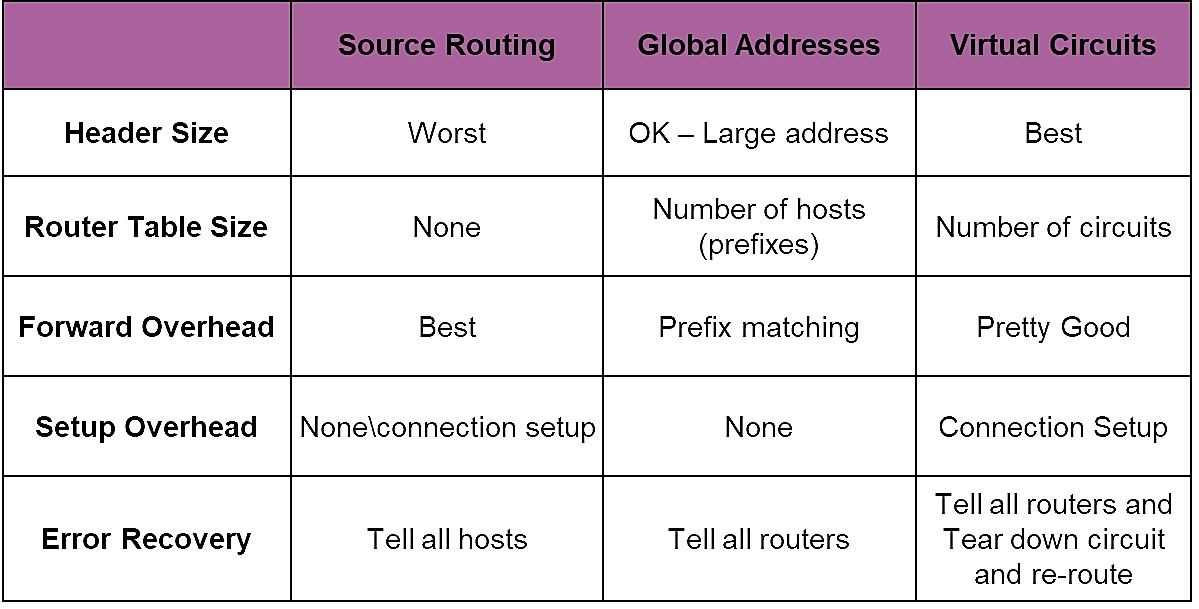
# שכבת הרשת

## שיטות להעברת מידע

שכבת הרשת אחראית על המסלול שהמידע יעבור עד הגעתו ליעד, מהרשת המקומית השולחת ועד לרשת המקומית המקבלת, דרך הנתבים. וכן כיצד כל נתב יודע לאן לנתב את המידע. שכבת הרשת מקבלת את המידע ברכיב המקור משכבת התעבורה ומחזיר את המידע לשכבת התעבורה ברכיב היעד.

יש מספר שיטות לקבוע כיצד יתבצע המעבר של המידע בין שתי נקודות קצה, נפרט על שלושה מהם:

1. **Virtual Circuit** - עוד לפני שליחת המידע שכבת הרשת מגדירה מעגל ווירטואלי, שהוא מסלול מדויק של קווים ונתבים שבהם יעבור המידע. שכבת הרשת עוברת בכל הנתבים במסלול ומעדכנת את הטבלאות ניתוב כך שידעו לאן לנתב את המידע. מעבירים את כל החבילות במסלול המוגדר, ולאחר מכן שכבת הרשת סוגרת את המעגל הווירטואלי ומעדכנת בכך את כל הנתבים. בשיטה זו סדר שליחת החבילות זהה לסדר הגעתן ליעד, החבילות עוברות באותו מסלול, והזמן שבין כל שליחת חבילה הוא קבוע.
2. **Datagram Networks** - בשיטה זו אין מסלולים קבועים, אלא עבור כל חבילה שנשלחת מכניסים אליה את כתובת היעד שאליו היא צריכה להגיע. באמצעות אלגוריתמי ניתוב (Routing Algorithms) ממלאים את טבלאות הניתוב של כל נתב. כאשר החבילה מגיעה לנתב הוא מסתכל בכתובת היעד שלה, ולפי כתובת זו וטבלת הניתוב יודע לאן לנתב את החבילה. מכיוון שטבלאות הניתוב יכול להשתנות כל הזמן, יכול לקרות מצב שחבילות יעברו במסלולים שונים מה שיגרום לחבילות להגיע שלא לפי הסדר בהן נשלחו. לפי שיטה זו עובד האינטרנט!
3. **Source Routing** - לפני ששולחים חבילה מגדירים בתוכה את כל מסלול הנתבים שהיא צריכה לעבור בהם. אמנם כיצד החבילה יודעת להגיע לנתב? עושים זאת באמצעות הצפה של הרשת בשאלה איפה נמצא הנתב. לאחר שמזהים היכן הנתב מכווינים את החבילה במסלול זה. הצפה של הרשת זהו תהליך שלוקח זמן ולכן שיטה זו פחות בשימוש.

## פרוטוקול IP

לפרוטוקול IP יש שני תפקידים עיקריים:

1. להוסיף לכל חבילה את השדות הנצרכים ב-IP Header, ומה הנתבים צריכים לעשות לפי שדות אלו.
2. לקבוע כיצד כל נתב שהחבילה מגיעה אליו יעביר אותה, כך שלבסוף החבילה תגיע ליעד.

נתאר באופן כללי את השלבים שבהם פועלת שכבת הרשת, ובהמשך נפרק כל אחד משלבים אלו ונסביר כיצד הפרוטוקול מבצע אותם:

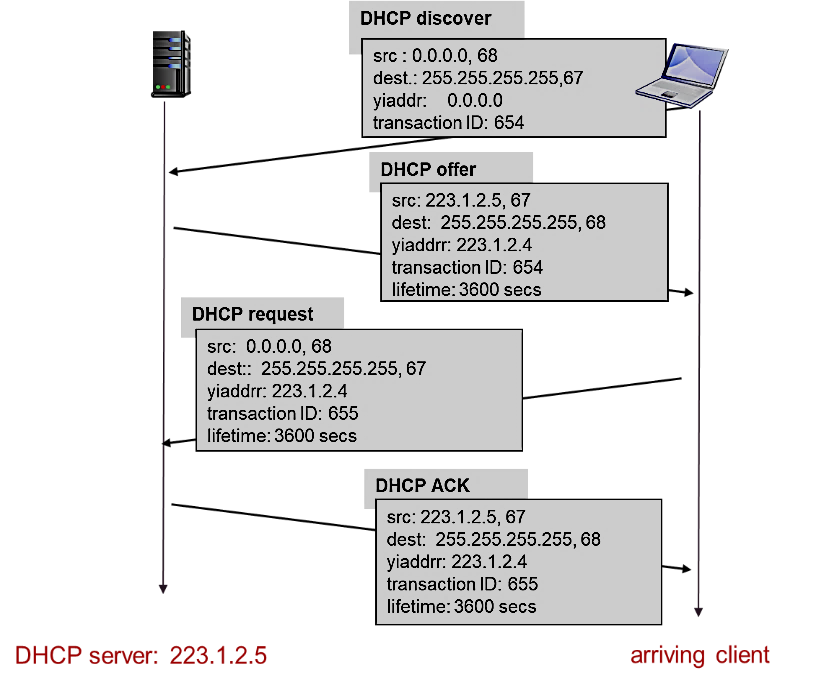
כל רכיב קצה מקבל כתובת IP או על ידי הקצאה ידנית או על ידי שרת DHCP. כאשר רכיב קצה רוצה לשלוח הודעה הוא מאתר את כתובת רכיב היעד באמצעות שרתי DNS. לאחר מכן שכבת האפליקציה בונה את הודעת HTTP ושכבת התעבורה יוצרת חיבור אמין/לא אמין. שכבת הרשת מקבלת את המקטע משכבת התעבורה ועוטפת את המקטע ב-IP datagram, שבו היא מוסיפה את כתובת ה-IP של רכיב היעד ורכיב המקור ושדות נוספים ב-IP Header. בשלב זה נהוג לקרוא ל- datagramחבילה (packet). יש שתי גרסאות לפרוטוקול IP: IPv4, ו-IPv6.

לכל רכיב קצה יש קשר (Link) אל נתב שדרכו הוא מוציא את החבילות לאינטרנט. הגבול בין רכיב הקצה ללינק נקרא ממשק (Interface). כאשר רכיב קצה ברשת רוצה לשלוח הודעה הוא מעביר את החבילה אל הלינק שלו, ומשם זה מגיע לנתב. באמצעות טכניקת NAT הנתב מתרגם את הכתובת הייחודית של רכיב הקצה אל כתובת אחת המשותפת לכל רכיבי הקצה ברשת, ומנתב זה החבילה יוצאת לאינטרנט. כאשר החבילה חוזרת לנתב, מתרגמים חזרה את הכתובת של הרשת אל הכתובת של רכיב הקצה, ומעבירים את החבילה ללינק המתאים המוביל אל רכיב הקצה. ברכיב הקצה שכבת הרשת מפרקת את החבילה ומעלה את המקטע אל שכבת התעבורה.

בכל נתב יש לינק אחד או יותר שמחוברים אליו. הגבול בין הנתב לכל אחד מהלינקים נקרא גם כן Interface. לכל Interface יש כתובת שאליו היא מובילה. כל חבילה שמגיעה לנתב יוצאת דרך לינק אחד בלבד. לכל נתב יש טבלת ניתוב (Routing Table) שבאמצעותה הוא יודע לאיזה לינק לנתב את החבילה. טבלאות הניתוב נבנות באמצעות אלגוריתמי ניתוב.

## DHCP

כדי לשלוח מידע באינטרנט שכבת הרשת צריכה לדעת מהי כתובת ה-IP של השרת, אמנם בנוסף היא צריכה לדעת את כתובת ה-IP של הלקוח כדי לדעת לאן להחזיר את התשובה של השרת. כיצד לקוח מקבל כתובת IP? יש שתי אפשרויות: או שמקצים כתובת IP ידנית דרך הגדרות הרשת במחשב או באמצעות שרת DHCP הנמצא ברשת שאליה התחברנו שהיא הדרך הנפוצה.

לכל רשת יש מספר שרתי DHCP הפועלים לפי פרוטוקול DHCP (Dynamic Host Configuration Protocol), ומטרתן הוא לספק קונפיגורציה ללקוח שהתחבר לרשת. קונפיגורציה כוללת: כתובת IP ללקוח, Subnet Mask, כתובת שרת ה-DNS, כתובת של הנתב ממנו יוצאים (Gateway). באמצעות מידע זה הלקוח יכול לשלוח ולקבל מידע באינטרנט. יש מספר שרתי DHCP כדי שאם אחד ייפול השאר יוכלו לספק קונפיגורציה. אם כל שרתי DHCP מסיבה כלשהי לא עובדים, אזי למחשבים שכבר התחברו לרשת יהיה אינטרנט, אך מחשבים חדשים לא יוכלו להתחבר.

תהליך קבלת הקונפיגורציה מורכב מ-4 הודעות:

1. **Discover** - הלקוח שולח הודעת "DHCP Discover" בפורט ידוע שמשמש ל-DHCP (67 לדוגמא) לכל מי שמחובר לרשת (Broadcast). מטרתה של ההודעה היא לגלות שרתי DHCP.
2. **Offe**r - אחד משרתי ה-DHCP קולט את ההודעה ומחזיר הודעת "DHCP Offer", המציעה ללקוח ששרת זה יכול לתת קונפיגורציה.
3. **Request** - הלקוח מבקש משרת זה קונפיגורציה באמצעות הודעת "DHCP Request".
4. **Acknowledge** - שרת DHCP מחזיר "DHCP Acknowledge", המכילה את כל הקונפיגורציה שהלקוח צריך.

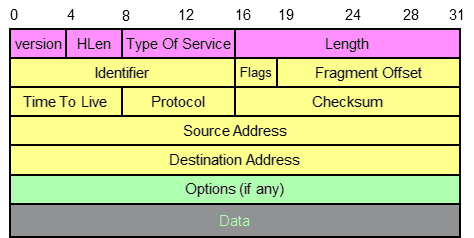
פרוטוקול DHCP זהו פרוטוקול בשכבת האפליקציה המשתמש בשכבת התעבורה בפרוטוקול UDP, זאת מפני שההודעות הן מאוד קצרות ואין צורך בחיבור אמין בין הצדדים.

לכל שרת DHCP יש מספר כתובות מוגבל שהוא יכול להקצות לפי גודל הרשת. הכתובות המוקצות הן לזמן מוגבל, לאחר זמן זה ניתן לחדש את "החוזה" בין הלקוח ל-DHCP.

## IPv4

גרסה 4 של פרוטוקול IP, שבה כל כתובת היא בת 32 סיביות. תאורטית, שיטה זו מאפשרת עד (מעל 4 מיליארד) כתובות שונות, אולם, מכיוון שטווחים גדולים של כתובות שמורים למטרות מיוחדות, מספר הכתובות השמישות קטן יותר. גרסה זו היא הוותיקה והנפוצה, אמנם עם הזמן גרסה 6 נהיית יותר ויותר שימושית. למידע נוסף ראה פרק ב' בסיכום זה.

### IPv4 Header

גודל IP Header סטנדרטי הוא 20 בתים, והגודל המקסימלי של חבילת IP הוא 65,515 בתים (64 קילובייט). ביחד עם פרוטוקול TCP מגיעים ל-40 בתים רק של Headers.

**Version** - מספר הגרסה של פרוטוקול IP. לפי מספר הגרסה הנתב ידע איך לפרש את שאר השדות. כל גרסה ה-Header שלה בפורמט שונה. התמונה לעיל זה הפורמט המתאים ל-IPv4.

**HLen** - גודל ה-Header. רוב החבילות לא יכילו Option ולכן בדרך כלל יהיה 20 בתים.

**Type Of Service (TOP)** - מורה האם לתת עדיפות לחבילה זו במצבים כמו עומס לדוגמא. הנתב לא חייב להתייחס לשדה זה.

**Length** - האורך של כל החבילה כולל ה-Header. מידע זה דרוש בשביל לדעת כמה באפרים להקצות.

**Identifier, Flags, Fragment Offset** - קשור לפיצול והרכבה של IP. יוסבר בסעיף הבא.

**Time To Live (TTL)** - מספר הנתבים המקסימלי בהם החבילה יכולה לעבור מרכיב המקור עד רכיב היעד, כולל הנתב של הרשת. בכל פעם שמגיעים לנתב שדה זה יורד ב-1. אם מגיעים לנתב ושדה TTL=0 הנתב יזרוק את החבילה.

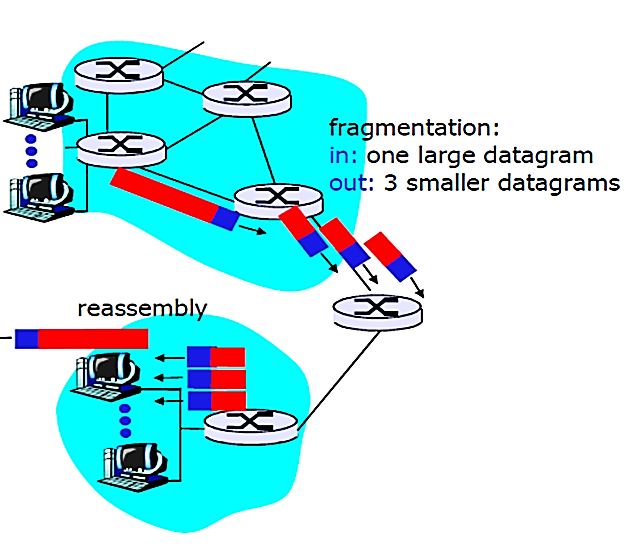
**Protocol** - שדה שמתמלא רק כאשר מגיעים לרכיב היעד. מראה לאן שדה ה-Data יעבור. לדוגמא 6 אומר שה-Data יעבור לפרוטוקול TCP, 17 ל-UDP, ו-1 ל-ICMP.

**Checksum** - בדיקת שגיאות שמידע הגיע כמו שנשלח.

**Source Address** - כתובת ה-IP של השולח. ממולא על ידי שולח המידע.

**Destination Address** - כתובת ה-IP של המקבל. ממולא על ידי שולח המידע.

**Option** - מידע נוסף על החבילה כמו: חותמת זמן, מסלול ספציפי של הנתבים, ערכים לצורך debug, וכו'. יכול להגיע עד ל-40 בתים.

**Data** - כל המידע שהצטבר מהשכבות מעל. ממולא על ידי שכבת התעבורה.

## פיצול והרכבה של IP

כאשר שכבת הרשת מקבלת מקטע משכבת התעבורה היא מפצלת אותה לחבילות. בכל רשת מוגדר הגודל המקסימלי של חבילה - MTU (Maximum Transfer Unit). כאשר חבילה עוברת מנתב אחד לשני, יכולה להיווצר בעיה אם בלינק שהחבילה צריכה לעבור בו גודל ה-MTU קטן מגודל החבילה הנוכחי, כך שהחבילה לא יכולה לעבור בלינק. במצב זה צריך לפצל את החבילה לחבילות קטנות יותר, שכל אחת נקראת fragment (רסיס). כל פרגמנט ממשיך לנוע באינטרנט באופן עצמאי עד שהוא מגיע אל רכיב היעד, לשם כך כל פרגמנט צריך להכיל את כל השדות שתיארנו לעיל בפרוטוקול IP.

כאשר החבילה מגיעה אל רכיב היעד היא מתחברת מחדש לחבילה המקורית, ועולה אל שכבת התעבורה כחבילה אחת. כיצד רכיב היעד יודע איזה רכיבים צריך להרכיב יחד? באמצעות השדות ב-IP Header שדילגנו עליהם בסעיף קודם:

**Identification** - לכל חבילה שיוצאת מרכיב המקור יש ID שונה. כאשר מפצלים את החבילה כל פרגמנט נשאר עם אותו ID. כך יודעים אילו פרגמנטים שייכים לאיזו חבילה.

**Fragment Offset** - מספר הביט הראשון של הפרגמנט מתוך כל הביטים בחבילה המקורית. באמצעות שדה זה רכיב היעד יכול לדעת מה הסדר של כל פרגמנט כדי להרכיב את החבילה חזרה.

**Flag** - IP הוא פרוטוקול לא אמין, לכן איננו יכולים לדעת בוודאות שכל הפרגמנטים יגיעו ליעדם. כדי שרכיב היעד ידע בוודאות שקיבל את הפרגמנט האחרון של החבילה המקורית, מוסיפים דגל לכל פרגמנט. הפרגמנט האחרון מקבל דגל 0, וכל שער הפרגמנטים מקבלים דגל 1. בנוסף קיים דגל "do not fragment" שאם דולק לא ניתן יהיה לפצל את החבילה.

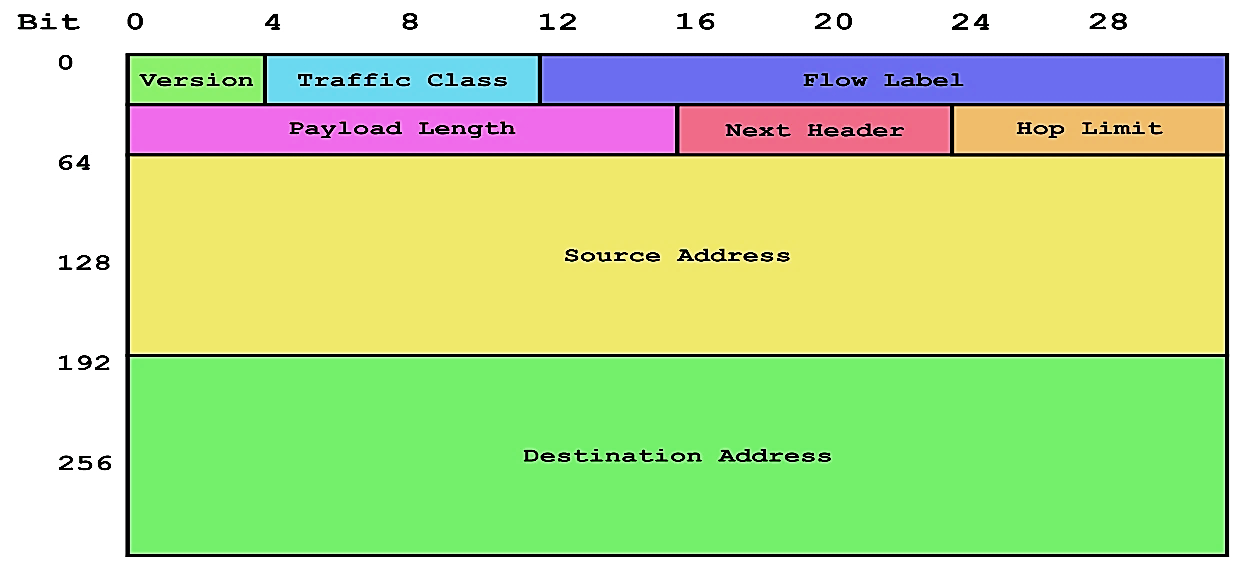
כאשר הגיעו רק חלק מהפרגמנטים של חבילה, רכיב היעד לא ישלח ack על חבילה זו. בשלב מסוים רכיב המקור ישלח את החבילה מחדש, גם את אלו שכן הגיעו, ולכן יש כאן בזבוז.

## IPv6

גרסה 6 של פרוטוקול IP, בה כל כתובת מורכבת מ-8 קבוצות של 16 סיביות, כלומר 128 סיביות. תקן זה מאפשר מרחב עצום של (מספר עם 39 ספרות) כתובות שונות, ופותר את המחסור בכתובות לצמיתות.

בניגוד לגרסה 4 גרסה 6 אינה מאפשרת פיצול של חבילות לפרגמנטים. במידת הצורך נשלחת הודעה לשולח המסבירה את השגיאה. בגרסה 6 אין מנגנון לבדיקת שגיאות Checksum, מהסיבה הפשוטה שגם שכבת התעבורה וגם שכבת הקו בודקות שגיאות, ואין צורך בבדיקה נוספת בשכבת הקו.

### IPv6 Header

גודל פתיח של גרסה 6 הוא 40 בתים. הסיבה לגודל זה היא בעיקר כתובת ה-IP של המקור והיעד המכילות כל אחת 16 בתים. גרסה 6 מחזיקה הרבה פחות שדות מגרסה 4 כדי שיהיה פשוט וקל יותר לעבד את המידע בחבילה ולהעביר אותה הלאה.

**Version** - מספר הגרסה של פרוטוקול IP. לפי מספר הגרסה הנתב ידע איך לפרש את שאר השדות. כאן כמובן הגרסה היא 6.

**Traffic Class** - זהה לשדה TOP בגרסה 4. משמש למתן עדיפות לחבילה כאשר יש עומס ברשת לדוגמא.

**Flow Label** - מזהה החבילות מאותו "flow". flow לא מוגדרת היטב.

**Payload Length** - מספר הבתים בחבילה לא כולל ה-Header.

**Next Header** - זהה לשדה Protocol בגרסה 4. שדה שמתמלא מראה לאיזה פרוטוקול בשכבה העליונה ה-Data יעבור. ב-IPv6 הורחבו האפשרויות לשדה זה כך שיוכל להראות תכונות נוספות.

**Hop Limit** - זהה לשדה TTL בגרסה 4. מספר הנתבים בהם רשאית החבילה לעבור. אם תגיע לנתב עם TTL 0 הנתב יזרוק את החבילה ויחזיר הודעת שגיאה לשולח.

**Source and Destination Address** - כתובות ה-IP של השולח והמקבל. ממולא על ידי שולח המידע.

### IPv6 to IPv4

מערכות מגרסה 6 יודעות לטפל בחבילות מגרסה 4, אך מערכות מגרסה 4 לא יודעות לטפל בחבילות מגרסה 6. כיצד נעביר חבילות אלו במערכות הישנות?

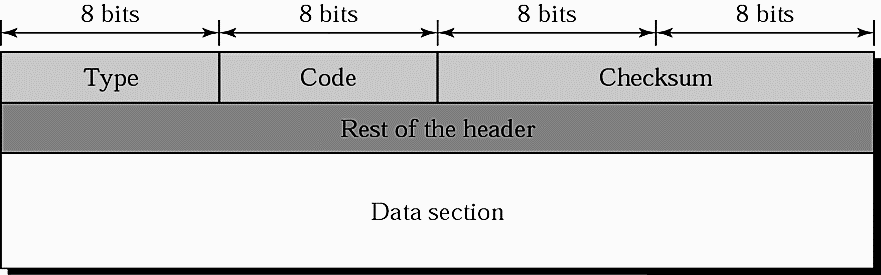
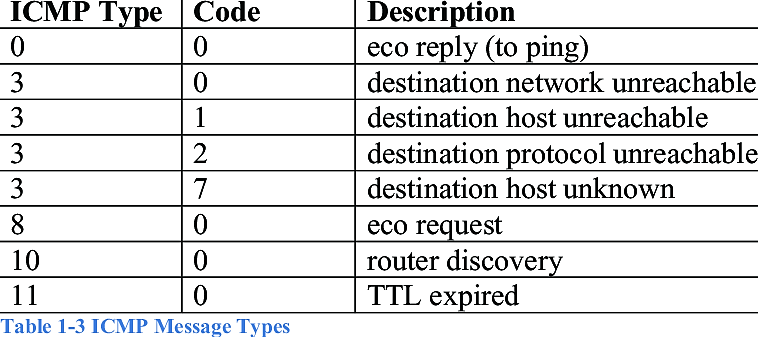
עושים זאת באמצעות **Tunneling**, בשיטה זו לוקחים את כל חבילת IPv6 ומכניסים אותה לתוך ה-Data של חבילת IPv4 חדשה שיוצרים. את החבילה המורכבת מעבירים במערכת הישנה. לאחר שמגיעים אל מערכת חדשה מפרקים את חבילת ה-IPv4 ומחלצים ממנה את החבילה בגרסה 6 וממשיכים להעבירה ברשת כרגיל.

## פרוטוקול ICMP

פרוטוקול ICMP (Internet Control Message Protocol) הוא פרוטוקול עזר המשמש רכיבי קצה ונתבים כדי להעביר הודעות ביניהם ברמת הרשת. לפרוטוקול ICMP יש מספר שימושים, העיקריים הם:

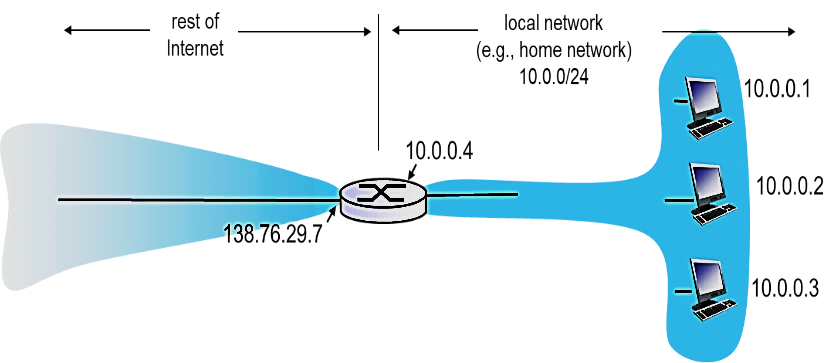
* **דיווח על שגיאות בחבילות של פרוטוקול IP** - כמו: אי יכולת להגיע לרכיב קצה, פורט או רשת, TTL נגמר, וכו'. הרכיב שבו התרחשה התקלה שולח הודעה חזרה לרכיב המקור על השגיאה.
* **Ping** - הודעה הנשלחת בפרוטוקול ICMP, ומטרתה הוא בחינת תקינות התקשורת בין נקודת המקור לנקודת היעד. צד אחד שולח echo request והשני עונה ב-echo reply.
* **Traceroute** - תוכנה שמטרתה למצוא את כל הנתבים במסלול מרכיב מקור לרכיב יעד, וכן את הזמן שלוקח לחבילה להגיע לכל נתב. בהתחלה שולחים חבילה עם TTL=1, החבילה מגיעה לנתב הראשון במסלול, אשר זורק את החבילה ומחזיר למקור הודעת ICMP עם כתובת הנתב, האומרת כי הייתה חריגה ממשך הזמן המוקצב. התוכנה שומרת את הנתונים ואת ה-RTT עד שהודעת ICMP חזרה. לאחר מכן חוזרים על הפעולה עם הודעה שבה TTL=2 המגיעה לנתב השני, וכן הלאה עד שמצליחים להגיע לרכיב היעד.
* **MTU Discover** - גילוי ה-MTU בכל נתב במסלול מרכיב מקור לרכיב יעד. שולחים חבילה עם MTU מקסימלי ודגל "do not fragment" דולק. מחכים להודעת ICMP שתדווח באיזה נתב ובאיזה לינק לא ניתן היה להעביר את החבילה, ומה ה-MTU הדרוש כדי להעבירה. נשלח הודעה חוזרת עם ה-MTU הקודם שקיבלנו, וכך נתקדם עד שנגיע לרכיב היעד.

הודעות ICMP נבנות בשכבת ה-IP, בדרך כלל מחבילת IP רגילה, אשר יצרה תגובת ICMP.‏ פרוטוקול IP עוטף את הודעת ה-ICMP המתאימה בכותרת IP חדשה, כדי לשולחה חזרה לרכיב המקור ששלח את ההודעה המקורית. הודעת ה-ICMP כוללת שתי שדות עיקריים: type המציין את סוג הגורם להודעה, ו-code המציין את הגורם הספציפי להודעה (ראה טבלה), וכן כוללת את 8 הבתים הראשונים בחבילת ה-IP שיצרה את השגיאה.



מכיוון שפרוטוקול IP עוטף את פרוטוקול ICMP, הוא נחשב לפרוטוקול שהוא מעל IP אך עדיין בשכבת הרשת. בדומה ל־UDP, פרוטוקול ICMP אינו מבטיח מסירה. לפרוטוקול ICMP יש גרסה המתאימה לפרוטוקול IPv4 ונקראת ICMPv4, וכן יש גרסה המתאימה לפרוטוקול IPv6 הנקראת ICMPv6.

## NAT

NAT (Network Address Translation) היא טכניקת ניתוב שמשתמש בה הנתב המשותף לכל רכיבי הקצה ברשת. טכניקת NAT אומרת שכל חבילה שמגיעה לנתב עם כתובת ה-IP הפרטית של רכיב הקצה, תוחלף לכתובת IP יחידה המשותפת לכל רכיבי הקצה ברשת. כך שכל הודעה שנשלחת מאותה רשת מגיעה ליעד עם אותה כתובת IP! משתמשים ב-NAT משתי סיבות:

1. **צמצום בכתובת IPv4** - שהרי במקום שלכל רכיב קצה יינתן כתובת IP חיצונית, כל רכיבי הקצה מיוצגים ככתובת אחת בלבד. האפשרות לחבר רכיבי קצה רבים לאינטרנט באמצעות כתובת אחת חוסכת הרבה כסף לארגונים וחברות.
2. **אבטחה** - ניתן להסתיר את מבנה הרשת הפנימית מפני אנשים מבחוץ כגון האקרים או ספקית האינטרנט.
3. **קל לשינוי** - שינוי ברשת הפנימית אינו דורש עדכון של העולם החיצוני.

הניתוב מתבצע באופן הבא:

* כאשר מתקבלת חבילה מהרשת הפנימית, כתובת ה-IP והפורט של רכיב המקור נשמרים בטבלת NAT הנמצאת בנתב תחת מספר כלשהו x. המספר x ישמש כאינדקס כדי למצוא את הכתובת והפורט של רכיב המקור כאשר תתקבל הודעת תגובה. כתובת המקור בחבילה מתורגמת לכתובת ה-IP של הנתב, ופורט המקור בחבילה מתורגם ל-x. לאחר מכן החבילה נשלחת ומגיע אל רכיב המקור כאילו נשלחה מהנתב עצמו.
* כאשר מתקבלת חבילה לנתב מהאינטרנט אשר מיועדת לאחד מרכיבי הקצה, באמצעות פורט היעד בחבילה הנתב מוצא את כתובת ה-IP והפורט של רכיב הקצה שאליו החבילה צריכה להיות מנותבת. הוא משנה את כתובת היעד ופורט היעד לערכים שמצא בטבלה, ומנתב את החבילה אל רכיב הקצה המתאים. רכיב הקצה מקבל את החבילה ומבחינתו הוא לא מודע לשינוי של ה-NAT.

### NAT traversal problem

הבעיה הגדולה בשימוש בטכניקת NAT היא כאשר רכיב קצה מהרשת החיצונית רוצה לשלוח הודעה אל רכיב קצה ברשת הפנימית, ללא הודעה קודמת מהרכיב הפנימי אל הרכיב החיצוני. כאשר ההודעה תגיע אל הנתב של הרשת הוא לא ידע לאן לנתב את החבילה. יש שתי פתרונות לפתור בעיה זו:

1. לעדכן כל NAT בכתובות ה-IP של כל רכיבי הקצה הנמצאים ברשת הפנימית, כך שאם יקבל הודעה מבחוץ לאחת מכתובות אלו ידע כיצד לנתב אותם.
2. **Relaying (ממסר)** - הרכיב הפנימי מתחבר לממסר חיצוני (סוג של שרת) שהרכיב החיצוני מכיר, ודרך ממסר זה הם מתקשרים.

## אלגוריתמי ניתוב

על מנת לעביר חבילה מרכיב קצה אחד לשני, שכבת הרשת צריכה לקבוע את המסלול של הנתבים שבו החבילה תעבור. קביעת המסלול נעשית באמצעות פרוטוקולי ניתוב (Routing Protocols). לכל פרוטוקול ישנו אלגוריתם ניתוב שבאמצעותו הוא מגדיר את המסלול. אנו נלמד שני אלגוריתמי ניתוב מרכזיים: דייקסטרה ובלמן-פורד.

כל אלגוריתם בונה גרף מכוון שבו הצמתים הם נתבים והקשתות הם הלינקים (קשרים פיזיים) בין הנתבים. לכל לינק מוגדר משקל המבטא את "העלות" של המעבר באותו לינק. המשקל נקבע לפי האורך של הלינק והעומס הנוכחי בו. כל המשקלים חיוביים. כמובן שהמשקל אינו קבוע אלא משתנה לפי מצב הרשת. כל אלגוריתם מקבל גרף כזה ואיזשהו צומת מקור A, המייצגת נתב, שממנה הוא מתחיל. מטרת כל אלגוריתם היא להחזיר את המסלול הקצר ביותר בין צומת המקור A לכל צומת אחרת בגרף.

לאחר שמריצים את אלגוריתם הניתוב, ניתן לבנות את טבלת הניתוב עבור אותו נתב מקור A. באמצעות טבלה זו עבור כל יעד שהחבילה תרצה להגיע נדע לנתב אותה אל הנתב הבא במסלול הכי קצר אל היעד.

## Link State Algorithm

אלגוריתם זה מתבסס על אלגוריתם דייקסטרה (Dijkstra). עוד לפני שמפעילים את האלגוריתם על הנתב A הוא צריך לדעת את כל האינפורמציה על מפת הרשת (הגרף), שהם כל הנתבים, הלינקים ביניהם, ומשקל כל לינק. כיצד הנתב מקבל כל מידע זה וגם מתעדכן שיש שינויים ברשת? שתי אפשרויות:

1. **Link State Broadcast** - בתחילה כל נתב מכיר רק את הנתבים המחוברים אליו ישירות באמצעות לינק. כל כמה זמן נתב שולח שידור הנקרא "Link State Broadcast", שבו הוא מפרט על כל הנתבים המחוברים אליו ועל העומס ברשת. כל נתב שמקבל שידור זה שומר את המידע, וכך בשלב מסוים כל נתב מכיר את כל מפת הרשת וגם מתעדכן בשינויים ועומסים.
2. **עדכון ISP** - ספק האינטרנט שאחראי על נתב מעדכן אותו כל זמן מסוים בשינויים ברשת.

אלגוריתם ניתוב זה נחשב לאלגוריתם ניתוב סטאטי, מכיוון שמתעדכן בקצב איטי.

### שלבי האלגוריתם

אלגוריתם דייקסטרה עובד לפי השלבים הבאים:

* מתחזק מערך D[v] שעבור כל צומת בגרף v שומר את המרחק מ-A ל-v. בהתחלה כל הערכים שווים לאינסוף, מלבד A שמאותחל ל-0.
* מתחזק מערך P[v] שעבור כל צומת בגרף v שומר את הצומת שהובילה לגילויה של v. בהתחלה כל הערכים מאותחלים ל-null.
* מתחזק קבוצה Q של כל הצמתים.
* בלולאה כל עוד Q אינה קבוצה ריקה:

- שולף את האיבר עם המרחק המינימלי S (באיטרציה הראשונה תמיד S=A).

- עובר כל שכן V של S.

- אם המרחק השמור ב-S ועוד משקל הקשת בין S ל-V קטן מהמרחק השמור ב-V, אזי מעדכנים את

המרחק ב-V לערך זה. ומעדכנים שהצומת שהובילה ל-V היא S.

* לאחר ש-Q ריק, המערך D[v] שומר את המרחקים הקצרים ביותר מ-A לכל צומת בגרף.
* ניתן למצוא את המסלול מ-A לכל צומת באמצעות הליכה אחורה במערך P[v] עד שנגיע לצומת A.

כל צומת שמוצאת מהרשימה כבר מקבלת את המרחק הקצר ביותר אליה. לכן כבר באיטרציה ה-k ישנם k נתבים שקיבלו את המרחק הכי קצר. סיבוכיות האלגוריתם היא O(mn) כאשר m מספר הקשתות ו-n מספר הצמתים.

## Distance Vector Algorithm

אלגוריתם זה מתבסס על אלגוריתם בלמן-פורד (Bellman-Ford) בגרסת התכנון דינמי שלו. באלגוריתם זה בתחילה כל נתב X מכיר רק את הנתבים השכנים שלו ואת המרחק ממנו אליהם. הוא שומר מידע זה במערך , המייצג את המרחק הקצר ביותר מ-X לכל נתב אחר y. מערך זה נקרא "Distance Vector". כל כמה זמן הנתב X מקבל מכל שכן שלו V את ה-Distance Vector שלו . כל השכנים מקבלים בצורה רקורסיבית את ה-Distance Vector מהשכנים שלהם בצורה דומה, וכן הלאה. לבסוף X מקבל את ה-Distance Vector של כל השכנים שלו. הוא עובר על כל המערכים ובודק האם המרחק מהשכן V עבור כל צומת אחרת y ועוד מרחק הקשת מ-X ל-V קטן ממה ששמור במערך *. אם כן, הוא מעדכן את המערך שלו לערך זה.*

כאשר נתב מעדכן אצלו מרחק קצר יותר הוא מעדכן את כל השכנים שלו בשינוי, כי יש סיכוי גבוה שגם הם יצטרכו לעדכן פרטים אצלם. כאשר יש שינוי במפת הרשת, הנתב שזיהה זאת מפעפע את השינוי לכל השכנים שלו, הם מעדכנים ומעביר את העדכון הלאה לכל השכנים שלהם, עד שהעדכון מגיע לכל הנתבים. אלגוריתם ניתוב זה נחשב לאלגוריתם ניתוב דינאמי, מכיוון שמתעדכן בקצב מהיר.

## השוואה בין האלגוריתמים

1. **סיבוכיות הודעות:**

LS - צריך להתעדכן בכל הלינקים ברשת, ולכן יש צורך O(nm) הודעות.

VD - שולחים הודעות רק לשכנים.

1. **סיבוכיות התכנסות:**

LS - עד שמתקבל המסלול הכי קצר לכל נתב מתעדכן לוקח גם כן O(nm).

VD - סיבוכיות משתנה. יכול גם לקחת הרבה זמן עד שמקבלים מסלול הכי קצר לכל נתב. אפשר גם להיתקע בלולאה.

1. **שגיאה באחד הנתבים במסלול:**

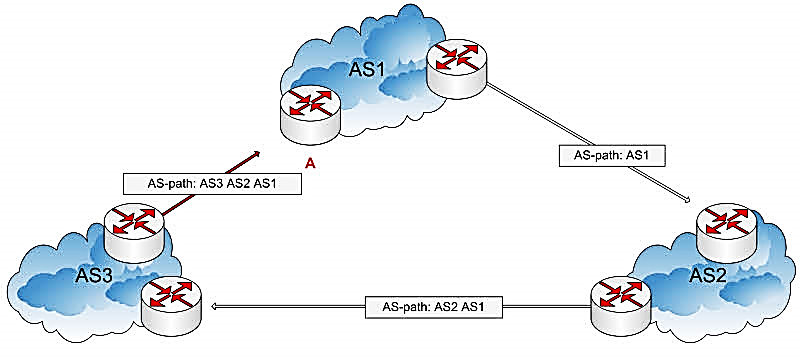
LS - כל נתב יכול לשלוח את הטעות רק בלינק אחת, אמנם באמצעות מנגנון Link State Broadcast יכול להגיע גם ליותר. אך בגלל שכל נתב מבצע את החישובים לעצמו יש מידה כלשהי של חוסן מטעיות אלו.

VD - נתב אחד יכול להעביר את הטעות לכל הנתבים (מעביר לשכנים והשכנים מעבירים הלאה).

**מסקנה:** אלגוריתם LS יותר יעיל ובטיחותי יותר, ולכן הוא בשימוש הרבה יותר נפוץ.

## ניתוב היררכי

עד כה הסתכלנו על האינטרנט כאוסף של נתבים המחוברים ביניהם ומבצעים אלגוריתם ניתוב המחשב את המסלולים הכי קצרים בין כל הנתבים. אמנם בפועל תיאור כזה אינו אפשרי, מאחר ובאינטרנט יש מיליונים של נתבים עם לינקים רבים ביניהם, ולא ניתן להפעיל אלגוריתם ניתוב על כולם ולשמור את הזיכרון על כולם באותו נתב. לכן האינטרנט מחולק לאזורים אוטונומיים "Autonomous Systems", או בקיצור AS, המכילים הרבה רשתות ונתבים. דוגמא ל-AS הם כל הנתבים והרשתות באותו ISP. כל הנתבים באותו אזור אוטונומי מריצים את אותו אלגוריתם ניתוב, ויש להם את כל המידע אחד על השני.



הקשר בין אזורים אוטונומיים נעשה באמצעות נתבים מקשרים הנקראים "Gateway Routers". נתבים אלו אחראיים להעביר חבילות אל מחוץ לאזור האוטונומי בו הם נמצאים. כל הנתבים המקשרים גם כן צריכים להריץ את אותו אלגוריתם ניתוב, וגם לדעת את כל המידע אחד על השני.

פרוטוקול ניתוב בתוך אזור אוטונומי נקרא "פרוטוקול ניתוב פנימי" Interior Gateway Protocols (IGP). ופרוטוקול ניתוב בין אזורים אוטונומיים נקרא "פרוטוקול ניתוב חיצוני" Exterior Gateway Protocols (EGP).

## פרוטוקולי ניתוב פנימיים (IGP)

ישנם שני פרוטוקולי ניתוב פנימיים מרכזיים: RIP ו-OSPF.

### RIP

RIP (Routing Informative Protocol) הוא פרוטוקול ניתוב פנימי באזור אוטונומי. המאפיינים שלו הם:

* משתמש באלגוריתם Distance Vector המבוסס על בלמן-פורד.
* משקל כל לינק הוא 1, לא מתחשבים בעומס ואורך הלינק.
* אורך המסלול המקסימלי הוא 15 לינקים שונים, יותר מכך מקבל ערך אינסוף. מכאן ש-RIP מוגבל רק לאזור אוטונומי עם קוטר של פחות מ-15 לינקים.
* נתבים שכנים מחליפים ביניהם Distance Vector כל 30 שניות. אם נתב לא קיבל Distance Vector מהשכן שלו אחרי יותר מ-180 שניות, מסיק כי הלינק או הנתב מתו, ומעדכן את הטבלה אצלו וגם את כל השכנים.
* כל Distance Vector מכיל לכל היותר 25 נתבים.
* RIP הוא פרוטוקול ברמת האפליקציה המשתמש בשכבת התעבורה בפרוטוקול UDP.

### OSPF

OSPF (Open Shortest Path First) הוא פרוטוקול ניתוב פנימי באזור אוטונומי. משתמש באלגוריתם Link State המבוסס על דייקסטרה. כל נתב מחזיק אצלו את המידע על כל הנתבים והלינקים ברשת. כל נתב מריץ אלגוריתם דייקסטרה, וכך מוצא את המרחק הקצר ביותר ממנו לכל שאר הנתבים. כל כמה זמן הנתב שולח שידור לכל הנתבים באזור האוטונומי המכיל מידע על המרחק מהנתב לכל השכנים שלו (Link State Broadcast). OSPF הוא פרוטוקול בשכבת הרשת.

פרוטוקול OSPF עדיף על פרוטוקול RIP ממספר סיבות:

* אבטחה - כל החלפת מידע בין נתבים מקבלת אישור, כך שרק נתבים אמינים משתתפים בפרוטוקול, מה שמונע מפורצים להכניס מידע לא נכון לטבלאות הניתוב.
* הפרוטוקול מאפשר שימוש במספר מסלולים עם אותו עלות. כך הנתבים יכולים לאזן עומסים על ידי שליחת חבילה כל פעם ממסלול אחר.
* הפרוטוקול מאפשר שלאותו לינק יהיה משקל שונה עבור שירותים שונים. כתוצאה מכך יכול לחשב מסלולים שונים לפי הצורך. לדוגמא, עבור כל מצב של עומס יבחר מסלול אחר.

### OSPF היררכי

פרוטוקול זה זהה ל-OSPF רגיל, אלא שבו ניתן לחלק את האזור האוטונומי לאזורים קטנים יותר, כך שכל אזור מריץ OSPF משלו. פרוטוקול זה שימושי כאשר האזור האוטונומי גדול יחסית. כל אזור מתפקד כאזור אוטונומי נפרד, כלומר הניתוב בתוך האזור מערב רק נתבים באותו אזור, ונתבים מחוץ לאזור אינם יודעים על קיומם של הנתבים בתוך האזור.

מתוך כל האזורים ישנו אזור אחד מרכזי הנקרא Backbone (שלד), ותפקידו העיקרי הוא לנתב את בין האזורים השונים. הנתבים שמקשרים בין האזורים הרגילים ל-Backbone נקראים "Area Border Router" (ARB). כל חבילה שעוברת בין אזורים שונים מגיעה קודם ל-ABR, משם ל-Backbone, ומשם לאזור הרצוי.

## פרוטוקולי ניתוב חיצוניים (EGP)

נלמד רק על פרוטוקול אחד מסוג זה. פרוטוקול BGP (Border Gateway Protocol) הוא פרוטוקול ניתוב חיצוני בין אזורים אוטונומיים. בפרוטוקול זה כל נתב מקשר שומר מסלול ממנו לכל נתב מקשר אחר Path Vector. העיקרון די דומה ל-Distance Vector, אלא ש-BGP שומר בכל מסלול את האזורים האוטונומיים בהם צריך לעבור ולא את המרחק.

כל זמן מה נתב מקשר משתף את המידע שאצלו עם השכנים שלו. השכנים מסתכלים במידע ומחפשים מסלול קצר יותר מזה ששמור אצלם, אם מצאו מעדכנים את המידע. BGP אינו מחליט איזה מסלול ייבחר מבין כל המסלולים האפשריים, אלא החלטה זו נשארת בידי מנהל האזור האוטונומי, באיזה אזורים אוטונומיים הוא רוצה לעבור בדרך ליעד.

שני נתבים יכולים לנהל ביניהם תקשורת באמצעות 4 סוגי הודעות שהפרוטוקול מאפשר:

OPEN - פתיחת תקשורת TCP בין שני נתבים.

KEEPALIVE - נתב מחזיר הודעה זו לאחר שהפתיחת קשר הצליחה.

UPDATE - משמשת להעברת אינפורמציה על המסלול.

NOTIFICATION - משמש כדי להודיע שהייתה שגיאה בהודעה הקודמת, וגם לסגירת קשר.

BGP הוא פרוטוקול בשכבת האפליקציה המשתמש בשכבת התעבורה בפרוטוקול TCP.

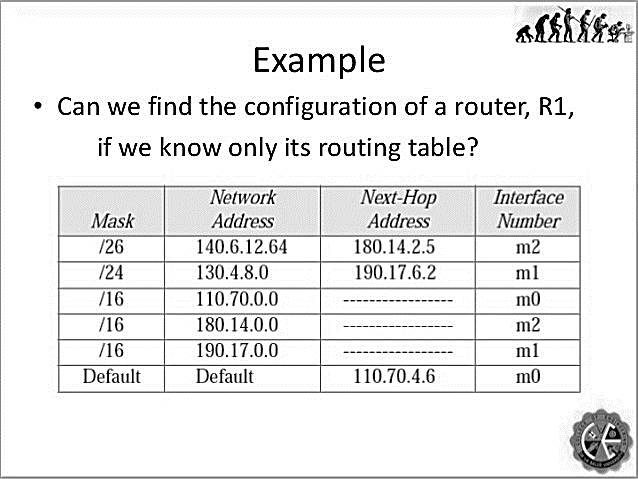
## הנתב

### מבנה הנתב

הנתב (Router) הוא רכיב בשכבת הרשת שתפקידו הוא לקשר בין רכיבי קצה ברמת ה-IP. לכל נתב יש טבלת ניתוב (Routing Table) משלו ובאמצעותה הוא מקבל החלטות ניתוב. כאמור לעיל, טבלת הניתוב נבנית באמצעות פרוטוקולי ניתוב מתאימים.

טבלת הניתוב מורכבת מארבע עמודות:

1. **Network Address** - כתובת הרשת שהנתב מכיר ויודע להוביל אליה.
2. **Network Mask** - ה-Subnet Mask של הרשת מהעמודה הקודמת.
3. **Next Hop** - מהו הנתב הבא במסלול כדי להגיע לרשת.
4. **Interface** - הלינק שדרכו החבילה צריכה לצאת כדי להגיע אל הנתב הבא.

נתבים מתקדמים מכילים טבלאות ניתוב עם הרבה יותר מידע, אך אלו השדות הבסיסיים.

### כיצד הנתב מקבל החלטת ניתוב

כאשר הנתב מקבל חבילה, הוא קורא ממנה את שדה ה-Destination Address ב-IP Header. מטרתו הוא למצוא את הנתב הבא שיוביל אל כתובת זו. הנתב מסתכל בטבלת הניתוב שלו מלמטה למעלה, ובכל שורה עושה פעולת AND לוגית בין כתובת ה-IP שקיבל מהחבילה וה-Network Mask באותה שורה. הנתב מחפש את השורה שבה הערך שקיבל שווה ל-Network Address באותה שורה, וגם ה-Network Mask שלה **הכי ארוך** מבין כל השורות שמקיימות את התנאי הראשון. חיפוש זה קרא "IP Lookup". לאחר שמוצא את השורה הכי מתאימה, הוא שולח את החבילה לכתובת הנתב שבאותה שורה דרך הלינק המתאים המוביל לנתב.

כל נתב צריך לדעת להתמודד עם כל כתובת אפשרית שהוא מקבל, לכן בכל טבלת ניתוב יש שורת ברירת מחדל (Default), שבה הרשת וה-Mask הוא 0.0.0.0/0. כאשר הנתב לא מכיר את הרשת שהחבילה רוצה להגיע, הוא שולח אותה לנתב ה-default gateway המופיע בשורה זו.