# שכבת האפליקציה

## אפליקציית רשת

שכבת האפליקציה אחראית לאפשר לאפליקציות רשת ברכיבי קצה להעביר ביניהם מידע ברשת. אפליקציית רשת מריצה תהליכים (תוכנות) על רכיב הקצה הקובעים את סוג התקשורת בין רכיבי הקצה. יש שני סוגים של תקשורת: תקשורת "לקוח-שרת" (client-server), שבהם תהליך לקוח - הוא תהליך שמתחיל את ההתקשרות, ותהליך שרת - הוא תהליך שממתין ליצירת קשר. ותקשורת "עמית לעמית" (peer to peer), שבה יש תהליכי לקוח ותהליכי שרת ביחד. בפרק זה נדבר בעיקר על תקשורת של לקוח-שרת, ובסוף הפרק נדבר על תקשורת עמית לעמית.

כדי לקשר בין רכיבי הקצה וכדי לדעת לאיזה סוג שירות אנו פונים, האפליקציית רשת מצרפת לכל בקשה כתובת IP של כתובת המקור והיעד, ומספר הפורט של השירות שבו אנו מעוניינים.

ישנם אפליקציות רשת שדורשות שירות תעבורה מסוים. שירות תעבורה יכלול את המרכיבים הבאים:

* **אמינות מידע** - וידוא שהמידע מועבר ליעד הנכון ואין איבוד מידע. לשם כך נשתמש בשכבת התעבורה בפרוטוקול TCP ולא בפרוטוקול UDP.
* **תקורת יצירת קשר** - כמות נתוני תקשורת הנוגעים למידע המועבר.
* **רוחב פס ועיכוב** - יש אפליקציות שדורשות איכות מינימלית של רוחב פס לשם שליחת המידע, ויש אפליקציות שדורשות רוחב פס גבוה.
* **גודל וסיבוכיות** - גודל הקוד והסיבוכיות הם לעיתים קריטיים.

## דף אינטרנט

דף אינטרנט הוא מסמך המוחזק ברשת האינטרנט (World Wide Web), וניתן להצגה תוך שימוש בדפדפן. קבוצת דפים הקשורים ביניהם על ידי קישורים יוצרת אתר אינטרנט. ההנחיות לדפדפן בדבר התוכן שעליו להציג ניתן לו בקוד HTML, אשר בדרך כלל מכיל הפניות לאובייקטים. אובייקט יכול להיות קובץ HTML אחר, תמונת JPEG ג'אווה applet, קובץ אודיו, וכו'.

### כתובת URL

לאתר עצמו ולכל אובייקט הנמצא בו יש כתובת URL (Universal Resource Locator) אליהם ניתן לפנות. בקשה לכתובת URL היא בקשה המופנית אל השרת בפרוטוקול HTTP, והיא מכונה HTTP Request. הכתובת מורכבת מכמה חלקים. לדוגמא: http://en.wikipedia.org/w/index.php?title=Webpage.

1. http - פרוטוקול התקשורת.
2. :// - מפריד את הפרוטוקול משם האתר.
3. en.wikipedia.org - שם האתר (Domain name).
4. /w/ - מיקום האובייקט באתר. זהו מיקום לוגי, לא בהכרח תואם למערכת הקבצים של השרת.
5. index.php - שם האובייקט וסוגו.
6. ? - מפריד בין שם האובייקט לפרמטרים.
7. title=Webpage - פרמטר וערך (במקרה זה, שם הערך שאותו מבקשים).
8. במקרה של פרמטרים נוספים, הם יופרדו באמצעות סימן "&".

במידה וחסר החלק הראשון המציין את הפרוטוקול, הדפדפן בדרך כלל עושה שימוש בברירת המחדל HTTP. במידה וחסר שם הקובץ או מיקומו, שרת HTTP עושה שימוש בקובץ או בתוכנית ברירת מחדל, המוגדרים בשרת.

## פרוטוקול HTTP

פרוטוקול HTTP (Hypertext Transfer Protocol) הוא פרוטוקול תקשורת שנועד להעברת דפי HTML ואובייקטים שהם מכילים (כמו תמונות, אודיו, סרטונים וכו') ברשת האינטרנט וברשתות אינטראנט. הפרוטוקול פועל בשכבת האפליקציה, בתקשורת "לקוח-שרת" (client-server). לקוח HTTP הוא הדפדפן, ושרת HTTP הם שרתי ה-web שהם שרתי התוכן המרכזיים ברשת האינטרנט. הדפדפן מבקש, מקבל ומציג, והשרת שולח אובייקטים בתגובה לבקשות.

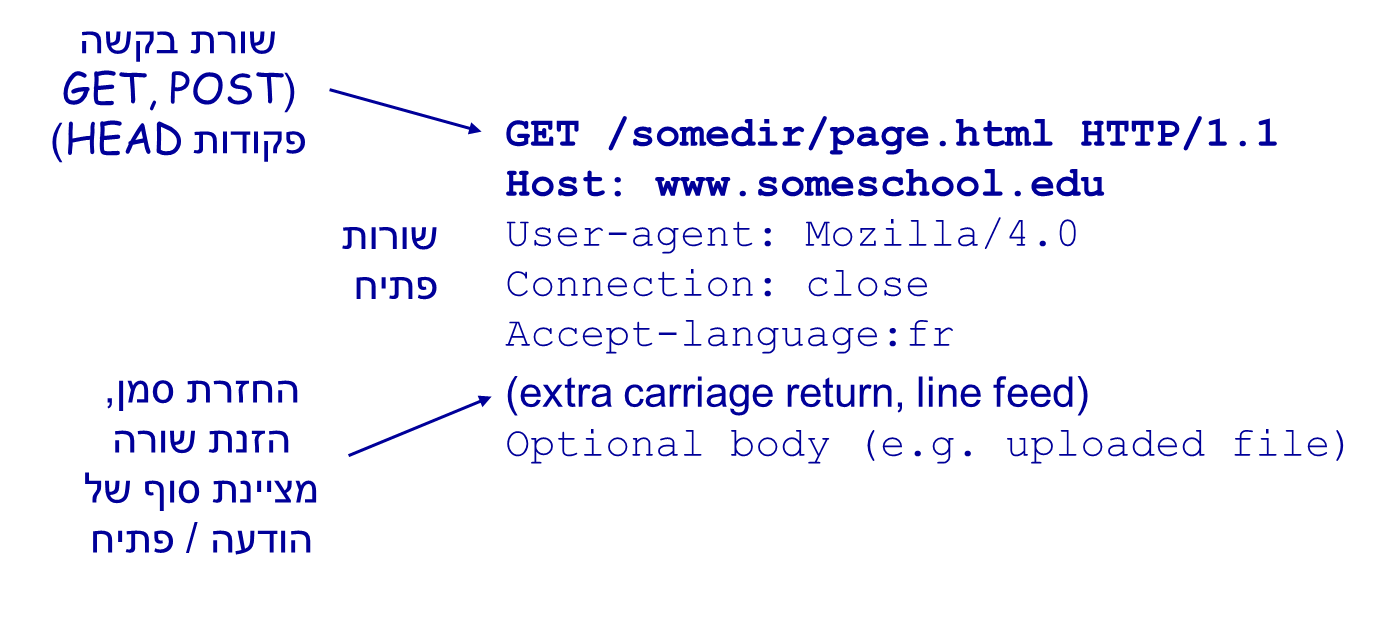
מעצם הגדרתו, פרוטוקול HTTP הוא stateless protocol - חסר מצבים, כלומר אינו שומר מידע על בקשות קודמות של הלקוח. על מנת ליצור תקשורת בין הלקוח לשרת שמבוססת על היסטוריית הבקשות-תשובות שלהם, נעשה שימוש בעוגיות (cookies). לדוגמא, שרת יכול לשתול במחשב הלקוח עוגייה עם אישור שהלקוח התחבר לחשבון מסוים עם סיסמה נכונה, על מנת שלא יצטרך להקיש סיסמה בכל התחברות מחדש לאתר שמתארח על השרת.

### התקשרות HTTP

* ראשית, הלקוח יוצר חיבור (התחלת שיחה) לכתובת ה-IP של השרת ולפורט שבו השרת HTTP נמצא באמצעות פרוטוקול TCP בשכבת התעבורה.
* השרת מקבל חיבור TCP מהלקוח, ושולח ללקוח הודעה חזרה שהוא מחובר.
* הלקוח שולח אל השרת בקשה (request) הכוללת את כתובת האובייקט המבוקש (למשל, דף HTML), ופרטים נוספים על הבקשה ועל הלקוח.
* השרת קורא את הבקשה, מפענח אותה, ושולח ללקוח תגובה (response) בהתאם.
* חיבור TCP נסגר.

### הודעת בקשה

בקשת HTTP מורכבת מכמה חלקים:

1. **החלק הראשון** - מתחיל במילה המתארת את שיטת הבקשה, כמו: GET, POST, HEAD. לכל שיטת יעוד שונה. אחר כך כתובת האובייקט המבוקש. ולבסוף גרסת הפרוטוקול שלפיו מורכבת הבקשה.
2. **החלק השני** - מכיל שדות כותרת המתייחסים לבקשה, ללקוח, או לתוכן הנמצא בגוף הבקשה. הוא פותח בשדה כותרת בשם host, שערכו הוא שם האתר אליו מיועדת הבקשה.
3. **החלק השלישי** - גוף הבקשה (אופציונלי).

### הודעת תשובה

תגובת HTTP מורכבת מכמה חלקים:

1. **החלק הראשון** - מתחיל בגרסת הפרוטוקול שלפיה נבנתה הודעת התשובה. לאחריו קוד מצב תלת ספרתי XXX המציין את התוצאה של ניסיון השרת למלא את הבקשה שנשלחה, והסבר טקסטואלי קצר על משמעותו. אלו הם קבוצות של הקוד מצב:

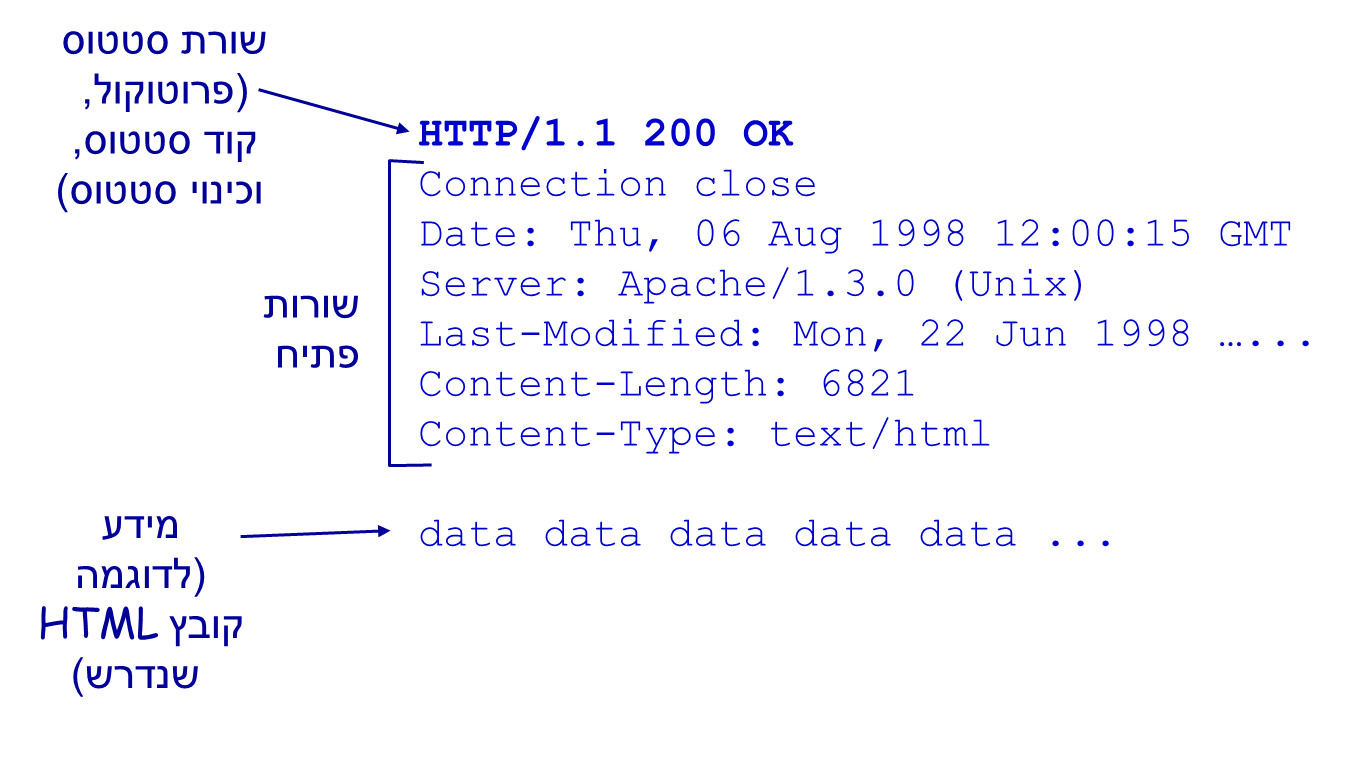
1XX - ההודעה מכילה מידע בלבד.

2XX - הבקשה ששלח הלקוח בוצעה בהצלחה על ידי השרת, והתשובה כוללת את האובייקט המבוקש.

3XX - הבקשה פוענחה בהצלחה, אך מסיבות כלשהן התשובה אינה כוללת את האובייקט המבוקש.

4XX - נמצאה שגיאה בבקשה עצמה – שגיאת לקוח. דוגמא מוכרת "404 Not Found"

5XX - השרת לא הצליח למלא אחר הבקשה כתוצאה מכשל פנימי – שגיאת שרת.

1. **החלק השני** - שדות כותרת המכילים מידע על הודעת התשובה ועל השרת.
2. **החלק השלישי** - גוף הודעה שתוכנו תלוי בשיטת הבקשה ובקוד המצב.

## גרסאות HTTP

RTT (Round Trip Time) הוא זמן שלוקח לשלוח חבילה קטנה מלקוח לשרת ובחזרה. כדי לחשב זמן זה נצטרך קודם לחשב:

1. זמן שידור - כמה זמן לוקח למידע לצאת מהלקוח.
2. זמן התפשטות - הזמן שעובר מהרגע שהביט האחרון עוזב את הלקוח עד שמגיע לשרת.
3. נכפיל ב-2 את זמן השליחה מהלקוח לשרת כדי לשכלל גם את הדרך חזרה.

נסמן:

D - גודל המידע בביטים (Data).

R - כמות המידע שעוברת בשנייה.

L - אורך הקו במטרים.

V - מהירות העברת מידע בשכבה הפיזית. בסיב אופטי שווה למהירות האור.

### HTTP 1.0

בגרסא הראשונה של HTTP, עבור כל בקשת HTTP מתחילים קודם ביצירת חיבור TCP, ואז בקשת אובייקט, ולאחר מכן הקשר נסגר. זהו חיבור לא עקבי (Non-persistent). אם כן, הזמן שלוקח להעביר אובייקט בגרסא HTTP 1.0 הוא פעמיים RTT, אחד עבור TCP ואחר עבור HTTP. אלא שצריך להוסיף את הזמן שידור שהשרת שלח את האובייקט. לכן הזמן להעברת אובייקט אחד הוא: . אם נרצה להעביר 4 אובייקטים (דף ראשי ועוד 3 תמונות) נקבל: .

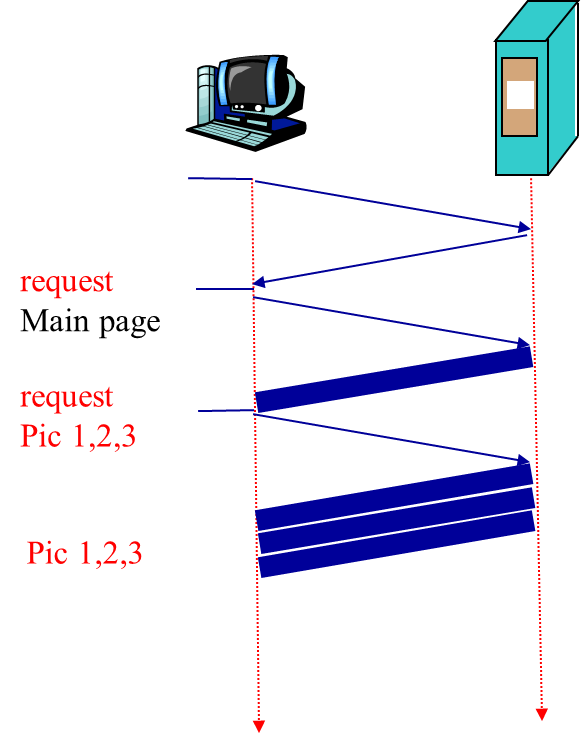
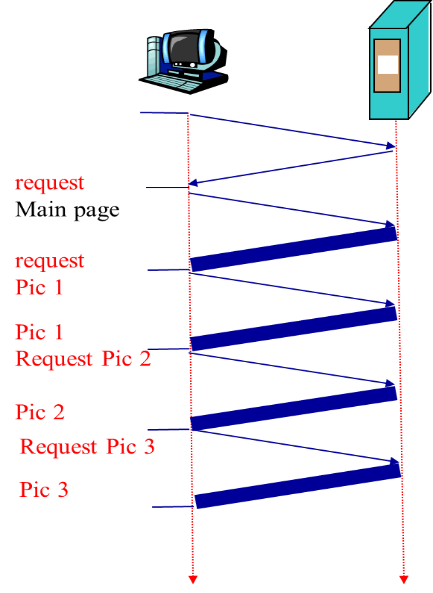
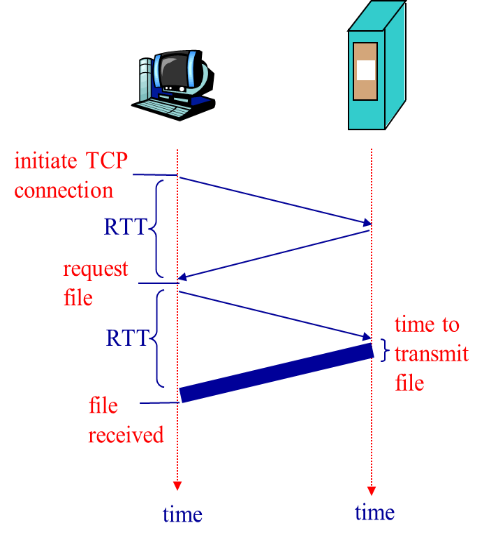
### HTTP 1.1

בגרסא זו לאחר שליחת אובייקט הקשר נשאר פתוח, כלומר יש צורך רק בחיבור TCP אחד בתחילת ההתקשרות. החיבור ייסגר כאשר הלקוח או השרת יעבירו: "Connection: close". אמנם עדיין לא ניתן לשלוח עוד בקשה לפני שהתקבלה תשובה לבקשה הקודמת. חיבור כזה נקרא חיבור עקבי (persistent) ללא pipelining. לכן כדי להעביר 4 אובייקטים (דף ראשי ועוד 3 תמונות) נקבל: .

### HTTP 1.1 p

בגרסא זו ניתן לשלוח בקשה אחר בקשה בלי לחכות לתשובה, וגם הקשר נשאר פתוח. חיבור כזה נקרא חיבור עקבי (persistent) עם pipelining. כדי להעביר 4 אובייקטים (דף ראשי ועוד 3 תמונות), ניצור קודם חיבור TCP ראשוני, ואז נבקש ונקבל את הדף הראשי, ולאחר מכן נבקש את שלושת התמונות בבת אחת. סה"כ נקבל: .

HTTP 1.0



HTTP 1.1

HTTP 1.1 p

## גרסאות חדשות של HTTP

### HTTP 2.0

* בניגוד לגרסאות קודמות, שבהם לאחר שמקבלים דף מבקשים מהשרת את כל האובייקטים בדף זה, ב-HTTP 2.0 השרת יכול לדחוף אובייקטים שהלקוח צריך גם אם לא ביקש אותם.
* מאפשר להגדיר עדיפויות להודעות כך שהודעות דחופות יעברו יותר מהר.
* דוחס את ה-HTTP Header בצורה יעילה ובכך מפחית את העומס ברשת.
* מאפשר למספר דומיינים להעביר נתונים באותו חיבור TCP.
* בגרסאות קודמות ניתן לפתוח באותו פס רחב 6 חיבורים מקבילים של TCP. גרסה 2 מאפשר לפתוח יותר מכך, ובכך לחסוך השהיות ברשת.

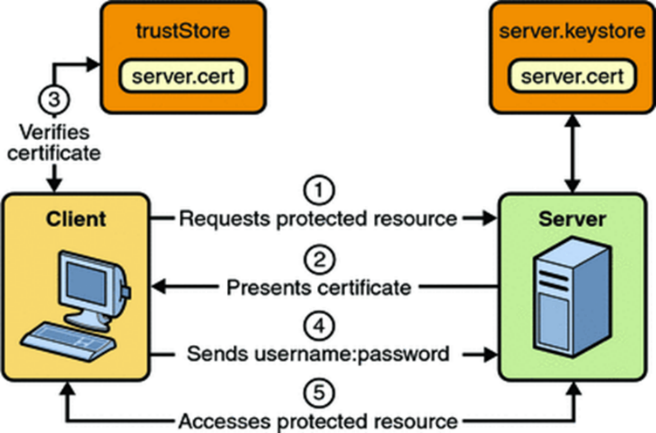
### QUIC

פרוטוקול הבנוי על HTTP 2.0, אלא שבשכבת התעבורה במקום להשתמש בפרוטוקול TCP האמין אך איטי משתמש בפרוטוקול UDP הלא אמין אך מהיר. את מה שמספק TCP שבעיקרו כולל פתיחת קשר ובקרת עומס, QUIC מעלה ומבצע בשכבת האפליקציה, ואף מייעל זאת. ב-QUIC במקום פתיחת קשר של 3 הודעות, כבר בהודעה הראשונה יש בקשת HTTP וגם פתיחת קשר..

## HTTPS

HTTPS (HTTP Secure) אינו פרוטוקול תקשורת בפני עצמו, אלא יישום של פרוטוקול HTTP. הוא זהה לו מבחינת המבנה, אך מורה לדפדפן להוסיף שכבת הצפנה כדי להגן על תעבורת המידע. השימוש ב-HTTPS חשוב במיוחד ברשתות לא מוצפנות, Wi-Fi למשל, בהן כל משתמש באותה הרשת מסוגל לבצע "רחרוח" אחר תעבורת המידע הנשלחת ברשת ולגלות מידע רגיש. יתרונות השימוש ב-HTTPS:

* HTTPS מקנה אימות של זהות השרת.
* יוצר חיבור בין שרת אינטרנט ללקוח שיוצר עמו קשר כנגד התקפת אדם באמצע.
* מאפשר הצפנה דו-כיוונית של תעבורת המידע בין הלקוח לשרת, אשר מגינה מפני האזנת סתר או שינוי של תוכן המידע העובר בין הצדדים.

במילים אחרות, הפרוטוקול מבטיח כי גולש האינטרנט הניגש לאתר כלשהו, אכן יתקשר עם אותו השרת שהתכוון להתקשר עמו.

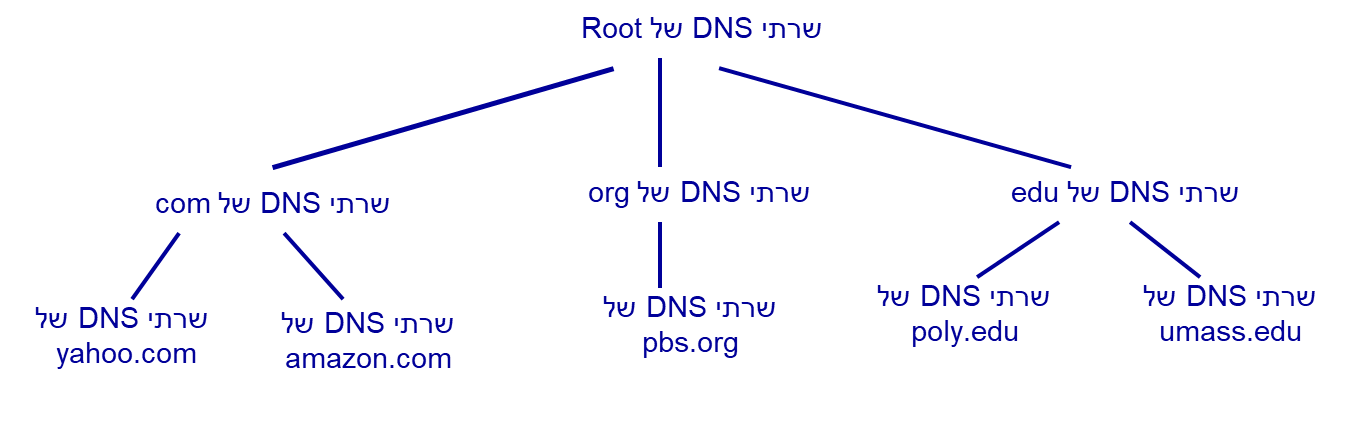
לכל שרת אינטרנט יש "תעודה" (Public key certificate). הרעיון הבסיסי ב-HTTPS הוא שהדפדפן מאמת את התעודה של השרת. בדפדפני האינטרנט שמור מראש, כבר בעת ההתקנה, מידע אשר מאפשר להם לאמת תעודה שנשלחת מהשרת. המקור המנפיק את התעודה לשרת חייב לקבל את אמונו של הדפדפן לצורך האימות, בין אם זהו מקור שמוגדר מראש לדפדפן כאמין או שהמשתמש עצמו הגדירו כאמין.

## פרוטוקול DNS

פרוטוקול DNS (Domain Name System) הוא פרוטוקול המאפשר תרגום של Domain Name (החלק הראשון בכתובת URL) לכתובת IP. שירות זה מאפשר למשתמשי האינטרנט האנושיים לפנות לכתובות אינטרנט בקלות, בלי שיידרשו לזכור כתובת IP מספרית.

כאשר אנו כותבים בדפדפן כתובת URL ולוחצים ENTER, מתחיל תהליך של שליחת בקשת HTTP לפי מודל השכבות. כאשר מגיעים לשכבת התעבורה, פרוטוקול TCP מעוניין לבצע הליך של פתיחת קשר, אמנם בשלב זה הוא לא יכול כי הוא לא יודע מהי כתובת ה-IP שהוא צריך לשלוח אליה. לכן מתחיל תהליך שבו נשלח שאילתה DNS, ובסוף התהליך מוחזר כתובת IP. מנקודת מבטו של הלקוח, ה-DNS הינו קופסא שחורה אליה הוא מזין כתובת URL ומקבל את כתובת ה- IP שלו.

## שרתי DNS - היררכיה

שרת DNS הוא שרת שמחזיק את המידע איזו כתובת IP שייכת לדומיין. אמנם כיוון שכל אדם משתמש בשירותי DNS, לא יכול להיות שיהיה שרת אחד כזה, אלא ישנם המון שרתים המחולקים להיררכיה בעלת שלוש רמות. לכל שרת בהיררכיה יש רשומות (zone files), של השרתים שנמצאים ברמה מתחתיו ואליהם הוא יודע להפנות. כאשר הוא מקבל שאילתת DNS הוא בוחן את הדומיין שמופיע בבקשה, ומפנה אל השרת DNS הבא בהיררכיה שיקרב אותו אל היעד. כך פועל גם השרת שאחריו, וכן הלאה עד שנגיע לשרת DNS שיודע את כתובת ה-IP של הדומיין שביקשנו. ככל שהשרת גבוה יותר בהיררכיה כך הוא מפוצל למספר מחשבים גבוה יותר כדי לא ליצור עליו עומס.

* + 1. **Root Servers** - בראש ההיררכיה נמצאים שרתים המכונים "שרתי שורש". יש בסך הכל 13 שרתי שורש בכל העולם (10 מהם בארה"ב). כל שאילתה DNS מופנית קודם לשרתים אלו. והם מפנים אל השרת המתאים ברמה השנייה.
    2. **TLD** - השרתים ברמה השנייה אחראים על שמות דומיין מהרמה העליונה הנקראים TLD (Top level domains), כלומר על כל השמות דומיין אשר משתמשים בסיומת אינטרנט מסוימת. ה-TLD מחולקים למדינות (כמו il, uk, וכו'), או לארגונים מאוד גדולים (כמו com, edu, וכו'). גם בתוך הרמה השנייה יש היררכיה כלשהי. למשל, שאילתה לשרת מרמת השורש לגבי mywebsite.co.il תופנה משרת ה-root לשרת il אשר אחראי על כלל שמות התחום הישראליים. שרת זה יוכל להפנות לשרת co, שהוא השרת האחראי על כתובות של אתרים מסחריים בישראל. שני שרתים אלו הם שרתי TLD ומסווגים ברמה השנייה. בסוף התהליך שרתים אלו יפנו אל שרתDNS ברמה השלישית.
    3. **Authoritative DNS Server** - שרתי DNS מהימנים אלו הם השרתים האחרונים בהיררכיה (בדוגמא לעיל - mywebsite). והם אלו שבפועל יספקו את כתובת ה-IP. מכיוון שרק הרמה התחתונה מספקת את כתובת ה-IP, כל שינוי של כתובת מטופלת אך ורק ברמה התחתונה, ללא צורך לעדכן את המערכת הכללית של שרתי ה-DNS.

## שרת DNS מקומי

בנוסף על שלושת סוגי השרתים לעיל, קיים שרת נוסף הנקרא שרת DNS מקומי (Local DNS Server). שרת זה אינו שייך להיררכיה שהוצגה לעיל, אך קשור אליה בצורה מלאה. כל ספק אינטרנט מחזיק לפחות שרת DNS מקומי אחד. כאשר לקוח מתחבר אל ספק השירות, ספק השירות מקצה לו כתובת IP של שרתDNS מקומי, (לרוב זה שהכי קרוב אליו). בכל פעם שהלקוח שולח שאילתת DNS, היא מופנית קודם לשרת המקומי, ובמידה ואין לו את הכתובת שמורה ב-cache (מטמון), הוא יעביר את השאילתה ל-root server, אשר יחזיר לו את השרת DNS הבא שהוא צריך לשאול. וכך הלאה עד שישיג את הכתובת ויחזיר אותה ללקוח.

לשרת DNS המקומי שני שירותים:

* משמש כמתווך (proxy) בינו לבין היררכיית השרתים שתיארנו לעיל.
* אוגר אצלו רשימת כתובות אחרונות שביקשנו, כך שאם נרצה לקבלם שוב לא יצטרך לבצע את תהליך ה-DNS מחדש. מאחר ודומיין וכתובות IP משתנים, לכל רשומה יש שדה TTL (Time To Live) שאומר עד מתי יש לשמור את הרשומה.

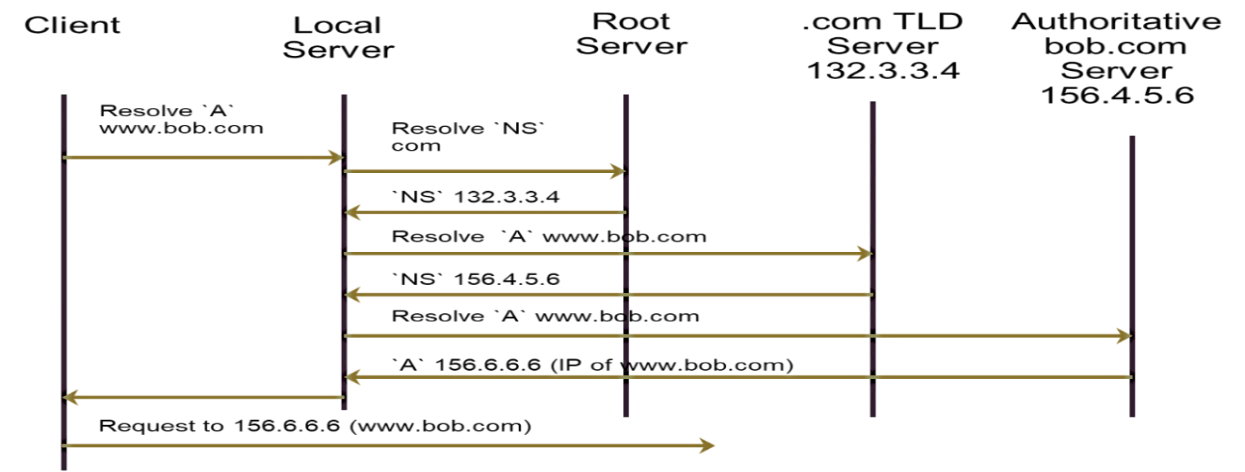
## שאילתת DNS

שאילתת DNS נשלחת מעל פרוטוקול UDP. כל שאילתת DNS מורכבת מ-5 שדות: "שם" - זהו שם הדומיין. "ערך" - בשדה זה יוחזר כתובת ה-IP. "סוג השאילתה" - מגדיר איזה מידע צריך להכיל. "מחלקה" - של שם הדומיין, בווינדוס תמיד מחלקת IN (Internet). TTL - הזמן המותר לשמור את הערך המוחזר משאילתה זו. סוגי השאילתות הם:

**A** - מחזיר את כתובת ה-IP של הדומיין בשאילתה.

**NS** - מחזיר את כתובת ה-IP של השרת DNS המהימן של הדומיין בשאילתה. מכיוון שלדומיין אחד יכולים להיות מספר שרתי DNS, לשאילתה זו יכולים להיות מספר תשובות. כמובן שגם יכול להיות שלשתי דומיינים שונים יהיה אותו שרת DNS.

**MX** - מחזיר את כתובתו של השרת המשמש את הדומיין לקבלה של דואר אלקטרוני.

**CNAME**‏ - "Canonical name", מחזיר את השם הקנוני - שם נוסף לאותו הדומיין.

## תכונות נוספות ב-DNS

### מטמון

כאשר שרת DNS מקבל תשובה לשאילתה הוא שומר אצלו את המידע במטמון. פעולה זו נקראת "DNS Caching". מכיוון ששמות דומיין משתנים וכך גם כתובות IP, מידע זה אינו קבוע אלא נמחק לאחר יומיים שלושה, לפי שדה TTL בשאילתה שבה נשמר מידע זה.

### השהייה

כתוצאה מפעולת המטמון, כאשר עושים שינויים בשם דומיין או בכתובת יש השהייה של עד 72 שעות שלאחריהם האתר יהיה נגיש דרך כל דפדפן. השהייה זו נקראת "DNS Propagation".

### אבטחה

יכולים להיות מספר סוגי התקפות על מערכת ה-DNS:

* 1. Man in the middle - גורמת לכך שהשאילתות לא יופנו אל שרתי DNS מהימנים, אלא אל שרת DNS של התוקף, היפנה אל אתר שיזיק למשתמש.
  2. Cache poisoning - התקפה על המטמון של שרת DNS כך שעבור שאילתות מסוימות יפנה לאתרים שיזיקו למבקשים עד שה-TTL יפוג. זוהי התקפה הרבה יותר מסוכנת שלא מזיקה רק למשתמש אחד, אלא לכל המשתמשים הפונים לשרת DNS זה, וגם יכולה להזיק לשרתי DNS אחרים בהיררכיה.

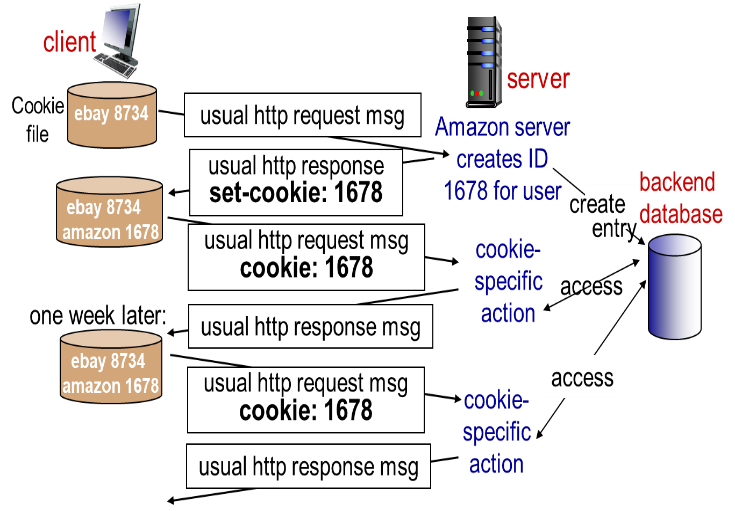
הפתרון להתקפות אלו הוא DNSSEC המסמן באופן מוצפן רשומות קריטיות, כך שמבקש השאילתה יוכל לוודא שהתשובה שהוחזרה היא אכן אמינה. הוא עושה זאת באמצעות שתי שדות נוספים בשאילתה DNS: DNSKEY הנותן מפתח לכל רשומה, ו-RSSIG הנותן חתימה מוצפנת לכל שאילתה.

התקפה נוספת על DNS היא מסוג DDoS - בשנת 2002 התבצעה התקפה על ה-root server שהעמיסה אותו בבקשות, ולא נתנה למשתמשים באינטרנט לבקש ממנו כתובות נוספות. אמנם בפועל לא היתה לזה השפעה על הגלישה באינטרנט, כלומר המשתמשים עדיין קיבלו את כתובות ה-IP הנכונים. זה קרה בגלל שה-zone file של ה-root server נמצא בהמון מקומות, ולכן שרתי ה-DNS המקומיים פשוט ניגשו למקומות אלו ועברו לשלב הבא בהיררכיה. התקפה משמעותית יותר תהיה התקפה על שרת DNS מקומי שלא תאפשר שירותי DNS, או התקפה על שרתי DNS מהימנים שלא יוכלו להחזיר את ה-IP.

## DoH (DNS over HTTPS)

פרוטוקול זהה ל-DNS רגיל אלא שבו הודעות ה-DNS נשלחות בתוך הודעת HTTPS. הסיבה לכך היא כדי למנוע את בעיות האבטחה שפירטנו בסעיף קודם. בפרוטוקול זה לא ניתן לצותת לבקשות ה-DNS של המשתמש כיוון שכל ההודעות מאובטחות ומוצפנות על ידי פרוטוקול HTTPS.

## Cookies

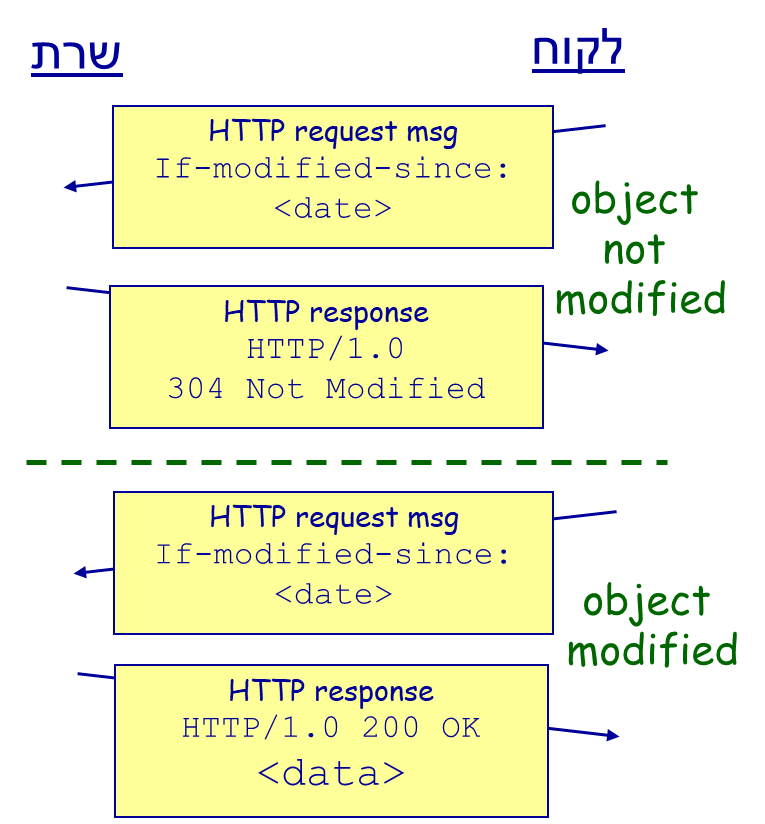
כאשר אנו גולשים באתר מסוים, נרצה לעיתים לשמור מידע כמו: פרטים אישיים, סיסמאות, עגלת קניות, התאמת תכנים ופרסומות, וכו', כך כשניכנס שוב לאתר לא נצטרך להזין את כל המידע מחדש. אך כמו שכבר ציינו לעיל, פרוטוקול HTTP אינו שומר מידע על בקשות קודמות של הלקוח. כדי ליצור תקשורת בין הלקוח לשרת המבוססת על היסטוריית הבקשות-תשובות שלהם (Web Sessions), נעשה שימוש בעוגיות (cookies).

עוגיות עובדות בצורה הבאה: בפעם הראשונה שאנו ניגשים לאתר ושולחים בקשת HTTP, השרת מחזיר תגובה מתאימה ובנוסף מחרוזת אותיות או מספרים הנקראת Cookie, שבאמצעותה האתר יזהה את הלקוח. המזהה לא תלוי בכתובת ה-IP כי היא יכולה להשתנות בהתאם לרשת. הדפדפן ישמור מידע זה בקובץ cookie, ומעתה בכל פעם שנשלח בקשה לאתר זה, הדפדפן יצרף את המידע מקובץ ה-cookie עם ה-header של הבקשה. במקביל השרת מתחיל לאסוף מידע על השימוש של הלקוח באתר ושומר אותו במאגר מידע. כאשר השרת מקבל בקשה עם המידע מקובץ ה-cookie הוא ניגש אל מאגר המידע ושולף ממנו את המידע ומציגו ללקוח.

הבעייתיות בשיטה זו, שקובץ ה-cookie יושב רק על המחשב שבו ביצעו את ההתקשרות הראשונה. אם אותו אדם יבצע התקשרות ממחשב אחר, האתר לא ידע עליו את הפרטים ולא יוכל לחלץ את קובץ ה-cookie מהמחשב השני. עוגייה יכולה להימחק באחת משתי דרכים או שהמשתמש מוחק ידנית את הקובץ בו נשמר המידע, או ש"פג התוקף" של העוגייה.

## Web Cache בשרתי Proxy

שרתי פרוקסי הם שרתים שמתווכים בין הלקוח לשרת. שרת הפרוקסי שומר ב-cache (מטמון) שלו דפים שכבר ביקשנו בעבר. כך שבפעם הבאה שנבקש דפים אלו, נקבל אותם מהשרת פרוקסי ולא מהשרת הראשי כדי שאפשר יהיה להראות אותם מהר יותר בפעם הבאה. שרת פרוקסי דומה ל-L.DNS רק לאובייקטים ולא לכתובות.



מנגנון ה-cache בשרתי פרוקסי עובד כך: הדפדפן שולח בקשה לאובייקט אל שרת הפרוקסי. אם האובייקט אינו נמצא שם, השרת פרוקסי פונה לשרת הראשי ומבקש ממנו את האובייקט. אם האובייקט נמצא שם, אזי השרת פרוקסי מוודא שאכן נמצא אצלו הגירסא המעודכנת של האובייקט. הוא שולח בקשת HTTP מותנית אל השרת הראשי המבקשת את האובייקט רק אם הוא שונה מאז הפעם האחרונה שהאובייקט נשמר אצל הפרוקסי. אם שונה יוחזר האובייקט מהשרת הראשי ויועבר ללקוח. אם לא שונה תוחזר הודעה 304 - Not Modified, והפרוקסי ישלח את האובייקט שנמצא אצלו. נמצא שהפרוקסי מתפקד הן כשרת מול הלקוח, והן כלקוח מול השרת הראשי.

היתרונות בשיטה זו הם:

* חיסכון בתעבורה - מצמצם את הבקשות מהשרת וכך מפחית את העומסים עליו. ובאופן כללי, שרתי פרוקסי מפחיתים את התעבורה הכוללת באינטרנט ובכך משפרים את התפוקה והביצועים.
* מהירות - השרת פרוקסי נמצא קרוב יותר אלינו מאשר השרת המקורי (proximity - קרבה). ולכן בקשה ממנו היא יותר מהירה.

## CDN

נניח שיש לנו תוכן כבד, כמו סרטוני וידאו, שמיליוני משתמשים מרחבי העולם מבקשים לצפות בו באותו זמן, כיצד נשמור את התוכן הזה? אפשרות אחת היא ליצור מגה שרת שיכיל את התוכן ויהיה מספיק חזק לענות לכל המשתמשים. אמנם פתרון זה בעייתי מכמה סיבות: מספיקה בעיה אחת בשרת כדי שכל המשתמשים לא יקבלו את התוכן, לאנשים שרחוקים משרת זה הגישה לתוכן תהיה איטית, ייווצר עומס על קו התעבורה שכן הרבה עותקים זהים נשלחים באותו קו רשת.

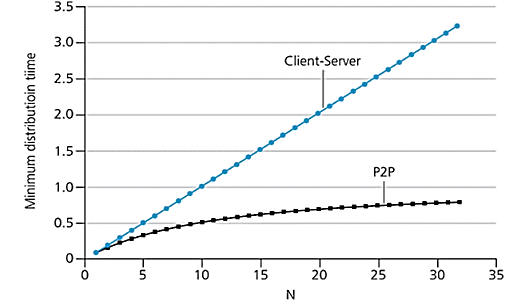
פתרון שני לבעיה זו הוא CDN (Content distribution networks), שהוא שכפול של התוכן לשרתים רבים ברחבי העולם, כך שכל משתמש פונה לשרת הקרוב אליו. בשיטה זו גם אם שרת אחד ייפול ניתן יהיה לבקש את התוכן משרת אחר, הגישה לתוכן יותר מהירה, ואין עומס על הקו רשת. כיצד שרת ה-DNS יודע מהו שרת ה-CDN הטוב ביותר עבור הלקוח? דרך אחת היא לתת את השרת הקרוב ביותר ללקוח (מספר קפיצות בין נתבים הכי נמוך) או השרת הכי פחות עמוס. דרך שנייה היא לתת ללקוח מספר אפשרויות והוא יבחר.

יתרון נוסף ב-CDN, שכאשר מבצעים שינוי בתוכן לא חייב לעדכן את כל שרתי ה-CDN, אלא ניתן לבחור אלו שרתי CDN לעדכן בהתאם לשינוי שנוצר וכך פחות להעמיס על השרתים. כך לדוגמא נוצר יוטיוב שהוא בעיקר בעברית או לשפות אחרות.

## DASH

DASH (Dynamic Adaptive Streaming over HTTP) הוא פרוטוקול בשכבת האפליקציה המאפשר להזרים למשתמש הצופה בסרטון וידאו את האיכות הכי גבוהה שהוא יוכל לקבל ללא תקיעות ובהתחשב בתנאי הרשת (adaptive streaming). בפרוטוקול זה שומרים מטריצה עם איכויות שונות של הסרטון, מזהים את העומסים ברשת, ובהתאם לכך מורידים/מעלים את איכות הסרטון למשתמש לפני שהוא מקבל תקיעות של הסרטון.

## תקשורת עמית לעמית - P2P

בניגוד לתקשורת לקוח-שרת שבה יש הבחנה ברורה בין הצדדים (לקוח מבקש ושרת מספק), בארכיטקטורת P2P (peer to peer) אין הבחנה זו אלא כל אחד מהקצוות מתפקד גם כלקוח וגם כשרת, כלומר כל אחד מהקצוות מסוגל לבקש מידע ולספק מידע, וכן ליזום או לסיים התקשרות.

רשתות P2P משמשות בעיקר לצורך שיתוף קבצים, שבהן כל משתמש מאפשר לשאר המשתמשים להוריד קבצים מהמחשב שלו, ובתמורה מקבל גישה לקבצים במחשבים של משתמשים אחרים. שימוש זה נפוץ בעיקר בתחום זה מפני שהורדת הקובץ בתקשורת P2P טובה יותר מאשר בלקוח-שרת, יתרון זה גדל ככל שיותר עמיתים מחוברים לרשת ה-P2P (ראה תמונה משמאל).

נציג שלושה גישות מרכזיות לתקשורת P2P:

1. **ספרייה מרכזית** - כאשר עמית A מתחבר, הוא מדווח לשרת הראשי על כתובת ה-IP שלו ועל הקובץ שהוא מעוניין בו. השרת משדך לו עמית B שכבר הוריד את הקובץ הזה כך שהוא יוכל להוריד את הקובץ ממנו. לאחר סיום ההורדה, עמית C יוכל להוריד את הקובץ מ-A. גישה זו הייתה בשימוש על ידי Napster בגרסתה המקורית.

בשיטה זו העברת הקבצים אכן מבוזרת אך המידע על מיקום הקבצים הוא ריכוזי. הבעייתיות היא: קריסה של השרת הראשי עלולה להוביל לקריסת המערכת כולה, ועומס רב על השרת הראשי גורר ביצועים פחות טובים.

1. **הצפת שאילתה** - בגישה זו אין שרת מרכזי, אלא כאשר עמית A מבצע חיפוש אחר קובץ מסוים משדכים לו רשימה של עמיתים אופציונאליים. A שולח את השאילתה לכל העמיתים ברשימה (ping). השאילתה עוברת בין כל העמיתים ברשימה, וכל עמית כזה בודק אם הקובץ נמצא אצלו, שולח הודעה חזרה ל-A (pong) דרך חיבור TCP שכבר קיים, וגם מעביר את השאילתה לרשימה שלו. A שכעת מקבל הרבה תשובות על מיקום הקובץ, יכול לבחור מהיכן להוריד את הקובץ. גישה זו הייתה בשימוש בחברת Gnutella.

גישה דומה אך קצת יותר משוכללת מגישה זו היא שכל רשימה היא קבוצה של עמיתים, ולכל קבוצה כזו יש ראש קבוצה, וקבוצות נוספות המחוברות אליה. כל שאילתה מגיעה לראש הקבוצה והוא בודק אם הקובץ נמצא אצלו בקבוצה, וכן מעביר את השאילתה לקבוצה אחרת. גישה זו הייתה בשימוש בחברת KaZaA.

1. **טורנט (Torrent)** - קובץ טורנט הוא מצביע אל שרת טראקר (Trucker), שהוא למעשה שרת המכיל את כלל העמיתים המשתפים קובץ כלשהו ברשת, וכן יודע לכוון את כל התנועה בין המשתמשים המורידים את אותו קובץ. השיתוף בין העמיתים לא מתבצע על כל הקובץ אלא על חלקים ממנו. כל עמית מקבל חלק מהקובץ מעמיתים אחרים **ובמקביל** משתף את החלקים שאצלו. תוכנה שמשתמשת בטורנט מוצאת מקורות דרך שרת, והעברה עצמה של הקובץ מתבצעת ישירות ממשתמש לאחר. תוכנות אלו הן תוכנות שיתוף מהירות יחסית לתוכנות שיתוף אחרות. תוכנה נפוצה מאוד שמשתמש בשיטה זו היא BTorrent.

כאשר עמית A מוריד טורנט של קובץ כלשהו ומפעיל אותו, הוא מתקשר אל הטראקר ומודיע לו על התחברותו. בשלב זה הוא עדיין לא מחזיק אצלו שום חלק מהקובץ. כל x זמן הוא מדווח לטראקר כמה חלקים יש אצלו, כך הטראקר עוקב באופן שוטף אחר כל העמיתים. לאחר ההתחברות, הטראקר שולח לו כתובות IP של רשימת עמיתים חלקיים מתוך כלל העמיתים הנמצאים בטורנט, ויוצר חיבור TCP אל כל עמית ברשימה זו.

בכל רגע נתון עמית A צריך לבקש מהעמיתים שמקושרים אליו את החלקים החסרים לו להשלמת הקובץ. קודם הוא מבקש את החלקים הנדירים ביותר, בכדי לעזור להפוך אותן ללא נדירות. עמית A מנהל מעקב אחרי העמיתים המקושרים אליו ובוחר 4 שקצב ההורדה מהם הכי מהיר. כל 10 שניות הוא מחשב את קצב ההורדה ומשנה בהתאם את העמיתים מהם הוא מוריד. כדי שגם עמיתים אחרים יוכלו לגשת לחלקים שאצל עמית A, אחת ל-30 שניות נבחר עמית באופן אקראי, והם יחליפו ביניהם חלקים. אם שניהם מרוצים הוא יצטרף ל-4 העמיתים מהם הוא מוריד, ואם לא ימצא עמית טוב יותר.

כל חלק מהקובץ שהורד והוא כבר מוכן, הופך להיות זמין למשתמשים אחרים. כלומר, היא חולקת אותו עם שאר המשתמשים. בו בזמן, השרת מפנה משתמשים אחרים אל עמית A, על מנת שיוכלו להוריד ממנו חלקים מהקובץ שהורדתם הסתיימה. לאחר שהקובץ הורד בשלמותו עמית A יכול לעזוב באנוכיות או לשמש כמקור לקובץ שלם (seed).