלווחabilities ביניהם - אפשר להבין מכך שמבנה האינטרנט מורכב מאוד. הינו רוצים שייהי את החיבור בין כולם, אך ישנו בעיות פוליטיות וכלכליות וכך' לחיבורים אלה.



עם הצלחת האינטרנט כמו מלא ISPs וכמה השאלה איך נחבר בין ספקי השירות האלה כדי לאפשר העברת Chbilut בין מארח מ-A ISP למארח מ-B ISP וכן נוצר ה- **IXP** - Internet Exchange Point שתפקידו לחבר בין הספקים.

עם הזמן כמו ISP אזריים שתפוקידם לחבר כמה access net מאותו איזור אל ה-ISPs הגדולים.

וגם כמו עוד ISP גדולים כמו גוגל יתרכן רב בניהול ISP משליהם לטובה שירות גלישה טובה יותר ומהירה יותר וכמו כן גם רוחית יותר.

השהיות

הלקוח משדר חבילה ↗

↖ **השהיית זמן שידור** - ממשך הזמן הדרוש לדחיקת כל הביטים של החבילה לחוט.

↖ **השהיית התפשטות** - מנהל כמה זמן לוקח לנו לשדר מיציאה החוט עד לנtab שלנו, המעביר מתבצע כך

שכל ביש בודד מהחבילה עובר עד שכולם מגיעים לנtab. ↗

↖ **השהיית עיבוד** - כשהחבילה מגיעה כמו צורך מהלקוח או מנtab אחר אל נתב (router) כלשהו, תפקידו לוודא שהחבילה הגיע כמו שצריך ואין שום תקלת במעבר מהלקוח עד אליו וזה הוא צריך להחליט לאן להעביר את החבילה. הוא בודק שגיאות ביתים, מחשב יציאה לקישור ובדרך"כ דורך זמן גדול מהרגיל. ↗

↖ **השהיית המתנה בתור** - אחרי שהנתב החליט לאן להעביר את החבילה, הוא מעביר את החבילה לリンק המתאים ובמצב זה יכול להיות תור, זמן המתנה משתנה בהתאם לעומסים.

אם החבילה הגיעה לתור מלא היא נזרקת ממש ויש מקרים שהיא נשלחת חוזרת לנtab ורק התחלה חוזר על עצמו ↗ או שהוא לא חוזרת ללוקוח בכלל ופושט נאבדת.

כדי לראות בעניינים את המסלול זהה, יש תוכנה שנקראת **traceroute** שנייה להפעלה בcmd, הוא מודד את העיקוב מהמקור אל הנתב לאורך האינטרנט עד ליעד.

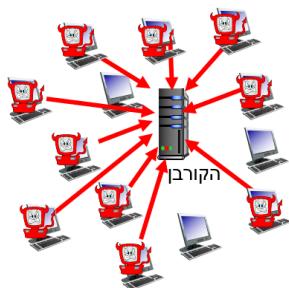
אבטחת הרשת

כשבנו את האינטרנט לא באמת לקחו בחשבון את עניין האבטחה. יש כל מיני סוגים של malware (תוכנות שנוצרו בכוונת תקיפה למחשבים), אם זה וירוס שצריך להפעיל אותו או אם זה בוטים שמשתלטים על המחשבים שלנו ללא ידיעתנו ומנצלים אותם לצורך תקיפה משותפת עבור גוף תקשורת מסוים.

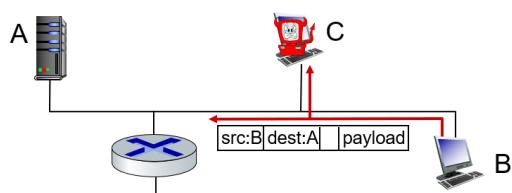
המיועד לה. Denial of Service - DoS - התקפה שנועדה לכבות מחשב או רשת, מה שהופך אותה לנגישה למשתמשים

- Distributed Denial of Service - DDoS

כמויות גדולות של נתונים מציפים את הקורבן ממוקורות רבים ושונים (אנחנו המקורות השונים ואנחנו לא מודעים לזה). מצד המגן - קשה מאוד לעצור את ההתקפה כי היא מגיעה מכמות גדולה של מקורות.



Packet Sniffing - כמו למשל שימוש בwireshark שאפשר להקליט את המידע של התעבורה, עם המידע הזה אנו יכולים לשכפל אותה ולהגיד שאנחנו בכל מחשב אחר B למשל, אני רוצה להציג לשרת ואני לא רוצה שיעלו עלי אז אני אגרום להם לחשב שהמקור הוא B ולא אני C - זה נקרא **IP Spoofing**.



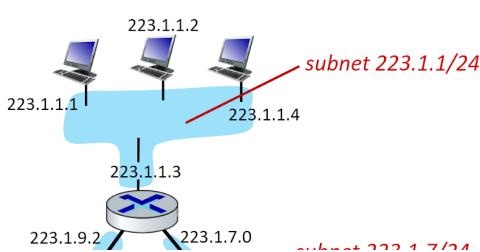
כתובת IP

כתובת פרוטוקול אינטרנט (כתובת IP) היא תווית מספרית המוקזית לכל מכשיר המחבר לרשת מחשבים שמשתמש בפרוטוקול האינטרנט לתקשורת. כתובות IP משרתות שתי פונקציות עיקריות: זיהוי משך מארח או רשות וכותבת המיקום שלה. תפקידו לחבר בין מארח / נתב לקבל פיזי. לרוב הנתבים יש כמה חיבורים כאלה. למארח בדרך כלל יש אחד או 2 חיבורים כמו אינטרנט חוצי או אינטרנט אלחוטי וכו'...

Subnets - חלוקה הגיונית של רשת IP הנוהג לחלק רשת לשתי רשותות או יותר.

זה תחת-רשת שלא עוברת דרך רואטור, כלומר בין מחשבים באותו התת-רשת אין להם צורך להשתמש בראוטר

כדי לתקשר אחד עם השני. כלומר אם אני מחובר לאינטרנט הביתי עם המחשב ואני שולח הודעה לאדם נוסף בבית שגם הוא מחובר באותו האינטרנט (אותו רואטור) אז אנחנו נתקשר תחת אותו subnet.



אם נסתכל בציור מימין נראה כי הסביבה הכחולה זה תת-רשת שלנו שיצא מהראוטר ומהחברים אליו 3 מחשבים, במקרה זה כל מחשב ייחסיק את אותה כתובות בעלת 1.1 אבל המיקום האחרון בכתובת שייכת למחשב ספציפי כלומר ייחודי אליו, כלומר 233.1.1/24 אפשר 24 כתובות שונות בתת הרשת הזאת (24 מחשבים).

subnet-mask - מסכה שנוחנת את הכתובת של הרשת בה נמצא המחשב

network-classes - ארכיטקטורת כתובות רשת מחלקת את שטח הכתובות עבור פרוטוקול האינטרנט גרסה 4 (IPv4)

לHASH מחלקות כתובות. כל מחלוקת, המקודדת בארכיטקטורה הביטים הראשונים של הכתובת, מגדרה גודל רשת שונה, כלומר מספר מארחים עבור כתובות (מחלקות C, B, A), או רשת ריבוי שידורים (מחלקה D). טווח הכתובות החמישי (E) שומר למטרות עתידיות או ניסיוניות. ידעת שיעורי רשת הופכת לבעה כשאתה מתחמודד עם ניתוב.

subnet part	host part
11001000	00010111
00010000	00000000

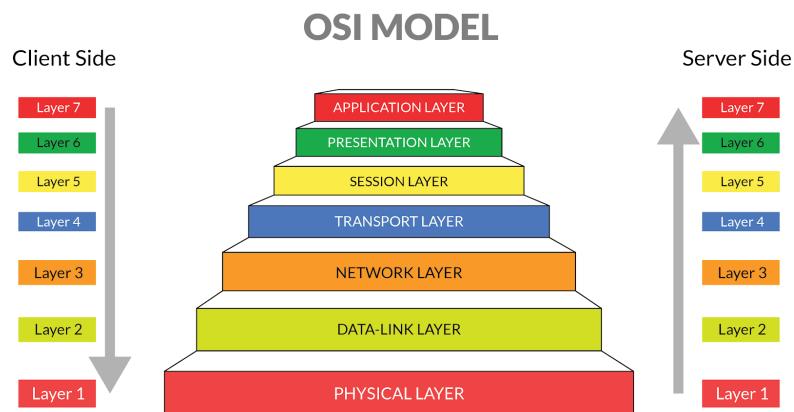
CIDR - Classless InterDomain Routing •

DHCP - Dynamic Host Configuration Protocol •

כלומר לקבלת כתובות באופן דינמי מהשרת.

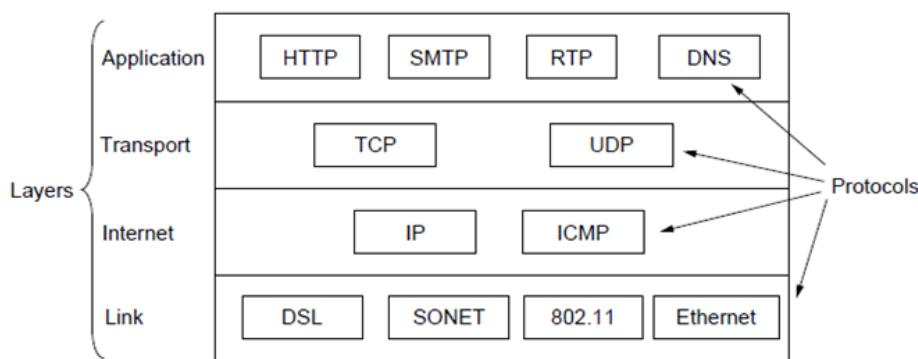
מודל ה- OSI

- **שכבה ה- Application** אחראית על נתינת שירות תקשורת למשתמש.
- **שכבה ה- Presentation** אחראית על הכנת המידע באמצעות תרגום, הצפנה ועוד' על מנת שנוכל לשולח או לקבל את המידע בפורמט המתאים.
- **שכבה ה- Session** אחראית לשילוח בחיבור וסנכרון.
- **שכבה ה- Transport** אחראית על העברת הודעה process 7-process, בשכבה זו יש כבר אמינות כי היא דואגת שהמידע יעבור באופן שלם.
- **שכבה ה- Network** אחראית על שליחת חבילות מ-host אחד לאחר - מעביר מקו ל-ip אחר.
- **שכבה ה- Data-Link** אחראית להעביר מידע שנקרא frame מנקודה לנקודה.
- **שכבה ה- Physical** אחראית להעביר נתונים של ממש מנקודה לנקודה אחרת, ע"י סיבים אופטיים, זרמי חשמל וכדומה...



מושג הכינוס (encapsulation) - מצד השולח - השכבה מקבלת מידע מהשכבה מעלייה ומבצעת את תפקידיה בזה שהיא מוסיף header ו- "עוטפת" את עצמה. ומהצד מקבל - "מקלף" את השכבות.

מחסנית פרוטוקול האינטראנט



שכבות האפליקציה

תפקידי השכבה

- **תקשרות עם משתמש הקצה** - שכבות האפליקציה אחראיות על התקשרות עם משתמש הקצה - ממשק המשתמש, טיפול בבקשתות שונות, וכו'.
- **עיבוד הנתונים** - בהתאם להחלטות שמבצע המשתמש, שכבת היישום יכולה לדוחס את הנתונים, להצפין אותם, או לעבד אותם בכל צורה אחרת לפני שהועברו על-גבי הרשת.
- **ניהול תהליכיים** - כל פעולה שהמשתמש מבצע ברשות מוגדרת כתהlixir בין תחנת המוצא לתחנת היעד. שכבת היישום אחראית על ניהול התהליכים הללו, כך שני המחשבים יכולים לבדוק בין נתוניים של תהליכיים שונים המתקיימים במקביל.

פרוטוקולי השכבה

- HTTP, FTP, SSL, SMTP, POP3, TFTP, NFS, RSP .
- בנוסף לפרוטוקולים שימושיים בשכבה זו גם מספר שיטות דחיסה והצפנה דוגמת - ZIP.

יש לא מעט אפליקציות רשות כלומר משהו שיוצר אינטראקציה בין מכשירים. אנו צריכים לדאוג שהאפליקציה תרוץ במכשירות הקצה (מחשב, שרת, טלפון וכו') ותדבר בעזרת הרשות כאשר אנו לא צריכים לטפל בנתבים ומוגדים וכך תפקידנו לדאוג לפתח את האפליקציה שלנו בלבד.
שנム 2 ארכיטקטורות לאפליקציות:

1. ארכיטקטורת Client-Server

Client - יוצר קשר ומתקשר עם השרת, הלקוח לא תמיד מחובר והוא יכול לנوع מכתבות רשות אחת לאחר, יכולות להיות כתובות IP דינמיות שונות.
Server - צריך לחתור ללקוח, צריך לדעת מה הכתובת שלו.

2. ארכיטקטורת Peer-to-Peer

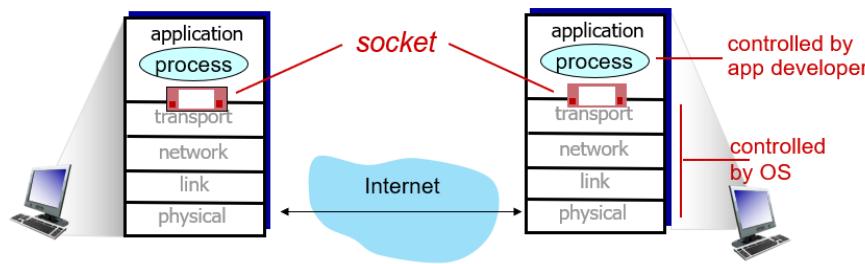
השרת לא תמיד זמין, מערכות הקצה שירויות ומתקשרות ישירות. עםיתים מבקשים שירותים אחרים (בדומה ל-torrent, skype), נווטנים שירות בתמורה לעמיתים אחרים. עמיתים מחברים לסריגן ומשנים כתובות IP וניהול שלו מורכב.

טהlixir התקשרות (Processes communicating)

טהlixir (Process) – מופע של תוכנית המנוהל על ידי מערכת הפעלה. שני תהליכיים באותו מחשב מתקשרים דרך מערך הפעלה. בין מחשבים שונים, תהליכיים מתקשרים דרך הרשת. כאשר שני תהליכיים (process) רוצים לדבר ביניהם הם צריכים להעביר הודעות אחד לשני. במקרה שלנו, נכון להגדיר שהטהlixir הראשון הנקרא (client) והשני זה השרת (server).

Socket

- כדי שתהליק יוכל לקבל/לשלוח מידע לרשת/מהרשת חייבים לה "צינור" הנקרא **Socket**
- שילוב של כתובת IP עם מספר Port מגדיר כתובת שקע (**Socket Address**) .
- Sockets Connection מזוהה ע"י צמד



: Sockets 3 סוגי

1. **Stream Socket** : קשר מהימן קצר לenza - TCP.
2. **Connectionless Datagram Socket** : קשר לא מהימן לprotoוקולים ברמות נמוכות יותר.
3. **Raw Socket** : גישה ישירה לprotoוקולים ברמת ה- Application ורמת ה- Transport. דוגמא לכך שלשכבה ה-application השפיעה על שכבה ה-Transport.

בד"כ במחשב פועלים מספר סוגים יישומים בו-זמןית (בעזרת מערכת הפעלה). כיצד המחשב יודע לסייע את הנתונים המתקבלים, לישומים השונים הפועלים בו-זמןית? לא מספיק רק כתובת IP אלא גם אני איזשהו כתובת פנימית איזשהו **Port** - פורט פנימי שטמפה אותו לשירות אליו אני רוצה לגשת. יש כל מיני סוג פורטים: פורט 80 - ניגשים אליוqua HTTP, פורט 25 - מייל (לא צריך אלא רק להכיר). הלוקו יוצר את הקשר עם השרת لكن הлокו צריך לדעת לאיזה פורט לגשת בשרת אבל כשהוא כבר מגיע לשרת, השרת כבר יודע מאייזה פורט הлокו ניגש בצד שלו וזה הוא השולח לו את הנתונים שהлокו דרש בהתאם לדרישות.

פרוטוקולים בשכבה הרשות (An application-layer protocol)

כדי להגיד מה זה הפרוטוקולים צריך שכל פרוטוקול יגיד לנו איזה סוג הودעות הוא משתמש. מה הסינטקס של כל הודעה, ומה המשמעות של כל אחד מהנתונים כי לא תמיד ההודעה תהיה טקסט לפעמים זה יהיה מספרים וכו' וגם כמibaseן הפרוטוקול צריך לדעת מתי לשלוח ולקבל הודעות. מבחרת פרוטוקולים יש 2 סוגים כלליים:

- 1. Open Protocol** - כל אחד יכול לגשת אליהם להגדיר ולממש אותם.

כלומר אם אנחנו רוצים שאנשים ישתמשו באפליקציה שלנו הם צריכים להכיר את הפרוטוקול של האפליקציה. - זה מאפשר לנו להשתמש בכל מיני דפנאים שונים לנו את אותן השירותים פחות או יותר כי הם יודעים למשש את הפרוטוקול לפי ההגדרה שלו כמו עם הפרוטוקול המוכר HTTP.

- 2. Proprietary Protocols** - פרוטוקול פרטי כמו skype, אני לא יכול ליצור לי איזשהו סקייפ מסויל שידבר עם סקייפ אחר, ככל עובדים עם סקייפ אחד בלבד.

דרישות האפליקציה משירות התעבורה

- העברה אמינה ומלאה של המידע.
- חלק מהאפליקציות צריכות עמידות בזמןניים ודילוי כמו משחקי מולטיפליר במחשבים.
- תפקוה של נתונים כמו למשל איקות צפיה של מולטימדיה.
- אבטחה.

סוגי שירותים בשכבה התעבורה

- 1. TCP Service** - דואג לשירות אמין כלומר דואג שהמידע יעבור בין 2 התהליכים ואם המידע לא הגיע הוא ישלח אותה שוב لأن צורך. הוא מבטיח גם לא להעmis את הצד השני במידע.

- 2. UDP Service** - שירות לא אמין - שולח את המידע ולא מבטיח שהמידע הגיע ויגיע אל היעד.

TLS - אחראי על האבטחה בשכבה התעבורה עם TCP, מאפשר הצפנה מקצועית לכך.

HTTP: פרוטוקול שכבה האפליקציה של האינטרנט

דף האינטרנט מורכב מאובייקטים, כך שכל אחד מהם יכול להיות מאוחסן בשרת או באינטרנט שונים. האובייקט יכול להיות קובץ HTML, תמונה JPEG, יישום ג'אווה, קובץ שמע, ... דף האינטרנט מורכב מקובץ HTML בסיסי הכלל מספר אובייקטים שהפניה אליהם ניתנת לכתובת אחר.



- **מצד הלקוח:** דפדפן שմבוקש ומתקבל הלוקו (באמצעות פרוטוקול HTTP) וגם "מציג" אובייקטים ברשות.
 - **מצד השירות:** שרת האינטרנט שולח (באמצעות פרוטוקול HTTP) את האובייקטים.
- ה-HTTP משתמש ב-TCP שמבצע את פעולה החיבור מהלקוח לשרת - (ההילך "לחיצת-היד") הוא יוצר את הסוקטאים בפורט 80. HTTP לא זכר שום דבר על הלוקו מבחינת הבקשות וכו'...

RTT - Round Trip Time - זה פרק הזמן של העברת החבילות מהלקוח אל השירות ובחרזה.

ישנו 2 סוגי של חיבורו HTTP :

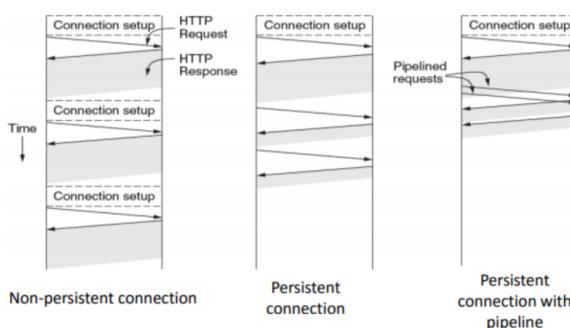
1. **HTTP Persistent (0.0) - פתייחת התחברות TCP, שליחת אובייקט אחדividually ולאחר מכן תסתמוך מה-TCP.**

זה שיטה לא טובה כי יש היום הרבה נתונים ברשות וזה יכול לגרום לעיכובים רבים כי אני כל הזמן פותח וסגור את ההתחברות.

2. **Persistent HTTP Persistent (1.1) - פתייחת התחברות TCP, שליחת אובייקטים ביחד ולאחר מכן תסתמוך מה-TCP. הרבה יותר יעיל ומשפר לי את זמן הגלישה.**

לגישה זו יש 2 תתי-גישות:

- א. **NP** - אנו נשלח בקשה רק לאחר שקיבלונו את התשובה לבקשת הקודמת.
- ב. **P** - נשלח כמו בקשה ואז מקבל את התשובות ביחד.



HTTP Request Messages

- POST method - מבקשת שרת אינטרנט לקבל את הנתונים הכלולים בגין הودעתה הבקשה, ככל הנראה לאחסונם.
- GET method - משמש לבקשת נתונים מקור מוגדר.
- HEAD method - כמעט זהה ל-GET, אך ללא גוף התגובה.
- PUT method - משמש לשילוח נתונים לשרת כדי ליצור / לעדכן משאב.

HTTP response status code

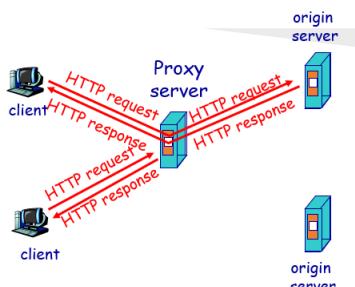
הסטטוס מופיע בשורה הראשונה בהודעת הקבלה של server-to-client, סלטוטים כמו:

- 200 OK - הבקשה הצליחה, האובייקט המבוקש בהמשך הודהה זו.
- 301 Moved Permanently - האובייקט המבוקש הועבר, מקום חדש צוין בהמשך בהודעה זו.
- 400 Bad Request - הודעת בקשה אינה מובנת על ידי השירות.
- 404 Not Found - המספר המבוקש לא נמצא בשרת זה.
- 505 HTTP Version Not Supported

HTTP Header

מאפשרת ללקוח ולשרת להעביר מידע נוספת עם בקשת HTTP או תגובה.

- כדי שחוויות הגלישה תהיה טוביה יותר בין אם זה קישור זמן הורדת הדף ועוד כמה,
- זכיר ש-HTTP לא זכר את הלוקוח ואת הדרישות שלו ולכן שיפורים מסוימים לטובות הלוקוח:
- **HTTP Cookies** - אתרי אינטרנט ודפנות לCookie משתמשים בעוגיות כדי לשמור על מצב כלשהו בין תיעודים כלשהם - זה יכול לחת למלצות טובות יותר, מצב הפעלה משתמש והתאמאה לצרכי הגלש. החיסרונו הוא הפגעה בפרטיות כי הוא יידע הכל על הגלש.
 - **Web Caches - Proxy Servers** (מטמון) - מצומצם את זמן התגובה לבקשת הלוקוח, המטמון קרוב יותר ללקוח, מאפשר לספקיו תוכן להעביר את התוכן בצורה יעילה יותר.



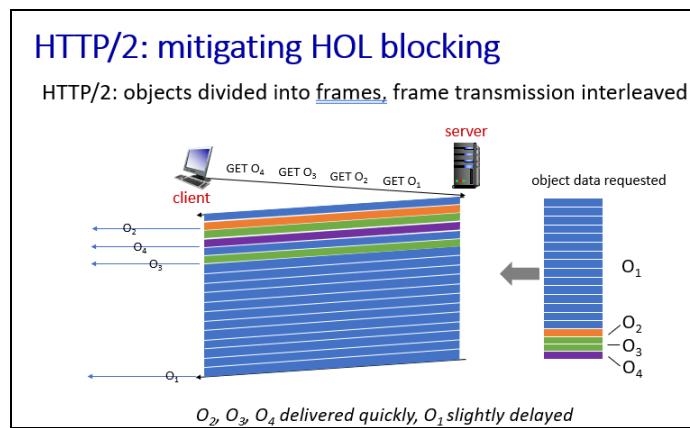
Proxy

שרת proxy משמש כשער ביןINTERNET.

זהו שרת מתווך המפריד בין משתמשי קצה לבין אתרי INTERNET בהם גולשים. שירותי פרוקסוי מספקים רמות שונות של פונקציונליות, אבטחה ופרטיות בהתאם למקרה השימוש שלו, לצרכים שלך או למבדיניות החברה שלך.

גרסאות ה-HTTP

- **HTTP/1.x** - הוא יוצר הרבה חיבורים במקביל ולא דוחס את header וגם לא עובד בסדרי עדיפות.
 - **HTTP/2** - דוחס את header, הוא מאפשר לנו-**Push** שזה בעצם יכולת של השרת לשלוח תגובות מרובות לבקשת לקוח יחיד, מספיק לו להתחברות TCP אחת בלבד.
- יש לנו **Stream** - כלומר הצמד של request ושל response.
- בנוסף גרסה זאת מחלקת את האובייקטים ל-**frames** וכן התעבורה עוברת מהר יותר. הבעיה בגרסה הזאת היא שעדיין TCP ייחיד זה כמו צינור אחד שמכניס בתוכו הרבה streams אבל אם משהו הולך לאיבוד אנחנו נהייה בבעיה כי ה-TCP צריך לשדר את זה עוד פעם וזה יכול לעכב לנו שידורים אחרים וגם אין לנו אבטחה.



- **HTTP/3** - יותר מאובטחת, מבקר כל שגיאת אובייקט וועמס באמצעות UDP.

DNS: Domain Name System

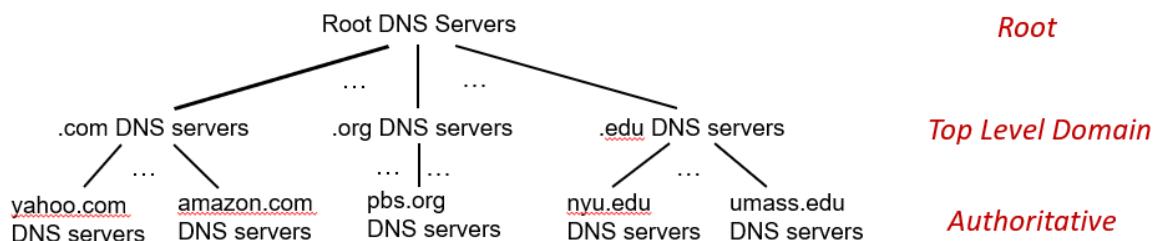
זה מערכת לתרגום כתובות URL ל-IP. המערכת בנויה משרתים המחזיקים מאגרי מידע בצורה מבוזרת לפי היררכיות כדי להתמודד בעיקר עם בעיות של נקודת כשל ייחוד, עומס בתעבורה וגם בעית מרחק הפיזי מתחנות היעד.

אם נרצה לגלוש לאתר מסוים אנחנו לא צריכים את הכתובת IP שלו אלא את ה-URL שלו ובשביל הדוגמה, כאשר אנו נלחץ ENTER בתהילך הראשון שקרה זה תהליך של DNS כלומר ההמרה הזאת מהשם URL אל כתובת-IP, בתהילך זהה מעורב פרוטוקול.
טהיליך זה מאפשר לנו להשתמש בשמות שלא באמות אמיתיים, אפשרות לבזר את העומס למשל יש כמה webservers אז בתהילך DNS אפשר ליצור מצב שבו אני מפנה חלק מהליקוחות לשרת אחר.

יש לנו כאן מערכת מאוד מורכבת כי מדובר בהרבה מאוד מידע וכן אנו מבזרים אותו ועושים לו היררכיה והיא מחולקת בדרך"כ ל-3 שכבות:

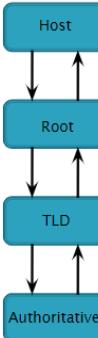
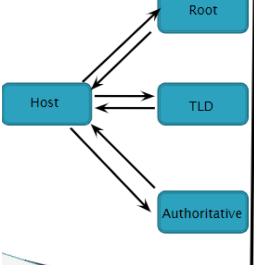
Top Level Domain Root DNS Servers

השכבה העליונה היא Root DNS Servers השכבה השנייה היא **com** והשכבה השלישית היא **Authoritative**.
בדרך"כ כאשר אנו הולכים לאיישתו כתובת למשל **amazon.com** אז מה שקרה זה שהЛОקח פונה **root** והוא יפנה אותנו ל-**com** Top Level Domain של **com** ואז הוא יפנה אותנו ל-שרת DNS של **amazon.com**



Local DNS name servers

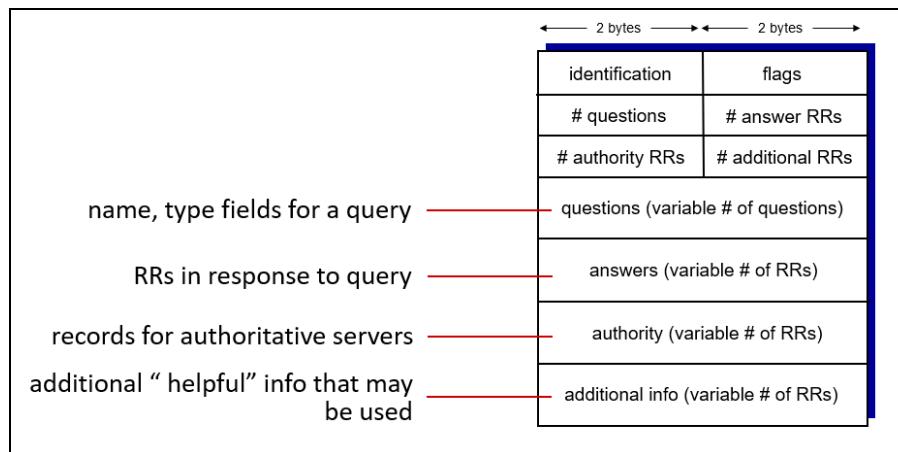
כאשר המארכ מבצע שאלות DNS, השאלתה נשלחת לשרת ה-DNS המקומי שלו. שרת זה אינו שיר בהכרח להיררכיה, הוא נמצא באמצעותו בין הלקוח בין היררכיה הוא עושה את כל השאלות, יכול לשמור את התשיבות שהוא מקבל וזה יכול לחפשו לנו את הזמן. שרת מקומי לרוב יהיה באותו subnet של המשתמש. יש 2 תהליכי שאלות, איטרטיבי וקורסיבי...

קורסיבי	איטרטיבי
 <p>שיטת זו, המחשב המבקש שולח בקשה אחת לשרת ה-Root, שרת Root, לא מחזיר תשובה עדין אלא שולח בעצמו את הבקשה לשרת ה-TLD. לאנו מחזיר תשובה עדין אלא שולח את השאלה לשרת המהימן, שמחזיר תשובה לשרת ה-TLD, שמחזיר תשובה לשרת ה-Root, ושמחזיר תשובה למבחן המבקש.</p>	 <p>בשיטת זו, המבחן המבקש, שולח בקשה לאחד משרתי ה-Root, Root, לאחר קבלת התשובה (כתובת של שרת TLD) הוא פונה (בעצמו) לשרת ה-TLD ולאחר קבלת תשובה, הוא פונה (שוב בעצמו) לשרת המהימן (Authoritative).).</p>

ב-database יש לנו רשומות(RR) וכל רשומה יש לה פורמט(name, value, type , ttl , ttl) ולפי הפורמט הזה אנחנו יודעים מה האינפורמציה שאנו רוצה לקבל בהודעה.
live to ttl = time to live כולם כמו זמן אני יכול לשמור את הרשומה הזאת, ה-type משפיע לי بما אני מצפה לראות ב-name וב-value:

type=A או AM	type=NS או NM
<ul style="list-style-type: none"> ה-name יהיה hostname. ה-value יהיה ה-IP Address. 	<ul style="list-style-type: none"> ה-name יהיה ה-domain למשל google.com ה-value של ה-hostname שהוא אמיתי לאותו domain.
type=MX או MM	type=CNAME או CM
<ul style="list-style-type: none"> ה-value יהיה השם של המסר שמחובר אליו השם. 	<ul style="list-style-type: none"> ה-name יהיה לו את ה-name alias הכוונה שיש לו את השם שונה לו לעבוד אותו כמו למשל com www.google.com אבל במקרה צריך לחפש אותו בכתובת אחרת שלא נראה לעיני הגולש. ה-values יהיה ה-שם המקורי שלו.

כך נראה פרוטוקול ההודעות של DNS, להודעות בקשה ותשובה של DNS יש את אותו הפורמט:



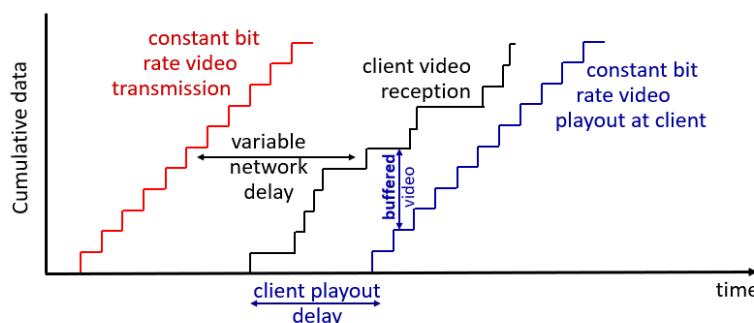
DNS Security

- DDoS Attacks - לפוצץ את שרת ה-root בתעבורה כבדה.
- להפצע שרת DNS בתעבורה כבדה.
- ניצול DNS עבור SDoS כך שנשלחות שאילותות עם כתובות מקור מזויפת: יעד IP.
- שליחת שגיאות מזויפות לשרת DNS.

תכנים מולטימדיה - Video Streaming and CNDs

תעבורה ידאו הוא ה拄רך העיקרי של רוחב הפס באינטרנט כמו למשל Netflix, YouTube וכו'... ידאו מוגדר מבחינתו כאוסף של תמונות המוצגות לנו בקצב קבוע, יש כל מיני פתרונות לחישכת רוחב הפס באמצעות טכניקות תעבורת נתונים המתאימים לידאו ולא פוגעים באיכות אופן פטאל.

האתגר העיקרי הוא שרחב הפס של שרת-לקוח משתנה לאורך זמן. שינוי רמות העומס ברשת (בבית, ברשות הגישה, בליבת הרשת, בשרת היידאו) ואובדן פקודות יכול לעכב את הפעלת היידאו או לגרום לאיכות ידאו יורדה. لكن קיים ה-buffer שבעצמן הפעלה טווע מראש ידאו כדי למנוע תקיעות ולצופה יהיה את האפשרות לצפות חלק לכל אורך הסרטון.



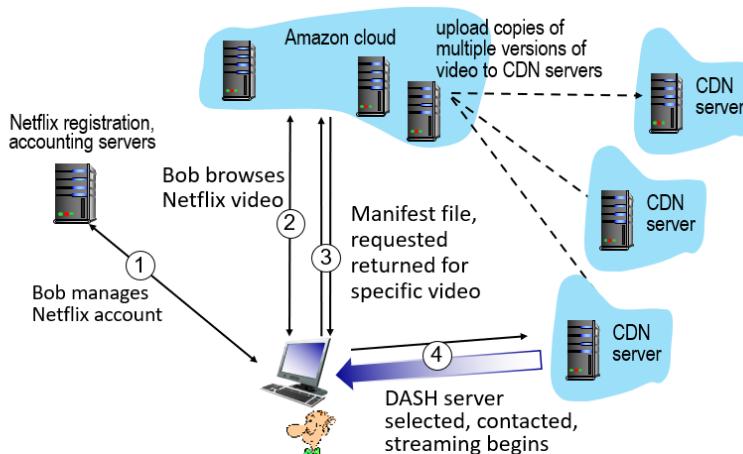
הרצון שלנו ליצור מערכת שרת-לקוח שתוכל לחתן לו את הסרטיים ולהתמודד עם כל מיני בעיות ברשת וגם עם כל מיני רצונות של הלקוח ולתת לו את האיכות הכי טובה באותורגע.

DASH : Dynamic, Adaptive Streaming over HTTP

זה הבסיס לכל streaming של מולטימדיה
מצד השרת אני לוקח את הסרט מחלק אותו לחתיכות, כל חתיכה מקודדת בכמה איזיות ובבנייה מעין מפה של URLs שכל אחד מהם מצביע על חתיכות שונות.
מצד הלקוח - מודד מעת לעת את רוחב הפס של השרת ללקוח, מבקש חתיכה אחת בכל פעם,
בוחר בקצב קידוד מksamלי בהתחשב ברוחב הפס הנוכחי ויכול לבחור שיורי קידוד שונים בנקודות זמן שונות
(תלו依 ברוחב הפס הזמן בזמן).

CDNs

זה אוסף של שרתים שנמצאים או בתחום ה-access network שלך או מחובר ליד DXו לשמה אני משכפל
תוכנים כמו סרטים תמונות מוזיקה וכו' ...
המטרה היא לספק זמינות וביצועים גבוהים על ידי הפצת השירות באופן מרחבו ביחס למשתמשי הקצה. הוא
עוזר לי לדוחף את התוכן קרוב אליו (לא בהכרח גיאוגרפית) אלא מהרבה מאוד בחינות.



שכבה התעבורה

כשאנו מדברים על שירות התעבורה הכוונה שאנו יוצרים מעין חיבור לוגי בין תהליכי שוכנים אצלם אצל המארח כלומר מבחינת המארחים הם אכן מדברים אחד עם השני.

פעולות פרוטוקולי תעבורה במערכות קצה:
השלוח: מפרק הודעות אפליקציה לקטעים (**segments**) וועובר לשכבה הרשת.
המקבל: מרכיב מחדש את ה-**segments** להודעות וועובר לשכבה האפליקציה.

שני פרוטוקולי תעבורה הקיימים ליישומי אינטרנט הם **TCP**, **UDP**.

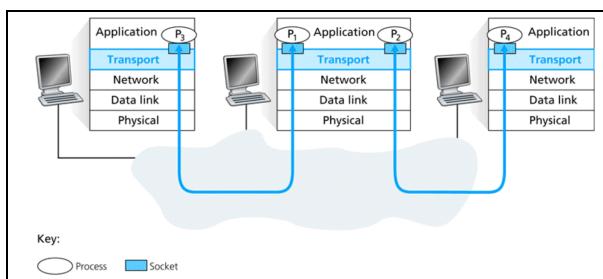
זכור שכבה זו מחברת בין פרוטוקולים ושכבה הרשות מחברת בין הוסטיהם זאת אומרת התעבורה צריכה לקבל מהפרוטוקולים את ההודעות ולהעביר אותן לרשות.
הרשת לא מביאה השמיע וגע, וגם לא מביאה שיגע לפיה הסדר, יכול להיות שביטים יהיו תקולים ובשביל זה אנחנו צריכים את שכבה התעבורה.

פורמט הנתונים שניתן לזהות על ידי IP נקרא **IP-datagram**.
הוא מורכב משני רכיבים, כולל **header** ומהנתונים, אותם יש להעביר.
לשודות ב-**datagram**, למעט הנתונים, יש תפקדים ספציפיים בביצוע העברת הנתונים.

Multiplexing and Demultiplexing

מבוססים על segments, datagrams וערכי השדות של header.
זה תהליך שקורה בכל השכבות.

- הוא הממשק שדרכו תהליך (ישום) מתקשר עם שכבה התעבורה.
- כל תהליך יכול להשתמש בשקעים רבים.
- שכבה התעבורה במכונה מקבלת רצף של קטעים משכבה הרשת שלה.
- מסירת קטעים לשקע הנכון נקראת **demultiplexing**.



- הרכבת קטעים עם המידע החדש והעברתם לשכבה הרשת נקראת ריבוב.
- נדרשים בכל פעם שונזר שיתוף תקשורת. **demultiplexing-I multiplexing**

איך Demultiplexing עובד?

המארח מקבל datagrams IP, לכל datagram יש כתובת IP של המקור וכתובת IP של היעד.
כל datagram נשא segment אחד של שכבה התעבורה וכל segment יש port של המקור ושל היעד.
המארח משתמש בכתובות IP וב-port כדי להפנות את ה-segment אל ה-socket המתאים.

משתמש במספר של ה-port של היעד (בלבד).
demultiplexing : UDP ↵
demultiplexing : TCP ↵
 באמצעות 4-tuple באמצעות :

1. כתובות IP של המוקור
2. כתובות IP של היעד
3. מספר ה-port של המוקור
4. מספר ה-port של היעד

ומשם ה-4-tuple: משתמש בכל 4 הערכים האלה כדי להכוון את ה-socket אל ה-segment הנכון.

UDP: User Datagram Protocol

- מtabס על IP-Protocol שלא מבטיח הגיעו לפי סדר והגעה באופן כללי.
- ההודעות יכול להאבד והוא גם לא מבטיח שלא יהיה שגיאות.
- UDP עוזר יותר בנושא השגיאות זהה שהוא מוסיף ל-header שלו שדה מסויים כך שהצד השני שמקבל יכול להיעזר בשדה זהה כדי להבין האם התרחשה שגיאה.

למה UDP?

1. הוא יכול לשדר מהר ופשוט לומר הוא לוקח את המידע מהאפליקציה,
2. מוסיף header, ושולח ל-IP.
3. יכול לעבוד כאשר יש עומס שונה מה-TCP שמוריד את הקצב בעת עומס.
4. לא מבזבז RTT כמו ב-TCP שעוד לפני קבלת מידע אני עושה פתיחת קשר.

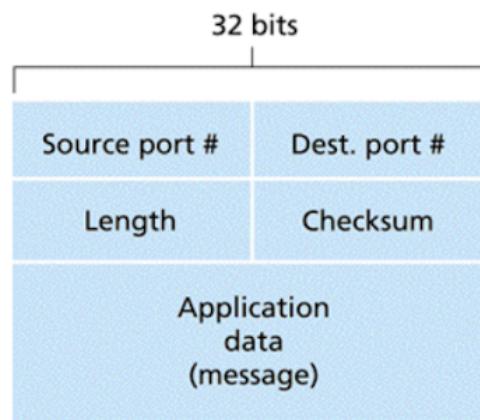
מי משתמש ב-UDP?

- אפליקציות סטרימינג (such as amazon prime, netflix, disney וכו'...)
- DNS
- SNMP - (פרוטוקול תקשורת לניהול התקני רשות ברשות IP)
- HTTP/3 הוא דוגמה לפרוטוקול שהפסיק להשתמש ב-TCP והוא עבר ל-UDP אבל כדי לעשות את זה הוא צריך להעביר מושכות שהוא ב-TCP כמו נניח reliability וה-control congestion control משכבות התעבורות אל שכבת האפליקציה.

UDP Segment Structure

- יש לו 4 שדות בלבד, כל שדה בניוי מ-16 ביטים ואת ה-header שלו.
- מזזה את תחילת UDP ששלח ה-segment Source port
- מזזה את תחילת ה- UDP אשר יטפל בנתוני האפליקציה. Dest. port

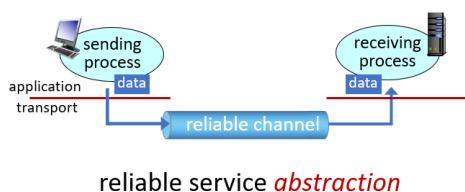
- מצין את אורך ה-*segment*, כולל header - Length
- מזזה שגיאות - Checksum



UDP Checksum

הוא לוקח את הביטים מסדר אותם כמו במטריצה שהרוחב שלה הוא 16, מחשב את הסכימה של הביטים ובסופו הוא מפעיל NOT כלומר ההופכי שלו, הצד השני לוקח את המידע אם הוא יעשה סכימה עם ה-*checksum* של השולח יצא לו רצף של אחדות - כלומר שהכל תקין, אחרת יש שגיאה.

העקרונות של העברת מידע באמצעות אמינה



ambilintano אנחנו היינו רוצים ליצור מצב שיש לי תחיליר של שלוח-מקבל ואני רוצה ליצור מצב שאין יוצר לאפליקציה איזשה שירות אמין-cailo הערכץ ביןיהם הוא אמין.

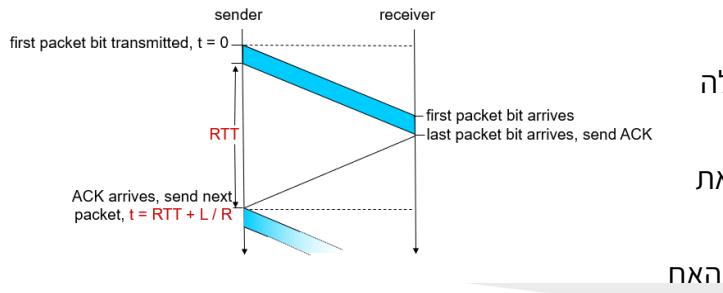
לכן היינו רוצים:

מנגרטן	שימוש
Checksum	חקיקו לזוות שגיאות של ביטים בחבילת העברת
Timer	משמש להעברה מחדש אם החבילה או ה-ACK אבדו (או התעכבו).
Sequence number	מספר רציף של זרימת חבילות. משתמש ב-sacks כדי לזוות חבילות שאבדו, ומשתמש במספרים כפויים לאתר חבילות כפולות.
Acknow-ledgment	מאשר חבילות שהתקבלו כהכלכה; מכיל מספר סידורי; עשוי להציג או להיות אינדייבידואלי.

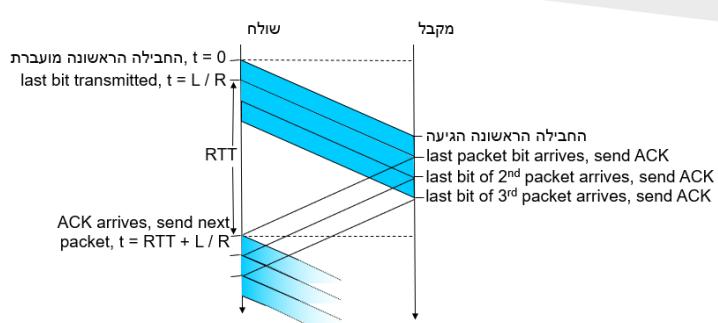
אומר לשולח שהחבילת לא התקבלה.	Negative ack-nnowledgment
שליחת מנות רק בטווח מוגבל, מגדילה את השימוש של השולח בהשוויה. ל-stop-and-wait.	Window, pipelining

פרוטוקלים/מנגנוןים מוכרים שספקים אמינות

1. Stop-and-wait - קלומר שידרת - תעצור



ותחכה, ורק אחרי שקיבלה ACK אז תמשיך. תרצו או לא תרצו החוקים האלה מביאים אותנו ל프וטוקול אמין כי אם המודיע לא הגיע לצד השני הוא ישדר את המידע עוד פעם ויתעקש על שידור החבילה עד אשר היא תגיע לצד השני. האה זה יעיל? - כמובן שלא, אבל זה עדין אמין.



2. Pipelining - מושפר את היעילות

הזאת - במקום לשדר חבילה אחת ולרכות, אנו נשדר כמה חבילות ייחדי.

3. Go-Back-N - מצד השולח:



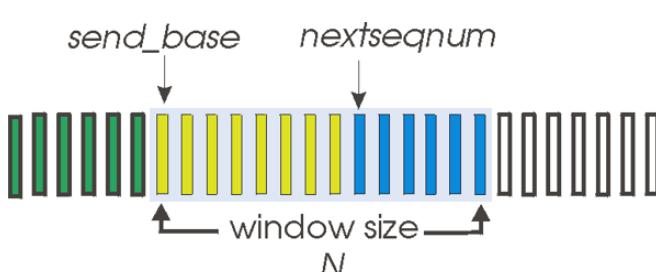
נתון לשולח חלון, ה-N הוא גודל החלון.

זה אומר שהחלון מאפשר לשדר N חבילות אחת אחרי השניה.

בכל רגע נתון הוא צריך לדעת מה החבילה הראשונה בחלון ומה הבאה שאני יכול לשדר. יroke - חבילות שוודרו וקיבלו עלייהם ACK קלומר טיפולנו בהם. לבן - חבילות שעדיין לא התעסקנו.

צחים - חבילות שנשלחו אבל לא קיבלתי עליהם עדין אישור.

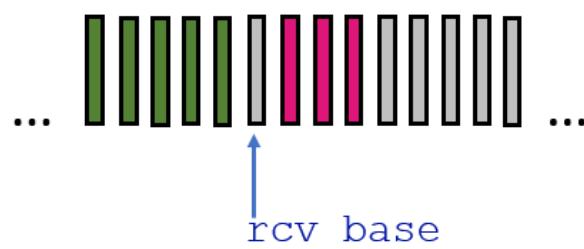
כחול - חבילות שאני יכול לשדר אבל עדין לא שלחתי אותם.



- המנגנון זהה עובד עם ACK מצטבר כלומר שאני מקבל ערך מסוים ב-ACK וגוזר ממנו שכל החבילות עד אותו ערך כולל - הגיעו.
- ה-ACK לא מאשר לי חבילת ספציפית אלא הוא מאשר לי את כל החבילות ברצף עד נקודת מסויימת בילי קפיצות.
- N of TIMEOUT לאייזחו חבילת N, שմשדר ממנה ועד סוף החלון ولكن השם של המנגנון.

מצד מקבל:

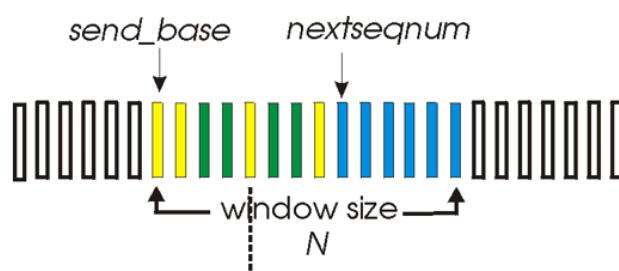
אם אני מקבל חבילת לא בסדר אני יכול לשמור אותה או אם אין לי מקום לשמר אותה היא נזרקת ואני מוציא ACK אך ורק על רצף של סדרה כלומר אני תמיד אסתכל על ה-`recv_base` ועלומר על ה-ACK האחרון ואם הוא עדין לא הושלם אז עדין לא סיימתי את התפקיד שלו.
ירוק - התקבל בהצלחה עם ACK.
ורוד - לא בסדר : התקבל אבל ללא ACK.
אפור - לא התקבל.



- Selective Repeat . 4

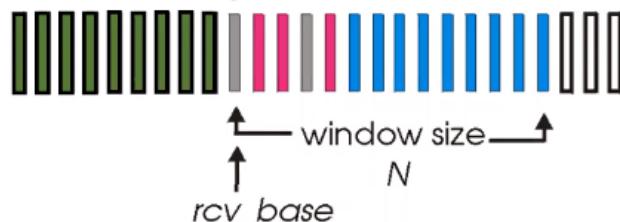
מצד השולח:

אני יוצא מנקודת הנחה שלצד השני יש חלון קבלה, لكن אני אומר - אני רוצה וمبקש לאשר אינדיידואלית כל חבילת שאתם מקבלים.
לכן אם יש שידור חוזר אני משדר חבילת אחת עליה אנו מדברים.
החלון אותו חלון כמו שאנו מכירים הכוונה שהוא באותו תתייחסות אבל כאן בಗל ששיעור הוא פרטני יכול להיווצר מצב שבתווך החלון יהיו לי חבילות מאושרת שאישרו אותם (היורוקות) ועודין יכול להיות חבילות שלא קיבלתי עליהם אישור ולאחר מכן תמיד חבילות שאני יכול לשדר ועודין לא שידרתי.

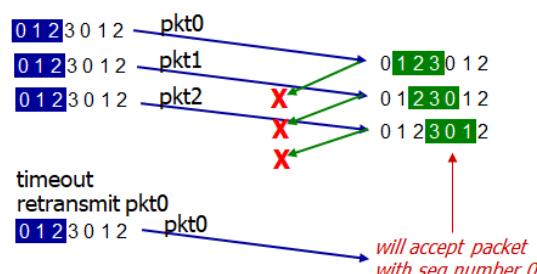


↳ מצד מקבל:

יש גם חלון קבלה שיכל להיות בסطטוס שונה מהחלון שלicha והוא מתייחס לחבילות שהוא קיבל, חבילות שחשפות לו וחבילות שיכל לקבל אבל עדין אין שום התיחסות עליהם.



חשוב! - הדילמה של המגנון זהה, הבעיה היא שלא הגיע ACK על החבילה הראשונה.



שאלה - איזה יחס נדרש בין גודל הרצף לגודל החלון כדי למנוע את הבעיה?

תשובה - כדי להתמודד עם המצב שהחלון שלicha שונה מחלון הקבלה ולא יהיה לי חוזרת על

אחד האינדקסים אני צריך שהיחס של גודל החלון השליחה לבין טווח הסדרה יהיה **פִי 2. קלומר**

שהסדרה תהיה גדולה פי 2 לפחות.

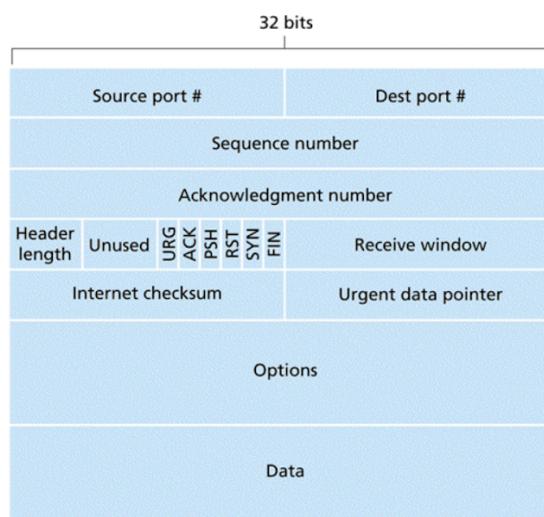
את אומرت שאם הסדרה הייתה $0,1,2,3,4,5$ וגודל החלון הוא 3 אז היה מתקיים כמו שצורך ללא שום בעיה.

TCP: Transmission Control Protocol

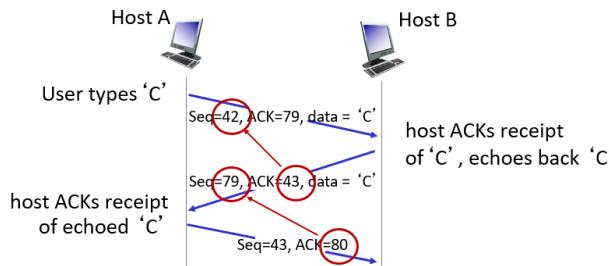
- הוא point-to-point הכוונה שיש לו שולח ומქבל והميدע יכול לעבור ב-2 הכוונים כלומר שם אני פותח קשר עם מישו אחר אז אני מדבר אליו והוא מדבר איתי.
- אני מעביר בזוורה של בתים ולא בזוורה של הודעות.
- אפשר ל'הברת הודעות משכבה האפליקציה ללא הגבלה.
- לרוב אני משתמש ACK מצטבר.
- עובד ב-Pipeline כלומר יכול לשЛОוח segment אחרי segment כלומר אני לא צריך לחכות לאישור כדי לשЛОוח את השני.
- פתיחת קשר לפני שליחת המידע.
- משתמש לא להעמים על הצד השני.

TCP Segment Structure

- source, dest, checksum - כמו שראינו ב-UDP.
- Sequence number and Acknowledgement number - משמשים ליישום העברת אמינה.
- Header length - אורך החדר.
- Flags field
- 1. URG - מצבין כי השולח סימן נתונים מסוימים כדחופים.
- 2. ACK - מצבין ששדה Acknowledgement number בתוכף.
- 3. PSH - מצבין כי יש להעביר אותו באופן מיידי (PUSHed).
- 4. RST - משמש לאייפוס חיבור, כלומר חיבור מבולבל או סירוב.
- 5. SYN - משמש ליצירת חיבור.
- 6. FIN - משמש לסיום חיבור.
- Receive window - מזזה כמה שטח זמני עברו נתונים נכנסים.



TCP Sequence numbers and ACKs

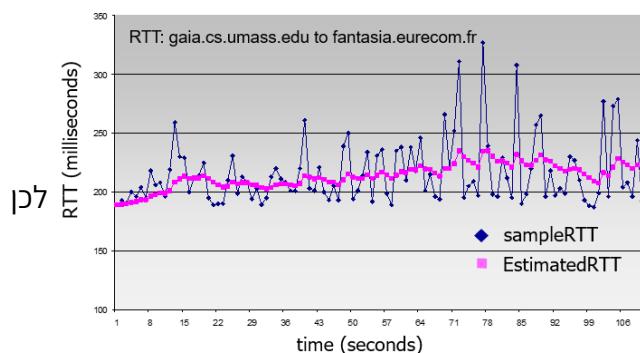


- Sequence numbers** • זרם בתים "מספר" של בית ראשון בנתוני הקטע (לא כמו ספירה רגילה מ-1 עד N) אלא לפ'.
- Acknowledgements** • מספר מס'ר בתים הבאים צפוי מהצד השני, ACK מצטבר

TCP round trip time, timeout

כמה זמן אני מחכה עד שאני מחליט שה-segment לא הגיב? מה הערך של timeout שאני אגדיר? אמרנו שהזיהו קשור ל-RTT אבל כולנו יודעים ש-RTT הוא לא ערך קבוע. لكن אנחנו צריכים להעריך את-RTT אבל איך עושים זאת. SampleRTT: זכרנו נמדד מהעברת ה-segment ועד קבלת ACK, לא כולל חישוב שידורים חוזרים. SampleRTT לא ערך קבוע, הוא משתנה כל הזמן, אך נעשה ממוצע של כמה מדידות אחרונות, ולא רק של ה-SampleRTT הנוכחי. פונקציית הממוצע המשוקל תראה כך:

$$\text{EstimatedRTT} = (1 - a) * \text{EstimatedRTT} + a * \text{SampleRTT}$$



ערך אופייני: $a = 0.125$ כלומר $a = \frac{1}{8}$. את ה-RTT בכחול אפשר לראות ש-segment אחריו segment ה-RTT לא אחיד באמת. לכן sample לא טוב לנו כי הוא כל הזמן משתנה אני צריך את הממוצע של האחרונים נסתכל על ה-Estimated אבל כשאנו משערכים משהו יכול להיות טעויות וכך הלאה: ולכן קיים הפתרון שבו נכניס לחישוב שלנו כמה תאים נוספים:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



הערכתה של RTT

"טוווח ביטחון"

כלומר נחשב את הטעות באותו השיטה, כאשר $\beta = \frac{1}{4}$

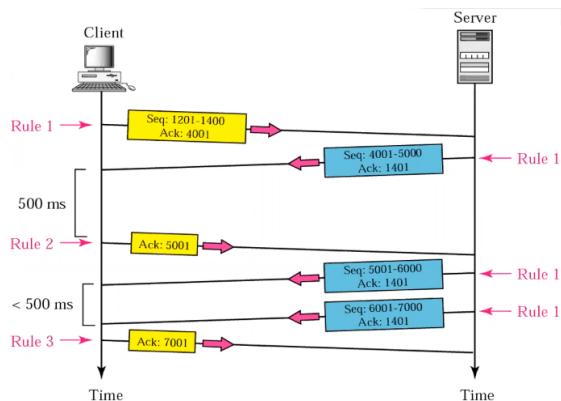
$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

הטעות המוצברת

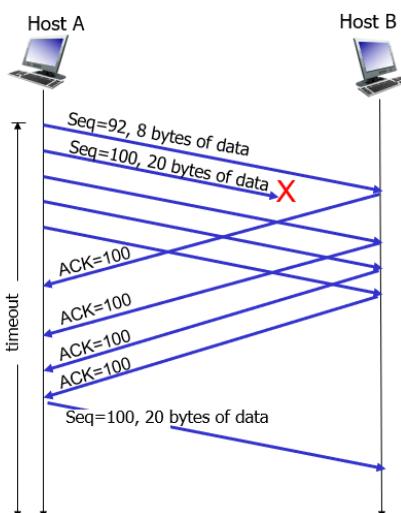
הטעות הנוכחיית

TCP ACK generations rules

1. אם אני רוצה לשלוח מידע, אין לי סיבה לשלוח את המידע וה-ACK ביחד זה מיותר. אני יכול לשלוח אותם ביחד זה נקרא (piggyback) כלומר אני לוקח את ה-ACK ומוכניס אותו לתוך ה-data header.
2. אם אני מקבל איזשהו segment וקיבלתי אותו לפי הסדר ואני לי מידע לשולח אול' כדי ליחסות קצר או אז שזהן הקצר עבר או שיגיע לי עוד segment לפי הסדר ואז יש לי שני segments שונים ואני יכול לאשר.
3. אם מגע לי משהו שראיתי כבר או שהגיע לי לא לפי הסדר או כל דבר אחר אז פשוט נתן לו לשלוח את ה-ACK כמה שיותר מהר.



TCP Fast Retransmit

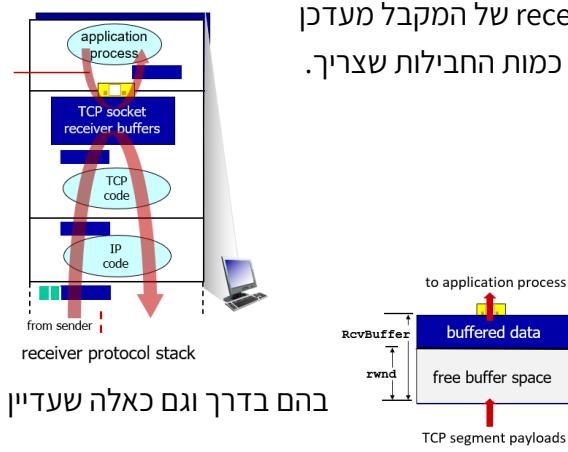


המנגנון עושה אופטימיזציה, אם אני מקבל שכפולים של ACK אז זה אומר שה-segment שידרתי לא הגיע אל היעד.

במוקם להתעתק וליחסות שה-timeout יגמר בלי לעשות כלום אז המנגנון ינצל את הזמן היקר זהה וישדר את ה-segment שוב פעם אל היעד כי מן הסתם ידוע לנו שבניסו הקודם הוא לא הגיע.

ההסכמה היא שאם חזרו אליו 3 פעמים אותו ACK אז נבצע שידור נוספת של ה-segment ללא הגיע אל היעד.

TCP Flow Control



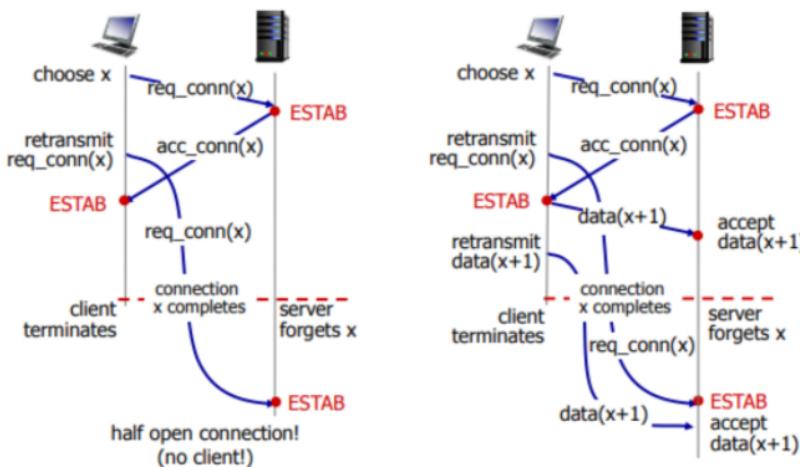
recv - זה אומר כמה מקום נשאר לו. כהה אני יכול להגביל את ה-sender הוא ישלח מידע בצורה כזו שהיא לא תעמיס את ה-receiver. המושג "in-flight" זה בעצם כל ה-segments שלם קיבלו ACK הם יכולים להיות segments שכבר שידרתי לא שידרתי.

TCP Connection Management

לפני שאני שולח מידע אני חייב לבצע "לחיצת ידים" ככלומר פתיחת קשר בין השולח למקבל. לחיצת 2 ידים זה כמו בין שני אנשים שפותחים בשיחה.

```
Socket clientSocket = newSocket("hostname", "port_number");
Socket connectionSocket = welcomeSocket.accept();
```

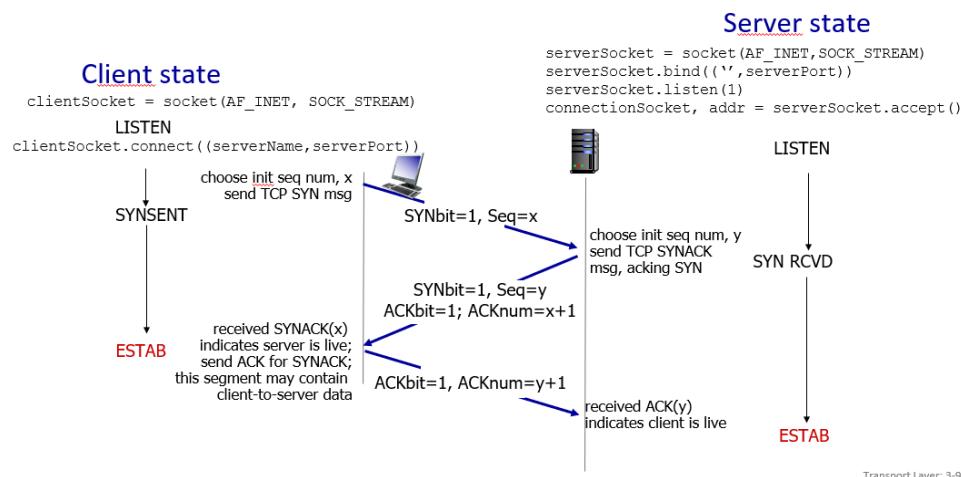
באיר השמאלי הקשר נפתח פעמיים ללא סגירה ולגנ יש בזבוז משאבים. באיר הימני הלקוח מבקש קשר וגם שולח מידע לשרת לפניו שקיבל ממנו accept. אחר כך הלקוח שולח שוב את המידע ואנחנו כבר במצב timeout וرك אז השרת מקבל accept על המידע ולכן נפתח כאן קשר פעמיים ככלומר בזבוז משאבים ושליחת מידע פעמיים.



זה לא מספיק לחיצת 2 ידיהם כי אני מבהיר משאבים, כי עד שהשלוח יעביר מידע למקבל, הקשר כבר יסגר וכל תהיליך לחיצת 2 ידיהם יהיה מבוזבז סתם.

נשתמש בלחיצת 3 ידיהם - בצדקה זו הקלינט/שרת יודע להתעלם מכפיילויות.

באיור אפשר לראות שה-client מניח שה-connection פתוח רק אחרי ה-SYNACK מה-.Server.received ACK(y) is alive Client-ACK מה-.Server מניח שה-connection פתוח רק אחרי ה-ACK כלומר Client-.header. אני שולח הודעה נוספת נוספת מצרי את ה-ACK לתוכו ה-data של ה-.header. אם יש מספר כלשהו כפוף אז השרת ידע להתעלם ממנו.



התנטקות מ-TCP

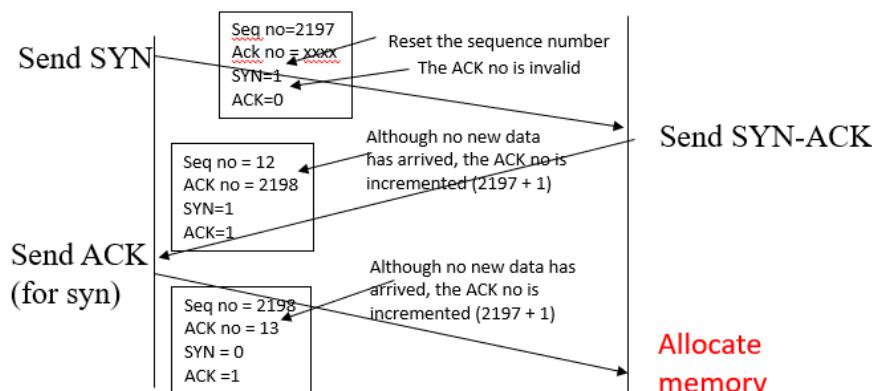
יש אפשרות המידע עובר באופן דו-צדדי (full-duplex) כלומר שמיידע ממש בין הקלינט לשרת יכול לעبور בו שני הכוונים. זה שהקלינט סיימ לשדר זה לא אומר שהשרת סיימ לשדר וכך יש הודעת FIN מכיוון הקלינט והשרת יאשר אותה ואחריה יהיה הודעת FIN מצד השרת והודעת ACK מצד הקלינט ורק אז נסגר הקשר.

SYN Flooding Attack

נניח ואני תוקף שרת כלשהו, אני מציק לשרת ומוציא אותו משירות בזה שאני פותח מולו הרבה מאוד התחברויות ולוקח לו את כל המשאבים שלו ומציף אותו בבקשתות הגורמות לעומס. אז מצד השרת, אם לcko מבקש יותר מדי משאבים אז אני אגביל אותו. כלומר לכל IP יהיה אפשרות להתחבר אליו ב-x התחברויות, אבל זה עדין לא יעוזר לי, כי התוקף לא מעוניין אותו לקבל תשובה הוא רוצה להוציא את השרת מאייזן ולכן יוכל לזייף את כתובתה ה-IP של המקור כלומר לשנות מלא הודיעות מכל מיינו סקן.

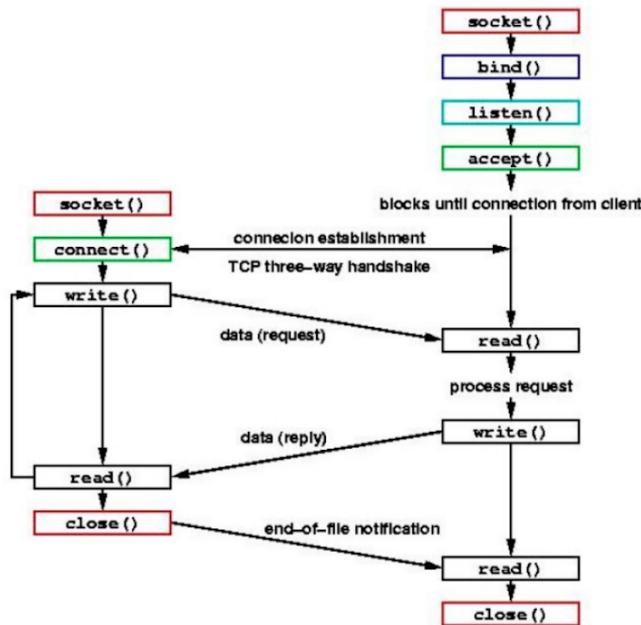
אז מה אפשר לעשות כדי להימנע מתקיפה שכזו?

בעזרת **SYN Cookie** - העוגייה מכילה מספר סודי שנוצר בהתקפס על ה-IP Address וגם מפתח סודי. השרת לא שומר מפתח זה מכיוון שהוא יכול ליצור אותו שוב מתי שהוא רוצה על סמך המידע. אז אם קליינט שולח SYN ומקבל SYN+ACK מהשרת אז הקליינט יהיה חייב לענות עם ACK שמתאים בדיק להודעתה-N-SYN שנשלחה אליו מהשרת. אחרי זה, השרת בודק את ה-ACK שאמור להיות ה-"סוד" שהופק מ- 2^{32} previously+1.

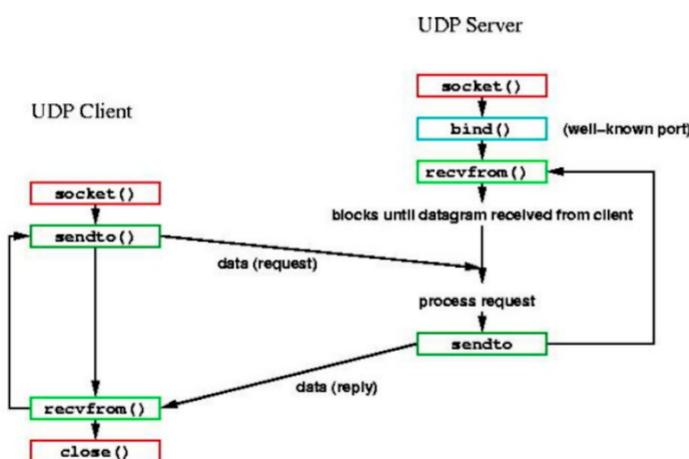


יצירת Socket במשפט C

עבור פרוטוקול TCP נראה את התהליך הבא בדיאגרמה:



עבור פרוטוקול UDP נראה את התהליך הבא בדיאגרמה:



כדי **להתחילה** את הכל צריך לדרש את הספריות הבאות מעל מערכת LINUX:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <arpa/inet.h>
```

קריאה מערך SOCKET הערך חזרה של הסוקט יהיה FILE DESCRIPTOR הערך של הסוקט יהיה INTEGER שפוצן יצירר לאייפה לקרוא ולכתוב.

```
int socket(int af, int type, int protocol);
```

הסביר: •

- af – זה פרוטוקול (לדוגמה עבור IPv4 הערך הוא: AF_INET ועבור IPv6 הוא AF_INET6).
- סוג הסוקט (ה-socket) SOCK_DGRAM יוצר לי סוקט של UDP וה-SOCK_STREAM יוצר לי סוקט של TCP.).
- .UDP: SOCK_DGRAM פרוטוקול שכבת תעבורה: עבור TCP: SOCK_STREAM, עבור UDP: SOCK_DGRAM ייתן את ברירת המחדל: בהתאם לסוג הסוקט.
- ערך 0 – יתנו את ברירת המחדל: בהתקף הערך -1. במידה ולא הוקם socket הפונקציה תחזיר את הערך -1.

דוגמה **לפתיחה** סוקט:

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

בדוגמה יצרנו socket בפרוטוקול IPv4 עם SOCK_STREAM המאפשר לנו לקיים תקשורת אמינה עם ערך 0, ברירת המחדל.

מצד הלוקה נפעיל בפעולות הבאות:
הקמת חיבור עם שרת (במקרה של פרוטוקול TCP, פרוטוקול UDP עובד ללא הקמת חיבור)
 ע"י קריית הפונקציה הבאה:

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

הפונקציה מקבלת INTEGER, כל פונקציה שנכשלה תחזיר לנו -1.

אני אומר לך "תעשה connect" דרך הסוקט sockfd שפתחנו לפני רגע.

תעשה connect לשרת שהkonfigurציה שלו נמצאת בתחום ה-serv_addr struct sockaddr ואננו מעבירים את הגודל של המבנה addrlen.

- sockfd – זה socket descriptor אשר מוחזר על-ידי פונקציה (socket מהשלב הקודם).
- serv_addr – מצביע למבנה sockaddr המכיל כתובת ו포רט של שרת אליו אנו מחברים.
- זה מבנה אותו מעביר לסקט או מעביר לפונקציה שנשתמש במהלך היזון.
- addrlen – גודל מבנה sockaddr בbytes.

מבנה `sockaddr` מכיל מידע מיידע על ה-IP של הסקוט עברו סוגי שונים של סוקטים.

```
struct sockaddr {
    unsigned short sa_family; // address family, AF_xxx
    char sa_data[14]; // 14 bytes of protocol address
};
```

- `sa` - יכול להיות `AF_INET` או `AF_INET6` שיצ hairy לנו איך אני הולך לקרוא את הביביות הבאים.
- `sa_data` - מכיל כתובת ופורט של יעד.

```
// (IPv4 only! for IPv6- please see struct sockaddr_in6 )
struct sockaddr_in {
    short int sin_family; // Address family, AF_INET unsigned
    short int sin_port; // Port number
    struct in_addr sin_addr; // Internet address
    unsigned char sin_zero[8]; // Same size as struct sockaddr
};
```

בثور מתוכנת, אני עובד עם מבנה מסוג `in_sockaddr` אני ממלא את המבנה בקטע קוד מלמעלה,
אחריו זה אני עושה `cast` מהמבנה הזה למבנה שיצרנו לפני שהוא ה-`sockaddr`.

- `sin_zero` - צריך לאפס לפניו כל שימוש על-ידי פונקציה (`memset`).
- גניך והייתי צריך להוסיף עוד מידע כלשהו אז אני צריך עוד קצת זיכרון בצד כדי שלא ננסה
ארכיטקטורה אחרת (נצבע על ערך זיכרון שלא לנו).
- המערך הזה הוא-caן ליתר הבינוח שהסבירנו אבל לא נשתמש בו לאחר מכן.
- `sin_family` - אותו הדבר כמו שראינו במבנה `sockaddr`.
- `port` - צריך להמיר לייצוג של Network Byte Order באמצעות הפונקציה (`htons`) הכוונה היא
לאזה פורט אנו רוצים להתחבר מהצד השני.
- `sin_addr` - צריך להמיר IP כתובות ביינארי (`IPv4`) ודצימלי (`IPv6`) לייצוג רשת על-ידי
הפונקציה (`inet_pton`). (אני פושט מלא בערך זה כתובות IP).

הfonקציה :`memset`

```
void memset(void *str,int c, size_t n)
```

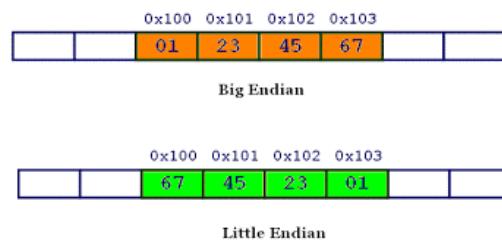
אותה הפונקציה שעזרה לנו לאפס את `sin_zero` שנמצא במבנה `in_sockaddr`.
היא מוחקת חתומים ראשונים של מחזורת `str` אשר המצביע אליה מתקבל כIFORMTER ובמקום רושמת את `c`.

לדוגמה:

```
struct sockaddr_in serverAddress;
memset(&serverAddress, 0, sizeof(serverAddress));
```

אני מעביר ל-`memset` את הכתובות, אומר לה לאפס (הארוגמנט השני) באורך של הכתובת ששלחתי לה (הארוגמנט השלישי).

Byte Order - בגלל שמייצרו המעבדים של המחשבים שלנו לא עובדים באופן אחיד, כמו למשל שיש מעבדים שעובדים עם Big Endian ויש כאלה שעובדים עם Little Endian. כלומר כל דבר שאני מעלה לרשות, שולח או מקבל אליו נתייחס אליו באופן של Big Endian.



- במחשבים ביתיים נשמרים ב-Host Byte Order.

cut נציג פונקציות להמרה (אחד מהם נמצא עוזר לפונקציה `port htons htons_in_port()`) שבסבבנה `in.h`:

```
htons() // host to network short
htonl() // host to network long
 ntohs() // network to host short
 ntohl() // network to host long
```

לדוגמה:

```
serverAddress.sin_port = htons(5000); // short, network byte order
```

אם נרצה להמיר כתובת IP (כתובת זו היא לא INTEGER) لكن יש לנו פונקציה בשם `(inet_pton()`)

```
struct sockaddr_in sa; // IPv4
struct sockaddr_in6 sa6; // IPv6
inet_ntop(AF_INET, "10.12.110.57", &(sa.sin_addr)); // IPv4
inet_ntop(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr)); // IPv6
```

עיקרון אנו שולחים לה את סוג הכתובת למשל IPv4, נעביר לה את הכתובת ב-String קלומר בצורה `*char` והפונקציה לבד תעשה העבודה של לחלק את הכתובת לארבע כתובות וכו' ואחריו כל זה היא תהפוך

אותו ליצוג בינארי ב-[Big Endian](#).
הארגון שלישית אומר לפונקציה שתשמור על השינוי שלו בכתובת שלוחתי לה.
הפונקציה מחזירה 1 - אם יש שגיאה או 0 אם ההמרה לא עברה בהצלחה.
יש לוודא שערך המוחזר על-ידי פונקציה `inet_ntop` גדול מ-0.
נשים לב שנעזרנו בפונקציה זו ב-`addr_in` שבמבנה `sockaddr_in`.

נניח שאנו מקבלים הודעה "մבחן", קיבלנו חבילה מסוימת שהיא ביצוג בינארי ב-[Big Endian](#).
איך נהפוך את הכתובות IP למשהו שרלוונטי לקרואיה בשביבנו?
נשתמש בפונקציה `inet_ntop`.

```
// IPv4:  
char ip4[INET_ADDRSTRLEN]; // space to hold the IPv4 string  
struct sockaddr_in sa; // pretend this is loaded with something  
inet_ntop(AF_INET, &(sa.sin_addr), ip4, INET_ADDRSTRLEN);  
printf("The IPv4 address is: %s\n", ip4);  
  
// IPv6:  
char ip6[INET6_ADDRSTRLEN]; // space to hold the IPv6 string  
struct sockaddr_in6 sa6; // pretend this is loaded with something  
inet_ntop(AF_INET6, &(sa6.sin6_addr), ip6, INET6_ADDRSTRLEN);  
printf("The address is: %s\n", ip6);
```

מצד השרת נועל כך:
כאשר אני פותח סוקט יש לי הרשות, אני צריך לחבר את הסוקט זהה לאיזשהו **פורט** ספציפי שאני אחראי עליו.
ואוכל לקרוא ולכתוב עליו.
לכן יש לו את הפונקציה () **bind** כלומר, קריאת מערכת.

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- זה `socket descriptor` אשר מוחזר על-ידי פונקציה `socket` שייצרנו בשלב הראשון.
- מצביע למבנה `sockaddr` המכיל כתובת ופורט של שרת אליו מחברים.
- גודל המבנה של `sockaddr` בbytes.
- הפונקציה מחזירה 1 - אם יש שגיאה. אחרת, היא תחזיר 0.

הפונקציה `setsockopt` תביא לנו שליטה על ההגדרות של הסוקט.
לפעמים מנסים להפעיל שרת אך קישור ה-`bind` נופל עם הودעה "Address already in use".
הבעיה נובעת מכך שביט קטן של סוקט שהוא מחובר עדין בשימוש זהה תופס פורט.

על מנת לאפשר שימוש חוזר של פורט, יש לנו משהו שנקרא SO_REUSEADDR FLAG שונוכל SO_REUSEADDR ויש עוד הרבה FLAGS להיעזר בהם. ניתן להוסיף את הקוד הבא:

```
int yes=1;
if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof yes) == -1){
    perror("setsockopt");
    exit(1);
}
```

אחריו פעולה bind השרת עובר למצב המתנה (מדובר על TCP בלבד).
אחריו ומדובר עכשו בצד השרת, הוא חייב להאזין במקום מסוים, הוא **מאזין**-IP שלו עם פורט מסוים עם ה-.bind

```
int listen(int sockfd, int backlog);
```

- זה socket descriptor - sockfd
- גודל מקסימלי של תור הבקשות לחיבור.
- תור הבקשות לחיבור מוגבל על-ידי כמות החיבורים שנitin להקים.
- הפונקציהמחזירה 1- אם יש שגיאה. אחרת, היא תחזיר 0.

הוצאת בקשה לחיבור מתוך תור הבקשות (מדובר על TCP בלבד).
ה-listen מוכנה לחיבורים ומי שיקבל את החיבורים ז' פונקציית **ה-accept**.
הוא מקבל את החיבור ותחזיר לנו סוקט שיופיע לחיבור זהה ספציפית (לקליינט).
אני מספק לו sockaddr (מבנה) שבעצם אם אני בתור קליאנט בא להתחבר לשרת זהה אז השרת זהה יאחסן את ה-IP שלו ואת הפורט שלו וכוכב ארגומנט sockaddr *.
ע"י קריית פונקציה:

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- הפונקציהמחזירה sockfd של סוקט חדש שנוצר ומכניסה לבנייה sockaddr מיידע אודות הלוקש שמתחבר (IP ו-Port).
- addrlen - משתנה שלם מוקומי אשר מגדר את גודל המבנה sockaddr.
- הפונקציהמחזירה ערך 1- אם יש שגיאה ואחרת תחזיר 0.

אם נרצה **שלוח** מידע לפי פרוטוקול TCP נשתמש בפונקציה () **send**.
חשוב לציין שם אם אני ב-TCP אז לפני send עשינו connect וכאן יש לנו חיבור ויש לנו לאן לשלוח.

```
int send(int sockfd, const void *msg, int len, int flags);
```

- sockfd - זה socket descriptor דרכו אנו שולחים את המידע.
 - msg - מצביע על המידע שהוא שולחים.
 - len - גודל המידע בbytes.
 - flags - ערך שלו הוא 0.
- הfonקציה מחזירה את כמות הביטים שנשלחה בפועל. כמוות היכולת להיות פחות ממה שרצינו לשלוח.
- יש לוודא כמה נשלח בפועל ולהשלים את מה שחרש.
 - הfonקציה מחזירה 1- אם יש שגיאה.

דוגמא לשילוח הודעה בפרוטוקול TCP:

```
char *msg = "Haim was here!";
int len, bytes_sent;
...
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
```

- הfonקציה send תחזיר לי כמה Bytes נשלחו בפועל והם ישמרו ב-`bytes_sent`.

כשרצנה **לקבל** מידע לפי פרוטוקול TCP :

```
int recv(int sockfd, void *buf, int len, int flags);
```

- sockfd - זה socket descriptor דרכו אנו קוראים את המידע המתתקבל.
 - buf - מצביע ל-buffer אשר קוראים מתוכו.
 - len - הגודל המקסימלי של ה-buffer בbytes.
 - flags - הערך שלו הוא 0.
- הfonקציה מחזירה את כמות הביטים שנקרו או בתוך ה-buffer.
- הfonקציה מחזירה 1- אם יש שגיאה.
 - הfonקציה מחזירה 0 אם החיבור נסגר.

אם נרצה **שלוח** מידע מיל UDP חשוב לציין שאין לנו חיבור קודם ולכן נשתמש בפונקציה הבאה:

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
           const struct sockaddr *to, socklen_t tolen);
```

- **sockfd** - זה socket descriptor דרכו שלחחים מידע.
- **msg** – מצביע למידע אשר שולחים
- **len** – גודל המידע בbytes
- **flags** – ערך של 0
- **to** – מצביע למבנה **sockaddr** המכיל IP ו-Port של יעד
- **tolen** – גודל המבנה **sockaddr**.
- הפונקציה מחזירה את כמות הביטים שנשלחה בפועל. כמות היכולה להיות פחות ממה שריצינו לשולח.
- יש לוודא כמה נשלח בפועל ולהשלים את מה שחרש.
- הפונקציה מחזירה ערך -1 אם יש שגיאה.

אם נרצה **לקבל** מידע מיל UDP חשוב לציין שאין לנו חיבור קודם ולכן נשתמש בפונקציה הבאה:

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
             struct sockaddr *from, int *fromlen);
```

- **sockfd** - זה socket descriptor דרכו קוראים מידע
- **buf** – מצביע לבאפר אשר קוראים בתוכו
- **len** – גודל מקסימלי של באפר בbytes
- **flags** – ערך של 0
- **from** – מצביע למבנה המכיל IP ו-Port של השולח
- **fromlen** – מצביע למשתנה מוקמי אשר יכול גודל המבנה **sockaddr**.
- הפונקציה מחזירה את כמות הביטים שהתקבלו.
- אם הפונקציה מחזירה ערך -1 אז יש שגיאה.
- אם פונקציה מחזירה 0 זה משקף כי חיבור נסגר.

סגירת חיבורים וסגירת סוקט

אחרי שסיימתי את השיחה שלי אזי אני צריך לעשות לסגור את הסוקט כדי שאוכל לשחרר את ה포רט.

```
int close(int sockfd);
```

- socket file descriptor - sockfd •

אם לא נעשה close זה אפשר פרצת אבטחה כאשר אני משאיר פורט פתוח לציבור וכן צריך לסגור.

יש עוד פונקציה שסגורת הגדרות ספציפיות בסוקט מסוים:

```
int shutdown(int sockfd, int how);
```

- ערך של ארגומנט how קובע אופן סגירת חיבור:
 - .1 SD_RECEIVE הערך 0 - סגירת חיבור לקבלת מידע.
 - .2 SD_SEND הערך 1 - סגירת חיבור לשילוח מידע.
 - .3 SD_BOTH הערך 2 - סגירות שני כיוונים - דומה לפונקציה ()close.
- הfonקציה מחזירה 0 אם הפעולה עברה בהצלחה, אחרת, -1.

Raw Sockets

- אפשר גישה למידע לפרוטוקולי רשות מלבד TCP ו-UDP כמו למשל ICMP,IGMP .
- למשל ICMP זה פרוטוקול בקרה, העובדה שלו זה נתנו שליחת הודעה בסיסית, בין היתר האחריות שלו זה לבדוק את הרשות בהודעת ping. עוזר בכך להבין אם הצד השני קיים.
- הפרוטוקולים האלה לא משתמשות בפורטים והם לא נמצאות בשימוש בשכבה התעבורה.
- למשל פרוטוקולים חדשים על IPv4.
- לבנות חבילות משלך (בזהירות).

למה ?Raw Sockets

- גישה לדברים אחרים.
- אפשר ברמת המתקנת אופציות לשחק עם החבילות.
- אפשר למשתמש פרוטוקולים חדשים.
- יכול לבנות ולמלא את ה-Header IP איך שאחננו רצאים (טוב לבדיקה ושליטה).

מגבילות?

- אין אמינות (רק בין קרטיים הרשות).
- אין פורטים.
- אין סוג תקשורת מסוים (יכול לעשות מה שבא לו).
- אין ICMP אוטומטי - איני צריך לבנות אותה בעצמי ולהגיד לה מה לעשות.
- ambil ל' גישה ושירה ל-data raw (המידע שלא עבר עיבוד עדין).
- דורש גישה מנהל.

תפעול ICMP-Raw Sockets

- אתחול סוקט:

```
socket (PF_INET, SOCK_RAW, IPPROTO_ICMP);
```

- אין לנו ()bind בגלל שאין לנו פורט.
- אין לנו ()connection.

- שלוחים מיידית בצורה:

```
sendto (s, buf, strlen(buf), 0, addr, &len);
```

יצירת Raw Socket

- צורת IP Header משלנו בצורה:

```
setsockopt (s, IPPROTO_IP, IP_HDRINCL, &on, sizeof (on));
```

- יכול לבצע .bind and connect

Principles of Congestion Control

מהו עומס? הרשות היא עמוסה אם יש יותר מדי תעבורות של מקורות ומידע ברמה שהרשות לא יכולה להתמודד איתה.

התופעה - השהיות ארוכות וחבילות שהולכות לאיבוד בגלל נניח buffer overflow (הכוונה לראוטרים שה-buffer שלהם מלא מידי).

יש לשים לב שלא מדובר כאן ב-flow control (כי הרבה יש שולחים לא רוצים להעמיס את הרשות).

מה גורם לעומס?

תרחיש 1

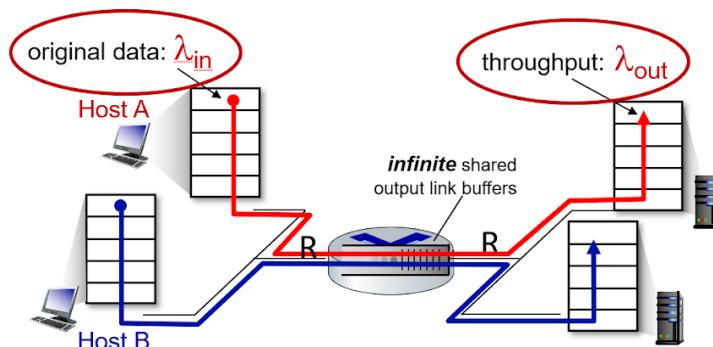
נתבון באירור משמאלי.

יש לנו 2 הוסטיהם: הוסט A משדר לשרת העליון ויש את הוסט B שמשדר לשרת התחתון.

קיים ראטור אחד בינויהם שיש לו תור אינסופי.

קצב הכניסה של הנטב = קצב היציאה ($= R$).

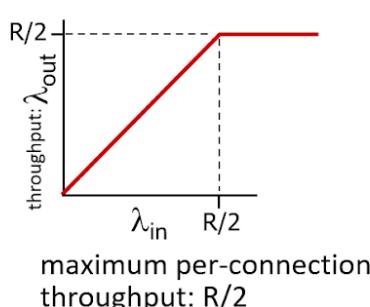
את התכולה המקסימלית הוא R מכל אחד מהכיוונים באירור.



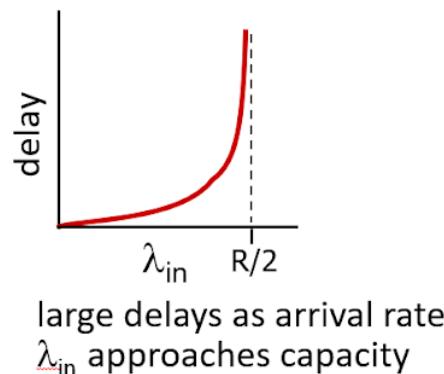
כאשר λ_{in} (כלומר הקצב שבו האפליקציה מייצרת מידע לתעבורה שצורך לשדר) ו- λ_{out} הם קצב של מידע שגם קיימים בשידור ההפוך בציור.

אין צורך בשידורים חוזרים כי ה-buffer (התור) הוא אינסופי (מדובר בתור שמשתף ביחיד את שני השידורים). אז מה קורה אם קצב של λ_{in} גדול?

از בקצב האופטימי והוגן אם הוא λ_{in} יעלה אז λ_{out} יעלה עד $A = R/2$ כי הוא יכול לקבל חצי אז אני יכול להעביר את זהcapacity שלו.



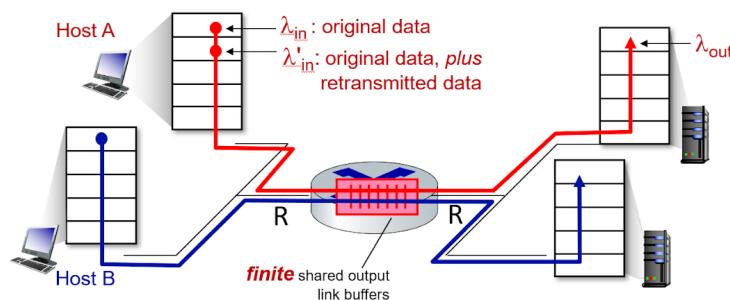
אבל ככל שהוא מגדילים את קצב התעבורה אנחנו יכולים ליצור תורים בנטב ואז השהייה גדלה מאוד וזה לא מצב אופטימי כי זה אומר שעד החבילה מגיעה לצד השני זה יקח הרבה מאוד זמן.



זאת אומרת שההעלאות את הקצב אבל באיזה נקודת זמן העלאה הזרת פוגעת בינו.

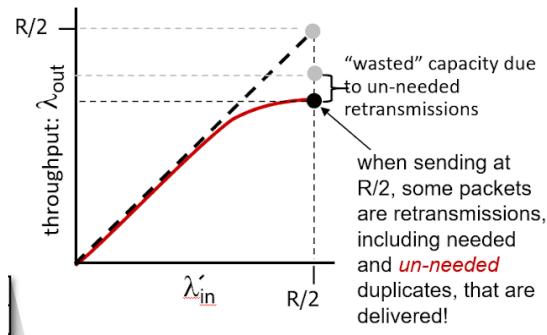
תרחיש 2

אותו המצב כמו מקודם רק שהפעם השרת עם buffer סופי. כזכור שיתכן שהשלוח צריך לעשות שידורים חוזרים או בගל timeout ולכן מבחןתנו יש לנו את ה- אל של האפליקציה כלומר $\lambda_{out} = \lambda_{in}$ ויש את ה- 'ג של שכבת התעבורה המכיל בתוכו גם את מה שהאפליקציה רוצה שאשדר וגם את השידורים החוזרים.



אין לנו אפשרות לדעת מה המצב של הנטב ומאהר והטור של הנטב סופי, יש לחת תשומת לב למצב הטור לפני שנשלח הודעה כלשהי מההוסטים כי אם נשלח הודעה בתור מלא אז ההודעות שלנו ילכו לאיבוד. ככל שנעה את ה- 'ג שהוא מכילה בתוכה 2 דבירים (המידע האמתי וגם את ה- retransmit) אז יהיה לי בעיה כי ה- λ_{out} לא מגיע לא שרצה וזה כי אני מבזבז חלק מהרווח פס שלוי כדי לשדר הודעות חוזרים וזה פוגע ברוחב הפס.

ויתר מזה, חלק מה-capacity הוא בשלל שידורים חוזרים.

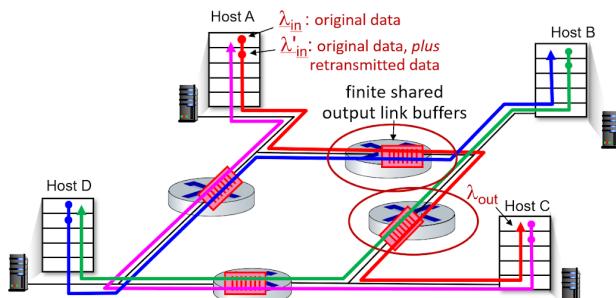


יכול להיות שיבוצע שידור חוזר שלא נדרש בכך כי הטימר שלו היה קצר מדי. ככלומר, עד שההודעה הגיע לשרת החופט ישלח הודעה נוספת כדי לבדוק אם הוא לא קיבל ACK שההודעה התקבלה בהצלחה בזמן שהוא הקצה ולכן מבחינתו נשלח שידור נוסף של הודעה זו.

אז מה המחיר לעומס?

- שידורים חוזרים.
- שידורים חוזרים שלא נצרכו.
- לינקים שעבוריהם מידע שימושכפל (כלומר מקטינים לנו את ה-throughput) וכן עומס הוא לא טוב (בשבילנו).
- שידורים חוזרים כפולים.

תרחיש 3

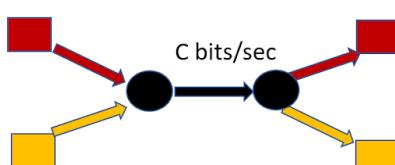


כאשר יש ל-4 הוסטים שמדוברים ביניהם מכל מני מסלולים אדום אם אצל הוסט מסוים הקצב יעלה עם השידורים החוזרים זה יכול לפגוע בקצב של הוסט אחר.

כלומר שיש כאן עוד מחיר, אם הabilities שלוי נזרקות באיזושהי נקודה אד כל מי שהתאמץ לפני להביא אותה לנתקב זהה בזבז את capacity שלו.

לכן התובנות שלנו מתרחישים אלה הם:

- throughput לא יוכל לעبور את capacity הכללי.
- השהיות גדולות ככל שאנו מגע ל-capacity (בעיה - תורמים יותר גדולים ודלילו).
- שידורים חוזרים בגלל אבדות שמודדים לנו את התפוקה האפקטיבית.
- אם יש לנו שידורים נדרשים שלא נדרשים זה עוד יותר פוגע ברשות.
- אם חיבור נזרקה אד כל מי שהתאמץ להביא אותה לנוקודה מסוימת בזבז משאבים.



יהיו 2 streams (אדום וצהוב) עםリンク מסוית (שחור) בעובד בקצב C .

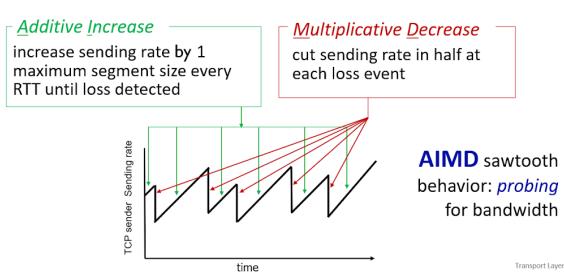
הבעיה כאן שאם התורמים בנחטיבים כמעט ריקים אד זה אומר שאנו לא מנצח את הרשות אבל אם התורמים מלאים אד הדילוי הוא גובהה וזה גם לא טוב.

נרצה לנצל את הרשות טוב יותר אבל להימנע מעומס ברשות.

מה הייתה רצחה לשפר?

- (Efficiency) Goodput = לצמצם ככל האפשר את השידורים החוזרים
- Fairness = איך לחלק את רוחב הפס בצורה ההוגנת ביותר בין כולם
- Low delay = שהabilities לא ימתינו עד אינסוף
- Fast = הקצאות מהירות

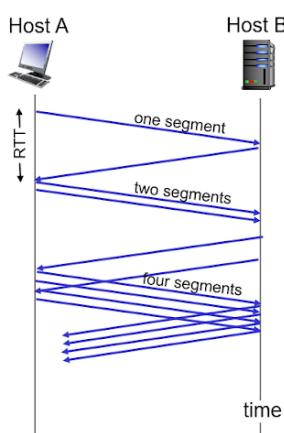
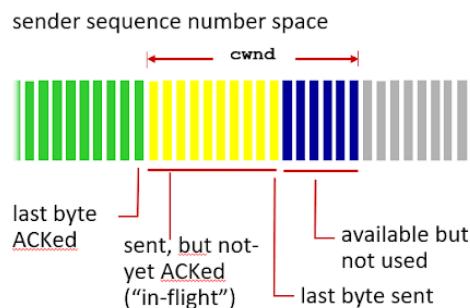
TCP Congestion Control



AIMD

הכוונה שאנו מגדיל את הקצב עד שאני רואה שיש עומס
ואז מקטין את הקצב.
עליה לינארית וירידה אקספוננציאלית.

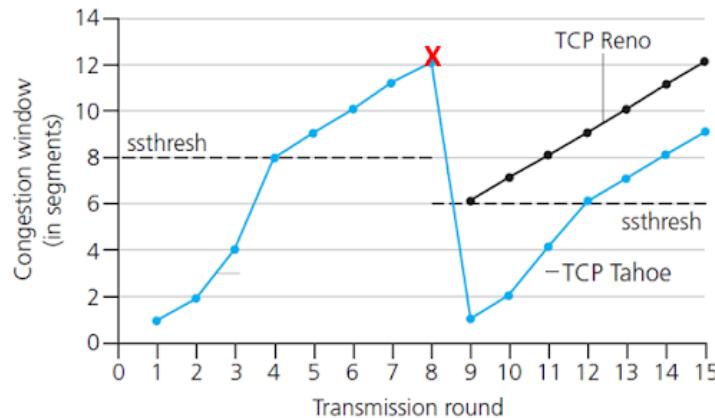
יש לנו חלון ואonto אני מגדיל ומקטין לפי השינוי כלומר ($TCP\ rate \approx \frac{cwnd}{RTT}$ (bytes/sec))
נרצה להתחיל לשדר ואני יודיעים מה גודל חלון הקבלה של הצד השני, لكن נרצה מצד אחד לנצל במקסימום את
חלון הקבלה של הצד השני ובמקביל לא להעמיס על הרשות ובנוסף לשדר כמה שיטור.



Slow Start

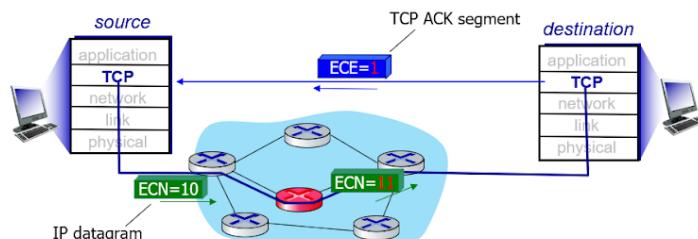
בתחילת השידור אני צריך לעשות תהליך שהוא אקספוננציאלי כלומר כל
RTT אני מכפיל את כמות החלונות שלו (אior משמאלי).

ברגע שאגיע ל-ssthresh אני עברו למצב של עליה לינארית.
כאשר הגיעו למצב של timeout אני יורץ ל-1 אבל ה-ssthresh נחתך בחצי.
אין משמעות יותר ל-ssthresh הקודם אלא יש משמעות ליצור אחד חדש
שידור בחצי מהמצב הקודם וכל לראות את הסיבה בעזרת האירור למטה.



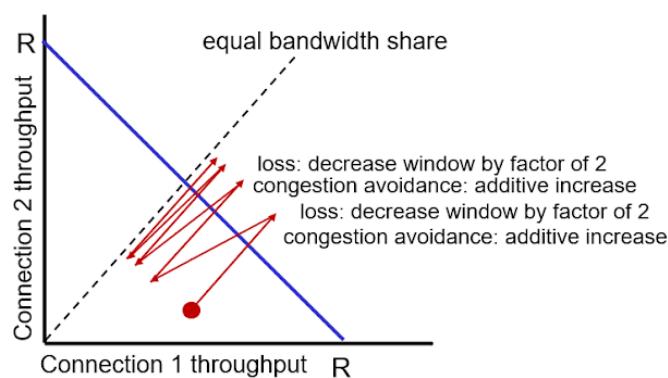
ECN

מהחר והנתבים לא יכולים להגיע ל-header של TCP אז הם מסכנים ב-header של IP, הם משנים ביטים לאלמנט כביש עומס ואז בצד השני (dest) הוא מעתק את הסימן מתוך header של IP datagram לתוכה TCP ACK segment ואז ה-source מקבל ב-source ACK segment ויכול באמצעותו להבין אם אותן segment שאותר עבר באיזשהו נתב שהוא עמוס.



TCP Fairness

אם יש כמה חיבורי TCP על אותו לינק היינו רוצים שהם יתחלקו ביניהם בצורה שווה. באior הבא נראה את הקצב של 2 חיבורי TCP, הכו המכוון זה החלוקה ביניהם. אם נעלם לינארית אז באיזושהי נקודה אנחנו נעmis יותר מדי ואז נרד בחצי מכשה עד שנשים לב שאנחנו מתכוונים לאלמנט שהרווח פס כביכול מוחולק בצורה שווה בין כלום. TCP מאפשר לי ליצור הוגנות בין הרבה חיבורים (לעומת UDP).



שכבות הרשות

שירותים ופרוטוקולים

השכבות ב-system ends-to-end לא מדברות בין עצמם עצמן כמו בשכבות התעבורה והאפליקציה. שכבת הרשות של הנטב מתערב בתהיליך בנגד לשכבות שנלמדו וגם היחסים לוקחים חלק. מצד השולח: עוטף סגננטיים בתוך datagram וועובר לשכבה הקישור (link layer). מצד המქבל: מקבל את ה-datagram ומופר סגננטיים ל프וטוקול בשכבת התעבורה.

- פורטים בשכבה זו הם פיזיים בניגוד לשכבות הקודמות שהם וירטואליים.

פונקציות עיקריות בשכבה

- Forwarding : להעביר פאקטה מ-link input ל-link output .
 - Routing : להבין מהו המסלול בין המקור אל היעד (בעזרת אלגוריתמים).
- נעדר ב-waze כדוגמה שתעזר לנו להבין את ההבדל בין 2 הפונקציות הללו:
ההבדל ביניהם הוא שה-routing זה כמו לדעת מהו המסלול המקורי ליעד וה-forwarding זה שבכל צומת שנגיע הוא אומר לנו לפנות ימינה או שמאליה.

Data Plane

הוא לokaלי, זה פונקציה שהרואוטר עצמה מבצעת.
הפעולה של להעביר את ה-datagram מ-port input אחד לאחרר.

Control Plane

בין הנטבים עצם, הקונטROL פליין עוזר לו לדבר עם נתבים אחרים כדי להבין מהם המסלולים האפשריים מהווט להוסט.

- Traditional Routing Algorithms : ממומש בתוך הנתבים וביניהם לעצם יוצרים מעין פרוטוקולי ניתוב למסלולים האפשריים והקצרים ביותר.
- SDN : יותר מתקדם, לוקחים מכל נתב את ה-control plane שלו ומאחדים את כלם למקום אחד מרכזי ועוד יש לנו ראייה מרכזית על הזרימה (יש לנו שליטה נרחבת יותר על הרשות).

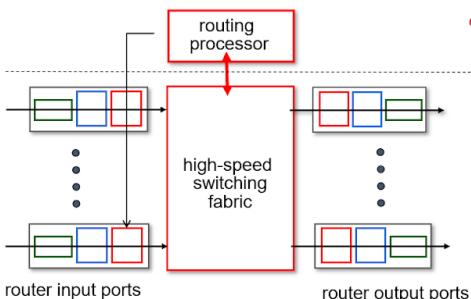
Best Effort Service

האינטרנט עובד בשיטת השירות בשם "Best Effort" אשר מתאר רמת שירות שלא מבטיחה שהמסר אכן יגיע אל יעדו, לא מחיבת העברת datagram מוצלחת, לא מחיבת זמן העברה וסדר העברה ולא מבטיחה שהעברת המסר תעמוד בסטנדרטים מסוימים (כגון השהייה ועיכובים).

יתרונות של שירות זה:

- השירות שלו פשוט, הרשות עצומה ומורכבת וכדי לבקש ממנה שינויים זה יכול ליצור בעיות, מנגנון זה מאפשר פרישה נרחבת של האינטרנט.
- אספקה מספקת של רוחב הפס מאפשרת ביצועים של יישומים בזמן אמיתי (למשל, קול אינטראקטיבי, וידאו) להיות "טובים מספיק" במשמעות "רובה הזמן".
- (מרכזו נתוני, רשות הפעלת תוכן) המתחברים קרוב לרשות הלקחות, מאפשרים לספק שירותי מיקומות רבים. שמחתנו יש מספק משאבים כדי לתת לנו את יכולות האלה דרך שכבת הרשות השנתים שיפורים כמו CDN וגם Data Centers הכל לשירות הלקוח וככה אפשר לאזן גם עומסים.
- שירותי "אלסטטיים" כמו כMOVN עניין ה-TCP עם Congestion Control

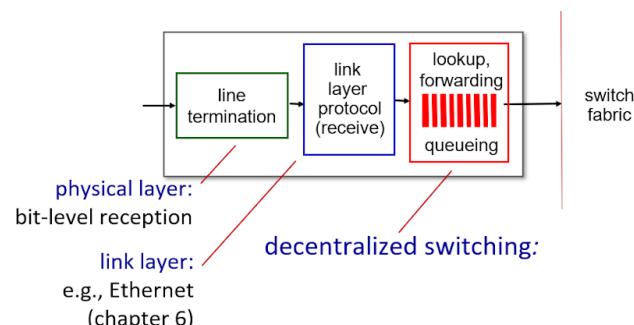
ארQUITטורת הנטב



portukan שוכנס אל הנטב - port Outport שיצא מהנטב ויש ביניהם תריליך (switching fabric). routing processor שמקבל הודעות ששרות לנטב ודואג להחלפת ההודעות בין הנטב זהה לבין שאר הנטבים. הוא גם בונה את ה-forwarding table שנמצא בכל input port שיצא מה-port routing processor אל ה-port האדום). (ה хрז שיצא מה-port routing processor אל ה-port routing processor בריבוע האדום).

ה-plane Data ב-airו זה כל הגוף מתחת לקו המקווקו.

Input port functions



בשלב decentralized switching-header אני אסתכל על איזהם שdots בתוך header ואני אעשה איזהו ביצוע ב-lookup כדי לקבל איזה match ואז אוכל לעשות כל מיני פעולות. לדוגמה, אני אקח מຕך header את ה-IP address, אעשה lookup בטבלת header, destination forwarding וברגע שיש match הפולה שלי תהיה להעביר ל-outport port שכתוב בטבלה.

Forwarding Table

Destination-Based Forwarding

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

כתוב IPI נבנה מ-32 ביטים כלומר 4 קבוצות של 8 ביטים כל קבוצה. בטבלה משמאלי כל שורת ביטים היא כתובות IPI. טווח כתובות ה-IPI שכתוב בשורה הראשונה הוא ¹¹ זהה כי כל הביטים זרים (מיימן לשמאלי) עד השלב המודגש (ה-10 גם חלק מהזרים).

Longest Prefix Matching

זאת אחת השיטות הכי טובות להתקאה בין כתובות.

- אם כל הכתובות משותפות ליציאה אותה (Link Interface) אז למה לא לנסתות לחפש ביניהם משותף?

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 000111** *****	2
otherwise	3

במוקם לעשوت הרבה בדיקות על מלא תווים כמו בשיטה הקודמת אני יכול לעשות שימוש בדיקות בלבד. איפה שיש match אני יוצא דרך ה-interface שלו ואם יש לנו כמה matches אז נחפש את ההתקאה עם התחלית הארוכה ביותר.

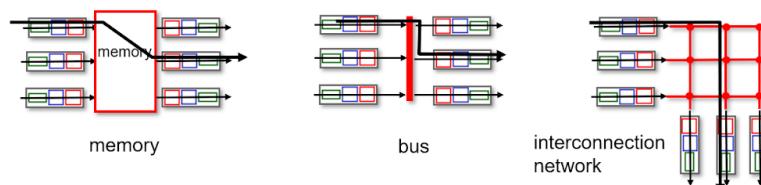
מבחן לוגית אם היתי יכול למיין את ההתחליות מהגדול לקטן אז ברגע שיש התקאה אחת נצא מידית מההתחלית הארוכה ביותר ולא נעשה בדיקות נוספות.

Switching Fabrics

מנגן המעביר חבילות מה-link input אל ה-link output הנכון.

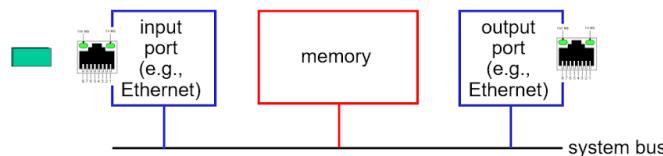
- קצב המיתוג: הקצב שבו אפשר להעביר חבילות מהKEN אל היציאות. לפעמים הקצב נמדד בכמה חיבורי כנישות/יציאות.

ישנם 3 סוגים עיקריים של : switching fabrics



שיטת memory

שיטת שפולה בדור הראשון של הראטורים ועובדת תחת שליטה ישירה על המעבד. החבילות נעות רק למערכת הזיכרון והמהירות מוגבלת על ידי הזיכרון.



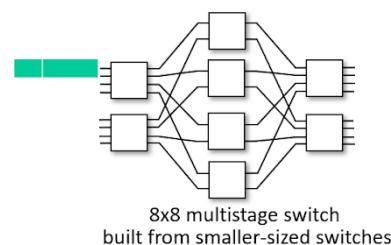
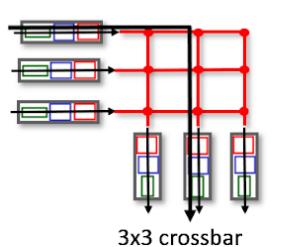
שיטת bus

ה-port input מעבירה את ה-datagram לשירות יציאת הפלט ב-sbus משותף, ללא התערבות בעיבוד הראטור. מכיוון שה-sbus משותף מועברת רק חבילות אחת בכל פעם בס-bus. אם הס-bus תפוס חבילות הנכנסות צריכות להמתין בתור. מהירות המיתוג מוגבלת ברוחב הפס של הס-bus.

שיטת interconnection network

ועוד כמה nets Crossbar, Clos networks מרובי-מעבדים פותחו בתחילת חיבור מעבדים במעבדים.

- Multistage switch: הוא מתג במידה אחה מכמה עמדות של מתגים קטנים. חבילה המגיעה ל-port input נעה לאורך האפיק המאוזן שמחובר אליו עד שהוא פוגש אפיק אנכי שמוביל אותה ל-port output הרצוי. רק אם האפיק האנכי לא פניו, החבילה מתחילה בתור ב-port input.



Input Port Queuing

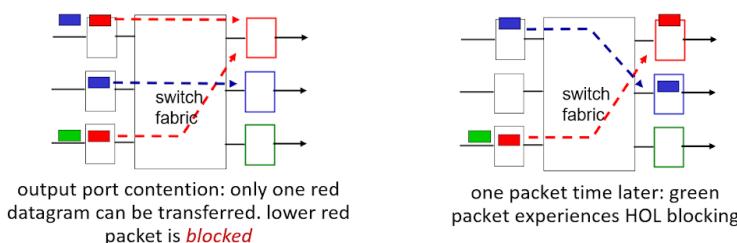
אם ה- switch fabric לא מהיר מדי (קרוב מאוד לזמן הקו) כדי להעביר את כל החבילות המגיעות ללא עיכוב, אז יוצרת תור ב- input ports.

נניח שיש לנו crossbar switching fabric שמקיים את התכונות הבאות:

- כל המהירות של הLINKים דומות.
- חבילה אחת יכולה לעבור מכל input ports ל- output port בויהו באותו זמן שלוקח לקבל את הchnih לה- link input.
- ניהול התור הוא FIFO.

במצב זה, מספר חבילות יכולות לעבור במקביל, כל עוד ה- output ports שלhn שונה. אם יש שתי חבילות שנמצאות ראשונות (בשני תורים שונים) ורוצות להגיע באותו תור ב- Output, אז חבילה אחת תיחסם ותחכה. הרו ה- switching fabric יכול להעביר רק חבילה אחת ל- output port בויהו ביחידת זמן אחת.

בציר הבא יש דוגמא שבה 2 חבילות (אדומות) נמצאות בראשם של שני תורים ב- input והן מיעדות לעבור לאותו output port . לתופעה זו קוראים **HOL blocking** (head of the line).



Output Port Queuing

בעיות

- buffering needed when datagrams arrive from different fabrics faster than the link can forward them.
- the buffer is full up to a certain point and datagrams can be lost due to overflow.
- what order do priorities have on the link?

פתרונות

1. ניהול ה- buffer

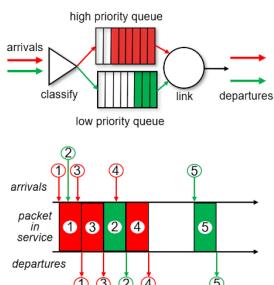
drop: איזה חבילה להוסיף וזרוק כאשר ה- buffers מלאים.

אם אין לנו מקום, נუיף את זנב התור.

עדיפות: זריקה על בסיס עדיפות

2. סימון

נסמן את החבילות בנקודת זמן כלשהי (בערך באמצע תהליך העברה).

3. FIFO**4. סדר עדיפות**

לتحת עדיפות כי אולי יש slow מפסים או חבילות שיותר חשובות מהאחרות. נניח שנרצה להעביר חבילות מסווג קול ויידאו יותר מחבילות מסווג אחר, ולכן נבחר עדיפות מסוימת לסוג חבילות הנכונות. הבעה של עדיפות היא שאנו יוצרים מצב שאנו מושרים את התור העדיף ורק אחרי שהוא מסיים את התור הזה נעבור לתור הבא בתור זהה והוא יוכל ליצור לנו בעיות כי התור עם העדיפות הנמוכה לא מקבל שירות בזמן ולכן יש פתרון שמעביר חבילה אחת כל פעם מכל תור באופן יחסית הוגן (זה האופציה הבאה).

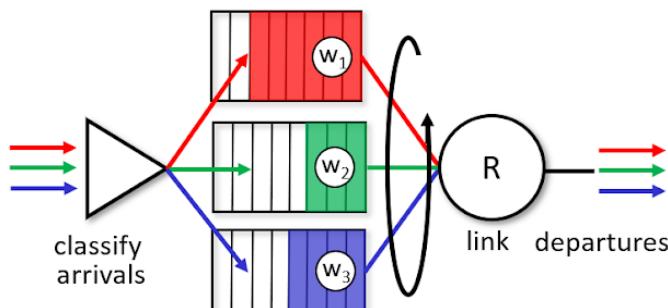
5. Round Robin - RR

נותן פתרון לבעה בפתרונות מספר 4, נعبر כל פעם על חבילה אחת מכל תור באופן יחסית הוגן. זה טוב, אבל החסרונו שלו הוא התפעול שלו כי הוא מבטל את העדיפות. ולכן במקרה הראה נאזרן את עניין העדיפותween RR.

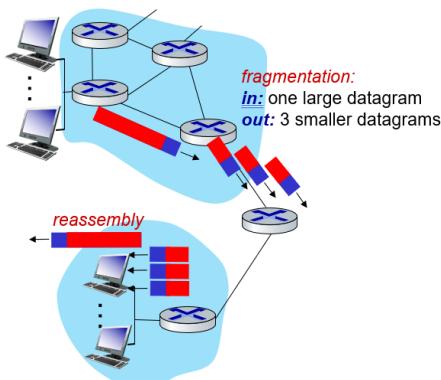
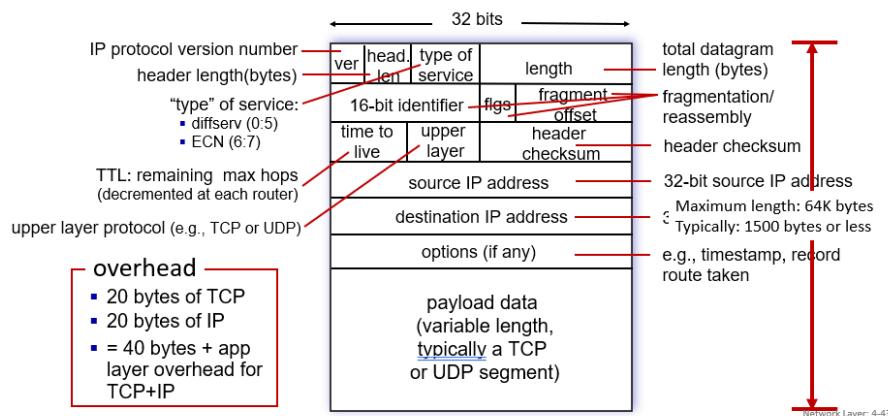
6. משקל הוגן בתור (WFQ)

שילוב בין העדיפות לבין ה-"מעבר בין כolumn" פשוט יותר משלו (ניתן לתרגם את זה לمعין מינימום שירות שכל תור מקבל).

לכל שלב i , יש משקל w_i המקבל את המשקל של כמות השירות בכל סיבוב הביצוע:

$$\frac{w_i}{\sum_j w_j}$$


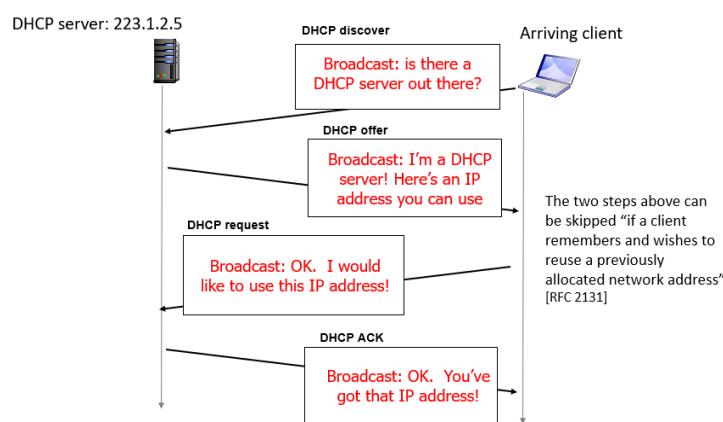
IP Datagram format



נניח ונרצה לשולח איזשהו דאטאגרם בגודל מסוים, יכול להיות שבאחד כבלי הרשת בדרך אל היעד יש כבל שלא יכול לספק את הגודל של הדאטא-גרם זהה בגלל שהוא גדול מדי בשיביל הכבל. ככל לינק יש MTU (גודל תעבורת מקסימלי עבור כל רמת לינק). יש כל מיני סוגים של לינקים - MTU השונים זה מה זה. IP-Datagram גדול מתחולק לחלקיים בתוך הרשת, ככלומר דאטאגרם אחד הופך לכמה דאטאגרמים, והם מורכבים מחדש בחזרה לדאטאגרם אחד שבום מגיעים אל היעד, וכך אפשר למנוע מצב של חסימה בגלל MTU כלשהו. במצב שיטת הפטון זאת לא אמינה כי יכול להאביד מידע בדרך כלל יותר מדי חלוקות...

DHCP

איך מארח מקבל כתובת IP? - ההוסט מקבל באופן דינامي כתובת IP מהרשת כאשר הוא "מצטרף" לרשת. הוא יכול לחפש את הכתובת האחורונה שהשתמש בה, אפשר שימוש חוזר בכתובות (אליאם בוzeitig התנטקות). ה-DHCP תומך בעיקר למשתמשים בנייד שמתחרבים ומתרנסקים מהרשת באופן דינמי.



התהיליך של DHCP:

בהתחלת אני מchipש לגלוות האם יש לי פה שרת DHCP כלומר DHCP discover ברגע שהוא עונה לי בין השורות הוא גם מציע לי איזשהו קונפיגורציה (DHCP offer), אני בוחר אותה כלומר DHCP request והוא מאשר (DHCP ACK).

ה-DHCP יכול להציג לנו יותר מלהזיר כתובת IP בתת הרשת. הוא יוכל לספק לנו את השם וכתובת IP של שרת DNS כלשהו.

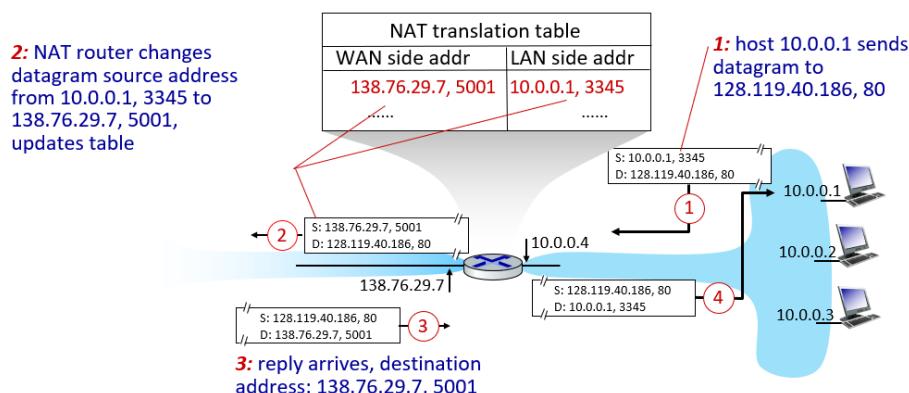
ICANN

את כתובת ה-IP של הרשות מקבלים מספק השירות שיש לו מרחב כתובות שקנה. ספק שירות יכול לאחד את תת הרשותות שמתוחתי לכתובת אחת וכל פניה לכוון תנוטב יישירות מהספק. שיטה זו טוביה בכך שהיא מפחיתה את מספר הכתובות שהרואוים צריכים לזכור.

נניח שאחד מהארגוני שנמצא מאחורי כתובת של ספק שירות מסוים רוצה לעבור לשרת אחר והוא רוצה לשמור על מרחב הכתובות שלו. על מנת לאפשר זאת הרואוור של ספק השירות הישן מוסיף כלל בנוסף לכללים הנוכחיים, שככל חבילה שנשלחת לכתובת הספציפית הישנה של הארגון תנוטב לספק החדש. ספק השירות מקבלים את מרחב הכתובות שלהם מארגון שנקרא ICANN.

NAT

ראouter NAT ממיר כתובת פנימית לכתובת חיצונית. הוא גם נותן יתרונות אחרים לדוגמה אנשים מבוזע לא בקהל יכולם להכנס לרשות הפנימית מבלתי שהרשות הפנימית יקרה איתם קשר. במקרה גם לא יכולים לראות מה המבנה של הרשות הפנימית וכי זה נותן לנו גם את האפשרות להבין מי מדובר עם מי בנושאים שאין יוצא מהרשת. הוא גם יכול להמיר את ה포רטים.



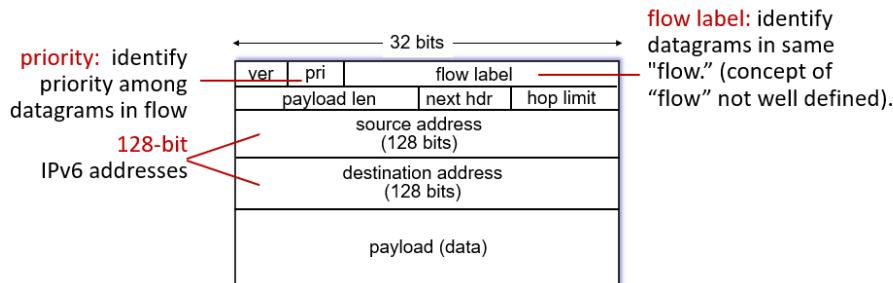
IPv6

המודיבציה הראשונית לפתח גרסה חדשה הייתה בשל העובדה שמרחב הכתובות בננות 32 ביטים הולך ונגמר. את רוב השינויים בין הגרסאות ניתן לראות בפורמט החבילה:

הגדלת מרחב הכתובות: אורך הכתובת גדול מ-32 ל-128 ביט (לכל גרגר חול בעולם יכולה להיות כתובת IP !!).

- Header קוצר יותר המאפשר מעבר מהיר יותר של חבילות.

- Flow Labeling ועדייפות.



פירוט:

- version - כמוון שכן הגרסה היא 6.
- Priority שדה זה דומה לשדה SOS בגרסה 4.
- מספרים בין 0 ל-7 משמשים למנתן עדיפויות כישיש תנוצה צפופה, מספרים בין 8 ל-15 משמשים למנתן עדיפויות כישיש תנוצה לא צפופה.
- Flow Label שדה זה מיועד לאפיין "flow" של חבילות.
- Payloads Length מספר הבתים בחבילה מעבר לאורך הקבוע שהוא 40 בתים (אורק ה- header).
- Next header מזהה פרוטוקול בשכבהعلילונה (TCP או UDP) כמו השדה protocol בגרסה 4.
- Hop limit כמו TTL.
- כתובת המקור והיעד.

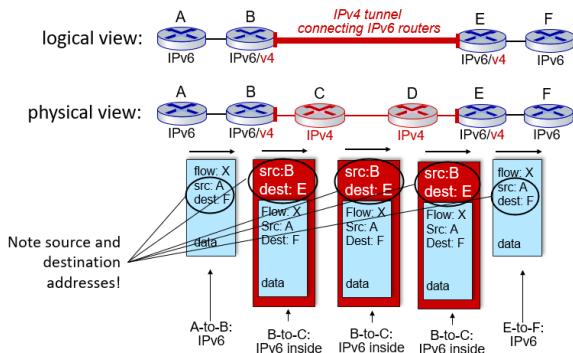
מה אין ב-IPv6?

- checksum
- fragmentation - יכולומר אם מגיעה לראטור חבילה גדולה יותר מהלינק היוצא, הראטור פשוט זורק את החבילה ושולח הודעה שגיאה לשולח.
- במקרה זה השולח יכול לשלוח שוב את המידע בחבילות קטנות יותר.
- Options

מעבר מ-IPv4 ל-IPv6!

איך כל האינטרנט שمبוסס על IPv4 יכול לעבור ל-IPv6?

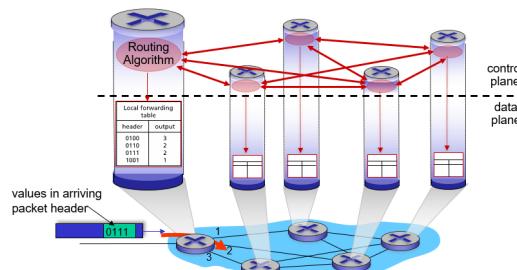
הבעיה היא שמערכות עם הגרסה החדשה יכולות לתקשר עם מערכות מהגרסה הישנה, אולם הפך זה בלתי אפשרי.



הבעיה זו היא הכתובה של קודקוד IPv6 שנמצא בצד השני של המנהרה (קודקוד E). אז החבילה עוברת בין קודקודים IPv4 מבלי שידעו בכלל שם מכילים את כל תוכן החבילה מגעיה לקודקוד E והוא מחלץ את תוכן החבילה שאויה צריך להעביר ושולח אותה לחבילה IPv6.

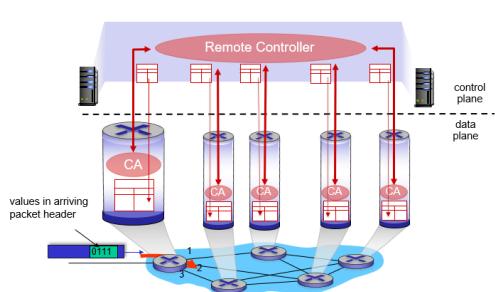
עד עכשיו עברנו על הפונקציה Data Plane וכעת נעבור ללמידה על הפונקציה השנייה בשכבה הרשות שנקראת Control Plane.

ישנם שתי גישות לבניית Control Plane בראשת:



Per-router control plane

מערכת מבוזרת, הנתבים ביןיהם מחליפים הודעות תחת פרוטוקול routing, מתוך התוצר של הפרוטוקול זה הם מעדכנים את טבלאות forwarding שלהם.

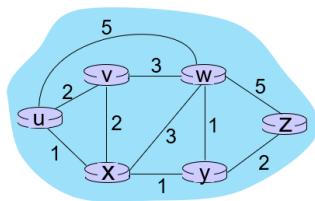


SDN control plane

מערכת מרכזית, אני לוקח את הנתב כביבול את ה-control plane ומרכז במקום אחד ושם מחליט על הטבלאות forwarding ואז אני מוריד את זה לננתבים.

התחלו את הרשות באלמנט מבוזר והרשות הולכת כרגע לאלמנט של ריכוז כמו ה-SDN שמאפשר לנו יותר יכולות מאשר הניתוב ה-per router.

Routing-Protocols



אנחנו רוצים להגיע למצב שאנו רוצים להבין מהו מסלול טוב. מסלול זה אוסף של נתבים שאנו יכול להעביר את החבילות מההתחלת לסוף. מסלול טוב זה אומר מבחינת מחיר, מהירות ופחות עומס. ניתן לחשב על זה כגרף.

מבחינת אפשרויות הינו רוצים למצוא איזשהו אלגוריתם שהוא:

- מתכווד מהר עם שינויים או אישי.
- נראה "**distance vector**" שהוא בעזרת אלגוריתם בلمן-פורד לגבי זה.
- בעל הסתכילות גלובלית כלומר שהוא רואה את כל הרשות בתבב או לא גלובלית כלומר שיכול להתמודד עם ידע מהשכנים שלו בלבד.
- נראה "**link state**" המבוסס על אלגוריתם דיקסטרה.

DIJKSTRA Algorithm

כדי להתחיל את האלגוריתם אני צריך להבין ברשות שלי את כל הטופולוגיה ורק אחרי זה אני מוצא את המסלול ה-"זול" ביותר מהמקור אל השאר.

ברגע שאחד הקודקודים רואה את כולם הוא יכול להרייך את המסלולים הזולים ואז לבנות לעצמו את ה-**forwarding table**.

כਮון שברגע שיש עדכון כלשהו, (משקל שהשתנה) אז צריך לבדוק את כולם ואז להרייך עוד פעם את הכל עם דיקסטרה.

נפטר

- $C_{x,y}$ עלות המשקל בין 2 קודקודים שכנים X ל-Y. הוא יהיה ∞ אם אין שכנות.
- (a) D אומדן שוטף של עלות נתיב בעלות נמוכה ביותר ממוקור ליעד A.
- (a) k צומת קודמת לאורך הדרכ ממקור ל-A.
- A קבוצה של צמתים שהדרך הנמוכה ביותר שלהם ידועה.

Initialization :

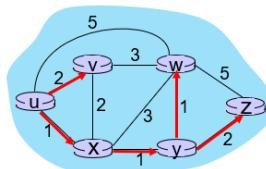
```

 $N' = \{u\}$ 
for all nodes v
  if v adjacent to u
    then  $D(v) = C_{u,v}$ 
  else  $D(v) = \infty$ 

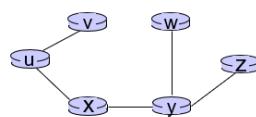
```

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' :
 $D(v) = \min(D(v), D(w) + C_{w,v})$
until all nodes in N'*

דוגמה

resulting least-cost-path tree from u:



resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from u to v directly

route from u to all other destinations via x

סיבוכיות האלגוריתם

שניהם נקודקים. לכל אחד מה-N איטרציות צריך לבדוק את כל הקודקודים. לכן מבחןת מספר ההשוואות נקבע ($O(n^2)$). אך ישנו מימוש הרבה יותריעיל המבצע ($O(\log n \cdot n)$).

הסיבוכיות של ההודעה

אחד הביעויות הכיוון קשות שלו זה העניין שכל נתב צריך לעשות broadcast על הلينקים שלו לכל שאר הראותרים. ככלומר צריכים ליצור לעצמינו איזשהו תהליכי/אלגוריתם שבו אני מסטר על המצב שלי לכולם וכולם מסטרים בכל השאר כדי שנוכל בכלל להתחיל (כי ההתחלת היא שאנו יודע מהו הטופולוגיה). لكن הסיבוכיות היא ($O(n^2)$).

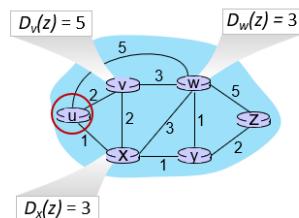
Bellman-Ford Algorithm

זהה $D_x(y)$: עלות המינימלית עבור מסלול מ- x ל- y .
 אזי $\{D_x(y) = \min_v \{C_{x,v} + D_v(y)\}$ כאשר $C_{x,v}$ זה עלות של השכן v של x .

בשונה מאלגוריתם דיקסטרה, אין לו ראייה טופולוגית כלומר הראייה שלו היא רק עבור שכנו כך שאם נרצה להעביר הודעה מ- x ל- y נראה זאת כמו "טיפה מתנפצת על שפת המים" כלומר x יעביר לשכן שלו v ו- v יעביר להלאה לשכניו עד שנגיע ל- y .

בסוף דבר $D_x(y)$ מחזיר לנו את המסלול הזהול ביותר בין x ל- y , אך לא את המסלול עצמו, כלומר בעזרת אלגוריתם זה x לעולם לא ידע את המסלול אליו אלא רק את המשקל המינימלי ביותר המצביע בינויהם.

Suppose that u 's neighboring nodes, x, v, w , know that for destination z :



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Distance Vector Algorithm

אלגוריתם זה מtabס על אלגוריתם בלמן-פורד.
כל קודקוד שולח את המרחק הוקטורי שלו אל שכניו.
כאשר x מקבל מרחק חדש זה, הוא מעדכן את המרחק שלו בעזרת אלגוריתם בלמן-פורד, וכך גם שכניו וכן
הלאה...

אלגוריתם זה עובד באופן איטרטיבי ובו יש תהליך קבוע ומוכנס עבورو כל קודקוד מחכה להודעה שנייה מרחק
משכני, ברגע שהוא מקבל הודעה זאת, הוא מחשב מחדש את המרחק שלו משכני ואז מעדכן לשאר היעדים
שלו וכך גם הם ייחסבו מחדש את מרחוקם וכן הלאה...

از מה ההבדלים בין Link State ו-Distance Vector?

נזכיר כי *Link State* מtabס על אלגוריתם Dijkstra.

סיבוכיות זמן ההודעה (עדכו) -

- עבור LS הסיבוכיות היא $O(n^2)$.
- עבור DV זמן התוכניות משתנה.

מהירות ההתוכניות -

- עבור LS : האלגוריתם הוא $O(n^2)$ וגם ההודעות $O(n^2)$, יכול להיות שינויים בעמדות.
- עבור DV: זמן ההתוכניות משתנה, יכול להיות לנו סיבובים אינסופיים.

מה קורה כאשר הראטור מתקלקל? -

- עבור LS: ראיור לפרסום עלות קישור שגוייה, כל נתיב מחשב רק את הטבלה שלו.
- עבור DV: יכול לפרסם עלות שגוייה של נתיב ("יש לי נתיב בעלות נמוכה באמת לכל מקום"). בנוספ, כל
טבלה של ראיור מנוצלת ע"י ראיורים אחרים: התפשטות שגיאות דרך הרשת.

Autonomous Systems (AS)

נתבים המצוברים לאזורי מكونים AS כלומר Autonomous Systems (AS).

ישנם 2 סוגי של AS:

intra-AS: ניתובים בתוך AS (רשות).

- כל הנתבים ב-AS חיברים להרץ אותו פרוטוקול ניתוב.

נתבים מ-AS אחרים יכולים לזרץ בפרוטוקולו ניתוב אחרים.

.gateway router הצלע המחברת בין ה-AS לבין AS-ים אחרים.

inter-AS: ניתובים בין AS-ים.

פרוטוקולי ניתוב הנפוצים ביותר של intra-AS הם:

- RIP: Routing Information Protocol, מבצע שינוי SVs בכל 30 שניות, פחות שימוש.

EIGRP: Enhanced Interior Gateway Routing Protocol המתבסס על DV.

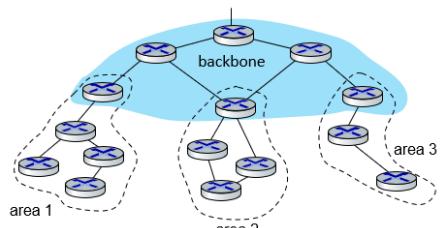
OSPF: Open Shortest Path First הוא משתמש באלגוריתם ניתוב של LS ונכנס לקבוצת פרוטוקולי

השער הפנימי, הפעילים בתחום מערכת אוטונומית (AS) אחת.

כל נתב מציף הודעת LS ישירות מה-IP לכל שאר הנתבים בכל ה-AS.

לכל נתב יש טופולוגיה מלאה ולכן משתמש ב-Dijkstra כדי ליצור forwarding-table.

בנוסף, כל הודעות OSPF מאותות (למנוע חדירה זדונית).



היררכיה OSPF

יש 2 שלבים של היררכיות: האזור המקומי וה-e-backbone.

- הודעת LS מוצפת אך ורק באיזור מקומי כלשהו, או ב-

e-backbone.

- כל נתב בעל ראייה טופולוגית על האיזור שבו הוא נמצא,

הוא יודע לגשת רק לעדדים מוגדרים באזור זה.

BGP : Border Gateway Protocol

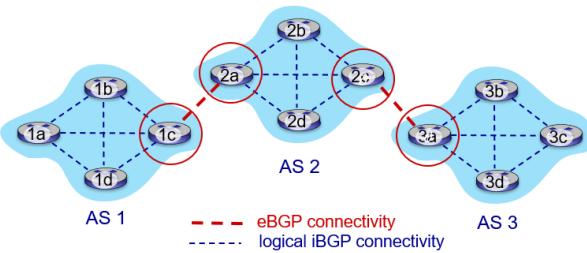
פרוטוקול הניתוב של ה-AS inter-AS הוא BGP : Border Gateway Protocol.

הוא "דבק" שמחזק את כל האינטרנטividually.

הוא מאפשר ל-subnets להודיע על קיומם ולידיעם שאיליהם הם יכולים להגיע לשאר האינטרנט.

פרוטוקול ה-BGP מספק לכל AS התיחסות ל-eBGP ו-

iBGP.



eBGP: מSIG מידע על הנטזות של subnet

מ-ASes שכנים.



gateway routers run both eBGP and iBGP protocols

- **BGP:** להפיץ מידע נגיש לכל הנתבים הפנימיים של AS. קבוע מסלולים "טובים" לרשותות אחרות על בסיס מידע ומדיניות נגישות (**Policy**).

ניתוב ה-BGP בעל 2 תכונות חשובות:

- **AS-PATH:** רשימה של AS-ים דרך עברה הודעתה ה-*prefix*.
- **NEXT-HOP:** מצין נתב פנימי AS ספציפי ל-AS הבא-הופ.

ניתוב מבוסס על מדיניות

gateway מקבל הודעתה ניתוב שמשתמש במדיניות מסוימת כדי לדעת אם לקבל את המסלול או לא. מדיניות AS קבוע אם להודיע על נתיב ל-AS-ים שכנים אחרים.

שני נתבי BGP ("עמייתם") מחליפים הודעות BGP באמצעות חיבור **TCP** קבוע למחצה.

הודעות BGP

- **OPEN** - פותח חיבור TCP לעמית BGP מרוחק ומאמת שליחת עמית BGP כלומר peer.
- **UPDATE** - מפרasm מסלול חדש (או מסלול ישן אם צריך).
- **KEEPALIVE** - משאיר את התחברות פתוחה למקרה של עדכונים ו-ACKs OPEN.
- **NOTIFICATION** - מודיע על שגיאות בהודעות קודמות, גם מיועדת כדי לסגור התחברות.

.intra-domain - בוחר gateway מקומי בעל הערות הנמוכה ביותר עבור **Hot Potato Routing**

ICMP: Internet Control Message Protocol

- ICMP בשימוש על-ידי hosts ונתבים כדי להעביר מידע ברמת הרשת.
- בעיקר בשימוש כדי לבדוק על שגיאות כמו מסרף שלא ניתן להגיע אליו וכו'..
- יכול לדעת מי הנתבים בדרך עד ליעד.
- משתמש ב-ping לבקשתות ותשובות.
- הודעות ICMP מחזיקה אותה IP Datagrams.

חולאם.