מבוא לרשתות תקשורת | 67594

הרצאות | פרופ' מיכאל שפירא

כתיבה | נמרוד רק

תשפ"ג סמסטר א'

תוכן העניינים

שבוע $\mathbb{I}$ | מבוא

הרצאה

נעסוק בעקרונות מרכזיים ברשתות תקשורת, ובראשם האינטרנט ואתגרים שיש להתמודד איתם בקשר אליו. האינטרט מאוד מורכב ואי אפשר להחליף אותו, אפשר לכל היותר לבצע בו שינויים שקשה מאוד לעשות והם איטיים.

אפליקציות באינטרנט

דוגמה הרשת משרתת מטרה כלשהי למשתמש כלשהו, לדוגמה סטרימינג של סרט. כיום סטרימינג קורה ע"י בקשה של הלקוח מהשרת של כמה שניות של סרט בכל פעם, באופן אינקרמנטלי. כך, אם הרשת לא יכולה להעביר (בין אם יש צוואר בקבוק אצל המשתמש או בכל מום אחר) מספיק סרט מספיק מהר, נקבל תקיעות ונצטרך להוריד רזולוציה כדי לקבל חוויה יותר טובה.

אם ידוע שיש בעיה ברשת, חלק מהשירותים שמספקים ווידאו בלייב שומרים באפר של כמה שניות מהשידור האמיתי כדי שאם יש תקיעה פתאומית באינטרנט החוויה תראה עדיין אחידה.

סטרימינג היא דוגמה לאפליקציה מעל הרשת - שימוש בפרוטוקולים שבקרוב נלמד. חווית הצפיה שתוארה למעלה נקראת Quality of Experience ואלג' הסטרימינג שהצגנו נקרא Dynamic Streaming over HTTP. העניין של QoE הוא בעייתי מאוד כי כולם מתחרים על אותו רוחב פס ושירותים רבים דורשים latency נמוך (לדוגמה שיחת זום, איכות יכולה להיות נמוכה אבל חייב להגיע מהר) לעומת throughput (כשקיבולת הרבה יותר חשובה מהזמן שלוקח שזה יגיע).

האינטרנט בעצמו הוא די פרוץ והרבה זמן מנסים לשפר את האבטחה של הרשת אבל זה מאוד קשה כי יש הרבה שחקנים והרשת מבוזרת ולהגיע להסכמה זה כמעט בלתי אפשרי.

בקרוב נדבר על מודל 7 השכבות של אבל לעתה נכווץ אותו ל-3 שכבות: החומרה (ראוטרים, מחשבים, סוויצ'ים וכו'), פרוטוקולים (הדרך שבה מדברות החומרות ביניהם, פרוטוקלים כמו TCP, BGP וכו') ואפליקציות (אתרים, אפליקציות וכו'). השכבה העליונה והתחתונה משתנות כל הזמן ומשתפרות בקצב תדיר, אבל השכבה האמצעית נשארת מאחורה כבר מאז שנות ה-90.

מהירות מעבר מידע

הרבה מהמידע באינטרנט עובר במהירות האור (לדוגמה בין יבשות) ולכן לכאורה אפשר לשלוח מידע מירושלים לניו יורק במהירות האור. הבעיה עם זה היא שראשית החומרה עצמה מאפשרת $\frac{2}{3}$ מהירות האור, רוחב הפס יכול להיות מוגבל, מספר תחנות הביניים בין היעדים יכול להיות משמעותי ויכול להיות שנלחם על מקום עם פקטות אחרות, לכן במקום ה-$30ms$ ההיפותטי של מהירות האור, זה לוקח כמעט פי 2.

בזמן 70 המילישניות האלה שנדמה שהן כלום זמן, מעבד ממוצע מספיק לעשות מיליוני סייקלים ולכן מבחינתו האינטרנט הוא סופר איטי. הזמן הזה הוא קריטי כשמדובר במניות, או דברים שדורשים תיאום בין אנשים שונים, וזו בעיה כי האינטרנט בפועל מיתרגם לתקשורת א-סינכרונית.

האינטרנט ועניינים נוספים

הרשת מחלוקת לשלושה משתתפים - קודקודים (ראוטרים, סוויצ'ים), קשרים (סיב אופטי וכו') ומשתמשי קצה (מחשב, טלפון אבל גם מקרר, טלוויזיה וכו').

כיום מספר המשתמשים ברשת גדל אקספוננציאלית, בגלל שלכל בן אדם יש הרבה יותר ממחשב אחד או טלפון אחד, אלא מקרר ומיקרו וטלוויזיה וכו'. יש הרבה סוגים שונים של משתמשי קצה, המון סוגים של קודקודים וקישורים, וגיוון גדול מאוד בדרישות אפליקציות (האם צריך דו-כיווניות, האם צריך מהירות, האם צריך אמינות וכו' וכו').

הרשת היא התשתית להעברת מידע בין משתמשים, ורק בה נעסוק. מעליה אפשר לבנות רשתות מבוזרות שונות שהן ברמת האפליקציה וזה א מה שיעניין אותנו.

רשת הטלפוניה

כדי להבין איך האינטרנט עובד, נבין קודם איך רשת הטלפוניה עבדה.

כשרוצים לדבר עם מישהו, קודם כל נרצה לדעת האם אפשר להגיע אליו באמצעות circuit, מסלול שהוא רק שלנו ושאי אפשר לעצור אותו באמצע. אחרי שיש קו, נעביר מידע על הקו עד שתסתיים השיחה. בסוף השיחה, נפנה את המשאבים.

המרכזן הוא זה שיוצר את הקו ומבצע switching - ניתוב של תקשורת שמגיעה מכניסה מסוימת ליציאה מסוימת. כיום טלפוניה מנותבת אוטומטית באמצעות אלג' מורכבים.

כיצד נוכל לנתב שתי כניסות לאותה היציאה?

- Time-Division - נחלק את הזמן לשברירים מחזוריים ונבצע round robbin על רוחב הפס.
- Frequency-Division - נחלק את הפס לתדרים שונים, כאשר כל תדר אחראי על משתמש אחד.

למה האינטרנט לא נראה כמו הטלפוניה

מה קורה אם אם משתמש לא משתמש בסלוט שלו (שותק בשיחה)? בטלפונים זה עבד בסדר, אבל באינטרנט זה לא פרקטי.

הרשת שתיארנו זה עתה היא מאוד יציבה, פשוטה, מהירה (העיכוב הוא רק המרחק הפיזי על הפס), צפויה, קל לגלות איפה יש בעיה.

אז למה לא להשתמש בה גם באינטרנט?

- אי-עמידות בפני כישלון: אם אין רוחב פס או אם יש בעיה באמצע, נוותר ולא נקבל שום תקשורת.
- בזבוז רוחב פס: טלפוניה לא שורדת תחת עומס כי רוחב הפס נגמר מתישהו ובאינטרנט זה לא מתקבל על הדעת. בטלפניה נשמור את רוחב הפס המקסימלי הנדרש לקוים הפתוחים עליה (P), לעומת האינטרנט שבו התקשורת המקסימלית אולי גבוהה אבל ממוצע הפקטות שמועבר (A) משמעותית יותר נמוך ולכן לא צריך לשמור את כל רוחב הפס המקסימלי כל הזמן.

  בטלפניה היחס $\frac{P}{A}$ הוא $1:3$. באינטרנט היחס הוא יותר מ-$1:100$.
- עיצוב מותאם אפליקציה: טלפוניה עוצבה לטלפונים, בניגוד לאינטרנט שנועד לאפליקציות מסוג אחר.
- זמן הכנה: בטלפוניה לוקח זמן לא זניח ליצור את הקו, בעוד באינטרנט צריך שזה יהיה (כמעט) מידי.

כדי להתגבר על כל הבעיות האלה, ב-64' הוצע השימוש בפקטות במקום קו שמור, ביזור הרשת והעברת הפקטות דרך הקודקודים ברשת ובהמשך גם ניתוחים סטיסטיים של מערכות כאלה.

רשתות ברודקאסט

ברשת ברודקאסט כל המידע מועברה לכל קודקוד וכולם חייבים להקשיב לפקטות. זה קורה ב-WiFi ולרוב בשאר LAN-ים (Local Area Networks), ומבחינה אינטואיטיבית גם בהרצאה.

יש כמה בעיות בשיטה זו, ביניהן:

- קשה להעביר את הפקטות מרחקים גאוגרפיים גדולים;
- קשה לתאם גישה של כמה משתמשים לרשת (בהרצאה רק אחד מדבר, אבל מה אם שני סטודנטים רוצים לדבר באותו הזמן?), זו בעיית ה-Multiple Access Problem.
- אין פרטיות בתקשורת (למרות שאפשר להצפין את המידע).

שבוע III | האינטרנט ועקרונות יסוד

הרצאה

רשתות switched

בניגוד לרשתות ברודקאסט, רשתות שהן switched מאפשרות לרבים לחלוק את אותו המשאב ולאנשים שונים לדבר עם אנשים אחרים באותו הזמן. מתחת לסיווג הזה יש עוד שני סוגי רשתות:

- circuit-switched - נקצה משאבים לקיבולת מקסימלית לאורך כל השיחה.
- packet-switched - נעביר פקטות בהתאם לכמה מידע נרצה להעביר ולא נקצה מראש את כל המשאבים שנדרשים.

פקטות

ברגע שלפקטה לא מוקצים משאבים, מסלול וכו', היא חייבת להכיל עליה את המידע שמספר לרשת מאיפה ולאן היא צריכה להגיע - כי אף אחד אחר לא יעשה את זה בשבילה.

הפקטה מכילה את גוף הפקטה (payload), שהוא המידע שאנחנו רוצים להעביר, ו-header שמכיל מידע שנועד רק לרשת (כמו פרטים על מעטפה).

כל פקטה מנתובת באופן עצמי ולכן היא יכולה לעבור מסלול שונה גם אם יצאה מיד אחרי פקטה אחרת. נצטרך לדאוג שהעובדה הזו לא תהפוך תקשורת לבלתי אפשרית ואיכותית.

כשפקטה יוצאת, היא עוברת דרך כמה תחנות ביניים וכל תחנת ביניים לכאורה מתעלמת מהגוף (אלא במקרים של אבטחה שלא נעסוק בהם), ומעבירה הלאה את הפקטה לתחנה הבאה. לקודקוד יש שיקול דעת לאן לנתב או לא לנתב פקטה (ואז היא תיפול). הרעיון הזה של שליחי ביניים נקרא store-and-forward - הקודקוד שומר את המידע וכשיכול (או לא), מעביר את המידע הלאה.

הערה בכל תחנת ביניים נוסף עוד דיליי של זמן עיבוד בקודקוד (וזה יכול להצטבר במרחקים ארוכים או קודקודים חלשים).

פרצים

מהתיאור הנ"ל, פקטות הן פתרון לא מוצלח ולא יציב בלי הבטחה של ביצועים כלשהם. אם כן, למה זה עדיין חשוב לנו?

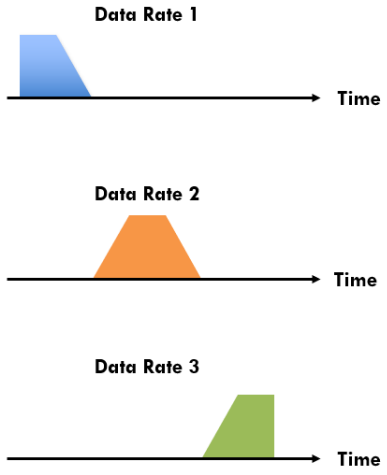
תקשורת באינטרנט הרבה פעמים היא בפרצים, כלומר דממה להרבה זמן ואז הרבה מידע יותר מרוחב הפס האפשרי, לדוגמה חיפוש בגוגל - נחפש, נסתכל על התוצאות, ואז שוב נחפש - וקבלת המידע מגוגל היא הפרץ. לעומת זאת, סטרימינג מגיע באופן יותר רציף מעל האינטרנט.

כיצד נטפל בפרצים? פתרון אחד הוא פשוט להפיל כל פקטה שאנחנו לא יכולים להעביר מיד. לחלופין, נוכל להוסיף באפרים ששומרים פקטות שמחכות לעבור. עם זאת, תמיד יהיו לנו יותר פקטות מאשר מקום בבאפר, ולכן תמיד נצטרך להפיל חלק מהפקטות.

דוגמה נניח שיש לנו שלושה פרצי מידע כבאיור (כחלק משיחה שלוקחת את כל ציר הזמן), כאשר במקסימום כל שיחה דורשת קצת יותר משליש מרוחב הפס.

אם היינו בטלפוניה, היינו מקצים לראשון ולשני מקום והשלישי היה נזרק בחוץ, והיינו מבזבזים בכל רגע נתון שני שליש מרוחב הפס.

עם זאת, היו מספקים משאבים לכולם כי הם לא רצו לדבר במקביל. ההנחה כאן היא שעומס התקשורת בא והולך אבל מתפלג באיזו שהיא צורה לא מתאומת שמאפשרת מעבר של רוב התקשורת בעומס סביר.



איור 1: דוגמה לתקשורת עם פרצים

Statistical Multiplexing

Statistical Multiplexing הוא הרעיון שלפיו נוכל לשרת משתמשים שונים על אותה התשתית גם אם רשמית אין לכולם מקום ביחד (לא כולם יכולים לחפש באותה השנייה בגוגל משהו, אבל בגלל שזה קורה בזמנים שונים במקומות שונים אנחנו כן מצליחים). גם ביטוח והלוואות עובדים ככה - הבנק לווה כסף בידיעה שלא כולם הולכים למשוך את הכסף שלהם ולכן מרשה לעצמו "להמציא" כסף, או בביטוח, שבו אם כל אחד היה בתאונת דרכים הביטוח היה בבעיה, אבל סטטיסטית זה לא סביר.

אפשר להסתכל על Stat Mux גם בתור חלוקה של הזמן לפריימים (שמחולקים אף הם לסלוטים). בכל פריים נייצר לכל היותר P פקטות ובממוצע נייצר A פקטות. אם P גדול מ-A משמעותית זה לא חכם להקצות משאבים לשיחה ספציפית כמו שהזכרנו כבר הרבה פעמים.

עם זאת, חוק המספרים הגדולים אומר לנו שסביר שערכים מהתפלגות יהיו קרובים לתוחלת, יותר מאשר לערך הכי גדול שאפשר - נקבל הרבה יותר התנהגות דומה ל-A פקטות לפריים מאשר P פקטות לפריים.

השוואה בין רשתות מנותבות Circuit ל-Packet

לרשת circuit-switched יש כמה יתרונות וחסרונות, ביניהם

יתרונות	חסרונות
רוחב פס מובטח	רוחב פס מבוזבז בפרצים
אבסטרקציה נוחה	חיבורים חסומים כשאין משאבים
מסלול העברת מידע פשוט	קודקודים חייבים להיות מודעים לשיחות שעוברות דרכם
תקורה נמוכה פר-פקטה	דיליי בהתחלת השיחה

החסרונות האלה הם קריטיים מדי לשירותים שלנו באינטרנט שבגללה אנחנו לא משתמשים בטלפוניה לאינטרנט.

קריטריון	packet	circuit
אמינות	אם יש כשל הרשת תחשב מסלול מחדש	אם הקו נופל לא נשחזר אותו
יעילות	ניצול של Stat. Mux לרוחב פס יעיל יותר	רוחב פס מוגבל ע"י הקוים שהוקצו להם משאבים
קלות מימוש	רשתות יכולות להתחבר בקלות כי הן לא מקצות אחת לשנייה משאבים	חיבור שתי מרכזיות דורש הקצאת משאבים מכל צד
טיפול בעומסים	צריך להתמודד עם עומסים (לזרוק פקטות, להוריד את הקצב,...)	לא רלוונטי

מודולריות

לחלק את הקוד לאבסטרקציות ומודולריות זה חשוב מכל מיני סיבות. בין היתר, אפשר לשנות את המימוש למימוש שונה/יותר יעיל בלי שאחרים ידעו. בנוסף, אפשר להוסיף עוד פונקציונאליות לקוד עם מודולים חדשים נפרדים, וזה בעיקר מאפשר להמשיך לשדרג ולהתקדם בחזיתות שונות באותו הזמן.

ברשתות מודולריות מקבלת נדבך נוסף - המימוש הוא מבוזר בין הרבה מכונות. לכן הושמו כמה עקרונות כדי לפתור בעיות כאלה.

- שכבות היררכיות - כל שכבה מדברת רק עם השכבה מעליה ומתחתיה ולא מודעת/מתעניינת באחרות.
- עקרון קצה לקצה - הרשת מאפשר חיבוריות וזהו, כל דבר אחר (כמו הצפנה) יעשה בידי המשתמשים עצמם. בנוסף, היא לא מבטיחה שפקטה תגיע, או שתגיע בסדר הנכון ולכן אפליקציות צריכות לדאוג גם לזה.

  העקרון הזה נועד כדי לאפשר אפליקציות שאנחנו עוד לא יודעים שיהיו להן דרישות אחרות לעשות מה שהן רוצות בצורה המיטבית ביותר.
- גורל משותף - בשיחה שמתקיימת, המקום היחיד שבו נשמר מידע על השיחה הוא בנקודות הקצה ולא בשום ראוטר או קודקוד באמצע.

הערה העקרון השני והשלישי מופרים ע"י ספקי אינטרנט באמצעות Firewalls וכל מיני מנגנונים אחרים כמו בדיקה של מעבר פקטות בלינקים חלשים מאוד (כמו סלולר). עם זאת, העקרונות האלה הם כן מנחים, רק לא ממומשים במציאות.

הדגמה ומוטיבציה למודל השכבות

נבנה את המשימות שלנו מלמטה למעלה.

- העברת אלקטורנים על כבל.
- העברת ביטים על כבל.
- העברת פקטות על כבל.
- העברת פקטות על רשת מקומית (LAN) עם כתובות מקומיות באמצעות ברודקאסט.
- העברת פקטות בין רשתות מקומיות שונות עם כותבות גלובליות.
- להבטיח שהפקטות מגיעות ליעד.
- לעשות משהו עם המידע הזה.

יש פה רבה משימות שונות ולשם כך אנחנו צריכים מודולים שונים - שכבות - שיהיו אחראים על כל משימה בנפרד.

דוגמה גם במציאות יש איזושהי שכבתיות. מנכ"ל א' שולח למנכ"ל ב' מכתב. המזכירה לקחה את המכתב, שמה אותו במעטפה, שלחה לחברת שליחויות.

זו שמה בעוד מעטפה למעבר פנימי בתוך תשתיות החברה, מעבירה למזכירה של המנכ"ל השני כשהיא הורידה את המעטפה הנוספת, המזכירה מוציאה מהמעטפה ונותנת את המכתב למנכ"ל.

סה"כ המנכ"ל שלח מידע והמנכ"ל האחר קיבל מידע. כל מה שנוסף הוא מטא-דאטא והמסלול שנוצר כאן הוא אנלוגי לחלוטין למה שקורה לפקטות באינטרנט.

מעבר לכך, כל שכבה תיקשרה באמצעות שפה ייחודית לה (מכתב, מעטפה ומעטפה פנימית) וזה אנלוגי לפרוטוקולים השונים של השכבות. אף שכבה לא צריכה להבין את השפה של אף שכבה אחרת.

שכבה	שפה	סוג מידע רלוונטי
מנכ"ל	מכתב	תוכן טקסטואלי
מזכירה	מעטפה	זהות הנמען
חברת שליחויות	מעטפה פנימית	מיקום הנמען

שבע המשימות שתיארנו למעלה מתאימות לחמש שכבות במודל השכבות באינטרנט.

- Pysical Layer - העברת ביטים (ואלקטרונים) בכבל.

- Link Layer - העברת פקטות ברשת מקומית (ובכבל).
- Network Layer - העברת פקטות בין רשתות מקומיות.
- Transport Layer - ווידוא הגעת הפקטות.
- Application Layer - לעשות משהו עם המידע.

תרגול

בתרגול חזרנו על הרבה מאוד הגדרות ומשפטים, אזכיר כאן את המרכזיים, להשלמות ראו קובץ התרגול

הערה בווי-פיי כולם שולחם הודעות לכולם וכדי למדל נכון את הרשת ולפתור את בעיית ההתנגשות, צריך הרבה הסת'.

הגדרה ההסת' של A בהינתן B היא $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$ עבור $P(B) > 0$.

דוגמה הטלנו קוביה וקיבלנו מספר זוגי. מה ההסת' לקבל 2? $P(\{2\} \mid \{2,4,6\}) = \frac{P(\{2\} \cap \{2,4,6\})}{P(\{2,4,6\})} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$.

הגדרה A ו-B הם ב"ת אם $P(A \cap B) = P(A)\,P(B)$.

משפט (בייס) $P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$.

משפט (נוסחת ההסת' השלמה) עבור $\bigcup B_i = \Omega$, מתקיים $P(A) = \sum P(A \mid B_i)\,P(B_i)$.

דוגמה נתון סיב אופטי ששולחים עליו הודעות משני סוגים: הודעה טובה שהסיכוי שלא תיפול אחרי זמן T היא e^{-T} והודעה רעה עם e^{-1000T}. ההסת' ליצירת פקטה טובה היא p.

1. מה ההסת' שהודעה שנוצרה ב-$T = 0$ תשרוד עד $T = t$? מנוסחת ההסת' השלמה התשובה היא $pe^{-t} + (1-p)\,e^{-1000t}$.
2. בהינתן שהודעה שרדה t שניות, מה ההסת' שהיא פקטה טובה? מבייס,

$$P(\text{טובה} \mid t \text{ שרדה}) = \frac{P(\text{טובה})\,P(t \text{ שרדה} \mid \text{טובה})}{P(t \text{ שרדה})} = \frac{p \cdot e^{-t}}{pe^{-t} + (1-p)\,e^{-1000t}}$$

הערה ישנם כל מיני מ"מ בדידים ידועים, להלן סיכום שלהם בטבלה

	Bernoulli	Binomial	Geometric	Poisson
Intuition	We make an experiment, and we could either fail or succeed	We make n independent Bernoulli experiments. We mark by X the sum of successes	We do Bernoulli experiments until we succeed. We denote by X the number of tries	Counting the number of events that occurred during some period of time
Probability mass function	$P(X=1)=p$ $P(X=0)=1-p$	$P(X=k)$ $=\binom{n}{k}p^k(1-p)^{n-k}$	$P(X=k)$ $=(1-p)^{k-1}p$	$P(X=k)$ $=\frac{\lambda^k}{k!}e^{-\lambda}$
Notation	$X\sim Ber(p)$	$X\sim Bin(n,p)$	$X\sim Geo(p)$	$X\sim Pois(\lambda)$
Exp.	$E(X)=p$	$E(X)=np$	$E(X)=1/p$	$E(X)=\lambda$
Var.	$Var(X)=p(1-p)$	$Var(X)=np(1-p)$	$Var(X)=(1-p)/p^2$	$Var(X)=\lambda$

איור 2: טבלת להשוואת מ"מ בדידים

דוגמה נניח שאנחנו שולחים הודעה מ-S ל-D דרך $n-1$ תחנות ביניים, כאשר ההסת' שפקטה תיפול היא p.

1. מה ההסת' שהפקטה תגיע ל-D? $(1-p)^n$ ההסת' שכל הקפיצות כן יצליחו.
2. בהינתן שאנחנו שולחים הודעה שוב ושוב עד שהיא תגיע, מה תוחלת מספר ההודעות שנשלח?

נגדיר X מספר ההודעות שנשלח עד שההודע התגיע. מתקיים $X\sim \text{Geo}\left((1-p)^n\right)$ שתוחלתו $\frac{1}{(1-p)^n}$.

3. עתה כל תחנת ביניים (לא כולל S) שולחת את הפקטה שוב ושוב עד שהפקטה מגיעה, מה ההסת' שנצליח?

$1-p$ כי זה תלוי רק בהאם S הצליח להעביר לתחנה הראשונה.

הגדרה נאמר כי $X\sim\exp(\lambda)$ אם $f_X(x)=\begin{cases}\lambda e^{-\lambda x} & x>0\\ 0 & \text{אחרת}\end{cases}$.

דוגמה במטח מטאורים, מגיעים מטאורים בקצב של $\lambda=50\frac{\text{מטאורים}}{\text{שעה}}$. נוכל לאמר שהמטח הוא תהליך פואסוני בקצב λ. בהתאם, מספר המטאורים ב-t שעות הוא מ"מ פואסוני עם פרמטר λt, כלומר $M\sim\text{Pois}(50t)$ (ל-$50t$ אין יחידות).

מה ההסת' שנצלם שני מטאורים במצלמה שנחשפה ל-15 שניות? נבחר $t=\frac{15}{60\cdot 60}$ ונחשב $P(M=2)\approx 0.018$ או בסימון חלופי $P_{k=2}\left(t=\frac{15}{60\cdot 60}\right)$.

מה הסהת' שנצלם יותר ממטאור אחד? נסמן $T=\frac{15}{3600}$, לחשב $P(M>1)$ ישירות דורש חישוב טור שזה לא נוח, לשם כך נחשב

$$\begin{aligned}P(M>1)&=1-P(M\le 1)\\&=1-P_{k=0}(t=T)-P_{k=1}(t=T)\\&\approx 0.019\end{aligned}$$

ונשים לב כי כל האפשרויות מעבר ל-$M=3$ תורמות לנו מעט מאוד להסת' (ככל שמרתחקים מהתוחלת דועכת ההסת' להיות שם).

הערה נסמן $P_{k=i}(t=T)=P(M=i)$ כאשר $M\sim\text{Pois}(\lambda T)$, ההסת' שהגיעו i פקטות בתהליך פואסוני עם פרמטר λ אחרי זמן T.

הגדרה תהליך סופר (counting process) מוגדר ע"י $\{N_t : t \geq 0\}$ כאשר $N_0 = 0$ ו-$N_t \in \mathbb{N}_0$, $\forall t \geq 0$ סופר את מספר האירועים שקרו עד זמן t (לכן זו פ' מונוטונית).

הגדרה תהליך פואסוני עם פרמטר λ הוא תהליך סופר שבו מספר האירועים באינטרוולים זרים ב"ת מספר, ושמספר האירועים בכל אינטרוול באורך t מתפלג לפי מ"מ פואסון עם פרמטר λt.

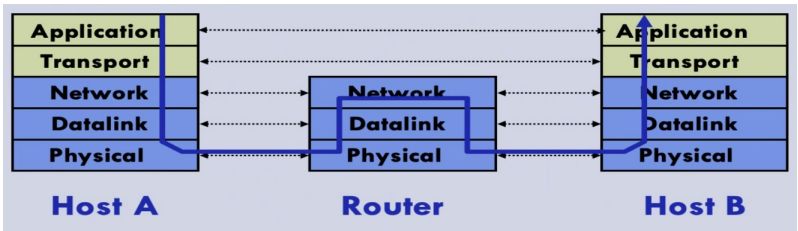
דוגמה מספר האנשים שעולים על אוטובוס הוא לא תהליך פואסוני, כי מספר האנשים תלוי בזמן (ולכן המ"מ שמייצג אינטרוול לא יכול להיות תלוי רק באורך הקטע אלא גם במיקום שלו על ציר הזמן).

שבוע $\mathbb{III}$ | מודל השכבות ותקשורת ב-LAN

הרצאה

הערה מתוך חמש השכבות, השתיים העליונות (Transport, Application) ממומשות רק בקצוות ולא בשום מקום אחר.

מסלול טיפוסי בין שני משתמשים נראה כך, כאשר העלייה והירידה באמצע מתארים את העובדה שכל שני קודקודים ברשת (שאינם משתמשי קצה), פותחים וסוגרים מעטפות גם הם כדי להעביר הלאה את ההודעה. כאן המעטפות הן header-ים שנוספים לתוכן הפקטה.



איור 3 : גרף להדגמת מעבר פקטה בין משתמשי קצה

פרוטוקולים

הגדרה פרוטוקול הוא הסכם איך לתקשר כדי להעביר מידע, או לתאם שימוש במשאב כלשהו. פרוטוקולים מגדירים סינטקס (מה הפורמט שבו מעבירים מידע) וסמנטיקות (איך מגיבים לאירועים והודעות מסוימות).

הערה פרוטוקול, בשונה מאלג', לא מתאר איך בדיוק נעשה משהו, אלא מה אנחנו צריכים שיעשה בכל מימוש של הפרוטוקול.

דוגמה הפרוטוקול הכי פופולרי בווב הוא HTTP, שיש לו סינטקס שמכיל הדרים, גוף הודעה, endpoint בכתובת כלשהי וכו'.

דוגמה פקטת IP מכילה הרבה מאוד שדות ולכל אחד מהם גודל ומיקום מגודר. אם נרצה להוסיף עוד שדה, להרחיב שדה וכו' וכו', נגיע לבעיה של interoperability - יש הרבה מאוד שחקנים באינטרנט וקשה להגיע להסכמה כדי שכל השחקנים בדרך ישתפו פעולה עם השינוי.

הערה ה-IETF הוא ארגון שכל מטרתו לאגד וליצור סטנדרטים לפרוטוקולים.

לכל שכבה יש כמה פרוטוקולים שונים (באפליקציה יש הרבה מאוד, בתעבורה יש הרבה וביניהם TCP, UDP וכו'), למעט בשכבה 3, שבה יש רק את IP.

חייבת להיות שכבה אחת שבה יש הסכמה בטוח על השפה שמדברים כי רק באמצעותה אפשר לבסס הסכמה על הפרוטוקלים האחרים שבהם יש שונות.

כל שכבה מקיימת את עקרון האנקספולציה - כל שכבה מכילה בתוך ההדרים שהיא מוסיפה את המידע שהיא צריכה ומשתמשת בשכבות מתחתיה כקופסה שחורה.

אפיון חמשת השכבות

נאפיין כל שכבה במודל חמשת השכבות באמצעות כמה קריטריונים.

- שירות - מה השכבה עושה.
- ממשק השירות - איך ניגשים לשירות (לצורך השכבה שמעל).
- פרוטוקול - איך peers מתקשרים (ממשק הפרוטוקול, איך ה-peer-ים משיגים את השירות, איך השכבה ממומשת בין קודקודים אבל לא בתוכם).

לכל שכבה יכולים להיות כמה מימושים שונים (כאן נכנסים חידושים והתאמות לצרכים שונים).

השכבה הפיזית

- שירות - מעבירה ביטים בין שני קודקודים שמחוברים בקשר פיזי.
- ממשק - מגדיר איך להעביר ולקבל ביטים (יש לספק את גודל ההודעה וכו').
- פרוטוקולים - קידודים שמאפשרים ייצוג ביטים (עוצמת זרם חשמל וכו').
- דוגמאות - סיב אופטי, גלי רדיו וכו'.

שכבת ה-Link

- שירות - לאפשר מעבר הודעות בין משתמשים באמצעות כתובות אבסטרקטיות מקומיות ולא דרך חיבור פיזי בלבד.
- ממשק - שליחת הודעות (מסגרות של ביטים) בין משתמשי קצה, קבלת ההודעות המיועדות למשתמש הקצה הנכון.
- פרוטוקולים - routing, כתובות MAC.
- דוגמאות - אתרנט, ווי-פי.

שכבת הרשת

- שירות - העברת פקטות לכתובת ספציפית עם כתובות אבסטרקטיות אבל גלובליות בין רשתות (העברת פקטה מרשת אחת בשכבה 2 לרשת אחר בשכבה 2).
- ממשק - איך לשלוח הודעות לרשתות אחרות ואיך לקבל הודעות שמיועדות לי.
- פרוטוקלים - יצירת רשתות ניתוב שמכילות מידע על לאן להעביר הלאה הודעה שמיועדת לכל כתובת גלובלית.
- דוגמאות - IP בלבד.

שכבת התעבורה

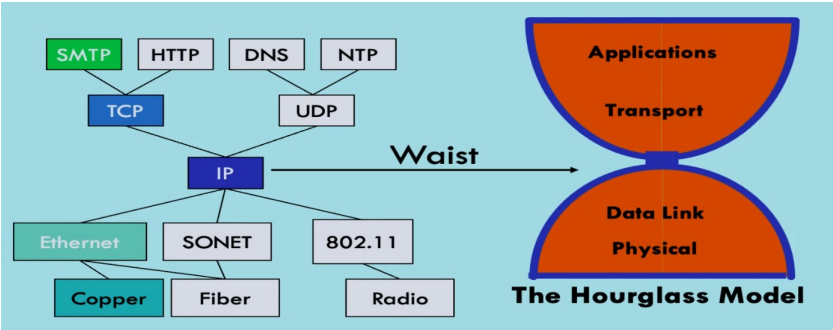
- שירות - תקשורת בין תהליכים, שמבחינה בין שיחות שונות שמתקיימות במקביל בין משתמשים, מספקת אולי אמינות בשליחת הודעות, וויסות קצב הודעות.
- ממשק - שליחת הודעות לתהליך ספציפי ביעד מסוים, העברת ההודעה המתקבלת לתהליך בתוך המכונה.
- פרוטוקול - הגדרת אמינות, פאקטיזציה של הודעות ארוכות, בקרת רצפים (שיגיעו לפי הסדר).
- דוגמאות - TCP, UDP אבל גם SCTP, T/TCP ועוד.

שכבת האפליקציה

שכבה זו היא השכבה השביעית, כאשר דילגנו על שכבות 5 ו-6 מסיבות אנכרוניסטיות.

- שירות - כל שירות שמוצע למשתמש הקצה.
- ממשק - תלוי באפליקציה.
- פרוטוקל - תלוי באפליקציה.
- דוגמאות - HTTP, SMTP, BitTorrent, Skype ועוד.

באיור הדגמה של "מודל שעון החול" - שכבה אחת חייבת להיות קבועה כדי שהשאר יוכלו להיות interpolable.



איור 4: גרף להדגמת מעבר פקטה בין משתמשי קצה

תקשורת ב-LAN

נתון לינק יחיד (בין אם כבל פיזי או תווך לוגי כמו ווי-פיי), כיצד נאפשר גישה מתואמת של כמה אנשים שונים?

הערה כדי לפתור את הבעיה של תקשורת ב-LAN (שכבת ה-datalink) נניח לעתה שיש לנו שכבה פיזית (נניח שקיימת), שכבת ה-datalink, ומעליה רק את האפליקציה.

הערה נקרא למשתמשי קצה קודקודים, לתווך הלוגי לינק ולפקטה בשכבה 2 פריים.

הפתרון שלנו צריך לאפשר כמה שירותים כאמור, ביניהם גישה ללינק עם כתובות בהדר הפריים והעברת מידע אמינה בין קודקודים (error detection וכו').

שכבת הלינק ממומשת בכרטיס הרשת שיש לנו במחשב/מכשיר, כך שרק כרטיסי רשת מדברים ביניהם בשכבה הזו ואין לאף גורם אחר חלק כאן.

הערה שכבת הלינק פותרת את הבעיה שפתר ה-switch רק בלי ה-switch. כשיש switch, אין בעיה של שיתוף המשאב של הלינק כי כולם מחולקים כבר לשיחות. המקום היחיד שבו יש עדיין דמוי broadcast domain זה בחלק של הכבל מכל קודקוד לסוויץ' (שם המחשב לכאורה מקבל ושולח מידע לכולם, עד שהסוויץ' ממיין הכל).

אנחנו מחפשים Multiple Access Protocol - אלג' מבוזר שקובע איך קודקודים מתקשרים ב-broadcast domain.

ה-MAP האידאלי

- כשקודקוד רוצה לדבר, הוא יכול לדבר בקצב R (רוחב הפס המלא).
- כש-M קודקודים רוצים לדבר, כל אחד יכול לדבר בקצב $\frac{R}{M}$.
- אין גורם ריכוזי שמתאם את השיחות.
- אין תיאום של שעונים, יישור של ביטים, והאלג' פשוט.

פרוטוקולים לגישה אקראית

כשלקודקוד יש פקטה לשלוח הוא שולח אותה בקצב המקסימלי האפשרי. אם שני קודקודים משדרים יש התנגשות ואז פרוטוקולים מסוג זה צריכים להגדיר איך לזהות התנגזויות ואיך לשחזר מהם מידע.

דוגמה אחת לפרוטוקול כנ"ל הוא Slotted ALOHA. נניח לשם פשטות (1) שכל הפריימים באותו הגודל; (2) שהזמן מחולק לסלוטים בגודל שווה (הזמן לשדר פריים יחיד); (3) קודקודים מתחילים לשדר רק בתחילת סלוט; (4) קודקודים הם מסונכרנים; (5) ואם שני קודקודים מתנגשים, כולם מזהים התנגשות.

תחת כל ההנחות הללו, כשנרצה לשלוח פקטה, נשלח אותה בסלוט הקרוב ואם הצלחנו בלי התנגשות יופי, ואם יש התנגשות נשדר את הפריים מחדש בהסת' p עד שנצליח לשדר ללא התנגשות.

יתרונות
אם רק קודקוד אחד מדבר הוא משדר בקצב המקסימלי
מאוד מבוזר, כל מה שצריך זה שהקודקודים יהיו מסונכרנים
פשוט

חסרונות
כשיש התנגשויות מתחילים לבזבז הרבה סלוטים
יש סלוטים ריקים
אפשר לזהות התנגשות לפני שנגמר הסלוט אבל לא נעשה עם זה כלום
אין לנו באמת יכולת לסנכרן

הגדרה goodput הוא היחס של סלוטים מוצלחים (ללא התנגשות ועם פריים) מסך הסלוטים. throughput הוא יחס הסלוטים עם פריים מסך הסלוטים.

ננתח את האלג' כאשר עתה תמיד נשלח פריים בהסת' p, גם בפעם הראשונה. נניח שלכל קודקוד יש אינסוף פריימים לשלוח, יש לנו N קודקודים והסת' שליחה p.

ההסת' שקודקוד יצליח לשלוח הודעה ללא התנגשות היא $p\left(1-p\right)^{N-1}$ (הוא שולח וכל השאר לא). ההסת' שקודקוד כלשהו יצליח הוא $Np\left(1-p\right)^{N-1}$ כי המאורעות שאנחנו מאחדים זרים. בשביל יעילות מקסימלית נמצא p^* שממקסם את הנ"ל, וכש-N שואף לאינסוף הערך הזה נותן יעילות של $\frac{1}{e} \approx 0.37$ שזה די גרוע.

כל הניתוח הזה לא פרקטי כי אין לנו סלוטים במציאות כי אין סינכרון כאמור, לשם כך יש את Pure ALOHA - כל קודקוד מתנהג כאילו אנחנו כן ב-slotted ALOHA כאשר ייתכן שהסלוטים של הקודקודים השונים לא מסוכנרים. במקרה כזה, ההסת' להתנגשות עולה כי לא נקבל רק התנגשויות בתוך פריימים, אלא גם בין פריימים.

תרגול

דוגמה בדומה לאנלוגיה למנכ"ל והמזכירה, אפשר להסתכל על מודל השכבות בדומה לפעילות של דואר ישראל.

נרצה לשלוח הודעה מירושלים, ישראל לטוקיו, יפן. קודם כל נצעק בשכונה האם מישהו מכיר את הנמען בטוקיו. אם לא, נשלח למרכז איסוף עירוני ושם יבדקו האם הנמען נמצא בעיר אחרת בארץ. אם לא, המכתב יועבר לנתב"ג ומשם יעבור את המסלול ההפוך ביפן.

כעת נעסוק בשכבה 2 ובפרט בפרוטוקולי גישה אקראית.

הערה נניח כמה הנחות משפטות: (1) כשקודקוד משדר הוא משדר ברוחב פס מלא; (2) אם יש התנגשות כל המידע אבד; (3) אפשר להתעלם מרעש בערוץ (אם יש כזה).

הגדרה רוחב הפס של ערוץ תקשורת כלשהו הוא כמות המידע שאנחנו יכולים לשלוח ליחידה זמן, נסמנו B ויחידותיו $\frac{bit}{sec}$.

הערה נסמן goodput ב-η.

דוגמה כמה זמן יקח לשדר פקטה כשפקטה היא בגודל 30 ביטים ורוחב הפס הוא 5 ביט/שנייה. כמה זמן יקח לשדר את ההודעה? $\frac{|packet|}{B} = \frac{30}{5} = 6$.

מה רוחב הפס בערוץ שמצליח לשדר 3 פקטות באורך 30 ביטים בתוך שתי שניות? $\frac{3\cdot 30}{2} = 45$.

לצורך ניתוח ALOHA, נסמן T הזמן לשליחת פקטה אחת, $X_p \sim Z$ מספר הפקטות שנשלחות באינטרוול בגודל T.

ב-ALOHA כפי שלמדנו בהרצאה, ב-ALOHA שולחים פריימים בסלוטים, כאשר ב-slotted מניחים שלכל הקודקודים יש שעון מסונכרן ולכן כל הסלוטים של כולם מתואמים, ואילו ב-pure יתכן שלכל אחד שעון אחר.

ננתח את האלג' במקרה הבינומי - $X_p \sim \text{Bin}(n,p)$ (כל קודקוד משדר בהסת' p) ובמקרה הפואסוני - $X_p \sim \text{Pois}(gT)$ (יש אינסוף קודקודים, ובאופן טיפוסי נצפה ל-g הודעות בשנייה, T גודל האינטרוול בשניות).

גישה בינומית (slotted)

דוגמה אם יש שלושה קודקודים, ההסת' ששניים יתנגשו היא $p^2(1-p)$, ההסת' לסלוט ריק הוא $(1-p)^3$ וכו'.

ראשית נחשב את ההסת' לפקטה מוצלחת (p_{suc}) ללא התנגשויות. לאחר מכן נסתכל על P_{suc} כמ"מ אינדיקטור על האם סלוט היה ללא התנגשויות ועם הודעה ואז נחשב את תוחלתו, וזה יהיה ה-goodput. נחזור על זה לארבעת המקרים

כמו שראינו כבר כמה פעמים, ההסת' p_{suc} היא $np(1-p)^{n-1}$ (בינומי קלאסי).

$T_{suc} = TP_{suc}$ הוא המ"מ המייצג את זמן השידור המוצלח בסלוט מסוים, כאשר $P_{suc} \sim \text{Ber}\left(\frac{np(1-p)^{n-1}}{p_{suc}}\right)$. מתקיים $E[T_{suc}] = Tp_{suc}$ ולכן ה-goodput הוא $\frac{T \cdot p_{suc}}{T} = p_{suc}$.

גישה בינומית (pure)

נסתכל על סלוט כלשהו, ונשים לב שעתה יכול להיות שמישהו נכנס לסלוט שלנו באמצע (או שאנחנו נכנסו לו). נצטרך שגם ביחידת הזמן הזו (סלוט מהזווית שלנו) וגם בזו הקודמת אף אחד לא שלח שום דבר, כלומר $p_{suc} = np(1-p)^{2(n-1)}$. ומשם שאר הניתוח זהה. במקרה כזה p האופטימלי הוא $\frac{1}{2n}$ שנותן $\frac{1}{2e}$ שזה חצי מה-goodput של ה-slotted.

גישה פואסונית (slotted)

דוגמה מה ההסת' שעל פני שני סלוטים לא נשלחה אף הודעה? $P_{k=0}(t=2T) = \frac{g^0 e^{-2gT}}{0!} = e^{-2gT}$, לחלופין נוכל לחשב את זה כמכפלת מ"מ ב"ת על שני הסלוטים. נשים לב שבתהליך הפואסוני שלנו אכן אינטרוולים זרים הם בעלי התפלגות ב"ת על מספר ההודעות (כיאה לתהליך פואסוני).

מה ההסת' p_{suc} הפעם? על פני סלוט אחד, אנחנו מחפשים את ההסת' שנשלחה הודעה אחת רק, כלומר $p_{suc} = P_{k=1}(t=T) = gTe^{-gT}$ ומאותה סיבה כמו לפני, ה-goodput הוא $\frac{E[T_{suc}]}{T} = \frac{TE[P_{suc}]}{T} = \frac{Tp_{suc}}{T} = p_{suc}$.

גישה פואסונית (pure)

מה היא p_{suc} הפעם? נסתכל שוב על יחידה אחד בגודל סלוט אחד, ונרצה שתשלח הודעה אחת בסלוט הזה ו-0 באחד שלפני, ובגלל שזה תהליך פואסוני זה ב"ת ונוכל להכפיל את ההסת', כלומר

$$p_{suc} = P_{k=0}(t=T)\,P_{k=1}(t=T) = e^{-gT}gTe^{-gT} = gTe^{-2gT}$$

וה-goodput בהתאם כמו לפני, הפעם $\eta_{max} = \frac{1}{2e}$ בדומה ל-pure בבינומי.

שאלה הודעות מגיעות כתהליך פואסוני עם פרמטר g. חלק מההודעות באורך S וחלק באורך $L > S$ כאשר ההסת' ל-S היא p (ו-$1-p$ ל-L).

- נניח שאנחנו משתמשים ב-slotted ALOHA עם סלוט בגודל L. מה ה-goodput?

ההסת' לפקטה מוצלחת היא $p_{suc} = P_{k=1}(t=L)$, ואז

$$T_{suc} = \begin{cases} L & p_{suc,L} \\ S & p_{suc,S} \\ 0 & \text{אחרת} \end{cases}$$

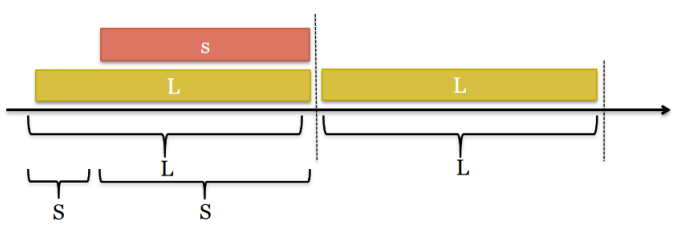
כאשר $p_{suc,L}$ זו ההסת' ששלחנו הודעה ארוכה והצלחנו ו-$p_{suc,S}$ בהתאמה. לכן $E[T_{suc}] = Lp_L + Sp_S$ ואז ה-goodput הוא

$$\eta = \frac{E[T_{suc}]}{L} = \frac{1}{L}\left(Lp_{suc,L} + Sp_{suc,S}\right) = gLe^{-gL}\left((1-p) + \frac{S}{L}p\right)$$

- נניח שאנחנו משתמשים ב-pure ALOHA. מה ה-goodput?

עתה כדי שנשלח הודעה מוצלחת בסלוט בגודל L, נחלק למקרה הקצר והארוך.

לשליחת הודעה ארוכה מוצלחת, נרצה (1) שתשלח הודעה אחת ארוכה בסלוט בגודל L שלנו; (2) 0 הודעות בסלוט בגודל S לפני; (3) ו-0 הודעות בסלוט בגודל $L-S$ לפניכן שהן ארוכות כי רק הן מתנגשות עם הסלוט הנוכחי, הקצרות לא (כמו באיור).



איור 5: הדגמה של הסלוטים המועדים להתנגשות, ה-S הימני הוא טעות ואמור להיות $L-S$

לכן סה"כ נקבל

$$\begin{aligned} p_{suc,L} &= \underbrace{P_{k=1}(t=L)(1-p)}_{(1)} \cdot \underbrace{P_{k=0}(t=S)}_{(2)} \cdot \underbrace{P_{k=0}(t=(L-S)(1-p))}_{(3)} \\ &= (1-p)\,gLe^{-gL} \cdot e^{-gS} \cdot e^{-g(1-p)(L-S)} \\ &= (1-p)\,gLe^{-g(2L(S-L)p)} \end{aligned}$$

כאשר הכפלנו ב-$(1-p)$ בחוץ ב-(1) כי אנחנו שולחים הודעה ורק בהסת' $1-p$ היא באורך L ובפנים ב-(3) כי אנחנו מעוניינים רק בהודעות באורך L שיכולות להשלח ולכן קצב ההודעות הרלוונטי משתנה מ-g ל-$g\left(1-p\right)$.

עתה ההסת' שנשלח הודעה מוצלחת ביחידת זמן בגודל S מחושבת לפי ההסת' (1) שתשלח הודעה אחת אינטרוול בגודל S; (2) שיהיו 0 הודעות בסלוט בגודל S לפני; (3) שיהיו 0 הודעות ארוכה בסלוט $L-S$ לפני הסלוט הקודם.

כלומר סה"כ אותו חישוב כנ"ל רק עם $P_{k=1}\left(t=S\right)$ במקום $P_{k=1}\left(t=L\right)$.

לסיכום,

$$\eta=\frac{L\cdot p_{suc,L}}{L}+\frac{Sp_{suc,S}}{S}=p_{suc,L}+p_{suc,S}=\dots$$

כאשר הפעם חילקנו כל אחד באורך ההודעה כי הודעה באורך S תורמת לנו S לרוחב הפס המנוצל בהצלחה והודעות ב-L תורמות L.

שבוע $\mathbb{IV}$ | שכבת הלינק לעומק

תרגול

דוגמה נניח שאנחנו מתקשרים עם סיב אופטי שלו 3 כבלים. מחשב אחד לא יודע לתקשר בשלושת הסיבים בו זמנית אלא רק באחד, ובו הוא משתמש בהצלחה $\frac{1}{2}$ מהזמן, אז ה-goodput יהיה $\frac{1}{6}$ (למעשה הוא חסום מלמעלה ע"י $\frac{1}{3}$).

הערה מעתה, η (ה-goodput) יהיה יחס הזמן המנוצל בהצלחה מסך הזמן, כפול השיקולים הפיזיקלים שנתונים לנו מבחוץ.

ראינו בתרגול הקודם את ה-goodput של גרסאות שונות של ALOHA, להלן טבלה שמסכמת אותם

	הסת' תחת תהליך פואסוני	הסת' כתלות ב-g,T
Slotted	$P_{k=1}\left(t=T\right)$	gTe^{-gT}
Pure	$P_{k=1}\left(t=T\right)P_{k=0}\left(t=T\right)$	gTe^{-2gT}

שאלה נתונים תשעה כפרים על מאדים בסידור הבא

1	2	3
4	5	6
7	8	9

כל כפר יכול לשמוע את ארבעת הכפרים הסמוכים לו (למעלה, למטה, ימינה ושמאלה), אפע"פ שהודעות מכוונות אך ורק לנמענים בתוך אותו הכפר.

הכפרים האי-זוגיים מתקשרים (בתוך הכפר) עם slotted ALOHA עם פרמטר g_o והזוגיים ב-pure ALOHA עם פרמטר g_e.

ניתן להניח כי בכל כפר יש אינסוף קודקודים (ולכן אפשר להזניח כל מיני תופעות כמו העובדה שלא משנה איזו יחידת זמן בגודל T נבחר, עדיין קצב ההודעות יהיה gT).

- מה ה-goodput של כל המושבה אם $g_o = 0$?

ה-goodput של כל מושבה זוגית הוא p_{suc} כפי שראינו בעבר, כאשר p_{suc} הוא (מהטבלה הנ"ל) $g_o T e^{-2g_o T}$. סה"כ ה-goodput הוא $4 \cdot \left(\frac{1}{4} g_o T e^{-2g_o T}\right) = g_o T e^{-2g_o T}$ כאשר אנחנו מכפילים ב-4 כי הניצול המוצלח שנוסף לנו הוא ב"ת לכן אפשר לסכום את כל הכפר והרבע שם כי כל כפר היא רבע מכלל המושבה.

- מה ה-goodput של כל מושבה אם $g_e = 0$?

בדומה לנ"ל, $g_e T e^{-g_e T}$ כי אנחנו ב-slotted.

- מה ה-goodput במקרה הכללי?

נחלק את הכפרים לשלוש קטגוריות: פינה, צלע ומרכז.

  - פינתי: ההסת' ש-1 (בה"כ) ישדר בהצלחה היא

$$p_{suc,corn} = \underbrace{P_{k=1}^{g_o}(t=T)}_{\text{הפינה משדרת}} \cdot \underbrace{\left(P_{k=0}^{g_e}(t=T)\right)^2}_{\text{השכנים לא משדרים כרגע}} \cdot \underbrace{\left(P_{k=0}^{g_e}(t=T)\right)^2}_{\text{השכנים לא משדרים קודם}} = g_o T e^{-g_o T} e^{-4g_e T}$$
$$= g_0 e^{-T(g_0 + 4g_e)}$$

כאשר נשים לב להבחנה בין זוגיים לאי-זוגיים מבחינת ה-pure/slotted לצורכי חישוב ההסת'.

  - צלע: ההסת' ש-2 (בה"כ) ישדר בהצלחה היא

$$p_{suc,side} \underbrace{P_{k=1}^{g_e}(t=T)}_{\text{הצלע משדרת}} \cdot \underbrace{P_{k=0}^{g_e}(t=T)}_{\text{הכפר לא משדר לפני}} \cdot \underbrace{\left(P_{k=0}^{g_o}(t=T)\right)^3}_{\text{השכנים לא משדרים כרגע}} \cdot \underbrace{\left(P_{k=0}^{g_o}(t=T)\right)^3}_{\text{השכנים לא משדרים לפני}} = g_e T e^{-2T(g_e + 3g_0)}$$

כאשר כאן אנחנו מסתכלים על המקרה הכללי שבו חלון הזמן שבחרנו לא aligned בדיוק על הסלוט של ה-slotted (במקרה כזה לא היינו צריכים את הביטוי האחרון).

נוכל להתעלם מהמקרה הספציפי הזה כי יש אינסוף נקודות קצה ובגלל שהכל רציף לא מסתכלים על חלון נקודתי.

  - מרכזי: ההסת' ש-5 ישדר בהצלחה היא

$$p_{suc,mid} = \underbrace{P_{k=1}^{g_o}(t=T)}_{\text{המרכז משדר}} \cdot \underbrace{\left(P_{k=0}^{g_e}(t=T)\right)^4}_{\text{השכנים לא משדרים כרגע}} \cdot \underbrace{\left(P_{k=0}^{g_e}(t=T)\right)^4}_{\text{השכנים לא משדרים קודם}} = g_o T e^{-T(g_e + 8g_o)}$$

לכן סה"כ ה-goodput הכולל הוא $\frac{1}{9}\left(p_{suc,mid} + 4 \cdot p_{suc,corn} + 4 \cdot p_{suc,side}\right) = \ldots$.

הערה לכאורה אם נשלחה הודעה בידי אחד הקודקודים ביחידת הזמן הקודמת ב-pure ALOHA, ההסת' שתשלח הודעה ביחידת הזמן הנוכחית ירדה ולכן ההסת' תלויות וכל החישובים שלנו לא נכונים. נראה הוכחה למה זה לא נכון. נסמן $X_{suc,i}$ מספר ההודעות ששלח הקודקוד ה-i בהצלחה ביחידת זמן T (0 או 1). בממוצע, על יחידת זמן בגודל T, בממוצע ישלחו

$$\sum E\left[X_{suc,i}\right] = E\left[\sum X_{suc,i}\right] = E\left[X_{suc}\right] = p_{suc}$$

כאשר השתמשנו בלינאריות התוחלת שמתקיימת גם כשהמ"מ לא ב"ת ולכן סה"כ באמת קיבלנו שלא משנה איזה חלון נבחר, תמיד נקבל בממוצע p_{suc} הודעות בשנייה.

שאלה נתונה רשת עם שני סוגים של פקטות כאשר קצב השליחה מתנהג כתהליך פואסוני $X_p \sim \text{Pois}(gT)$. פקטות מסוג A הן בגודל S ופקטות מסוג B הן בגודל αS כאשר $\alpha > 1$. כולם משתמשים בפרוטוקול slotted ALOHA עם סלוט $T = S$. בהסת' p נשלחת פקטה מסוג A ובהסת' $1-p$ מסוג B. יש אינסוף קודקודים.

- מה הוא p_{suc}, ההסת' לפקטה מוצלחת?

נוכל להסתכל על התהליך כשני תהליכים נפרדים, אחד $S_p^\alpha \sim \text{Pois}((1-\alpha)g)$ ואחד $S_p \sim \text{Pois}(pg)$ בהתאם לסוגים.

ההסת' לפקטה מוצלחת באורך αS היא

$$\begin{aligned} p_{suc}^\alpha &= \underbrace{P_{k=0}^\alpha(t=(\lceil\alpha\rceil-1)S)}_{\text{אף ארוך לא יכנס לפני}} \cdot \underbrace{P_{k=1}^\alpha(t=S)}_{\text{אנחנו היחידים כרגע}} \cdot \underbrace{P_{k=0}^\alpha(t=(\lceil\alpha\rceil-1)S)}_{\text{אף ארוך לא מתחיל עד שנסיים}} \cdot \underbrace{P_{k=0}(t=\lceil\alpha\rceil S)}_{\text{אף קצר לא מתחיל עד שנסיים}} \\ &= e^{-g(1-p)(\lceil\alpha\rceil-1)S} \cdot (1-p)\,gSe^{-\alpha g(1-p)S} \cdot e^{-\alpha g(1-p)(\lceil\alpha\rceil-1)S} \cdot e^{-\alpha gp\lceil\alpha\rceil S} \end{aligned}$$

ההסת' לפקטה מוצלחת באורך S היא (בדומה לחישוב הנ"ל)

$$p_{suc} = P_{k=1}(t=S)\,P_{k=0}^\alpha(t=S)\,P_{k=0}^\alpha(t=(\lceil\alpha\rceil-1)S)$$

וההסת' לפקטה מוצלחת בכללי היא $p_{suc,tot} = p \cdot p_{suc} + (1-p)\,p_{suc}^\alpha$.

- מה ה-goodput?

ה-goodput הוא

$$\eta = \frac{E[T_{suc}]}{\text{סך הזמן}} = \frac{S\alpha p_{suc}^\alpha}{S\lceil\alpha\rceil} + \frac{Sp_{suc}}{S} = \frac{\alpha}{\lceil\alpha\rceil}p_{suc}^\alpha + p_{suc}$$

תיקון שגיאות

לעתים בערוץ יש רעש ואז צריך לתקן ולזהות את השגיאות. ישנן כמה שיטות לזיהוי ותיקון שגיאות.

- repetition code - נחזור על כל ביט שלוש פעמים ונתייחס לביט לפי החלטת הרוב, כך נזהה כל אי אחידות ונתקן נכון רק אם יש רוב לא הרוס.

נוכל להסתכל על שלושת הביטים כקוורדינטות בעולם תלת ממדי, ואז כל האפשרויות ל-3 ביטים יוצרים קוביה תלת-ממדית. יש רק שתי קוורדינטות נכונות ולמעשה כדי להכריע, עושים הטלה ולוקחים את התוצאה (דמיינו גאומטרית). כאן התקורה היא 200% כי יש לנו n ביטים ואנחנו שולחים $3n$ ביטים.

- parity code - נוסיף ביט אחד שהופך את ההודעה לזוגית (0 אם היא כבר זוגית ו-1 אם היא אי-זוגית). התקורה כאן היא $1 + \frac{1}{n}$.

  שיטה זו נותנת לנו זיהוי של עד שגיאה אחת אבל אין לנו יכולת לשחזר את המקור.

  עכשיו נחלק את ההודעה למטריצה ונוסיף לכל שורה ועמודה ביט זוגיות (2-d parity). כך נוכל לזהות שלוש שגיאות ולתקן אחת.

שבוע $\mathbb{V}$ | CSMA ואלג' MAP

הרצאה

אנחנו עדיין מנסים לתקשר בתווך משותף (רשתות ברודקאסט) שאין בו סוויץ' באמצע. התוצאה של Slotted ALOHA שמניח הרבה דברים הייתה goodput של $\frac{1}{e} = 37\%$, וב-Pure ALOHA קיבלנו goodput של $\frac{1}{2e} = 18\%$.

הערה אם אנחנו יודעים לזהות התנגשות, אנחנו יודעים גם שכשנתחיל לדבר באמצע סלוט של מישהו אחר אנחנו נתנגש איתו, לכן אין סיבה להתחיל לדבר.

Carrier Sense Multiple Access

ב-CSMA, נאזין לפני שנשדר, ואם נזהה שהערוץ שקט, נשדר פריים שלם, ואחרת נחכה עד שיתפנה.

אם יש איזושהי הסטה מינימלית בין אנשים תמיד, אז לכאורה לא נוכל אף פעם לקבל התנגשות (גם אם התחלנו לדבר באותו הזמן אחרי שהיה שקט אחד יזהה את האחר). עם זאת, ההאזנה ותחילת השידור לא קורים באותו הזמן, ואז יכול להיות שקודקוד אחד התחיל לשדר אחרי ששמע שקט, והאחר גם לא שמע שום דבר ϵ זמן אחריו כי השידור לא הגיע אליו עדיין ועכשיו שניהם מדברים.

מעבר לזה, אם מצב כנ"ל קורה, לא רציונאלי להמשיך את השידור כי הקודקוד השני יודע שהוא גורם להתנגשות.

CSMA/CD וזיהוי התנגשויות רחוקות

CSMA כנ"ל, כאשר נניח שאנחנו מזהים התנגשויות בזמן קצר, ומבטלים שידורים אם מזהים התנגשות כדי למנוע בזבוז של הערוץ. לאחר ביטול השידור נגריל זמן המתנה ונחזור לשלב ההמתנה לשקט.

הערה מודל כזה אפשרי ב-LAN קווי אבל באלחוטי אין לנו דרך לדעת האם יש התנגשות בהכרח כי הסיגנל שלנו משמעותית יותר חזק מזה שמגיע אלינו.

במקרה שבו אנחנו מאוד רחוקים מקודקוד אחר שמתנגש איתנו, נוכל לשמוע על ההתנגשות הרבה אחרי שסיימנו לשדר את הפריים וחשבנו שהוא היה מוצלח.

במקרה הכי גרוע תחילת השידור של הקודקוד הרחוק היא ב-$t = PROP - \epsilon$ כאשר $PROP$ זמן פעפוע ביט מאיתנו לקודקוד הרחוק ביותר ברשת. במקרה כזה הרחוק יזהה התנגשות ב-$t = PROP$, ונגלה אנחנו שהייתה התנגשות רק ב-$t = 2PROP - \epsilon$.

אם כן, לאחר $2PROP$ אם לא התוודענו על התנגשות ניתן להניח ששידרנו בהצלחה. מכאן שכדי להשתמש ב-CSMA/CD צריך אורך מינימלי של פריים כדי שיתקיים $B \cdot L > 2 \cdot PROP$ כלומר שהזמן שלוקח לשדר פקטה (אורך כפול רוחב פס) יהיה יותר מ-$2 \cdot PROP$, ואז נזהה את ההתנגשות לפני תום שידור הפריים ונדע אם צריך לשדר שוב או לא ולא נאלץ לשמור פריימים לשידורים עתידיים וכו'.

חישוב ה-goodput של CSMA/CD

נחשב את ה-goodput במקרה של CSMA/CD. נניח שהזמן מחולק לסלוטים ושסלוט הוא פעמיים זמן הפעפוע המקסימלי ברשת (רק בשביל החישוב התאורטי, לא קשור לפרוטוקול).

בכל רגע נתון, נשלח פריים בהסת' p (גם בפעם הראשונה וגם בשידורים מחדש). ההסת' שרק אחד שידר בסלוט היא $\binom{N}{1} p (1-p)^{N-1} = \alpha(p)$ וזה מקסימלי עבור $p = \frac{1}{N}$, שאז נקבל $\alpha_{max} = \frac{1}{e} \approx 0.4$. נסמן A מספר הסלוטים המבוזבזים בתוחלת לפני ששידרנו בהצלחה פריים, אז מתקיים $A = \alpha \cdot 0 + (1-\alpha)(1+A)$ וכאשר $\alpha = \alpha_{max}$ נקבל $A = 1.5$ (מדובר בתוחלת של מ"מ גאומטרי). לכן ה-goodput שלנו הוא

$$
\begin{aligned}
\eta_{CSMA/CD} &= \frac{\text{זמן השידור של סלוט אחד}}{\text{כמות הסלוטים שנדרשו עד לשידור מוצלח כולל}} \\
&= \frac{TRANSP}{TRANSP + E\,[\text{מספר הסלוטים המבוזבזים לשידור מוצלח של פריים}]} \\
&= \frac{TRANSP}{TRANSP + A\,(2 \cdot PROP)} \\
&= \frac{TRANSP}{TRANP + 3 \cdot PROP} \\
a = \frac{PROP}{TRANSP} \quad &= \frac{1}{1+3a}
\end{aligned}
$$

כאשר $TRANSP$ הוא זמן השידור של הודעה אחת (גודל הסלוט). במציאות מתקבל $\eta_{CSMA/CD} \approx \frac{1}{1+5a}$. כדי למקסם את ה-goodput נרצה למזער את a, כלומר להקטין את היחס. אם כן, ברשתות LAN קטנות, $PROP$ הוא קטן ואז אפשר להרשות לעצמנו $TRANSP$ לא עצום (לא יקח שעה לשדר הודעה) שייתן goodput טוב.

הערה אם כל האינטרנט היה ברדוקאסט, היינו צריכים $TRANSP > 2 \cdot PROP$ כלומר הרבה זמן לשידור הודעה, וגם $a \gg 1$ והיה לנו goodput נורא.

כתובות MAC

לכל כרטיס רשת צרובה כתובת MAC באורך 48 ביטים. כתובות כאלה נקראות כתובות פיזיות כי הן לא מספרות לנו שום דבר על המיקום שלנו בעולם, אלא רק על מי ייצר את הכרטיס שלנו וכיוצ"ב. נשתמש ב-MAC-ים ברשתות LAN כדי לקבוע את המען של הפריים שלנו. יש גוף שמרכז את חלוקת כתובות ה-MAC לחברות שמייצרות NIC-ים.

הערה כיום כבר אין יותר מדי רשתות ברודקאסט טהורות, כשלרוב יש סוויצ'ים ששוברים את התקשורת החופשית באמצע. הפרוטוקול המרכזי שמתמשים בו בשכבת הלינק כיום הוא Ethernet.

Ethernet

האלג' של את'רנט עובד בדומה ל-CSMA/CD, עם שינויים קלים.

1. כרטיס הרשת מקבל פקטה משכבת הרשת ויוצר פריים מתאים לה.
2. אם כרטיס הרשת מזהה שהערוץ פנוי, הוא משדר, אחרת יחכה שיהיה פנוי.
3. אם הכרטיס שידר פריים שלם בלי לזהות התנגשות, סיימנו.
4. אם זוהתה התנגשות, נפסיק לשדר ונשלח jam signal שמוודא שאם שידרנו קצת מדי זמן האחרים יבינו שזה לא רעש אלא שידור שהפסיק באמצע.
5. אחרי ביטול השידור בהתנגשות, נחכה זמן אקראי $k \in \left\{0, \ldots, 2^{m-1}\right\}$ לפני חזרה לשלב 2, כאשר m מספר הפעמים שהתנגשנו עבור הפריים הנוכחי.

תרגול

הבעיה ב-ALOHA כאמור היא שאנחנו לא מתנהלים עם התנגשויות כמו שצריך. לצורך כך הומצא CSMA, שמטפל בהתנגשויות באמצעות הפסקת שידור לאחר זיהוי התנגשות ובדיקה האם הערוץ פנוי לפני שמתחילים לשדר. נשווה בין כמה תתי-פרוטוקולים כאלה

פרוטוקול	הערוץ פנוי?	הערוץ תפוס?	השלכות
1-persistent	שלח הודעה	המתן עד שפנוי ושלח מיד	יש הסת' גבוהה להתנגשויות בערוצים עמוסים
non-persistent	שלח הודעה	חכה פרק זמן ונסה שוב	נקבל ניצולת נמוכה בעומס נמוך
p-persistent	שלח בהסת' p	המתן עד שפנוי ושלח בהסת' p	פשרה בין השניים הנ"ל

Exponential Backoff

כמה זמן ראוי לחכות ב-non-persistent? הדרך הכי פופלרית לחישוב זמן ההמתנה הוא Exponential Backoff.

יהי פרמטר c קבוע מראש (לרוב 2). אחרי הניסיון ה-k לשידור פריים, נחכה jT יחידות זמן כאשר j מוגרל אחיד על $\left[0, c^k - 1\right]$. בכל פעם שנצליח לשדר, נאפס את k שלנו.

ברגע שזיהינו התנגשות עם אחרים, נשדר jam signal כדי שאחרים ידעו שהתנגשנו ושאין מה להתחייס לפריים הנוכחי.

חישוב ה-goodput של CSMA/CD (יותר לעומק)

נבצע את אותו הניתוח שעשינו בהרצאה, רק קצת יותר מפורט. נניח שאין לנו exponential backoff, שיש לנו N תחנות, זמן השידור המקסימלי ברשת הוא T_{prop} וזמן השידור של פריים הוא T_{trans}.

נניח שהפרוטוקול משדר בהסת' p, שהזמן (קונספטואלית) מחולק לסלוטים בגודל $2T_{prop}$ יחידות זמן, ושלקודקודים תמיד יש מה לשדר.

הפרוטוקול שלנו ישדר אם פנוי, אחרת יחכה לסלוט הבא, וכשיזהה התנגשות ישלח jam signal, יפסיק את השידור וינסה בסלוט הבא בהסת' p.

לרשת יש שלושה מצבים אפשריים.

- מתיחות - שני קודקודים משדרים ויגלו את ההתנגשות רק לאחר זמן לכל היותר $2T_{prop}$.
- שקט - אף אחד לא משדר
- משדר - קודקוד אחד משדר בהצלחה פריים.

ההסת' לשידור מוצלח כרגיל הוא $Np\left(1-p\right)^{N-1}$, והיא מקסימלית כאשר $p=\frac{1}{N}$, נסמן הסת' מקסימלית זו ב-S. זה לא מספיק לנו כדי לשדר כי אנחנו צריכים להתחשב בזמן שמתבזבז כשאנחנו במצב מתיחות.

השאלה שלנו כאן היא מה ההסת' לאינטרוול מתיחות באורך $T-1$ סלוטים, ולאחריו שידור מוצלח? לכן בתוחלת, הזמן הנדרש לשידור מוצלח הוא $2T_{prop}\cdot\frac{1}{S}$ בתוחלת (מספר הסלוטים כפול תוחלת המ"מ למספר הסלוטים עד לשידור מוצלח), ולכן הזמן המבוזבז בתוחלת הוא $2T_{prop}\left(\frac{1}{S}-1\right)$.

מכאן שה-goodput הוא

$$\begin{aligned}\eta &= \frac{T_{trans}}{T_{trans}+\text{זמן מבוזבז}}\\ &= \frac{T_{trans}}{T_{trans}+2T_{prop}\left(\frac{1}{S}-1\right)} \xrightarrow[n\to\infty]{} \frac{1}{1+\frac{2T_{prop}}{T_{trans}}\left(e-1\right)}\end{aligned}$$

וכאשר $T_{trans}\gg T_{prop}$ נקבל goodput שואף לאחד.

הערה היינו יכולים לעשות את אותו הניתוח גם במקרה של slotted ALOHA, כשאז היינו מציבים $T_{trans}=2T_{prop}$ כי זמן השידור של הודעה הוא בדיוק גודל הסלוט.

שאלה נתונה רשת בפרוטוקול CSMA/CD עם רוחב פס 10 Mbps וקצב פעפוע ביט של $2\cdot10^8\frac{m}{s}$. נרצה להוסיף עוד רשת דומה, מרוחקת מרחק של 20 ק"מ מזו. הרשתות מחוברות באותו התווך כנ"ל והמרחק המקסימלי בין קודקודים הוא 20 ק"מ.

- האם ניתן להריץ CSMA/CD כאשר גודל ההודעה הוא 64 בתים?

לא! נצטרך ש-$T_{trans}>2T_{prop}$ אבל כאן יש לנו זמן פעפוע של ביט אחד לאורך הרשת של

$$\tau=\frac{20km}{2\cdot10^8\frac{m}{s}}=10^{-4}s=100\mu sec$$

ואילו זמן השידור של 64 בתים הוא

$$\frac{8\cdot64}{10Mbps}=\frac{8\cdot64}{10^7\frac{b}{s}}=51.2\mu sec$$

שזה פחות מ-$2\times$ הזמן לשידור ביט לאורך הרשת. אם נגדיל את גודל ההודעה פי 4, אז זמן השידור יגדל פי 4 ואז זה יהיה חוקי.

- מה יהיה השינוי אם נגדיל את רוחב הפס ל-$100Mbps$?

זמן השידור יקטן פי 10 ולכן נגדיל את ההודעה פי 10.

שאלה נתונים שלושה קודקודים $A - B - C$ (בסדר הזה) עם 5 ק"מ מרחק בין כל שתי התחנות. נניח רוחב פס $B = 3Mbps$ ומהירות פעפוע $v_{prop} = 6 \cdot 10^7 \frac{m}{s}$.

- כדי ש-CSMA/CD יעבוד כראוי, מה גודל הפקטה המינימלי כדי ש-A יוכל לשלוח הודעות ל-C?

$$T_{prop} = \frac{10}{6 \cdot 10^4} s = \frac{1}{6} 10^{-3} s$$
$$T_{trans} = \frac{x}{B} \geq 2T_{prop} \iff x \geq 2\frac{1}{6 \cdot 10^4} \cdot 3 \cdot 10^7 = 10^3 bit$$

- כדי ש-CSMA/CD יעבוד כראוי, מה גודל הפריים המינימלי כדי ש-A יוכל שלוח ל-B הודעות כראוי?

אותו הדבר כנ"ל, עדיין יכולה להיות התנגשות עם C ולכן אין שינוי.

שאלה נתונה רשת עם slotted ALOHA בגודל סלוט $T_{trans} > 2T_{prop}$.

- נניח שנוסיף CSMA (לא CD), האם זה תשפר את ה-goodput?

לא! ה-CSMA יגרום לנו לחכות עד שנראה ערוץ פנוי, אבל כולם יראו ערוץ פנוי בתחילת הסלוט (או מיד אחריו) כי כולם aligned ולכל הפחות לא נקבל goodput יותר גבוה.

- האם הוספת CSMA/CD תשפר את ה-goodput?

גם לא! אנחנו לא מתחילים לשדר עד הסלוט הבא ולכן גם אם נזהה התנגשות ונפסיק לשדר, הסלוט הנוכחי מבוזבז כי לא נתחיל לשדר עד לתחילת הבא.

- האם הוספת CSMA/CD תשפר את ה–goodput במקרה של pure ALOHA עם אותם הנחות על הסלוט?

כן! לכל הפחות לא נהרוס שידורים קיימים ולכן נשפר את ה-goodput.

- האם הוספת CSMA/CD תשפר את ה-goodput מעבר ל-CSMA לרשת נ"ל?

גם כן! נפסיק שידורים שכבר התנגשו בהם ונאפשר לאנשים שעוד לא התחילו לשדר הזדמנות לשידור בסלוט מנקודת מבטם.

מתי כדאי להשתמש ב-CSMA/CD באופן כללי? כש-T_{prop} קטן יחסית ביחס ל-T_{trans}, ואז כשמזהים התנגשות נפסיק את השידור די מהר ולא נבזבז (לפחות לא אנחנו אישית) את הערוץ עם פריימים מושחתים.

שבוע $\mathbb{VI}$ | סוויצ'ים בשכבה 2

הרצאה

נסיים לעסוק בשכבת הלינק, ונדון ב-Ethernet.

ב-Ethernet אחרי התנגשות מחקים זמן מוגרל אחיד על פני $\{0, \ldots, 2^m - 1\}$ כאשר m מספר ההתנגשויות ולכן ככל שנתנגש יותר, כך ההסת' להתנגשות יורדת, כי ההסת' ששנינו חיכינו אותו הזמן הוא די קטן. ה-goodput של Ethernet הוא $\eta = \frac{1}{1+5\frac{t_{prop}}{t_{trans}}}$ שזה כמו ב-CSMA/CD, די טוב, כי אם אנחנו שולטים בגודל הרשת והפקטות, נוכל להשיג goodput לא רע בכלל.

שכבת הלינק ברשתות אלחוטיות

בתקשורת אלחוטית, בגלל שהאות שלנו כל כך הרבה יותר חזק מאותות שמגיעים, לא נוכל לזהות התנגשות בזמן שאנחנו מדברים ולכן במקום לזהות התנגשויות, נצטרך להימנע מהתנגשויות.

בכל רשת תקשורת אלחוטית, יש Access Point שהוא קודוקד מיוחד ברשת שיכול להוות סדרן לשאר הקודקודים (שהם משתמשים ברשת). ה-AP יקבע מי רשאי לשדר, ואז נחלק את הפעולה של כל קודוקד למצב שידור ומצב האזנה.

802.11

האלג' לשידור פריים ברשת אחלוטית בפרוטוקול 802.11 הוא כדילקמן (ומשתמש ב-CSMA/CA שיפורט בהמשך):

1. אם הערוץ פנוי לאיזשהו זמן (DIFS), נשדר את כל הפריים בלי לעצור ונחכה לאישור קבלה. אם לא נקבל, נעבור ל-2.
2. אם הערוץ תפוס, נתחיל טיימר עם זמן אקראי שסופר אחורה כל פעם שיש פרק זמן שקט בערוץ.

   כשהשעון נגמר, נשדר ואם לא נקבל אישור קבלה, נגדיל את זמן ההמתנה ונחזור להתחלה.

כל עוד לא מנסים לשלוח הודעה, ב-802.11 מאזינים ואם מקבלים פריים שלם טוב מאשרים קבלה.

CSMA/CA

איך נבצע את החלוקה למצב שידור והאזנה של הקודקודים?

- אם קודקוד רוצה לשדר, הוא צריך לשלוח בקשת Request-to-Send ל-AP.
- אם יש אישור, ה-AP שולח Clear-to-Send ברשת וכולם שומעים את זה (עד כדי התנגשויות עם RTS-ים אחרים אבל הם מאוד קצרים אז זה זניח). משם הקודקוד המשדר יכול להתחיל לשדר (לדוגמה באמצעות האלג' של 802.11).

הערה אם יש התנגשות ב-RTS-ים, ה-AP יזהה את זה ויאשר רק אחד מהם.

סיכום ברודקאסטים

למדנו בשכבת הלינק על כמה אלג', הפשוטים ביותר הם ALOHA, S-ALOHA, CSMA. ראינו שבמציאות משתמשים בברודקאסט חוטי ב-Ethernet עם CSMA/CD ובברדוקאסט אלחוטי ב-802.11 עם CSMA/CA.

Switch

סיימנו לעסוק במה שקורה ברשתות ברודקאסט. כיצד נחבר רשתות ברדוקאסט? ניזכר שסוויץ', כמו שהוא שובר רשתות ברודקאסט, הוא גם מחבר אותן. מה נדרוש שסוויץ' יעשה?

- יעביר פריימים של את'רנט ועל בסיס ה-MAC address שמופיע על הפקטה, יעביר הלאה למי שצריך את הפריים, כשהוא מתקשר עם יחידות אחרות באמצעות CSMA/CD.
- שקיפות - משתמשי קצה לא מודעים לקיום שלו וחושבים שהם שולחים ישירות למען שלהם הודעה.
- plug-and-play - לא צריך לקנפג את הסוויץ' בשום דרך והוא ידע כבר בעצמו איך לנתב ברגע שיחובר לרשת.

לכל קודקוד יש חיבור מפורש לסוויץ' (כאשר החוט שמחבר בין השניים הוא רשת הברודקאסט היחידה במקרה הזה), וכך כמה אנשים שונים יכולים לשלוח פקטות למענים שונים בלי התנגשויות.

איך סוויץ' יודע איזו יציאה רלוונטית לאיזו כתובת? הוא שומר switch table שאומרת לו לכל חיבור אילו כתובות MAC מתאימות לו. כל רשומה מכילה את כתובת ה-MAC של משתמש קצה, החיבור שאיתו מדברים איתו ו-Time To Live - משך הזמן שעבורו הרשומה הזו רלוונטית.

איך סוויץ' יוצר רשומות בטבלת הניתוב?

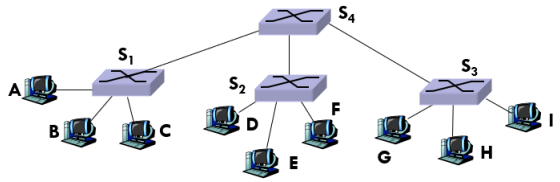
Self-Learning

ללמידה עצמית אין שום קשר ללמידת מכונה. למידה עצמית היא התהליך שבו סוויץ' לומד אילו כתובות יש סביבו.

- כשסוויץ' מקבל פריים, הוא מוסיף (או דורס אם צריך) רשומה לכתובת המקור של הפקטה בטבלת הניתוב שלו לפי החיבור שממנו קיבל את ההודעה.
- מעבירים הלאה את ההודעה:
  - אם אנחנו לא יודעים באיזה חיבור נמצא המען נשלח את ההודעה לכולם ונקווה שמתישהו המען ישלח הודעה בעצמו ונוכל להוסיף אותו כרשומה (לרוב זה יקרה כי השכבות העליונות מחייבות תשובה גם אם היא לא מכילה תוכן חדש).
  - אם אנחנו כן יודעים באיזה חיבור נמצא המען:
    - אם הוא נמצא באותו החיבור כמו הכתובת המקור, נזרוק את הפקטה (לא צריך להעביר לאף אחד).
    - אחרת נעביר אותה לחיבור הרלוונטי.

אם יש לנו כמה סוויצ'ים מחוברים, בגלל שהסוויצ'ים שקופים, האלג' עדיין עובד!

דוגמה אם נרצה להעביר מידע מ-A ל-G, כל החיבורים והסוויצ'ים יעבדו מעולה כי S_1 לא מודע לזה שיש סוויצ'ים בדרך ל-G, מבחינתו אם A רוצה לשלוח ל-G הודעה, הוא צריך לשלוח את זה דרך החיבור הימיני עליון שלו וזה יגיע לשם מתישהו (כמובן אחרי שביצע למידה עצמית וגילה שאכן G שם).



איור 6: טופולוגיה מורכבת של סוויצ'ים, הכל עדיין עובד כמו שצריך

הערה כבר כאן אנחנו מבינים שכתובות MAC אינן הפתרון האולטימטיבי לזיהוי משתמשי קצה כי אם כל אחד יצטרך לזכור את כל הכתובות של כולם בלי אבסטרקציה, נצטרך מיליארדי רשומות בכל ראוטר פעוט.

דוגמה מה קורה אם יש לנו לולאה בסוויצ'ים? לדוגמה אם שני סוויצ'ים מחוברים אחד לשני בשני ברודקאסט דומיינים שונים?

אם קודקוד A ישלח הודעה בברודקאסט אחד, היא תגיע לשני הסוויצ'ים, ואז סוויץ' אחד ישלח את ההודעה בברודקאסט האחר, זה יגיע לסוויץ' השני, הוא ידרוס את הרשומה של A אצלו.

לאחר מכן, אם צריך להעביר משהו ל-A זה לא יגיע אליו אף פעם וגם נשלח את הפקטה שוב ושוב בלי שום סיבה.

הערה הבעיה הנ"ל קורת רק כשיש לנו מעגלים בטופולוגיית הסוויצ'ים. אם לא היו לנו מעגלים, לא היה לנו שום redundancy וזה לא טוב.

לכן כן יהיו לנו פיזית מעגלים כלשהם בסוויצ'ים, אבל בפועל נשתמש רק בחיבורים מסוימים כך שיהוו עץ פורש של גרף הסוויצ'ים שנוצר.

Spanning Tree Protocol

כדי לגרום לסוויצ'ים לנוון פורטים (חיבורים) לשם הגעה לעץ פורש, צריך לתאם בין הסוויצ'ים באמצעות פרוטוקול עץ פורש (STP). האלג' שבו משתמשים מכיל שלושה שלבים (שבמציאות לא קורים אחד אחרי השני כי הכל מבוזר).

1. בחירת שורש העץ.

   (א) כל קודקוד יזכור את המספר הסידורי שלו (שהוא ייחודי).

   (ב) בכל פעם שנקבל מספר סידורי מבחוץ, אם הוא נמוך יותר ממה שאנחנו מחזיקים כרגע נבחר אותו.

   (ג) מדי פעם נשלח את המספר הכי נמוך שלנו לכל השכנים שלנו.

   מתישהו כל המערכת תתייצב על איזשהו מספר.

2. חישוב המסלול הקצר ביותר לשורש.

   ברגע שכל קודקוד ידע מה המסלול הקצר ביותר שלו לשורש, הגרף שנוצר מהמסלולים הללו הוא עץ פורש. נשתמש באלג' בלמן-פורד (מבוזר).

   (א) כל קודקוד ישלח לשכנים שלו מדי פעם את המרחק שלו מהשורש.

   (ב) כשקודקוד מקבל הודעה כנ"ל, הוא יעדכן (אם הוא צריך) את האב שלו לשכן הכי קרוב שלו לשורש, ואת המרחק שלו מהשורש (מרחק השכן + 1).

   מתישהו התהליך הזה יתכנס וכולם ידעו מה האבא שלהם במסלול (שיהיה מוסכם על כולם בסוף).

   נוכל למשקל במשקולות אי-שליליים חיבורים (לדוגמה המרחק שצריך לעבור בהם) וככה להשיג נקודת מבט אפילו יותר מדויקת על הטופולגיה.

תרגול

נלמד על החצי השני של שכבה 2 - בהינתן כמה רשתות ברודקאסט - איך נחבר ביניהן באמצעות סוויצ'ים ואיך נמנע לולאות בשביל תפקוד איכותי של הרשת.

הסוויץ', ברגע שמתחבר, מתחיל ללמוד מי מחוברים לאן באמצעות אלג' שראינו בהרצאה (בכל הודעה שמקבלים, מעדכנים את הטבלה ושולחים לפורט הרלוונטי או שעושים flooding). מהאלג' הזה נוצרת בעיה ברגע שיש לולאה בטופולוגיית הסוויצ'ים.

כדי להימנע מלולאות, נגרום לסוויצ'ים לנוון חלק מהפורטים שלהם כך שהטופולוגיה תמפה עץ פורש של הסוויצ'ים (אלג' STP). נחלק את הפורטים של כל סוויץ' לשלושה סוגים:

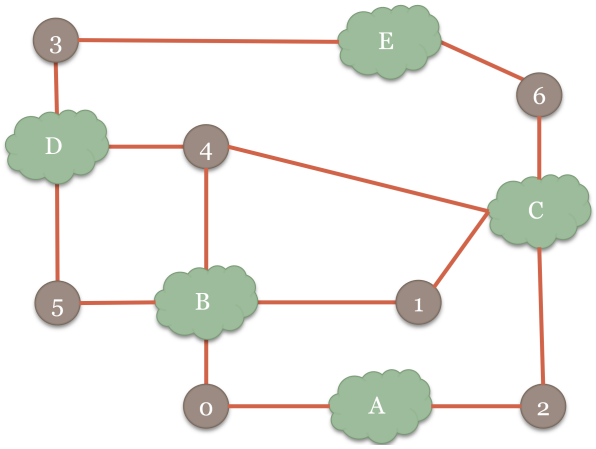
- **R**oot **P**ort - פורט שמחבר את הסוויץ' בדרך לשורש העץ.
- **F**orwarding **P**ort - כל פורט שאינו RP והוא פעיל, מכוון סוויצ'ים אחרים לשורש, וגורם לכך שהסוויץ' אחראי על כל הרשתות שמחוברות דרך הפורט הזה.
- **D**isabled **P**ort - פורטים מכובים, שמקשיבים להודעות "שלום" חדשות (למקרה שקודקוד נופל וצריך לשנות את העץ).

הערה לכל סוויץ' יכול להיות RP אחד וכמה FP-ים ולכל ברודקאסט יש FP אחד וכמה RP-ים.

הערה כשנריץ STP מסומלץ, קודם נמצא את ה-FP-ים מכל הרשתות כך שיכוונו לסוויץ' הכי קרוב לשורש, ואז נמצב את ה-RP-ים בהתאם.

הערה במקרה שב-STP סוויץ' מקבל הודעה משני סוויצ'ים שונים בעלי אותו גובה, הוא מכריע לפי ה-SID (Switch ID) הנמוך מביניהם, ואם זה שווה, הוא מכריע לפי מספר הפורט שממנו קיבל הודעה מכל אחד מהסוויצ'ים (הנמוך מביניהם).

דוגמה נתונה הרשת הבאה, כאשר עננים הם רשתות ברודקאסט וסוויצ'ים מסומנים בריבועים (SID כשם הסוויץ')



איור 7: טופולוגיה של סוויצ'ים

נריץ את האלג' ובסופו של דבר נגיע לכך ש-0 הוא השורש, ואז $1, 2, 4, 5$ כולם בגובה 1, $3, 6$ בגובה 2. ה-RP-ים עתה יהיו 5-B, 1-B, 2-A, 3-D, 4-B ואילו ה-FP יהיו כל מה שאינו RP כאמור או DP, שהם 4-C, 6-C, 6-E.

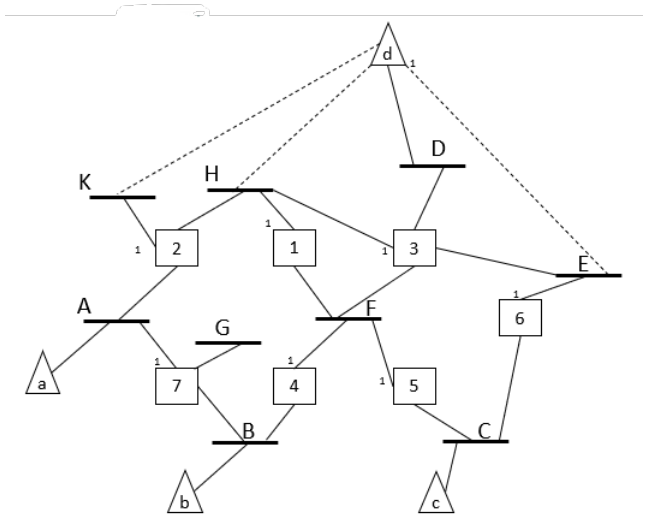
- נפל שורש העץ, איך ה-STP יארגן מחדש את הרשת?

השורש הוא 1. דרגות העצים הן 1 ל-$2, 4, 5, 6$ ו-2 ל-3. עתה ה-FP-ים הם 6-E, 4-D, 1-B, 1-C ו-RP-ים הם 6-C, 3-D, 4-B כאשר כאן 2 ו-5 נפלו כי לא צריך אותם בכלל (הפורטים אליהם מכובים בכל הסוויצ'ים).

- חזר 0 אבל נפל 1, איך נתארגן שוב?

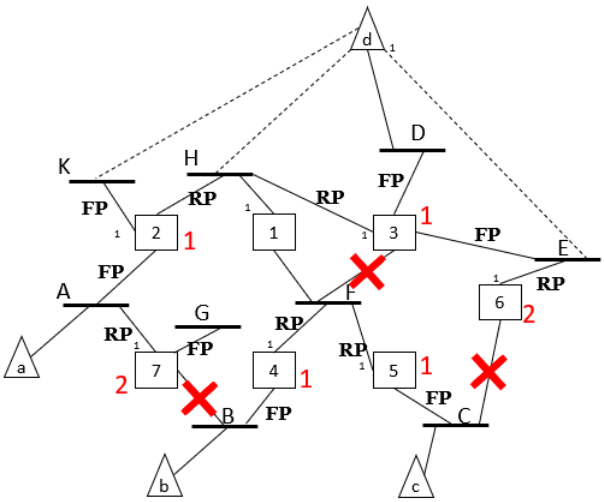
השורש הוא 0. דרגות העצים הן 1 ל-$2, 4, 5$ ו-2 ל-$3, 6$. ה-FP-ים הם 0-B, 4-D, 3-E, 2-C, 0-A ו-RP-ים הם 3-D, 4-B, 2-A ואז $5, 6$ נופלים.

דוגמה נתונה הרשת הבאה (משולשים הם יחידות קצה בתוך רשת, קווים הם רשתות LAN וריבועים הם סוויצ'ים). הפורטים של כל סוויץ' מתחילים מ-1 עם כיוון השעון. נריץ את STP עליו.



איור 8 : טופולוגיה של סוויצ'ים ויחידות קצה

נבחר כשורש את 1 ואז הדרגות יהיו 1 ל-$2, 3, 4, 5$ ו-2 ל-$6, 7$. באלג' יצא העץ הבא



איור 9 : STP שנוצר מהטופולגיה הנ"ל

- כיצד תראה טבלת הסוויצ'ינג של סוויץ' 2 אחרי הרבה הודעות בכל הרשת?

פשוט נסתכל אילו פורטים לא מנוונים מאפשרים הגעה מכל יחידת קצה לסוויץ' (יש מסלול אחד מכל רשת לסוויץ' 2)

כתובת MAC	פורט
a	3
b	2
c	2
d מרשת E	2
d מרשת D	2
d מרשת H	2
d מרשת K	1

שבוע $\mathbb{VII}$ ו שכבה 3 ו-IP

הרצאה

נמשיך את תיאור אלג' ה-STP שלא סיימנו בהרצאה הקודמת, היזכרו בשני השלבים הראשונים - מציאת השורש וחישוב עץ מסלול קצר ביותר אליו מכל קודקוד.

הערה לברדוקאסט דומיינים אין שום יכולת חישוב, ולכן כל דבר שנעשה ב-STP וכל תקשורת סוויצ'ים אחרת תהיה חייבת להיות על הסוויצ'ים.

3. חישוב עץ פורש לברודקאסט דומיינים.

(א) נשדר בפורטים שלנו את המרחק שלנו מהשורש.

(ב) אם נשמע מפורט מסוים (כלומר ברודקאסט דומיין ספציפי) שיש מישהו יותר קרוב לשורש מאיתנו, ננון את הפורט הזה כי אנחנו לא הסוויץ' האופטימלי בדומיין, אחרת הפורט הזה הוא ה-designated port של אותו הדומיין.

(ג) בכל מקרה, נשמור את הפורט שמוביל אותנו לשורש, ה-root port, פתוח להאזנות, ואם יש לנו designated ports כלשהם אז גם להעברת ההודעות הלאה.

הערה השלב השני ב-STP הוא חישוב עץ פורש לטופולגיית הסוויצ'ים והשלב השלישי הוא חישוב עץ פורש לטובת הברדוקאסט דומיינים (הם עצמם פסיביים).

הערה הודעות בעניין STP (כל מה שאנחנו משדרים כל הזמן לסוויצ'ים האחרים) נקראות הודעות **B**ridge **P**rotocol **D**ata **U**nits, ואותן אנחנו נשלח דרך כל הפורטים גם אחרי שניוונו חלק מהם לאי-שליחת הודעות עם תוכן של יחידות קצה.

הערה במציאות, כל שלושת השלבים שתיארנו (מציאת השורש, יצירת עץ פורש לסוויצ'ים ואז לברודקאסטים) קורים בו זמנית, כלומר נשלח הודעות BPDU שמכילות את השם שלנו, מי לדעתנו השורש ומה המרחק שלנו ממנו ביחד.

כתובות IP

אם אנחנו יכולים לחבר ברודקאסטים עם סוויצ'ים, למה לא לעשות שכל האינטרנט יהיה LAN אחד גדול? יש בעיית יעילות מבחינת STP וניוון של הרבה מאוד פורטים ושליחה למרחקים וכו', אבל הבעיה המרכזית היא שאנחנו חייבים לעבוד עם כתובות בעלות משמעות גאוגרפית.

כתובות MAC מספרות לנו רק מי ייצר את כרטיס הרשת אבל הן קבועות ולא משנה באיזו מדינה נהיה הן לא ישתנו. לכן נצטרך להוסיף עוד שכבה שמכילה ניתוב היררכי (נשלח לצ'כיה, ומשם לפראג, ומשם לרובע של המען ומשם למען עצמו), כי אחרת כל סוויץ' היה צריך לשמור כל כתובת של כל קודקוד שמישהו אי פעם תקשר איתו, שזה מיליארדים של ערכים שאי אפשר ליעל את ההחזקה שלהם.

הכתובות הגלובליות האלה נקראות כתובות IP, והן באורך 32 ביטים (8 תווים הקסדצימאלים) שניתנים (ויכולים להשתנות) לכל משתמש קצה או נתב (שהן הישויות שקיימות בשכבה 3).

במקום לשמור את הכתובות של כל עיר בעולם, נוכל לשמור קבוצות היררכיות של כתובות IP, הרעיון הזה נקרא **C**lassless **I**ntraDomain **R**outing איך נעשה את זה? נשמור subnets, לדוגמה $200.23.16.0/23$ משמעו הסאבנט שמכילה את כל כתובות ה-IP ש-23 הביטים הראשונים שלהם הם $200.23.16.0$ (המרה לביטים) ואז 9 הביטים האחרים יכולים להיות מה שנרצה, כלומר $2^8 = 512$ אפשרויות בתוך מרחב הכתובות הזה.

תרגול

כל מי שרוצה לתקשר ברשת צריכה להיות כתובת IP - כתובת וירטואלית (שמשתנה תלוי ברשת) בניגוד ל-MAC הפיזית.

כשנשלח פקטה בשכבה 3, נצטרך חוץ מתוכן ההודעה להוסיף גם את כתובת ה-MAC של השולח והמען (משכבה 2 עוד), וגם את כתובת ה-IP של השולח והמען. התפקיד של ראוטרים בשכבה 3 הוא למפות סאבנטים של IP לפורטים השונים שלו.

כשיש לנו הודעה לשלוח, אם אנחנו יודעים שהמען הוא ב-LAN (כי הראוטר סיפר לנו מה-subnet שלנו), נשלח עם פרוטוקול משכבה 2 בהתאם לאיזה ברודקאסט אנחנו. אחרת, נשלח לראוטר את ההודעות ואצלו ממופים כבר פוטרים שיודעים לאן להעביר את זה הלאה.

חוץ מאת כתובת ה-IP שקיבלנו מהנתב (ראוטר), המחשב שומר גם את ה-subnet mask שקיבל מהראוטר, שהיא גם 32 ביטים, שמסמלים את כל הביטים שחשוב שיהיו זהים כדי שנדע אם כתובת IP אחרת היא איתנו באותו ה-LAN.

דוגמה ה-subnet mask $255.255.248.0$ אומר שאת $3 + 8 = 11$ הביטים האחרונים אפשר לשנות ולהישאר ב-LAN שלנו (נמיר לבינארי ונבדוק כמה אפסים יש בסוף).

הערה אפע"פ ש-2^{32} כתובות זה מספיק לכל העולם, הן חולקו באופן לא הוגן (ל-MIT יש הרבה מאוד כתובות אבל אז לא נשאר לארגונים שלא הספיקו לתפוס מרחבים בזמן). לכן לאט לאט עוברים לכתובות IPv6.

דרכים לקבלת IP

- מבקשים ממנהל הרשת כתובת שמתאימה ל-MAC שלנו שמעתה תשמש לנו כל פעם שנחבר את המחשב - לא פרקטי לחיבור ספונטני ובסקייל גדול.
- מקבלים אחד מה-ISP (בין אם בוחרים ובין אם מקבלים).

- מדברים עם רכיב ברשת שאוטומטית מקצה לנו כתובת IP.

DHCP

פרוטוקול ה-DHCP מכיל ארבעה שלבים, שניים אופציונאליים ושניים חובה (בהתאמה).

1. משתמש קצה ישלח הודעה בברודקאסט הודעה שמבקשת כתובת IP - הודעה מסוג DHCP Discover - עם כתובת IP מקור 0.0.0.0 (כי אין לו עדיין) וכתובת IP יעד 255.255.255.255 (כתובת שמורה לברודקאסט), ושדה transaction ID כדי למנוע התנגשויות בין שני משתמשי קצה שמבקשים IP.

2. שרת ה-DHCP עונה עם הודעת DHCP Offer שמכילה שדה yaddr עם כתובת IP שהשרת מציע למשתמש הקצה. ההודעה תהיה עם כתובת IP יעד שהיא 255.255.255.255 כי מי שביקש IP עדיין אין לו אחד, ובנוסף שדה lifetime שקובע לכמה זמן ה-IP יהיה תקף.

3. משתמש הקצה עונה עם DHCP Request, ששוב עם 0.0.0.0 ב-IP המקור, ו-255.255.255.255 לכתובת היעד ושלושת השדות שקיבל בהצעה.

4. השרת עונה עם הודעת DHCP ACK שמאשרר את הבקשה עם כתובת יעד שהיא ה-IP ב-yaddr עם השדות האחרים.

מעבר לפרוטוקול הנ"ל, שרת ה-DHCP לרוב גם יענה עם כתובת ה-IP של שרת ה-DNS ב-LAN ושל ה-default gateway וה-subnet mask של ה-LAN.

הערה ייתכנו כמה שרתי DHCP, והראשון שיציע לרוב "ינצח". עם זאת, נקבל כאן race condition שיכול להוות בעיית אבטחה.

הערה כדי לשלוח הודעה עכשיו, נצטרך את הכתובת ה-IP וה-MAC של המקור, שיש לנו עכשיו את שתיהן (אחת מובנת ואחת מ-DHCP). את כתובת ה-IP של היעד נניח שיש לנו באמצעות בקשה ל-DNS שזה שרת נוסף שמתרגם כתובות טקסטואליות לכתובות IP. כיצד נקבל את כתובת ה-MAC של היעד?

ARP

כל משתמש קצה מחזיק טבלת ARP שממפה בין כתובות IP לכתובות MAC. לכל כתובת יש TTL כי כתובות IP יכולות להשתנות עם הזמן. אם יש לנו כניסה תקפה בטבלה - נוסיף אותה לפקטה ונשלח.

אחרתת נצטרך לקבל את ה-MAC ממישהו אחר - נשלח בברודקאסט (בשכבה 2) בקשה לכתובת ה-IP שאנחנו רוצים שנקראת ARP Request ונקבל חזרה ARP Response שמכיל את כתובת ה-MAC הרצויה ממישהו שיודע.

אם אנחנו רואים ש-IP הוא לא ב-LAN שלנו (אי תאימות של סאבנט), נשלח את ה-ARP Request לראוטר (שהוא השער שלנו לעולם החיצון) והוא כבר ידאג להשיג לנו את הכתובת. איך נדע מה הכתובת של הנתב? ה-DHCP נתן לנו אותו.

הערה אם לא ברור לנו איך יש לנו פיסת מידע כלשהי, כנראה שהיא הגיעה מה-DHCP.

שאלה איך נדע האם כתובת ה-IP הנוכחית שלנו פעילה? נשלח ARP Request עם ה-IP שלנו ואם אף אחד לא יענה נדע שהיא פנויה.

שאלה כיצד נוכל לשים כמה DHCP-ים באותו ה-LAN? נחלק את ה-IP-ים שהם מחלקים לתתי טווחים.

דוגמה נרצה לשלוח הודעה למישהו ב-LAN אחרי שהצטרפנו עכשיו לרשת - חסרות לנו כרגע כתובת ה-IP שלנו וה-MAC של היעד.

1. נשלח בברודקאסט DHCP Discover ונקבל חזרה DHCP Offer.
2. נשלח בברודקאסט DHCP Request ונקבל חזרה DHCP ACK - עכשיו יש לנו כתובת IP.
3. נשלח ARP Request בברודקאסט ונקבל חזרה ARP Response - עכשיו יש לנו את כתובת ה-MAC של היעד.

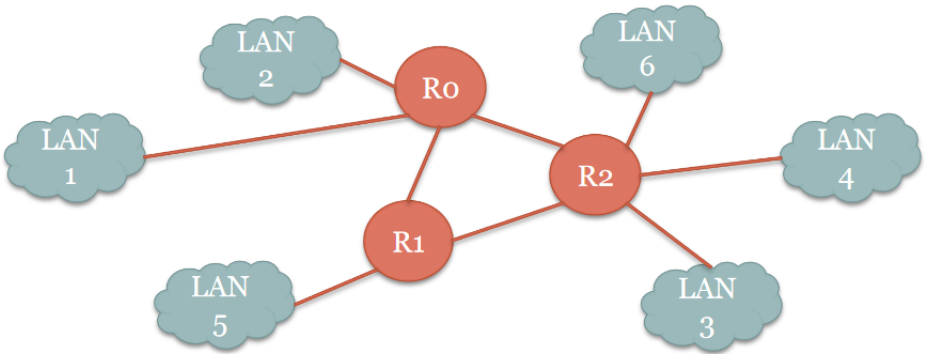
הערה בכל הדיונים כאן הסוויצ'ים לא לוקחים שום חלק בגלל שהם שקופים לחלוטין מבחינת יחידות קצה בשכבה 2.

דוגמה נרצה לשלוח הודעה למישהו ב-LAN אחר. עתה הראוטר יצטרך לעזור לנו לתקשר בין LAN-ים. נעשה זאת באמצעות תקשורת Hop-By-Hop, לפיה נשמור על כתובת ה-IP של המקור והיעד אבל נשנה בכל קפיצה בין יחידות בשכבה 3 את כתובת ה-MAC.

נניח שיש לנו נתב שמחבר שני LAN-ים שבאחד המקור A (כתובת ה-IP שלן) ובאחד היעד B (גם IP).

1. נשלח פקטה עם IP מקור יעד $A \to B$ (בהתאמה) ו-MAC משלנו (נסמן a) לשל הראוטר ב-LAN שלנו (נסמן c).
2. הראוטר ישלח פקטה עם IP $A \to B$ ו-MAC $d \to b$ כאשר d כתובת ה-MAC של הראוטר ב-LAN השני.
3. B יענה עם הודעה $B \to A$ ו-MAC $b \to d$.
4. הראוטר יעביר עם פקטה $B \to A$ ו-MAC $c \to a$.

שאלה כמה LAN-ים יש ברשת הבאה? 9, כי כל LAN חיצונית נחשבת ובנוסף כל צלע שמחברת בין שני ראוטרים (יצא במקרה הזה שיש LAN לכל רגל של ראוטר, לא תמיד ככה).



איור 10 : רשת שמורכבת מכמה ראוטרים ו-LAN-ים

שאלה מה קורה אם יש לנו DHCP רק מאחורי ראוטר ולא ב-LAN שלנו? נתקין DHCP Relaying Agent שיאפשר את פרוטוקול ה-DHCP Relay שעתה נתאר :

1. כשמשתמש ירצה לקבל IP, הוא ישלח DHCP Discover בברודקאסט.
2. הראוטר ישמע את זה וידע שהוא צריך להעביר (ביוניקאסט, כלומר ישירות) את ההודעה ל-DHCP שנמצא ב-LAN אחר, יחד עם שדה GIADDR שאומר ל-DHCP מאיפה הגיעה הבקשה ומכוח כך מאיזה אוסף כתובות להציע כתובת חדשה.

3. ה-DHCP יקבל את זה ויענה ביוניאקסט לראוטר עם DHCP Offer.
4. הראוטר יעביר בברודאקסט ב-LAN המקורי את ה-DHCP Offer.
5. נמשיך את ה-DHCP Request, ACK באותה הרוח (relaying דרך הראוטר ל-DHCP ב-LAN האחר).

DNS

נרצה שמשתמשים יוכלו לגשת לאתרים לא באמצעות כתובת IP שרירותית אלא באמצעות URL טקסטואלי. כאמור נשתמש בשרתי DNS כדי למפות טקסט ל-IP. כיצד הם יפעלו?

נשאל את שרת ה-DNS המקומי שלנו מה הכתובת של ה-URL שלנו. אם הוא לא ידע להגיד הוא יתחיל לחקור: הוא ילך אחורה בכתובת שקיבל רכיב רכיב (לדוגמה ב-$www.google.com$ נתחיל ב-com ואז $google$ ואז www) וישאל בכל פעם את שרת ה-DNS האחראי על הרכיב הנוכחי. אלו יפנו את השרת המקומי שלנו, ה-resolver, לשרת אחר יותר ויותר ספציפי עד שהשרת האחרון פשוט יענה לנו עם כתובת ה-IP הנדרשת ויחזיר לנו תשובה.

הערה השרת הראשון שאליו נלך כדי לקבל את כתובת שרת ה-DNS שיפנה אותנו לפי הרכיב הראשון שנקרא (האחרון בכתובת), קוראים ה-root DNS.

דוגמה נרצה לשלוח הודעה ל-URL מחוץ ל-LAN שלנו.

1. נקבל כתובת IP משרת ה-DHCP- ארבע הודעות (Discovery, Offer, Request, ACK).
2. נגלה מה-MAC של ה-DNS המקומי שלנו באמצעות ARP (ה-IP ידוע לנו מה-DHCP) - שתי הודעות (Request, Reponse)
3. נגלה מה-IP שמתאימה ל-URL שלנו עם DNS Request - שתי הודעות (Request, Response).
4. נגלה מה-MAC של הראוטר (ה-IP ידוע לנו מה-DHCP) - שתי הודעות (Request, Response).
5. נשלח את ההודעה הסופית עם ה-IP המקור והיעד (הסופי) ו-MAC הראוטר שלמדנו בשלבים הקודמים.
6. נקבל תשובה כפקטה מ-MAC הראוטר אבל IP המקור הרחוק (האמיתי), שמיועד לנו.

שבוע VIII | שכבה 3 לעומק

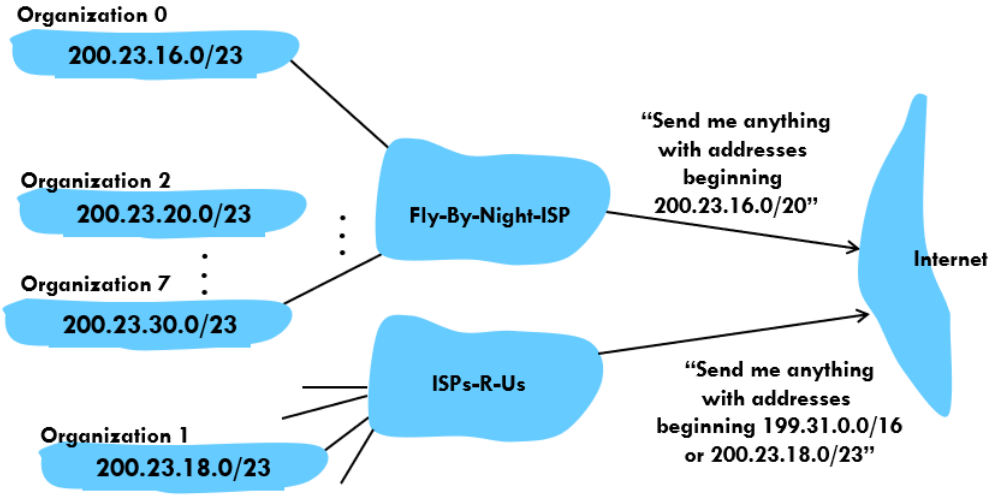
הרצאה

את הסאבנטים ניתן להסיק ע"י ניתוק נתב מטופולגיית משתמשי הקצה, ואז נקבל את כל רכיבי הקשירות שיכולים לדבר רק בתוך עצמם בשכבה 2, הלא אלו הסאבנטים.

דוגמה נניח שיש לנו ספק אינטרנט שמשרת 8 ארגונים. לכל ארגון יש סאבנט עם subnet mask של 23 ביטים, כלומר 2^9 כתובות לארגון. הספק חושף לאינטרנט שכל הודעה לסאבנט שמכיל את הארגונים שלו, צריכה להגיע אליו, והוא זה שיעביר הלאה את ההודעות לארגונים.

אם אחד הארגונים עובר ל-ISP אחר, יחד עם כתובות ה-IP שקנה (לכן לא ישתנו), עתה כל התקשורת לארגון תנותב בדרך פלא אליו דרך ה-ISP האחר, אפע"פ שה-ISP המקורי עדיין מכריז גם שהכתובות של אותו הארגון עדיין צריכות לעבור אליו. איך זה קורה?

כי הנתב של ה-ISP החדש יכריז שהוא זה שצריך להפנות אליו את התעבורה, רק שהוא יכריז את זה על סאבנט יותר ספציפי (רק של הארגון, במקום שמונת הארגונים יחד), וככלל מפנים תקשורת בשכבה 3 לממשק הכי ספציפי לכתובת שלנו (ראו איור).



איור 11: העברת סאבנט ל-ISP אחר

הדוגמה הנ"ל מדגימה מתקפה די חמורה - נוסיף ראוטר שמגדיר את מרחב הכתובות של גוגל בסאבנטים יותר ספציפיים ועכשיו כל התעבורה תעבור אלינו והצלחנו לצנזר את גוגל! זה לא כזה פשוט אבל יש מתקפות כאלה שקורות בטבע לא מעט.

הערה כל כתובות האינטרנט האלה, כדי שתהיה להן משמעות גאוגראפית/ארגונית, צריכות להתחלק ע"י גורם ריכוזי שמחלק (בתקווה באופן הוגן) את הסאבנטים. הארגון הזה נקרא ICANN. בעבר חולקו כתובות IP כמו זבל ולכן לעברית יש שני סאבנטים עם $/16$, כלומר 2^{17} כתובות, ול-MIT יש $/8$ כלומר 2^{24} שזה המון ולא לצורך.

פקטה של IP, מלבד כתובת ה-IP וה-MAC של המקור והיעד, מכילה גם עוד מטא-דאטה כמו כמה זמן הפקטה נמצאת במערכת כדי שלא תשלח יותר מדי פעמים בסיבובים, אבל גם איזה פרוטוקול מחכה לנו בשכבה מעליו (TCP, UDP וכו').

הערה את המטא-דאטה אי אפשר להצפין כי אז הראוטר לא ידע לאן להעביר את הפקטה, וזו בעיית פרטיות - בלי לדעת מה המידע, אם אני שולח פקטה למקום זדוני, אפשר לשייך אותי לפעילות זדונית.

Bootstrapping בשכבה 3

בעיית הבוטסראפ - איך להתחיל לעשות משהו מכלום - בהקשר של האינטרנט עד שכבה 3, כוללת שלוש בעיות:

- איך נקבל כתובת IP ששייכת לנו? DHCP.

- איך נקבל את כתובת ה-IP של משתמש שאנחנו רוצים לתקשר איתו? DNS.
- איך נקבל את כתובת ה-MAC של כתובת IP כלשהי? ARP.

ניזכר ב-DHCP שלמדנו בתרגול, שבו שולחים DHCP Request ו-ACK ולפעמים גם לפניכן DHCP Discover ו-Offer. את שני השלבים הראשונים לא צריך לדוגמה במקרה שאנחנו מחדשים את ה-IP שלנו, ואז אנחנו כבר יודעים איזה IP אנחנו רוצים ולכן אנחנו לא צריכים לבקש הצעה חדשה.

בגלל שתהליך אישור ה-IP קורה רק לאחר סיום ה-DHCP ACK, במהלך שתי (או ארבע-) ההודעות בפרוטוקול, לקוח ה-DHCP תמיד ישים את עצמו עם IP 0.0.0.0 ותמיד ישדר בברדוקאסט עם IP 255.255.255.255, ואילו השרת גם ישדר בברודקאסט אבל יזדהה ב-IP שכבר יש לו כמובן.

הערה DHCP הוא בכלל פרוטוקול בשכבה 5, כלומר אנחנו שולחים פקטה מעל UDP, שהוא פרוטוקול שכבה 4 שנלמד בקרוב, שהיא למעשה פקטה מעל IP.

הפקטה הזו יורדת לשכבת ה-UDP, שיורדת לפקטה מעל שכבת ה-IP, שיורדת לפקטה בשכבה 2, שנשלחת לראוטר שאיתנו ב-LAN, והוא מרכיב חזרה את כל הרכיבים עד שהוא מקבלת פקטת DHCP חזרה, ואז עונה תשובה ועושה את כל התהליך בדרך חזרה.

רשומות DNS ו-hostname resolving

כתובות אינטרנט ל-IP זו לא העתקה חח"ע או על. למה? אין סיבה, לדוגמה, שבישראל נופנה לשרת של גוגל בארה"ב, במקום אחד קרוב יותר אליו בישראל - זה חשוב הן מבחינת פיצול עומס הן מבחינת Quality of Service ועוד הרבה סיבות.

לכן DNS-ים שונים יכולים לענות תשובות שונות בזמנים/מקומות/תנאים שונים. הרעיון של caching של מידע שלא חייב להגיע ממקור ספציפי (בניגוד לזום לדוגמה), הוא השירות שמציעות Content Delivery Networks, שהן קבוצות שרתים שמספקים את אותו המידע (לדוגמה סרט בנטפליקס) מכמה מקומות שונים, הן בשביל redundancy והן בשביל לשפר את איכות השירות מהסיבות שכבר הזכרנו.

בתור משתמש קצה, אנחנו מתקשרים רק עם ה-DNS המקומי שלנו, ה-resolver, והוא יעשה את העבודה הקשה ויענה לנו בסוף עם תשובה. ה-resolver ישאל את ה-root שלו מה ה-IP שמתאימה לכתובת שלנו. אם הוא יודע, הוא יענה, אחרת, הוא יפנה אותנו למישהו שאמור לדעת. התהליך הזה ממשיך ככה שוב ושוב עד שיגיע לשרת ה-DNS ה-authoirative, כלומר זה שיש לו תשובה ולא מפנה אותנו לעוד גורם.

הערה במציאות יש הרבה קאשינג ככה שאפילו אם אין לנו את ה-IP של URL כלשהו, יכול להיות שיש לנו IP של DNS כלשהו באמצע ככה שלא נצטרך להתחיל את התהליך החל מה-root אלא שנוכל להתחיל אותו האמצע. יש 13 שרתי root, רובם כמובן בצפון אמריקה.

איך נראת רשומה בשרת DNS? יש הרבה מאוד סוגים של רשומות, נביט בכמה מהן:

- Type=A - רשומה של DNS סמכותי, שמכילה שם של משתמש קצה (URL נגיד) ואת ה-IP שלו.
- Type=NS - רשומה שמפנה ל-DNS סמכותי, שמכילה שם של דומיין (סיפא של URL נגיד) ואת שם ה-DNS הסמכותי עליו (מי הבא בתור בחיפוש של ה-resolver).

הערה כל רשומה מכילה בנוסף TTL - לכמה זמן הרשומה רלוונטית. ככל שה-DNS יותר גבוה (חולש על דומיין יותר גבוה), כך הזמן שנשמור אותו יהיה יותר ארוך, כי סביר שהוא יותר יציב מהבנים שלו.

דוגמה נבקש מה-resolver את ה-IP של $www.google.com$.

1. נבקש מה-resolver את ה-IP של ה-URL.
2. ה-resolver יבקש מה-root את ה-DNS הסמכותי ל-com..
3. ה-root יענה עם שתי רשומות - רשומת NS שמכילה את שם השרת הסמכותי לכתובת הזו, ו-A שמספרת לנו מה ה-IP של השרת הסמכותי שעכשיו סיפרו לנו עליו.
4. ה-resolver יבקש מהשרת של com. את ה-IP של $google.com$.
5. השרת הסמכותי של com. יענה עם שתי רשומות בדומה למה שענה ה-root.
6. ה-resolver יבקש מה-DNS הסמכותי שקיבל בשלה הקודם את $www.google.com$.
7. ה-DNS הסמכותי יענה לו עם רשומת A שמכילה את ה-IP שאנחנו צריכים.

תרגול

מעל שכבה 3 כבר יש לנו את היכולת לתקשר בין שני אנשים במקומות שונים. איך אם כן, נוודא מעבר אמין של תקשורת ברשת? עד כה אין לנו שום הבטחה על סדר, נכונות ועוד תכונות של התוכן שמועבר ולכן נרצה פרוטוקולים שיבטיחו לנו את זה.

Stop & Wait

- מקבל כשיקבל פקטה, יענה ACK אם קיבלנו פקטה ו-NACK אם קיבלנו פקטה שהושחתה בדרך.
- שולח ישלח פקטה, יחכה T_{out} כלשהו. אם קיבל NACK או כלום, נחזור בשנית על התהליך.

דוגמה נדגים למה זה אלג' לא משהו.

1. השולח שלח הודעה שהגיעה למקבל.
2. המקבל עונה ACK שנופל בדרך.
3. השולח לא קיבל ACK בזמן אז הוא שולח את אותה הפקטה שוב.
4. המקבל קיבל פקטה שהוא חושב שהיא חדשה, אפילו שהיא העתק של הקודמת ולא מידע חדש.

הבעיה נפתרת ע"י מספור ההודעות - כך נדע להבדיל בין פקטה שהיא העתק של פקטה קודמת ופקטה חדשה

דוגמה נדגים למה זה אלג' עדיין לא משהו.

1. השולח שולח הודעה שהגיעה למקבל.

2. המקבל עונה ACK שלוקח לו זמן להגיע לשולח.
3. השולח לא קיבל ACK בזמן ולכן שולח את הפקטה שוב.
4. השולח מקבל ACK שהוא חושב שהוא על ההודעה השנייה (ההעתק) ששלח אבל היא לא.
5. השולח שולח את הפקטה השנייה.
6. המקבל עונה ACK על העתק הפקטה הראשונה שהגיעה אליו רק עכשיו.
7. השולח חושב שהמקבל אישר לו את קבלת הפקטה השנייה.

בשלב הזה, המקבל חושב שהפקטה הבאה שתישלח היא השנייה, ואילו השולח חושב שזו השלישית!

הבעיה נפתרת ע"י מספור הודעות ה-ACK.

מהו ה-T_{out} האידאלי שימנע שליחת הודעות חוזרות שוב ושוב, שעדיין יאפשר תקשורת מהירה?

הזמן שצריך להתחשב בו ל-T_{out} הוא הזמן בין שליחת הביט האחרון של ההודעה מהשולח ועד קבלת הביט האחרון של ה-ACK מהמקבל.
הזמן הזה הוא T_{prop} (זמן השליחה למקבל) + T_{prop} (זמן המענה של המקבל לשולח) + T_{ack} הזמן שלוקח לקרוא הודעת ACK.

הערה בעיקרון יש גם את הזמן שלוקח לשולח לבדוק שההודעה תקינה T_{pt}, אבל לרוב נניח שהוא 0.

דוגמה גודל הפקטה שלנו הוא $2000bit$, גודל ה-ACK/NACK הוא $200bit$, קצב השידור הוא $10Kbps$, המרחק בין התחנות הוא $1000km$, מהירות השידור $2 \cdot 10^8$, $T_{pt} = 0$.

- מה הוא T_{prop}?

  $T_{prop} = \frac{distance}{prop\ speed} = \frac{10^6 m}{2 \cdot 10^8 m/s} = 5 \cdot 10^{-3}$.
- מה הוא T_{ack}?

  $T_{ack} = \frac{200bit}{10^4 bit/s} = 2 \cdot 10^{-2} s$.
- נניח ששולח ומקבל משתמשים ב-Stop & Wait עם שדרוג: השולח שולח L פקטות ברצף ואז מחכה T_{out} זמן שבו הוא מחכה להודעות ACK על כל ההודעות, ואם לא קיבל על כולן, ישלח את כל הפקטות יחד. מה הוא T_{out} האופטימלי, בהינתן ש-T_{ack} זניח?

  להודעה האחרונה יקח T_{prop} זמן להגיע ואז עוד T_{prop} לקבל ACK על ההודעה האחרונה, כלומר $2T_{prop}$.
- נניח עתה כי $L = 4$, $T_{prop} = 2$ (ביחידות זמן כלשהן), ו- $T_{packet} = 1$. מה יקרה אם פקטה 3 תיפול?

  פקטה מספר אחד תסיים לצאת מהשולח בסוף (יחידת זמן) 1 ותגיע בשלמותה למקבל בסוף 3 ($T_{prop} = 2$). משם יצא ACK חזרה לשולח שיגיע בסוף 5.

  בדומה השולח יקבל ACK על פקטה 2 בסוף 6 ועל פקטה 4 בסוף זמן 8, אבל לא המקבל לא ישמור את תוכנה אצלו בזיכרון כי הוא עובד רק לפי הסדר והוא לא קיבל את 3.

  אחרי זמן 4 השולח יתחיל לספור $T_{out} = 2 \cdot 2 = 4$ יחידות זמן, ובזמן 8 יקבל הודעות ACK על $1, 2, 4$ אבל לא על 3 כי היא נפלה באמצע.

  השולח ישלח את הכל מחדש והפעם כן נקבל את הכל ונוכל להתקדם בחיים.

שאלה מה-goodput של Stop & Wait, כאשר ההסת' שפקטה (עם תוכן או ACK) נופלת היא p?

סך הזמן הכולל ששידרנו בהצלחה הוא T_{packet}. נגדיר X מ"מ מספר השליחות הממוצע של הפקטה. זהו מ"מ שמתפלג $\text{Geo}\left((1-p)^2\right)$ כי אנחנו שולחים פקטה ורוצים ACK וחוזרים על התהליך הזה שוב ושוב עד שנצליח. התוחלת של X אם כן היא $S = \frac{1}{(1-p)^2}$.

מספר הניסיון הלא מוצלחים שלנו הוא $S-1$, והזמן שמבזבז כל ניסיון כושל הוא $T_{packet}+T_{out}$ (שולחים ומחכים את כל T_{out}).

ניסיון מוצלח (יש לנו 1 כזה), לוקח $T_{packet}+T_{prop}+T_{pt}+T_{prop}+T_{ack}$ כי צריך לשלוח את הפקטה, ואז לוקח לה זמן להגיע, שהמקבל יעבד אותה, ישלח חזרה הודעת ACK, ושנקרא אותה.

סה"כ נקבל

$$\eta = \frac{T_{packet}}{\left(\frac{1}{(1-p)^2}-1\right)(T_{packet}+T_{out}) + (T_{packet}+2T_{prop}+T_{ack}+T_{pt})}$$
$$(*) = \frac{T_{packet}}{\frac{1}{(1-p)^2}(T_{packet}+T_{out})}$$

$(*)$ נציב את T_{out} האופטימלי, שהוא כמו שחישבנו למעלה בדיוק $T_{packet}+2T_{prop}+T_{ack}+T_{pt}$.

שאלה נחזור לשאלה עם החלון.

- מהי ההסת' שחלון בגודל L כמו בדוגמה הנ"ל יפול בדרך, בהנחה ש-ACK-ים לא נופלים?

ההסת' שחלון הוא מוצלח היא $(1-p)^L$, ולכן התשובה היא $1-(1-p)^L$, ו-$P_{suc} \sim \text{Ber}\left((1-p)^L\right)$ הוא המ"מ אינדיקטור שמייצג האם שידרו חלון בהצלחה.

- מה ה-goodput של S&W עם חלון בגודל L?

תוחלת הזמן האפקטיבי שבו שלחנו הודעות $T_{suc} = L \cdot T_{packet} \cdot P_{suc}$ היא $L \cdot T_{packet}(1-p)^L$ (L פעמים לוקח T_{packet} זמן, ובהסת' $(1-p)^L$ שידרנו בהצלחה).

סה"כ הזמן שאנחנו משדרים הוא $L \cdot T_{packet} + 2T_{prop}$ (שולחים את כל ההודעות ומחכים $T_{out} = 2T_{prop}$). לכן סה"כ

$$\eta = \frac{L \cdot T_{packet}(1-p)^L}{L \cdot T_{packet} + 2T_{prop}}$$

- נציע שינוי לאלג' : לא נשדר ACK-ים חדשים על הודעות שכבר קיבלנו בניסיון קודמים וכך השולח יוכל להזיז את החלון שלו לפי הסדר (אם קיבלנו את $1,2,4$ בחלון הקודם, השלח ישדר הפעם את $3,4,5,6$). מהו זמן המחזור בין כל שני חלונות T_{period}, הזמן שעובר בין תחילת שני חלונות?

הזמן הזה הוא $4T_{packet}+T_{out}$ (נניח מעתה $L=4$) כי אנחנו משדרים הכל, ומחכים T_{out} לפני שמתחילים לשדר שוב.

- מה ההסת' שאף פקטה לא התקבלה בזמן מחזור כלשהו, כך שהשולח יאלץ לשלוח שוב את אותו החלון?

p, כי צריך רק שהראשונה תיפול ונצטרך לשדר שוב מהתחלה.

- מהו ה-goodput?

הזמן הכולל נשאר זהה לחישוב הקודם - $T_{tot} = 4T_{packet} + T_{out}$.

מהו T_{suc}? נחלק למקרים שהראשונה נפלה, השנייה נפלה וכו'.

$$T_{suc} = \begin{cases} 4T_{packet} & (1-p)^4 \\ 3T_{packet} & (1-p)^3 p \\ 2T_{packet} & (1-p)^2 p \\ T_{packet} & (1-p) p \end{cases}$$

כאשר החישוב (לדוגמה על $2T_{packet}$) הוא שאנחנו רוצים ששתי ההודעות הראשונות יצליחו, אבל השלישית תיכשל, ובדומה השאר.

לכן סה"כ ה-goodput הוא $\eta = \frac{E[T_{suc}]}{T_{tot}} = \ldots$.

- כמה פקטות נצטרך לשלוח בממוצע כדי שפקטה מספר שלוש תגיע בהצלחה?

נסמן Y מ"מ שסופר את מספר החלונות שנצטרך לשלוח עד שפקטה 3 תגיע בהצלחה. נסמן X_i מספר הפעמים ששלחנו את פקטה i עד שתגיע בהצלחה.

נשים לב כי $Y \neq X_1 + X_2 + X_3$ כי אם 1 הצליח להישלח, יכול להיות שגם 2 הצליח באותו החלון וכך גם בין 2 ו-3 ולכן יש חפיפה בין הזמנים האלה.

$X_1 \sim \text{Geo}(1-p)$ כי אנחנו שולחים שוב ושוב עד שלא ניפול, ולכן $E[X_1] = S = \frac{1}{p-1}$, וזה נכון גם ל-$X_2$ ו-X_3.

התוחלת של Y אם כן, היא $(S-1) + (S-1) + S$ כלומר כל הניסיונות של שליחת 1 בלי האחרון המוצלח, שבו גם ל-2 יש סיכוי להצליח בו. על זה נוסיף את הניסיונות של 2 (שמתחילים למעשה באחרון המוצלח של 1), בלי האחרון המוצלח. על זה נוסיף את הניסיונות של 3 (שמתחילים באחרון של 1) יחד עם האחרון כי אין לא חפיפה עם שום דבר אחריו.

שבוע $\mathbb{IX}$ ו DNS ובעיות אבטחה

הרצאה

דוגמה נניח שהקמנו סטראט-אפ FooBar ואנחנו רוצים אתר עם יכולת לנהל תעבורה בתוך הארגון.

1. נקבל בלוק כתובות IP מה-ISP שלנו.
2. נרשום את foobar.com אצל DNS שאחראי על ה-TLD $.com$, ונוסיף אצלו רשומה שמפנה ל-DNS הארגוני שלנו.
3. נקים DNS ארגוני שלנו ובתוכו רשומות מסוג A לכתובות שלנו.

DNS הוא גורם סופר ריכוזי ובעייתי ורבות מהמתקפות הן עליו. בלי ה-DNS אין לנו שום גישה לאינטרנט באמת.

אם אנחנו שרת מייל, חשוב מאוד למנוע גישה למיילים שאנחנו אחראיים עליהם, כי באמצעותם אפשר לגשת להרבה מאוד שירותים רגישים. אם מצליחים להשתלט על ה-DNS של שרת המייל - התוקף ניצח.

אם אנחנו אתר עם מידע רגיש, חשוב מאוד שלא ישתלטו על ה-DNS שלנו כי אז אפשר יהיה לחקות אותנו לגנוב מידע ממשתמשים תמימי דעים (שלא שמים לב שאין לנו סרטיפיקאט HTTPS לדוגמה).

בעיות אבטחה ב-DNS

1. אפשר להספים את ה-DNS עד שיקרוס ואף אחד לא יוכל לגשת לאינטרנט.

   זה קרה ב-2002 ו-2015 במתקפת DDoS על ה-root DNS-ים, אבל לא הזיק באמת כי הכל cached במספיק רמות שמעט מאוד היו צריכים את ה-root בשלב ההוא.

2. אם אנחנו בבית קפה, מי ששולט ב-DNS המקומי יכול לענות לנו מה שהוא רוצה על שאילתות אליו ולהתחזות לגורמים רגישים, ולנו לא יהיה מושג. מעבר לכך, מישהו אחר ברשת יכול לענות לפני ה-DNS ונתחשב רק בתשובה שלו.

3. Cache Poisoning - אם תוקף מצליח להכניס רשומה אחת אפילו (בעלת השפעה) ב-cache של ה-DNS המקומי שלנו, הוא יכול לבחור TTL הכי גבוה שאפשר והמתקפה תחיה הרבה מאוד זמן.

   יש פה משהו לא אינטואטיבי מבחינת אבטחה - אנחנו נותנים ל-DNS לספר לנו כמה אמין הוא (כמה זמן להחזיק רשומה) ואנחנו מאמינים לו גם.

Cache Poisoning

כדי שמתקפה הזו תצליח התוקף צריך להקדים את ה-Authoritative וגם להכיל Transaction ID זהה לזה של הבקשה. זה יכול לקרות או באמצעות ניחוש שזה לא סביר (off-path attack), או באמצעות האזנה לבקשה של ה-resolver (man-in-the-middle attack).

אם אנחנו off-path הסיכוי שנקלע ל-transaction id וגם נשדר בדיוק בזמן שפגה הרשומה ב-cache הוא מאוד נמוך, לכאורה.

התקיפה הכי טובה שאנחנו יכולים לעשות היא כזו: נקים מאות לקוחות של ה-resolver, שיגישו שאילתות לכתובות אקראיות, ובאותו הזמן נציף את ה-DNS עם תשובות עם transaction ids אקראיים. מפרדוקס יום ההולדת, ההסת' שנצליח להכניס רשומה היא 1. אם התוקף מפסיד במרוץ לתשובה, הוא יצטרך לחכות הרבה זמן עד שהרשומה תפוג. עד כאן נשמע שאי אפשר לעשות יותר מדי, אבל ב-2008 מייקל קמינסקי הוכיח אחרת.

המתקפה של קמינסקי היא כזו: נציף את ה-resolver עם שאילתות מהצורה $i.google.com$ לכל i. בכל בקשה כזו יש מרוץ בינינו, התוקפים, ל-DNS האמיתי, כי אף אחד לא חיפש את הכתובת הזו כי אין מה לחפש שם. בסופו של דבר, נצליח להכניס רשומה זדונית ל-cache של ה-DNS המקומי. התגובה הזדונית שלנו תהיה מהצורה

google.com NS www.bad.google.com

www.bad.google.com A 6.6.6.6

כלומר מפנים את ה-resolver אלינו (הרשעים) בכל השאלות על כתובות מהצורה $*.google.com$. כך, בשלב כלשהו רשומת ה-NS ל-google.com כן תפוג ואז אנחנו מתחרים עם התשובה הזדונית שלנו עם ה-DNS של com. (שהיה עונה לנו שלכל השאלות בעניין $*.google.com$, יש לדבר עם X האמיתי של גוגל). אם ניצחנו את ה-DNS של com. במרוץ וגם צדקנו ב-transaction ID, הרי שעכשיו אנחנו שולטים בכל ההפניות של לקוחות ה-ISP ל-$www.google.com$ לשנתיים.

המגבלה היחידה במתקפה הזו היא רוחב הפס שלנו כתוקפים להספמה, כי אפשר להתחיל את המתקפה מתי שרוצים בלי קשר ל-TTL של הרשומה אצל ה-resolver.

הפתרון היחיד בטווח הארוך כרגע הוא DNSSEC - דרך ל-DNS להוכיח שהוא באמת קיבל את הזכות להיות מתחת למי שמעליו בהיררכיה, וכך נוכל לשאול את מי שמעליו אם הוא באמת הגורם האמיתי והוא יוכל לאמת או להפריך זאת. הבעיה היא שזה דורש מנגנון עם מפתחות פומביים פורטיים וכזכור קשה להכניס מנגנונים חדשים לאינטרנט.

תרגול

נמשיך בחיפוש אחר אלג' מוצלח לשליחת הודעות אמינה מעל שכבה 3.

Go Back N

אלג' Go Back N הוא גרסה משופרת של S&W. נניח שיש לנו חלון באיזושהו גודל.

המקבל בכל פעם שמקבל פקטה חדשה, אם היא ברצף עם ההודעות הקודמות שקיבל (לדוגמה קיבלנו $1, 2$ ועכשיו 3), שולח ACK עם מספר הפקטה + 1 - סימון לכך שהמקבל אומר לשולח "קיבלתי את כל ההודעות עד X (לא כולל)".

השולח מזיז את החלון בהתאם כשהוא מקבל את ה-ACK ושולח בכל פעם את הודעות מהחלון לפי הסדר, ופותח טיימר על כל פקטה, כאשר אם לא קיבל אישור אחרי איזשהו T_{out}, הוא שולח את הפקטה שוב.

אינטואטיבית, השולח שולח פקטות ועל כל אחת מהן יש לו טיימר נפרד שלפיו הוא שולח שוב ושוב את הפקטה עד שהיא מגיעה, ואז כשמצליחים לשלוח, הוא נפטר מהטיימר של הפקטות שיצאו מהחלון ויוצר טיימרים חדשים לפקטות החדשות שיש לשלוח.

למה החלון עוזר לנו? אם פקטה נפלה בשליחה זה לא שונה מ-S&W ממוקבל. אבל, אם ACK מהמקבל נפל - יקרה דבר אחר. תלוי בגודל החלון והטיים-אוט, השולח יסיים לשלוח את כל החלון ולא יוכל להזיז את החלון כי לא קיבל ACK על הפקטה שנפלה באמצע. לכן הוא ישלח שוב את הפקטה שלדעתו נפלה אבל מהר מאוד יקבל מהשולח ACK שלא נפל שמודיע לו שהוא יכול להזיז את החלון ביותר מצעד אחד.

דוגמה נפל ה-ACK של פקטה 4 אבל $4, 5, 6$ התקבלו, אז המקבל ישלח ACK 7 וזה יגיע לשולח במוקדם או מאוחר והוא יוכל להזיז את החלון בכמה צעדים כבר.

את הפקטות שאנחנו שולחים אנחנו ממספרים עם מספר כלשהו של ביטים b (מספר נמוך כי אנחנו לא רוצים שכל הפקטה יהיה header של אינדקס), ואז ממספרים את הפקטות מודולו 2^b (פשוט אין יותר ביטים לייצוג המספר).

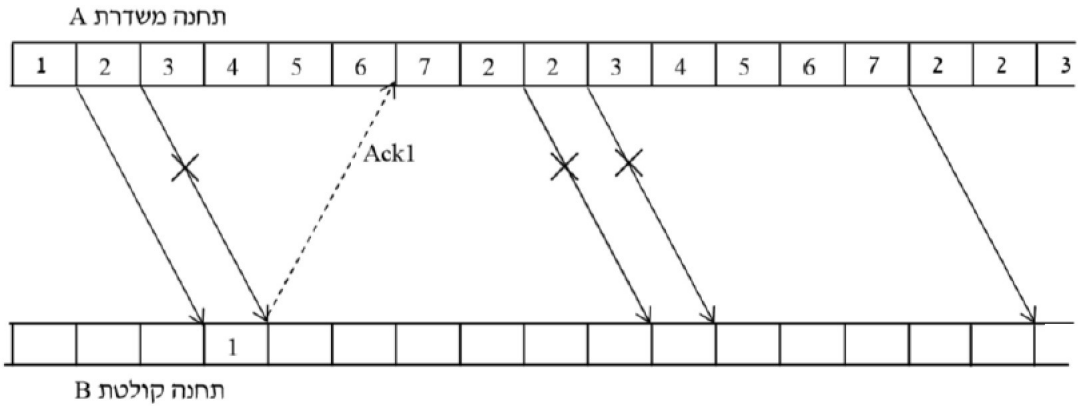
דוגמה נראה למה צריך למספר את הפקטות במודולו גבוה מגודל החלון. נניח שיש לנו חלון בגודל 10 וכל הפקטות הגיעו למקבל אבל כל ה-ACK-ים לא התקבלו ע"י השולח. לכן השולח ישלח את כל הפקטות שוב, והשולח יחשוב שזה כבר רצף של הודעות חדשות כי מבחינתו הוא אישר קבלה. כך קורה שהשולח והמקבל כבר לא מסכימים על תוכן הפקטות ונהרס לנו המידע.

מהדוגמה הנ"ל נובע בחלון בגודל N, צריך לפחות $\lceil \log(N+1) \rceil$ ביטים כדי בשביל אינדוקס הפקטות (ואז בדוגמה הנ"ל היה יוצא שהנשלח את הפקטה ה-10 והמקבל יודיע שהוא קיבל את כל האלה שלפני והם ידעו שהם מדברים על אותו הדבר).

שאלה נסתכל על GBN בוריאציה הבאה: ACK מצליח בהסת' 1, פקטה נופלת בהסת' p. כש-A מזהה כישלון (נגמר הטיים-אווט), הוא שולח את הפקטה L פעמים גם אם הוא מקבל עליה ACK כבר. המקבל עונה רק פעם אחת על כל פקטה חדשה ומתעלם מהשאר.

- בהינתן ש-$L = 2, T_{prop} = 2, T_{packet} = 1, T_{pt} = 1$, גודל החלון 6 ופקטה 2 נכשלת שלוש פעמים (והשאר מגיעות), כיצד יראה תרשים התקשורת בין השולח למקבל?

ראשית $T_{out} = 2T_{prop} + T_{pt} + T_{ack} = 2 \cdot 2 + 1 + 0 = 5$. לכן הודעה מספר 2 תיכשל, ולאחר 5 פקטות אחרות שישלחו $(3-7)$, השולח יקבע שההודעה לא נשלחה בהצלחה וישלח את 2 פעמיים, אלו לא יגיעו, הוא ישלח שוב גם את השאר, ואז בפעם הבאה שישלח את 2 פעמיים זה כן יצליח.



איור 12: תרשים התקשורת בין השולח והמקבל

- מה ה-goodput של הפרוטוקול?

את ה-goodput אפשר לחשב או לפי הסתכלות על חלון זמן וחישוב כמה פקטות בתוחלת נשלח בהצלחה בזמן הזה (T_{suc}), או לפי הסתכלות על פקטה וחישוב כמה זמן יקח לה בתוחלת להישלח בהצלחה, אנחנו נעשה לפי השיט השנייה.

מתקיים $\eta = \frac{T_{packet}}{E[T_{waste}]}$ כאשר T_{waste} מ"מ המייצג כמה זמן ביזבזנו כולל השליחה המוצלחת בסוף.

נסמן p_k ההסת' שהצלחנו לשדר בבליץ L ההודעות ה-k ו-$T_{waste,k}$ הזמן שביבזבנו אם נכשלנו $k-1$ בליצים ואז הצלחנו ב-k.

לכן $E[T_{waste}] = \sum_{k=0}^{\infty} p_k T_{waste,k}$ (תוחלת שלמה).

עבור $k = 0$, ההסת' היא $1-p$. ההסת' שנצליח אחרי בליץ השידור הראשון היא $p_1 = p\left(1-p^L\right)$ (לא הצלחנו פעם אחת, ואז לא נכשלנו לפחות פעם אחת), ובצעד ה-k-י, $p_k = p \cdot \left(p^L\right)^{k-1}\left(1-p^L\right)$.

הזמן שאנחנו מבזבזים לאחר k בליצים הוא $T_{packet} + k\left(T_{out} + LT_{packet}\right)$ (טיימר הטיים-אווט מתחיל מסוף הבליץ). את הטור לא נחשב ונשאיר לסטודנטית המשקיעה, אבל נראה שהתוצאה הסופית יוצאת

$$\eta = \frac{1-p^L}{1-p^L + p\left(a+L-1\right)}$$

כאשר $a = \frac{T_{out}+T_{packet}}{T_{packet}}$.

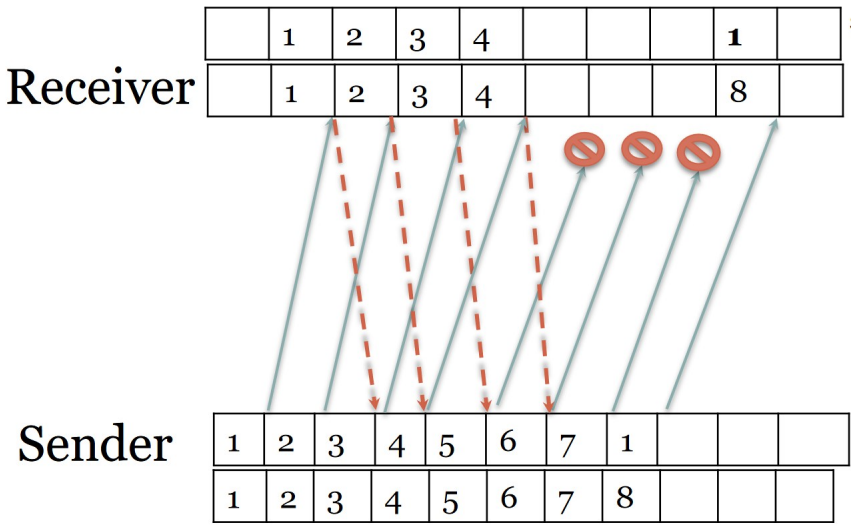
- מה ה-goodput של GBN קלאסי?

זה פשוט המקרה הנ"ל עם $L = 1$! לכן התשובה היא $\frac{1-p}{1-p+pa}$.

Selective Repeat

נניח שלמקבל יש באפר (בגודל חלון השולח) שהוא שומר בו פקטות שקיבל, השולח שולח בחלון זז, ו-ACK-ים משמעם רק "קיבלתי את ההודעה ה-i" (ולא ACK צובר כמו ב-GBN). המקבל, כמו השולח, מזיז את הבאפר שלו לאינדקס הראשון שעוד לא קיבל בכל פעם, הוא מעביר הלאה את המידע שעתה קפץ מעליו בהזזת החלון.

דוגמה נניח גודל חלון $N = 4$, ומספור פקטות עם $2N - 1$ מספרים (מודולו 7). נראה למה $2N - 1$ אינדקסים לפקטות זה לא מספיק. בתרשים הבא השולח מצליח לשלוח את $1 - 4$ ואז $5 - 7$ נכשלות אבל $8 = 1 \mod 7$ כן מצליח להישלח. המקבל לא יודע האם ההודעה החדשה שממוספרת ב-1 היא 1 (ייתכן שה-ACK שלו לא התקבל) או 8!



איור 13: תרשים התקשורת שגורם להתבדרות

לכן נדרש לפחות $2N$ מספרים בחלון, כלומר $\lceil \log 2N \rceil$ ביטים.

שאלה נניח שיש לנו SR אופטימלי, כלומר, גודל חלון ובאפר אינסופי, אינסוף ביטים למספור הפקטות, ACK לא נכשלים ו-T_{ack} זניח. בנוסף נניח שפקטות נופלות בהסת' p.

- מה ה-goodput?

מספיק לנתח לפקטה אחת (כי כל הפקטות ב"ת ומתפלגות אותו הדבר ולכן החישוב עובר הכללה טריוויאלית לאינסוף הודעות). נחשב ראשית באמצעות $\frac{T_{packet}}{E[T_{waste}]}$. T_{waste} מתפלג גאומטרית עם פרמטר $1 - p$ (ההסת' שמצליחים) כפול T_{packet} ($= T_{out}$ T_{packet} כי השאר זניח), לכן תוחלתו $\frac{1}{1-p}T_{packet}$ ואז $\eta = 1 - p$.

שבוע $\mathbb{X}$ | ניתוב תוך-ארגוני

הרצאה

כשחולקו כתובות IP, הן חולקו באופן לא הגיוני/שוויוני וזה יוצר כיום בעיות. לשם כך המציאו את IPv6, בו יש 40 ביטים לכתובת במקום 32 ושינויים בפורמט ה-header כך שיהיה יותר מתאים לסוג התקשורת שיש כיום.

בלתי אפשרי להשיג הסכמה בין כל הגורמים הנדרשים לשינוי הזה, והוא לא יכול לקרות באופן פתאומי, אלא בהכרח באופן אבולוציוני - כל NIC חדש שמיוצר תומך ב-IPv6, אבל אף אחד לא יעז לעבור ברגע אחד לתקשורת עם העולם החיצון עם IPv6 כי הפקטות שלו פשוט יפלו כי לא כולם תומכים. מי שכן משתמשים בפרוטוקול כרגע הם ארגונים שיכולים לשלוט במה שקורה בתוכם - ISP-ים, ארגונים כמו גוגל וכו'.

למה כל האינטרנט לא יכול להיות רשת IP אחת גדולה? למה היינו צריכים STP ולחבר ברודקאסטים באמצעות סוויצ'ים ולא פשוט לתקשר בשכבה 3? אין סיבה טובה! זה היה אולי אפילו עובד יותר טוב, אבל כבר מאוחר מדי - הרשת התפתחה בדרך כלשהי ואכאמור אי אפשר לעשות שינוי מיידי, בטח לא בסדר גודל כזה.

כשמאפיינים רשת, יש שני דברים שצריך להגדיר - הטופולוגיה של הרשת, כלומר איך נראה הגרף שמחבר בין יחידות הקצה, ואלג' הניתוב - איך מעבירים בין יחידות קצה את הפקטות.

טופולוגיה של רשת

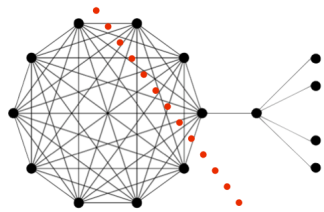
לכל רשת יש כמה מאפיינים חשובים ברמת הטופולוגיה:

1. קוטר - המרחק הקצר ביותר הארוך ביותר בין שני קודקודים (לכל שני קודקודים יש מרחק קצר ביותר, אז הזוג הכי רחוק במטריקה הזו).
2. דרגת קודקוד - כמה חיבורים (פורטים) יש לכל קודקוד.
3. Bisection Bandwith - רוחב הפס הנמוך ביותר שקיים כשחצי כלשהו של הרשת מנסה לשלוח מידע לחצי שני של רשת (הכללה של Min Cut - Max Flow).

רשת "נוחה" היא רשת עם קוטר קטן (כולם קרובים אחד לשני), דרגה נמוכה (יקר לייצר מכשירים עם הרבה), ו-Bisection Bandwith גבוה (מעבר תקשורת אופטימלי), אבל כמובן שזה טריידוף כי השיקולים האלה מתנגשים אחד בשני.

בהינתן רשת, יש לנו $\frac{1}{2}\binom{n}{\frac{n}{2}}$ אפשרויות לחלוקה לשתי קבוצות שוות ($\frac{1}{2}$ בגלל המקרה הסימטרי), כך שחישוב ה-Bisection Bandwith הוא לא יעיל.

דוגמה האם הרשת הבאה היא אופטימלית? כמובן שלא, כי יש צוואר בקבוק חמור בין ארבעת הקודקודים מימין, לבין שאר הגרף המחובר. עם זאת, אף חלוקה לשני חצאים של הרשת לצורכי חישוב Biscetion Bandwitch לא תתפוס את זה. לעומת זאת, חלוקה לשליש-שני שליש כן תאפשר את האבחנה הזו.



איור 14 : רשת מאוד לא אופטימלית

מסקנה Bisection Bandwith היא מטריקה לא מספיק מבחינה!

הגדרה ה-Edge Expansion של גרף הוא

$$\min_{S\subseteq V, 0|S|\leq \frac{n}{2}} \frac{|\{e : V\backslash S\text{-ל } S \text{ צלע בין } e\}|}{|S|}$$

כאשר הנרמול מאפשר לנו לצפות מחלוקות עם צדדים מאוזנים ליותר צלעות מאשר חלוקות מוטות מאוד.

הערה מיקסום של Edge Expansion הוא המטרה הסופית בבניית רשת טובה, אבל לא תמיד אפשר לעשות את זה.

סוגי בניות

- בנייה לא מפורשת - הגדרה של גרף באמצעות קודקודים וצלעות, כשלשמות האיברים בקבוצות האלה אין משמעות אינהרנטית (אפשר לשנות אותם בה"כ) ואין תבנית בהכרח לפיה הקודקודים מחוברים.
- בנייה מפורשת - קודקודים מחוברים עם תבנית/חוקיות כלשהי, כך שיש לגרף מבנה כלשהו. שמות של קודקודים אומרים הרבה מאוד עליהם ואפשר רק באמצעותם לבנות את כל הגרף (ולכן אם משנים אותם הגרף משתנה).

דוגמה נביט בגרף הבא,

איור 15 : גרף מערך לינארי

הקוטר של הגרף הוא 3, הדרגה היא 2 וה-Bisection Bandwith מתקבל בחתך באמצע. ניתן היה להגדיר את הגרף מפורשות על ידי מתן שמות לקודקודים $1, \ldots, N$ והגדרת צלע בין כל קודקוד i ל-$i+1$.

דוגמה היפר קוביה n-ממדית היא גרף עם 2^n קודקודים ממסופרים, עם צלע בין כל שני קודקודים שנבדלים בייצוג הבינארי שלהם רק באחד (בין 010 ל-000 יהיה צלע).

הקוטר של הגרף הוא n, כי כדי לעבור בין כל שני קודקודים נצטרך לכל היותר להפוך את כל הביטים של הייצוג, ויש n כאלה. הדרגה היא גם n כי כל ביט שמשתנה זה קודקוד שאנחנו מחוברים אליו.

מה החתך הכי גרוע שיש? נביט בחתך שבצד אחד יש לו את כל הקודקודים עם 0 בקוורדינטה הראשונה, ובצד השני 1 בקוור' הראשונה. כדי שתהיה צלע בין שני קודקודים בצדי החלוקה, הם צריכים להיות זהים בכל שאר הקוור' שלהם, כי אנחנו בר יודעים שהקוור' הראשונה שלהם שונה. לכן יש 2^{n-1} צלעות ביניהם, שזה בפרט המקרה הכי גרוע שיש (כל מקרה אחר יאפשר לקודקודים עם תנאים יותר נוחים להיות מחוברים).

אלג' ניתוב פשוט בין כל שני קודוקדים בקוביה הוא החלפה בכל פעם של הביט הראשון ששונה בין הכתובת של התחנה לכתובת היעד (בהכרח יש צלע כי יש שינוי של בדיוק ביט אחד).

דוגמה בעץ יש דרגה קבועה. הקוטר שלו הוא $\log N$ (מספר הקודקודים), וה-Bisection Bandwith הוא $\mathcal{O}(1)$ כי אם נחלק את הקודקודים לשתי קבוצות, תהיה בדיוק צלע אחת שתחבר ביניהם כי אחרת היה מעגל ואז זה לא היה עץ.

אלג' ניתוב קלאסי הוא לעלות למעלה (מהעלים הרחוקים מהשורש לשורש) עד שמגיעים לאב קדמון משותף (במסלול אל השורש), ואז לרדת למטה עד ליעד.

למה כל זה מעניין אותנו? אנחנו מנסים בהינתן תתי-רשתות IP בתוך ארגון (כלומר רשת overlay בשכבה 3 מעל רשתות בשכבה 2), לרשת אותן ביחד. כלומר לעשות ניתוב אינטרה (בתוך) דומיין, באמצעות ראוטרים שמחוברים לתתי-הרשתות שלהם.

הערה אלג' ניתוב מחליט על מסלולים להעברת פקטות בין יחידות ברשת, אבל ברמה הפרקטית אלג' הניתוב נותן לנו מימוש/חוקים לפיהם ניצור טבלת forwarding שבה אנחנו (ובתקווה גם אחרים) יתנהגו, כדי לנתב בהצלחה פקטות.

ברוב הארגונים מחשבים מסלולים קצרים ביותר באמצעות משקולות סטטיות שהוגדרו מבחוץ בין תתי הרשתות. בגלל שראוטרים הם מבוזרים ואין אף אחד מרכזי שמנהל את מעבר הפקטות, אנחנו צריכים לדאוג שכל ראוטר יודע מה ה-next hop של כל סאבנט של IP.

הערה נבחין שוב בין ניתוב שהיה אפשרי במקרה הריכוזי/ניתוח תאורטי, לבין forwarding שהוא הדרך להעביר פקטות כשמדובר בשת"פ מבוזר.

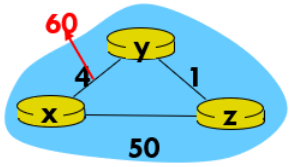
סכמות ניתוב

- Distance Vector Routing - תהליך איטרטיבי שבו מחשבים עם בלמן-פורד מבוזר את המרחק הקצר ביותר בין כל שתי יחידות קצה (בניגוד ל-STP שהיה ביחס לשורש בלבד).

  אם $d_x(y)$ הוא המרחק הקצר ביותר בין x ל-y, ומוגדר איטרטיבית ע"י $d_x(y) = \min_v \{w(x,v) + d_x(v)\}$.

  האלג' בעייתי במקרה שמשנים משקלים ברשת. נניח שבאופן דינמי מנהל הרשת שינה את המרחק בין שני ראוטרים. הקודקוד יזהה את השינוי, יעדכן את המידע אצלו, יחשב את ה-DV מחדש ויפעפע את השינוי לשכנים.

  דוגמה נניח שנתונה לנו הרשת הבאה, שבה שונה המשקל מ-4 ל-60.



איור 16 : מקרה קלאסי של Count-to-Infinity

y ישנה את המסלול שלו ל-x ללעבור דרך z, כי z טוען שיש לו מסלול ל-x באורך 5 (שזה שקר, אבל הוא לא יודע את זה). לאחר שעידכן אצלו את ה-DV, y יספר ל-z שיש לו מסלול באורך 6 ל-x. z מקשיב לו, מעדכן את המידע אצלו, ומודיע ל-y שיש לו מסלול במרחק 7 ל-x. ככה זה ימשיך במשך 44 איטרציות לפני שזה יתייצב ו-z יעבור לצלע עם 50, וזה לא תקין. בין היתר כי פקטות ישלחו בלופ אינסופי ביניהם ויפלו.

אפשר לפתור את הבעיה אם לא נעדכן את מי שעכשיו עידכן אותנו במצב, מה שנקרא Poisoned Reverse. עם זאת זה לא פותר את הבעיה הכללית אלא הופך את מקרי הקצה שבהם זה לא עובד ליותר מורכבים אך קיימים.

LS נמנע מהבעיה הזו כי קודקוד מחויב לספר לכל השאר על כל שינוי וכולם מעדכנים אצלם באופן עצמאי את ה-forwarding table.

- Link-State Routing - כל ראוטר מציף את השכנים שלו במידע כדי ללמוד את הטופולוגיה (כולה, לא רק מרחקים קצרים). לאחר איסוף המידע, כל נתב מריץ דייקסטרה כדי לחשב את המסלולים הקצרים ביותר שלו מאחרים.

הערה המקרה של Count-to-Infinity זו דוגמה טובה לעיקרון Good News Travel Fast, Bad News Travel Slow - כשהכל בסדר האלג' עובד טוב אבל אם יש בעיות המצב יתייצב רק לאחר זמן רב.

אפיון אלג' ניתוב

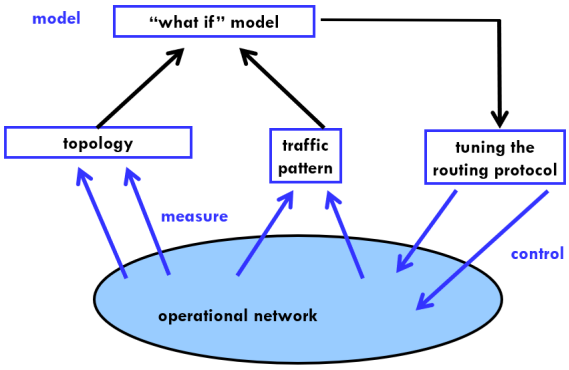
1. האם הניתוב מבוסס על מידע גלובלי או חלקי? Distance Vector מבוסס על מידע חלקי (אני מקשיב רק לשכנים), ואילו Link-State נותן לכל נתב את כל המידע, ושהוא יעשה איתו מה שהוא רוצה (בתקווה דברים טובים).
2. מורכבות התקשורת - כל שתי רשתות מספרות אחת לשנייה את המרחקים ביניהם ב-LS, במקום רק מספר לינארי במספר הקודקודים ב-DV.
3. רובוסטיות לנפילה של ראוטר ועוד.

דוגמה ברשת ARPAnet של צבא ארה"ב עדכנו בזמן אמת את המשקולות של חיבורים באמצעות טרנספורמציה לינארית על גודל התור שהמתין לעבור בצלע. השינויים התכופים וההריאקטייבים האלה גרמו לתגובותיות-יתר לעומס ומשם להתבדרות ואוסיצליות בתעבורה, כמו גם העמסה על מסלולים לא יעילים מבחינת משאבים.

עם השנים נוספו פלסטרים על גבי פלסטרים כדי לגרום לרשת להתנהל כמו שצריך, כשבסוף הוחלט לעבור למשקלים סטטיים שמשתנים ע"י בני אדם.

מודל השכבות הוא שפותר לנו (בעקיפין) את הבעיה - אם שכבה 3 ויעול המשקולות הסטטיים הם תחת Traffic Engineering שמאפשר ניתוב יעיל עד נקודה מסוימת, שכבה 4 ופרוטוקולים שלא יודעים על עומס באמת אבל דואגים למתן את העומס שלנו עצמינו הם אלה שמבצעים Congestion Control.

ניהול רשת מוצלח הוא תהליך איטרטיבי שלא נפסק שכולל הסתכלות על על המצב הנתון, קביעת פרמטרים טובים יותר בהתאם למידע הנתון לנן, צפייה מהצד על ההשפעה של השינוי וחוזר חלילה (ראו איור)



איור 17 : שיטת Measure, Model, and Control

תרגול

הערה חשוב לשים לב שאפע"פ שפרוטוקול כלשהו עוזר לנו לניהול שכבה כלשהי, הוא לא בהכרח קורה בשכבה הזו, לדוגמה DNS עוזר לשכבה 3 אבל קיים בשכבת האפליקציה.

אנחנו עוסקים עכשיו באיך לנתב באופן אופטימלי פקטות בטופולוגיית ראוטרים שמחוברים בחיבורים ממשוקלים ידנית ע"י מנהל הרשת.

הערה הפלט של אלג' הניתוב הוא טבלת ניתוב (עם רשומה פר-ראוטר) שבה ניוועץ לאיזה **שכן** נעביר את הפקטה, בכל פעם שנתבקש להעביר פקטה לראוטר כלשהו (שהוא לא בהכרח שכן).

Distance Vector

אלג' ניתוב דינמי, שממומש כיום ב-RIP (מריץ את האלג' כל חצי דקה). נריץ בלמן-פורד מבוזר שהתוצאה שלו תהיה וקטור (מתעדכן עד שמתייצבים) של מרחקים שלו מכל קודקוד אחר, ע"י $D_s(y) = \min_{v \in \Gamma} \{c(s,v) + D_v(y)\}$ (המסלול הקצר ביותר דרך שכן כלשהו).

במהלך ריצת האלג' נשמור טבלה עם שורה לוקטור של כל אחד מהשכנים שלנו (ואחת שלנו), שהוא עצמו מכיל עמודה לכל קודקוד אחר בגרף, אילוסטרטיבית יראה כך :

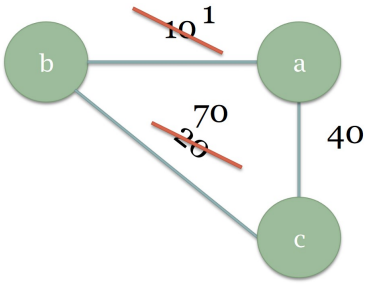
	מרחק קוד' 2 ברשת	מרחק מקוד' 1 ברשת	
			הוקטור שלי
			הוקטור של שכן 1
			הוקטור של שכן 2
			...

את הטבלה נאתחל עם ∞ בכל הערכים, חוץ מבשורת הוקטור שלנו שם ידוע לנו המרחק שלנו מהשכנים שלנו.

איטרציה באלג' מכילה שני שלבים: הפצת הוקטור שלנו ועדכון הוקטור שלנו באמצעות וקטורים של אחרים. בכל עדכון שנקבל של וקטור שכן כלשהו, נעדכן בשורה ראשונה (הוקטור שלנו), אם צריך, את הערך בעמודה כך שיתאים למציאות החדשה (אם המרחק המינימלי ירד או עלה).

נפסיק את התהליך כשנתייצב, ומשם נייצא טבלת ניתוב שמתאימה לכל קדוקוד ברשת את השכן שלנו שדרכו נשלח פקטה לאותו הקודקוד, ואת המרחק עד לקודקוד (המחיר) - הלא זו טבלת הניתוב.

דוגמה נציג בעיה ב-DV בשינוי לרעה של משקלים. נביט בשינוי הבא במשקלים



איור 18: דוגמה לכשל ב-DV בשינוי משקלים לרעה

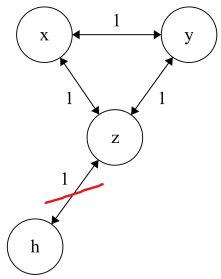
נניח שכבר קרה השינוי מ-10 ל-1 וכולם התייצבו, ועכשיו שינו את $b-c$ ל-70.

באיטרציה הראשונה, c ו-b ישימו לב שיש שינוי במשקולות שלהם ויעדכנו את המסלולים שלהם: c יעדכן את המסלול ל-a שיהיה ישיר (מחיר 40), ו-b יעדכן את המסלול ל-c שיעבור דרך a (מחיר 22, כי עושים b,a,b,c - שזה לא קורה במציאות). כאמור המסלול b,a,b,c הוא לא מסלול אמיתי ולכן יש לנו חישוב לא נכון. בפעם הבאה יעלה ל-a להגיע ל-c 23 (1 ועוד b ל-c). b יעדכן שלוקח לו 24. התלות הזו בין a ל-b תמשיך עד שנגיע ל-40, ואז a יעדיף לעבור דרך c ונגמרת התלות. התופעה הזו נקראת Count to Infinity.

מקור הבעיה הנ"ל הוא שכל אחד מצביע לשני ולא מודעים שהשכן עובר דרך עצמו. לכן אפשר להגיד לשכנים אם אנחנו בונים עליהם במסלולים שלנו ואז זה יפתור את התלות הזו שנקראת Poisoned Reverse. טכנית, כשנפיץ את הוקטור שלנו לכל שכן y, אם המחיר שחושב לקודקוד כלשהו ברשת z עובר דרך y (בקפיצה הקרובה ברשת), אז נשים בעמודה של z בוקטור שנשלח ∞ (כך y לא יעבור דרכנו בשליחה ל-z). בשאר הערכים לא ניגע.

הפאץ' הזה פותר את הדוגמה הפשוטה הנ"ל עם שלושה קודקודים, אבל זה עדיין יכול לא להיות יציב אף פעם.

דוגמה נביט בגרף בטופולוגיה הבאה, כאשר נניח שהרצנו את האלג' על הרשת המקורית, התייצבנו, ואז הוסר החיבור בין z ל-h



איור 19: דוגמה לכשל ב-DV גם מניעת Poisoned Reverse

- באיטרציה הראשונה, z יודיע ל-x ו-y שהוא לא יכול להגיע ל-h. x ו-y יעדכנו אצלהם את הנתונים על בסיס המידע שיש להם כרגע בלי תקשורת ביניהם, כלומר ש-x יעדכן שיש לו מסלול ל-h דרך y באורך 3 (כי עוד לא סיפר לו y שאין לו מסלול) וכך יעשה גם y.
- באיטרציה הבאה x יספר ל-z שיש לו מסלול באורך 3 (לא דרכו בקפיצה הראשונה) ולכן z יעדכן שיש לו מסלול באורך 4.
- באיטרציה הבאה y מעדכן אצלו שיש לו מסלול באורך 5 דרך y (שהוא לא יודע שעובר דרכו).

כל זה ימשיך עד אינסוף ואף פעם לא יתייצב!

Link-State

אלג' גלובאלי, ממומש כיום ב-OSPF (מריץ את האלג' כל חצי שעה). כל ראוטר יאסוף את המרחק של כל ראוטר מכל ראוטר אחר ברשת ויריץ דייקסטרה בעצמנו וכך נקבע את טבלת הניתוב.

בכל איטרציה, לא נריץ את דייקסטרה כולו שוב, אלא נשתמש בטריק הבא: נשמור אוסף של קודקודים שעברנו דרכם שאנחנו בטוחים במסלולים הקצרים ביותר שלנו אליהם, N'. בכל איטרציה נמצא את הקודקוד עם המרחק המינימלי (שידוע לנו כרגע באמצעות N' ושכניהם) שלא ב-N', נוסיף אותו ל-N', ונעדכן את המרחקים של כל השכנים של הקודקוד הזה שלא ב-N' עדיין. כך באינווריאנטה אנחנו מיוצבים כבר על כל מה שב-N', ואחר מספיק איטרציות כל הקודקודים יהיו ב-N' ונסיים את הריצה.

שבוע $\mathbb{XI}$ | ניהול תעבורה ו-ECMP

הרצאה

כדי לקבוע ניתוב מוצלח, צריך מישקול מוצלח של החיבורים בין ראוטרים. נאפטם את המשקולות בשלושה שלבים: מדידת הזרימה במציאות, בדיקת שינויים שונים, השמה השינויים.

התאוריה שלנו לבעיה הזו היא תורת מקסום זרימה. זו לא בעיית Max-Flow קלאסית כי אין לנו רק שני קודקודים שמדברים אלא כל שני קודקודים, וכי הפתרון של Max-Flow לא בהכרח ניתן למידול כמסלולים קצרים ביותר (כמו שעושים DV, LS) אלא כמסלולים יותר מורכבים שאי אפשר להשתמש בהם במציאות.

הגדרה בעיית Max Flow מקבלת רשת $G = \langle V, E, c \rangle$ ופולטת f זרימה עם $|f| = \sum_v f(s, v)$ מקסימלי.

הערה מלבד הבעיה שהמסלולים קשים למידול ושיש לנו הרבה קודקודים ולא רק שניים, Max Flow יכול להרעיב קודקודים מסוימים כדי להשיג זרימה יותר טובה, שזה מאוד גרוע.

הגדרה בעיית Max Multicommodity Flow מקבלת רשת $G = \langle V, E, c \rangle$ ומטריצת ביקוש $D \in \mathbb{R}_+^{|V| \times |V|}$ ופולטת f זרימה כך ש-$\sum_v |f_v|$ מקסימלי (f_v הזרימה מקודקוד v), כשאנחנו לא עולים על הביקוש.

הערה הרעיון הוא שאין סיבה להעביר יותר זרימה מהביקוש, כי נוכל ליפול לזרימות מאוד גבוהות שמספקות ביקוש מאוד גבוה בצד אחד ונמוך במקום אחר וזה לא טוב - אין סיבה להתאמץ מעבר לביקוש שיש.

הערה בניגוד ל-Max-Flow = Min-Cut, אין בעיה דואלית שימושית ל-Max Multicommodity Flow ולכן קשה לעבוד איתה בהנדסת תעבורה.

הגדרה בעיית Minimize Congestion מקבלת $G = \langle V, E, c \rangle$ וממזערת את העומס על הצלע הכי עמוסה, כלומר למזער $\max_e \frac{f(e)}{c(e)}$ כשמספקים את כל הביקוש, אולי עם חריגה מעל לקיבולת.

הערה הבעיה הזו שימושית וקיבלה הרבה התייחסות אפע"פ שלכאורה אין לנו באמת את הקיבולת הנדרשת, כי אנחנו מניחים כאן שמישהו תכנן לפניכן את הרשת כך שנוכל לעמוד בקיבולת בצורה כזו או אחרת.

יש עוד הרבה פ' מטרה שונות שאפשר למקסם, כמו הוגנות (נמקסם α שעבורו כל אחד שלח לפחות α מההודעות שלו בהצלחה תוך . . . זמן).

הערה בעוד Multicommodity flow מצליח לאפטם (גם אם לא פרקטית) מסלולים בין מקור ויעד ואיך עומס מתפלג על פני מסלולים, ניתוב IP באמצעות האלג' שראינו מנסה לעשות את שני הנ"ל, רק בסיבוכיות משמעותית יותר נמוכה (אפילו שכל הבעיות שעכשיו הזכרנו הן ב-LP).

כל הבעיות האלה כאמור בתאוריה הן נחמדות, אבל הדבר הפרקטי היחיד שמנהל רשת יכול לעשות הוא לקבוע משקולות סטטיים ולתת ל-OPSF ואח' לנתב את התעבורה בהתאם למשקולות הללו.

הערה מזעור עומס הוא חשוב גם מבחינת עמידות, לדוגמה מפני כשל בחיבור אחד קריטי.

Equal-Cost Multipath

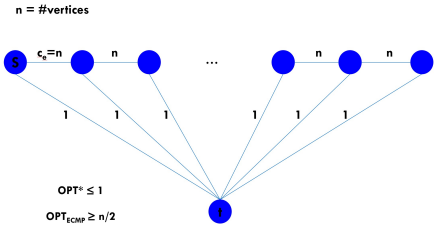
ECMP הוא אלג' שמחליט בהינתן משקולות הרשת לאיזה ראוטר להעביר הודעה. האלג' רץ בראוטר i ומחשב מרחקים קצרים ביותר לכל ראוטר j, כך שבסוף הוא חישב את nxt_{ij} לכל j שהוא אוסף השכנים של i שהם השלב הבא במסלול קצר ביותר ל-j.

במהלך forwarding של פקטות ל-j, הוא יפזר את הפקטות באופן אחיד על פני כל השכנים ב-nxt_{ij} (כל פעם יבחר שכן אחר כך שהעומס יהיה שווה).

הגדרה בעיית האופטימיזציה של קביעת המשקולות הסטטיים מקבלת גרף עם קיבולות $G = \langle V, E, c \rangle$ ומטריצת ביקוש $D \in \mathbb{R}_+^{|V| \times |V|}$, ומוציאה משקולות לכל חיבור כך שזרימה תחת שימוש ב-ECMP תהיה אופטימלית (תחת פ' מטרה כלשהי).

האם אפשר להשיג פתרון אופטימלי לכל פ' מטרה עם ECMP?

דוגמה נבדוק האם אפשר לקבוע משקולות כך ש-ECMP ימזער עומס ($\frac{f(e)}{c(e)}$) על הצלע העמוסה ביותר. נביט בדוגמה הבאה, בה יש לנו רק מקור אחד (s) ויעד אחד (t) במטריצת הביקוש



איור 20: רשת בצורת מניפה שמכשיל את ECMP

הערך האופטימלי שאפשר לקבל על עומס הצלע העמוסה ביותר הוא פחות מ-1 כי אפשר להזרים מ-s 1 דרך כל צלע עם קיבולת 1, ואת שאר התעבורה להעביר הלאה בקודקודים בשורה למעלה, כך שבכל פעם הזרימה שעוברת לקודקוד הבא תפחת באחד. העומס המקסימלי כאן הוא 1 והוא מתקבל על הצלעות ל-t.

מה ECMP יעשה? נראה שלא משנה אילו משקלים נקבע, ECMP ינתב באופן שמשיג עומס מירבי של לפחות $\frac{n}{2}$ (מאוד רע, משמעותית גדול מ-1). נניח שקבענו משקולות. לכן כשתעבורה יוצאת מ-s, או שהיא תתחלק חצי-חצי על הצלע עם $c = 1$ או שהכל יעבור ימינה. אם האפשרות הראשונה - הרי לנו עומס מרבי של לפחות $\frac{n}{2}$. אם השנייה, נמשיך הלאה - בצומת הבאה או שנתפצל או שנלך עם הכל ימינה. זה במקרה הגרוע ימשיך עד לקודקוד האחרון ואז כל התעבורה תעבור בצלע האחרונה עם קיבולת 1, כך שלא משנה מה יש לנו לפחות צלע אחת בקיבולת 1 עם עומס $\frac{n}{2}$.

הערה הדוגמה הנ"ל מראה שאנחנו לא יכולים להשיג ביצועים טובים בגלל מגבלת סיבוכיות, אלא בגלל מגבלת אקספרסיביות (בדומה לפער בין מכונת טיורינג לאוטומט).

הגדרה בעיית Link-Weight מקבלת כקלט גרף עם קיבולות $G = \langle V, E, c \rangle$ ומטריצת ביקוש, ומחזירה משקולות לחיבורים שהם אופטימליים מבחינת היותם קרובים ביותר לפתרון האופטימלי אם לא היינו כפופים ל-ECMP.

הערה בעיית ה-Link-Weight היא באמת מה שאנחנו רוצים לפתור, רק שהיא NP-קשה, ואפשר גם להוכיח שהיא NP-קשה אם יש רק מקור ויעד אחד, ואם מנסים לקרב מינימום עומס מירבי עם כל קבוע שהוא.

עם זאת הקושי של הבעיה בתאוריה מגיע מכל מיני דוגמאות פתולוגיות, אפע"פ שבמקרה הממוצע אפשר לקרב לא רע בכלל עם ECMP.

הערה לסיכום כל הקונספטים יחד: מנהל הרשת קובע משקולות כך שכשכל ראוטר יריץ DV או LS ואז כשיקבל פקטות ויריץ ECMP בהינתן הנתונים שיש לו, נקבל זרימה עם עומס מירבי מינימלי על כל צלע שהיא ברשת.

ECMP מפצל באופן אחיד על פני ה-next-hops שלו את התעבורה, אבל במציאות התעבורה היא פקטות. פקטות מאותו המקור חשוב שיעברו את אותו המסלול כי לדוגמה בשיחת וידאו, אם פקטות סובלות מפערי שידור שונים, זה פוגע משמעותית ב-QoS, לכן חשוב שיעברו את אותו המסלול.

לשם פיזור התעבורה באופן הזה, נשתמש בפ' האש על ה-header של פקטה (לפחות על השדות הרלוונטיים ליעד) ובהתאם לערך שיצא (מודולו מספר החיבורים ב-nxt_{ij}), נשלח ל-next-hop הרלוונטי את ההודעה. הרעיון כאן הוא שכל הפקטות של שיחת זום אחת יהיו בעלות אותם שדות ב-header ולכן ינותבו לאותו המקום.

דוגמה אם יש לנו מעט זרימות קטנות והרבה זרימות גדולות, יכו ללקרות מצב שההאש ימפה לנו שתי זרימות גדולות לאותו החיבור ואז יהיה עומס לא מאוזן, ועל כל זה יש מחקר רלוונטי עד היום.

Transport Layer

נעלה עוד שלב בהיררכיה - נניח שיש לנו דרך לתקשר בתוך ארגון עם ניתוב (בתקווה) אופטימלי. שכבה 4 ממומשת אצל משתמשי הקצה. המטרה של שכבה 4 היא להתאים תהליכים במחשב המקור לתהליכים במחשב היעד, בהנחה שיש לנו חיבור ישיר בין שני משתמשי קצה (או אבסטרקציה של חיבור כזה, כשכבה 3 היא שמאפשרת את האבסטרקציה הזו באמצעות IP ו-ECMP וכיוצ"ב).

שכבה 4 יכולה לשרת כמה מטרות שונות:

- TCP - שליחה אמינה ועם סדר של פקטות, מתבטאת בניהול עומס התקשורת, ניהול זרימה ויצירת הקשר.
- UDP - תקשורת לא אמינה בלי סדר - המינימום שצריך מעל IP כדי שאולי יגיעו הפקטות (יותר מהיר ככלל).
- ...

אף אחד מהפרוטוקולים האלה לא יכולים להבטיח לנו דיליי מקסימלי כלשהו, או רוחב פס כלשהו, כי זה לא תלוי בכלל במשתמשי הקצה.

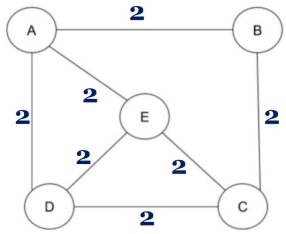
תרגול

ניהול תעבורה חולק היסטורית לשני חלקים שונים - מוצאים דרך קצרה ביותר באמצעות DV, LS בהינתן משקולות w, ואז מוצאים זרימה אופטי' בהינתן המסלולים. מהי זרימה אופטי'?

- MinCong - מזעור הצלע העמוסה ביותר, כלומר מציאת $\underset{f}{\text{argmin}} \max_{e} \frac{f(e)}{c(e)}$, בהינתן שקיימנו את כל דרישות הביקוש על חשבון חריגה מקיבול.
- MaxMCF - מקסום הזרימה הכוללת, כלומר $\underset{f}{\text{argmax}} \sum_{v} |f_v|$ כאשר $\sum_{u} f(v,u)$ הזרימה לכל השכנים מ-v, כשלא בהכרח משיגים את הביקוש הנדרש אבל לא חורגים מהקיבול.

הערה שתי הבעיות הנ"ל הן בעיות תכנון לינארי, כלומר $\underset{x}{\text{argmin}}\ c \cdot x$ בהינתן $Ax \leq b$ כאשר b, c וקטורים, A מטריצה. c יהיה $1 \ldots 1$ ותהיה לנו קוור' ב-x צלע והאילוצים יהיו כמו של בעיות הזרימה במקור - $f_e \geq 0$ ו-$\sum f_{e,in} = \sum f_{e,out}$ לכל צלע e. נשים לב שאנחנו לא אוכפים את הקיבול כי אנחנו מניחים שהקיבולים שנתונים לנו הם מספיקים.

שאלה נניח שיש לנו את הקומודיטים הבאים, מסודרים כטאפלים של מקור, יעד וביקוש: $(A, C, 5), (C, E, 3)$ ואת הטופולוגיה הבאה.

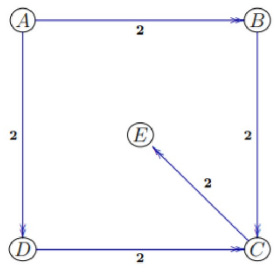


איור 21: טופולוגיה עם קיבולים מסומנים

- מה הפתרון האופטימלי ל-Max-MCF?

נצטרך למצוא את זרימה שמקיימת את הביקוש, כלומר סה"כ זרימה בגודל 8 (5 מתוך A ו-3 מתוך C). אם מתעלמים מהקיבולים, אפשר להשיג הרבה זרימות שמספקות את הביקוש. נוכיח עתה כי אי אפשר לספק את הביקוש עם תנאי הקיבול.

ראשית נשים לב כי הזרימה הבאה היא חוקית, ונוכיח שהיא אופטימלית



איור 22: זרימה אופטימלית תחת אילוצי הקיבול

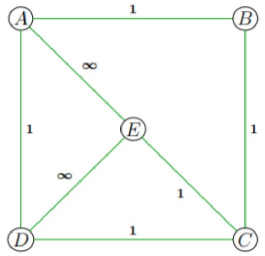
נסתכל על חתך מוכלל, שהוא קבוצה של קודקודים, כך שלכל קומודיטי יש נציג אחד בלבד (כלומר לא גם המקור וגם היעד) בקבוצה.

$\{C, E\}$ הוא לא חתך מוכלל כי לקומודיטי השני יש שני נציגים. $\{B, C\}$ הוא חתך מוכלל, והזרימה שלו, כלומר הזרימה דרך כל הצלעות בין קודוקדיו לאלו שמחוצה לו, היא 6. גם $\{C\}$ הוא חתך מוכלל חוקי, והזרימה דרכו שווה לקיבול דרכו, מה שאומר שהזרימה אופטימלית (כן יש דואליות בין חתך מוכלל ל-MCF). לכן לא תיתכן זרימה יותר טוב מהזרימה הנ"ל כי אז היינו חורגים מהקיבול. בתרגיל עדיף קודם למצוא חתך עם קיבול מינימלי ואז לנסות למצוא זרימה שממלאת את הקיבול הזה (כאן עשו הפוך).

השאלה הזו לא קשורה בכלל לאיך שהאלג' שלמדנו עובדים כי היא עוסקת בתאוריה ובמציאות יש משקולות ועושים ECMP וכו'.

- האם אפשר למשקל בעזרת OSPF (DV או LS לצורך העניין) את הרשת כך ש-ECMP יניב זרימה אופטימלית? אם כן הראו אותה ואם לא הסבירו מדוע.

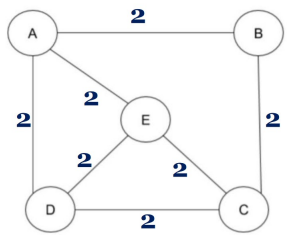
נביט בזרימה הבאה, שאינטואטיבית מפזרת את $A \to C$ על פני הרשת ו-$C \to E$ באופן ישיר



איור 23 : משקול הרשת לצורך פיזור ECMP מוצלח

נקבל במקרה הזה שני מסלולים קצרים ביותר מ-A ל-C ואחד מ-C ל-E, כך שבסופו של דבר נקבל את אותה הזרימה שהראנו בסעיף הקודם כלומר אכן השגנו זרימה אופטימלית.

- נניח עתה שהקומודיטים שלנו הם $(B, D, 5)$, $(C, E, 3)$. מה הפתרון האופטימלי לבעיית MinCong?



איור 24 : אותה הטופולוגיה שוב לנוחות הקריאה

נחפש את החתך עם הקיבול הקטן ביותר שוב, תחת הקומודיטים החדשים. הפעם זה $\{B, C\}$ (הסטודנטית המשקיעה תבדוק שזה אכן כך). גודל הקיבול הוא 6. נחפש זרימה עם גודל 6. מהגדרת החתך, נהיה חייבים לעבור דרך הצלעות שבחתך, והעומס המינימלי יתקבל אם נפזר באופן אחיד ככל הניתן את העומס על פני הצלעות (בפרט כי הקיבול של כולן זהה). לכן הפתרון האופטימלי נותן לנו עומס מרבי של $\frac{8}{6}$ (מפזרים אחיד זרימה כוללת של 8 על פני קיבול כולל של 6), כלומר שהצלעות $\{A, B\}$, $\{E, C\}$, $\{D, C\}$ יהיו שלושתן בעומס $\frac{4}{3}$ (בפרט עם זרימה דרכן של $\frac{8}{3} = 2 \cdot \frac{4}{3}$).

- האם יש משקול כך ש-ECMP יניב פתרון אופטימלי?

  לא! נסתכל על התעבורה שיוצאת מ-B.

  - אם הוא מזרים את כל 5 היחידות שלו דרך צלע אחת ב-ECMP קיבלנו עומס $\frac{5}{2} > \frac{4}{3}$ שזה לא אופטימלי.
  - אם הוא מזרים את 5 היחידות בשתי צלעות, סה"כ הזרימה שיוצאת מ-C תהיה 2.5 שהוא מקבל מ-5 ועוד 3 שיוצא מעצמו, כלומר סה"כ 5.5, שזה נותן עומס $\frac{5.5}{4} > \frac{4}{3}$ שזה גם לא אופטימלי.

שבוע $\mathbb{XII}$ | שכבה 4

הרצאה

השכבה 4 היא השכבה הראשונה שבה יחידות הקצה הם תהליכים (אפליקציות), שמאפשרים הפרדה לוגית של המידע המתוקשר. שכבה 3 (לכאורה, נראה בהמשך למה לא) לא מועדת להפרדה הזו.

הערה העובדה ששכבה 4 ממומשת בקצוות היא בעייתית במקרה שבו פקטה נופלת מילמטר לפני שהיא מגיעה ביעד. במקרה כזה עדיף שהלינק ישלח הלאה ממש קצת את הפקטה שוב. מצד שני יש אפליקציות שלא צריכות את השירות הזה (פקטה ישנה לא רלוונטית כבר) וזה מוסיף הרבה מורכבות, כך שכבר עדיף להיצמד לעיקרון ה-End-to-End.

שכבה 4 עושה Multiplexing(-De) - השולח עושה דימולטיפלקסינג, כלומר מאחד את כל התקשורות מכל התהליכים שלו לפקטות TCP מעל פקטות IP, כשאנחנו מזהים את האפליקציות עם פורטים, והמקבל צריך להפריד את ההודעות לפי הפורטים המזהים ולהעביר אותם לתהלכיים הרלוונטיים אצלו.

הממשק של שכבה 4 אצל תהליכים הוא סוקטים, ומערכת ההפעלה מסדרת פקטות שהיא מקבלת ומעבירה אותם לפורטים הרלוונטיים.

- ב-UDP מופיע ב-header כתובת IP ופורט רק של היעד, כך שאם שני תהליכים שולחים לאותו היעד דרך סוקטים זהים (מבחינת פורט ו-IP), לא ניתן יהיה להבחין בין השיחות. זה נקרא Connectionless demultiplexing.
- ב-TCP מזהים כל פקטה עם כתובת IP מקור ויעד ופורט מקור ויעד, ולכן סוקטים של תהליכים שונים מגדירים שיחות שונות. זה נקרא Connection-oriented demux, שכן רק באמצעות זיהוי של שני הגורמים בשיחה אפשר להגדיר שיחה (connection).

ניהול שיחות ב-TCP

בגלל שמדובר בשיחה ולא שליחה שרירותית של הודעות, לפני שאפשר לדבר, ב-TCP צריך לבסס את השיחה והזיכרון שלה אצל הלקוח והשרת, באמצעות לחיצת היד המשולשת.

1. הלקוח שולח TCP SYN עם sequence number התחלתי אקראי (לצורכי אבטחה).
2. השרת עונה עם SYNACK עם מספר סידורי התחלתי שלו ומקצה אצלו זיכרון לשיחה.
3. הלקוח עונה עם SYNACK, שכבר יכול להחיל מידע של השיחה.

כשהשיחה נגמרת (האפליקציה סוגרת סוקט), צריך לשחרר את המשאבים שהוקצו.

1. הלקוח שולח TCP FIN לשרת.
2. השרת עונה עם ACK, סוגר את הסוקט אצלו ושולח FIN.

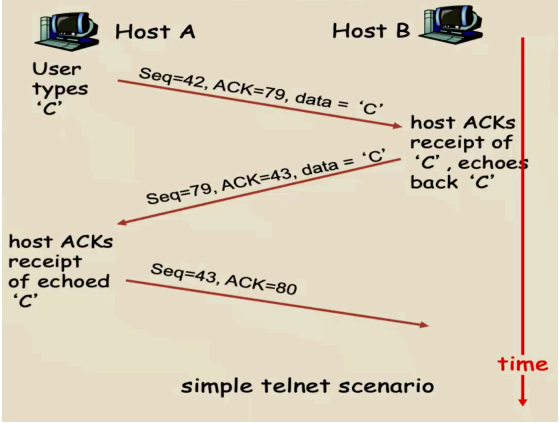
הערה ב-UDP אין שום דבר מהסוג הזה כי אין שיחה ולכן לא צריך לפתוח או לסגור אותה. עם זאת UDP חשוב כאמור בכל אפליקציה שהיא Real Time, וגם במקרים שיש שאילתות קטנות לא צריך את התקורה המוגזמת של TCP (לדוגמה DNS).

יותר חשוב מהנ"ל, חייב להיות פרוטוקול שלא צריך לזהות את עצמנו, כי אז אי אפשר לעשות DHCP לדוגמה. כלומר UDP הוא פרוטוקול בוטסראפינג, ולכן קריטי לאינטרנט!

מלבד התחלת וסיום השיחה הנ"ל, TCP מספק גם סטרים ביטים אמין ולפי הסדר (הוא דואג לסדר מחדש אם צריך), והוא pipelined - כלומר הוא דואג לניהול העומס וגודל הפקטות (הגודל המקסימלי נקרא MSS).

הערה גם כשיש לקוח ושרת ורק הלקוח שולח הודעות, התקשורת עדיין דו-כיוונית (צריך לענות ACK-ים וכו').

דוגמה נניח ששרת מריץ אפליקציה פשוטה שמקבלת קלט ועונה את אותו הפלט. התקשורת תראה כך (נניח שכבר עשינו לחיצת יד משולשת)



איור 25: דוגמה לתקשורת דו כיוונית ב-TCP

נשים לב שבכל שליחת הודעה, אנחנו מודיעים מה המספר הסידורי של הפקטה שאנחנו שולחים (seq), ומה המספר הסידורי האחרון שקיבלנו מהצד השני (שדה ה-ACK). גם אחרי שכל הדאטא נשלח, משתמש A עדיין עונה ל-B שהוא קיבל את ההודעה, גם אם אין לו ערך מוסף בצורת data לשלוח לו.

הערה מעתה נניח שיש לקוח ששולח הודעות בתקשורת חד כיוונית והשרת רק נותן פידבק, אפע"פ שזה לא באמת מה שקורה, כי זה יקל על הניתוח. בנוסף, נניח שהמספר הסידורי הוא מספר הפקטה, אפילו שבפועל הוא מספר הבתים שנשלחו מתוכן ההודעה.

מצבי השולח ב-TCP

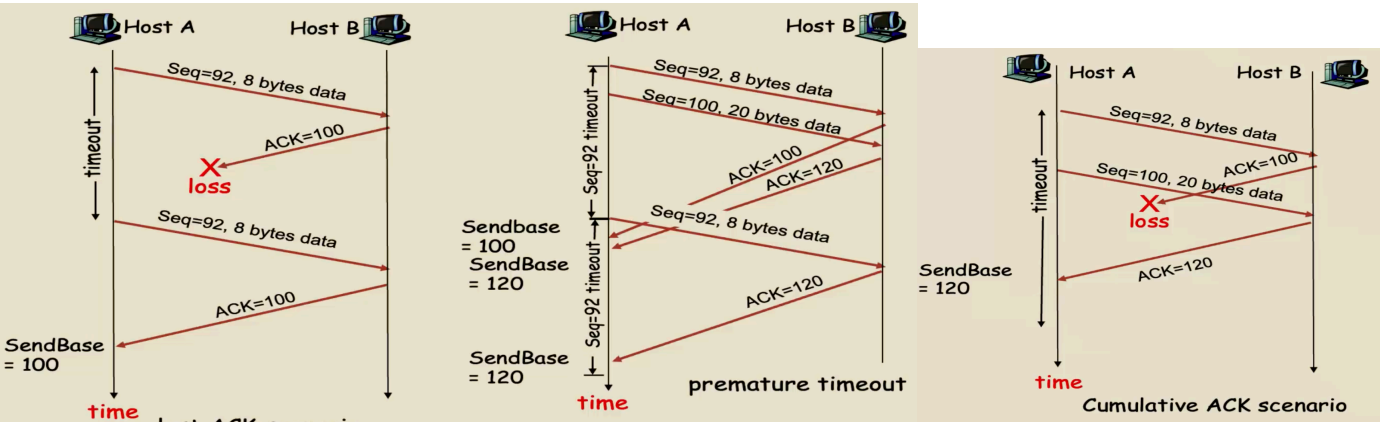
- מידע התקבל מהאפליקציה: צור סגמנט עם מספר סידורי, התחל טיימר להודעה האחרונה שעוד לא קיבלנו עליה ACK אם לא רץ כבר.
- טיים-אוט: אם הטיימר נגמר ולא קיבלנו אישור, שדר מחדש את הסגמנט והתחל מחדש את הטיימר.
- ACK התקבל: אם מתייחס לסגמנטים שלא קיבלו ACK עדיין, עדכן את המצב אצלי, ואם עדיין יש פקטות שלא התקבל ACK עליהם (שהם לא מה שעכשיו אושר), התחל את הטיימר שוב.

הערה בגלל שיש לנו טיימר אחד לכל פקטה, המנגנון האחרון שתואר גורם לכך שנוכל לחכות לפקטה אחת יותר מאינטרוול טיים-אוט אחד (נחכה קצת לפקטה שלפני, ואז עוד אינטרוול טיים-אוט שלם לנוכחי).

דוגמה נביט בשלוש דוגמאות לכשל ברשת. משמאל ACK נפל ואכן לאחר שנגמר הטיים-אוט נשלח שוב את ההודעות וזה יעבוד.

בדוגמה השנייה הטיים-אוט על ACK של 100 (כלומר השולח קיבל את מספר סידורי 92 שהכיל 8 בתים) לא הגיעה בזמן ולכן השולח שלח אותה שוב, אבל כשישלח שוב יקבל ACK מהמקבל של 120, כי המקבל כבר קיבל את מספר סידורי 100 שהוא בוגדל 20 בתים, כלומר טיפלנו כמו שצריך כמה בבעיה הזו.

בדוגמה השלישית ACK ראשון נופל אבל ACK שני מגיע בזמן בגלל שהוא קומולטיבי, הוא מכפה על ה-ACK הקודם שנפל וממשיכים הלאה.



איור 26: דוגמאות לבעיות ברשת והטיפול של TCP

הערה חשוב לזכור שה-ACK הוא קומולוטיבי, וכך מפשט מאוד מה כל צד צריך לזכור.

חישוב הטיים-אוט ב-TCP

איך נקבע את הטיים-אוט? נרצה שזה יהיה לא קצר מדי נגד שידורים מיותרים ולא ארוך מדי נגד תגובותיות איטית. התשובה היא זמן ה-RTT - ה-Round Trip Time. כדי לחשב את הזמן הזה, נחשב כמה זמן לוקחת התקשורת בין כל פקטה ו-ACK שלה. הדגימות האלו יהיו די רועשות, ולכן נרצה לשערך את ה-RTT באופן עדין. החישוב שעושים במציאות הוא

$$\text{EsteimatedRTT} = (1-\alpha)\cdot\text{EstimatedRTT} + \alpha\cdot\text{SampleRTT}$$

כאשר $\alpha = 0.125$ לרוב. ההשפעה של דגימות קודמות קטנה אקספוננציאלית ולכן מצד אחד כן נגיב למצב האקטואלי, אבל גם לא נעשה אוסילציות כל הזמן.

זה לא מסיים את החישוב של הטיים-אוט, כי צריך עדיין לקבוע את הקשר שלו ל-RTT. לשם כך נחשב את DevRTT שהוא הסטייה המצופה מה-RTT הממוצע, שיחושב בדומה ע"י

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

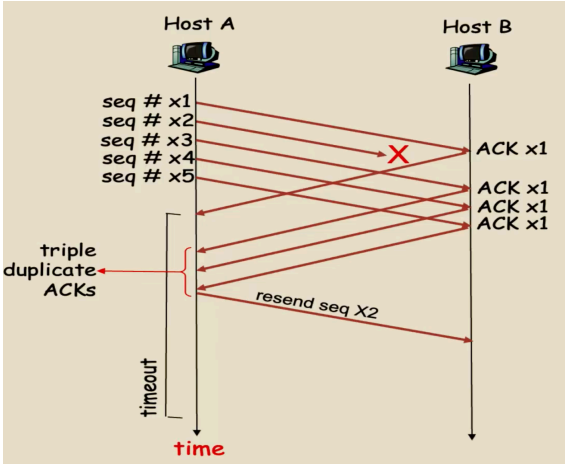
ואז לסיכום נקבע

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

ב-TCP יש רעיון שנקרא Fast Retransmit. לרוב הטיים-אוט הוא די ארוך, ולכן נוכל לקבל כמה ACK-ים מפקטות אחרות לפני שנקבל ACK או לא נקבל על הפקטה הנוכחית. נשים לב שאם קיבלנו כמה ACK-ים זהים (על מספר סידורי קודם), כנראה שזה אומר שהפקטה שאנחנו מחכים שתגיע לא הגיעה.

לכן לאחר 3 ACK-ים על אותו המספר הסידורי, כל סגמנט אחרי המספר הזה נחשב כנאבד ושולחים שוב את ההודעות. היתרון בזה זה שזה יכול לקרות גם לפני שהטיימר (הארוך יחסית) נגמר.

דוגמה נניח ש-1 הגיע אבל 2 נפל (והשאר גם הגיעו). לאחר 3 ACK-ים משוכפלים, נשלח שוב את 2 גם אם הטיים-אוט לא נגמר.



איור 27 : דוגמה ל-Fast Retransmit