בס"ד

206847584_322772880_325612695_328144951_325202158

חלק 1:

שאלה 1:

ננתח את שכבת התעבורה ונבחן מה הם הסיבות שיכולות לגרום לאיטיות ברשת. הסיבות האפשריות הן:

1. עומס ברשת-כמות גבוהה של תעבורת רשת יכולה לגרום לעיכובים
2. שימוש ב-TCP ואיבוד חבילות- אם התעבורה משתמשת בפרוטוקול TCP אז איבוד של חבילות במהלך התעבורה יגרום לשליחה מחודשת של חבילות ועיכוב בזמן העלאת הקובץ.
3. שימוש ב-TCP עם גודל חלון קטן מידי- יכול להגביל את קצב ההעברה של החבילות ברשת מכיוון שכל פעם נוכל שתהיה רק כמות קטנה של חבילות ברשת ולכן ייקח יותר זמן להעביר את הקובץ.
4. שימוש ב-TCP עם timeout לא מתאים. במקרה שאנחנו בוחרים timeout קטן מדי נשלח חבילות מסוימות כמה פעמים ללא צורך ונעמיס על הרשת. Timeout גדול יגרום לכך שאנחנו נחכה יותר מדי זמן עד שנשלח חבילה פעם נוספת במקרה שהיא נאבדה.
5. רוחב פס נמוך- אם רוחב הפס של הרשת קטן אז אנחנו נוכל להעביר כמות קטנה של מידע בכל זמן ולכן ייקח הרבה זמן.
6. תעדוף חבילות- אם הגדרות הרשת נותנות תעדוף לחבילות אחרות ברשת ולכן למשתמש הנוכחי ייקח הרבה זמן עד שהחבילות שלו ישלחו

פתרונות:

1. ניתן להוסיף נתבים ושרתים כך שהעומס יתחלק על פני מספר גדול של יותר של שרתים, ולכן העומס יפחת. כמו כן ניתן לתת תעדוף לחיבלות של הלקוח הספציפי הזה ברשת ואז החבילות שלו יעלו בקצב יותר מהיר משאר חבילות.
2. לחפש את הרכיב הבעייתי שגורם לאיבוד החבילות ולהחליף/לתקן אותו. (אנחנו מניחים שהבעיה היא לא בעומס הרשת.
3. במידה ונזהה המתנות רבות מדי ברשת נסדר את גודל החלון כך שיהיה בגודל האופטימלי.
4. נסדר את ה-timeout בהתאמה לרשת על פי הנוסחה שלמדנו בשיעור
5. רוחב הפס תלוי ברכיבים הפיזיים ולכן נצטרך להגדיל את הרכיבים בעלי רוחב הפס הנמוך ברשת.
6. נשנה את הגדרות התעדוף של הרשת. כך שהלקוח הנוכחי יהיה זה שמקבל עדיפות.

שאלה 2:

הפרוטוקול TCP נועד לזהות חבילות שהועברו לא לפי הסדר/ נפגעו במהלך המעבר שלהן או שלא הגיעו כלל. ובמקרים אלו צריך לשלוח את החבילות האלו מחדש. מעבר לכך ב-TCP גם מונעים עומס יתר בכך שמתאמים את קצב השידור לקצב ההורדה של המכשיר המקבל.(congestion control).

ניתוח השפעת בקרת הזרימה של TCP על הרשת:

בקרת הזרימה של פרוטוקול TCP עשויה לגרום לשליחה חוזרת של מידע או מגבלה של כמות מידע שיכול להישלח. ולכן הוא עשוי לגרום להאטה של הרשת.

ניתוח המקרה המתואר בשאלה שבו השולח הוא בעל ביצועים גבוהים משמעותית מאשר הביצועים של המקבל.

כאשר השולח בעל ביצועים טובים הוא יוכל לשלוח כמות גדולה יותר של מידע מאשר שהלקוח יכול לקלוט מה שיוביל בשלב למסוים לאיבוד של מידע. כאן נכנס מנגנון congestion control של TCP. מנגנון זה בודק בכל timeout האם הגיעו 3 ack רצופים בעלי אותו מספר. מחלק את גודל החלון ב-2 ואז נוסיף 3. ברגע שנקבל את ה-ack עבור החבילות נגדיל את החלון חזרה.

שאלה 3:

תפקיד הניתוב ברשת הוא בעל השפעה משמעותית על ביצועי הרשת ויעילותה בשל מגוון גורמים:

1. רוחב הפס של הנתבים במסלול- אם נבחר מסלול עם נתב בעל רוחב פס נמוך אז כמות המידע שנוכל להעביר ברשת תהיה מוגבלת.
2. כמות נתבים בדרך – לכל נתב יש זמן עיבוד ולכן מעבר במספר רב של נתבים יעלה את זמן העיבוד הכולל. ככל שנעבור ביותר נתבים כך הסיכוי שלנו לאבד חבילות יגדל ולכן ב-TCP זה יגדיל את הסיכוי לשליחה חוזרת של חבילות.
3. אורך פיזי של המסלול- ככל שאורך המסלול יהיה יותר גדול ככה במידה וקצב ההעברה זהה אז זמן המעבר במסלול הארוך יותר יהיה איטי יותר
4. סוג החומרה הפיזית- למשל באיזה כבל השתמשנו על מנת להעביר את המידע למשל סיבים אופטים יביאו תוצאה טובה יותר מאשר כבל ETHERNET, כמו כן חומרה פגומה עלולה לגרום לאיבוד מידע ויוצרת צורך בשליחה חוזרת, לכן נעדיף מסלול עם חומרה תקינה.

5. עומס ברשת-אם נבחר מסלול עמוס אז זמן השליחה יהיה גדול יותר, מכיל בתוכו את זמן ההשהייה בכל נתב בדרך.

נבנה גרף שבו כל נתב הופך לקודקוד, צלעות יהיו בין כל 2 נתבים שיש ביניהם קו תקשורת ונבנה פונקציית משקל לצלעות שמתחשבת בחמשת בפרמטרים שכתבנו למעלה. לאחר מכן נבצע dijkstra. על מנת לקבל את המסלול הקל ביותר וזה יהיה המסלול שנבחר.

שאלה 4:

פרוטוקול MPTCP משפר את ביצועי הרשת על ידי:

שיפור ביצועים- פרוטוקול MPTCP משתמש במספר סוקטים של TCP בו זמנית ולכן הוא יכול להאיץ את ההעלאה וההורדה במיוחד כאשר אחד מהחיבורים לא יציב או איטי.בנוסף זה עוזר להתמודד עם בעיית HOL על ידי הגדלת קצב העלאת הנתונים.

גיבוי לחיבורים- במקרה של אובדן אחד החיבורים נוכל להשתמש בסוקט חליפי לא נצטרך לפתוח סוקט חדש אלא נוכל להשתמש באחד הסוקטים האחרים שכבר קיימי

השהיית נתונים- מפחית את ההשהיה הכללית של התקשורת על ידי פיזור הנתונים על מספר מסלולים.
ומאפשר לבחור את המסלול עם ההשהיה הקטנה ביותר לכל חלק של נתונים.

איזון עומסים דינאמי- יכול להפנות את התנועה לחיבורים פחות עמוסים על מנת לפזר את העומס על הרשת.

שיפור חווית המשתמש- ניתן לשמור על חיבור רציף ואיכותי גם כאשר התנאים ברשת משתנים למשל מעבר בין חיבורי אינטרנט שונים.

שאלה 5:

הגורמים שמובילים לאיבוד חבילות רבות בשכבת הרשת הם:

1. עומס יתר על נתבים- כשנתבים מקבלים כמות גדולה יותר של חבילות מאשר מה שהם מסוגלים לעבד הם עלולים לאבד חבילות.
2. תקלות בחומרת הנתבים- תקלות בחומרה כמו כרטיסי רשת פגומים , חיבורים לא תקינים לרשת או זיכרון קטן מדי יכולים לגרום לאיבוד חבילות.
3. ניתוב שגוי עלול להוביל למצב שתהיה לנו לולאה שתיתקע בין כמה נתבים ולכן ה-ttl של החבילה ייגמר לפני שהיא תגיע ליעד.
4. תקיפת רשת שיזרוק חבילות מהנתבים ברשת.

הגורמים שמובילים לאיבוד חבילות רבות בשכבת התעבורה הם:

1. פרוטוקל TCP עלול לזרוק חבילות במקרה שהצד השולח שולח יותר נתונים מאשר מה שהצד המקבל יכול לקלוט.
2. שימוש בחלון גדול מדי – נשלח כמות גדולה יותר של חבילות מאשר שהלקוח מסוגל לקבל ולכן יזרקו מספר רב של חבילות

פתרון בעיות בשכבת הרשת:

1. להגדיל את כמות הנתבים ברשת על מנת להוריד את העומס כל אחד או לשדרג אותם.
2. נוודא תקינות של החומרה ונחליף רכיבים פגומים.
3. בחירת מסלול על ידי dijkstra
4. שימוש בחומת אש או מערכות זיהוי התקפות אחרות.

פתרון בעיות בשכבת התעבורה:

1-3 תיאום של כמות החבילות שניתן להעביר בבת אחת.

בס"ד

206847584_322772880_325612695_328144951_325202158

חלק 2' – תשובות לשאלות על מאמרים:

Flow Pic: Encrypted Internet Traffic Classification is as Easy as Image Recognition

1. מה התרומה העיקרית של המאמר?

תרומתו העיקרית של המאמר הינה מציאת שיטה חדשנית לזיהוי תעבורה מוצפנת ברשת באמצעות המרת גודל המידע אל תמונה ושימוש ברשת נוירונים לטובת זיהוי סוג התעבורה מהתמונה.
המרת המידע אל תמונה (flowpic) תורם בכך שאין צורך לעבוד עם תוכנות ידניות אלא כלל המידע על גודל החבילות מומר לתמונה.
שימוש בטכנולוגיה חדישה של סיווג תמונות בעזרת רשת נוירונים ובכל הגעה אל דיוק מירבי.
הגעה לדיוק רב אפילו ברשת מוצפנת כמו בשימוש בvpn ו tor .

2. באיזה פיצ'רים המאמר משתמש ואיזה מהם חדשניים?

פיצ'רים שאינם חדשניים:

1. שימוש בגדלי חבילות וזמני הגעה, שיטה זו שימשה המון שיטות לסיווג תעבורה אך כאן נעזר בה באופן חדשני.
2. הגדרת סשן לפי 5 מאפיינים עיקריים ויצירת tuple של: פרוטוקול, כתובת IP של המקור, כתובת IP של היעד, פורט מקור ופורט יעד.
3. חלוקה לזרמים קדמיים ואחוריים, כאן נשתמש בפיצ'ר זה על מנת לייצר תמונה של הזרימה ברשת אך מספר מחקרים השתמשו בשיטה זו גם כן.

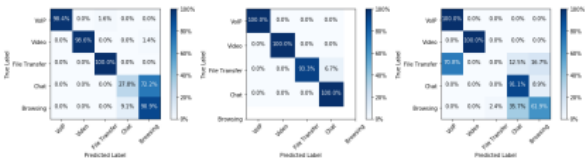
פיצ'רים חדשניים:

1. Flow pic - במקום לבצע את הזיהוי באופן ידני נוכל להעביר את כל גדלי החבילות אל ואז זמני הגילוי שלהם לתמונה חזותית באופן הבא: כאשר הציר האופקי מתאר את זמן החבילה והאנכי מתאר את הגדול של החבילה.
2. שימוש ברשת נוירונים CNN לסיווג התמונה: בדרך כלל שימוש בCNN הוא לסיווג תמונה אך כאן נעזר בו כדי לזהות תמונה שמייצגת תעבורת רשת.
ארכיטקטורת CNN כוללת 3 שכבות: שכבת קונבולוציה המחלצת מאפיינים מהתמונה , שכבת pooling המקטינה את גודל הנתונים אך שומרת על המידע הרלוונטי ושכבת fully connected המקבלת את ההחלטה הסופית על סוג התעבורה.
רשת הנוירונים מזהה תבניות באופן עצמאי מכיוון שהיא רשת לומדת, כמו כן רשת CNN יודעת לזהות דפוסים של תבניות ולא מחפשת מידע מדויק ומסיבה זו ניתן לזהות גם מידע מוצפן ולסווג אותו לתעבורה המתאימה.

3. מהם התוצאה העיקרית מהמאמר ומהם התובנות מהתוצאות?

תוצאות:

Class	Accuracy (%)			
VoIP	Training/Test	Non-VPN	VPN	Tor
	Non-VPN	**99.6**	99.4	48.2
	VPN	95.8	**99.9**	58.1
	Tor	52.1	35.8	**93.3**
Video	Training/Test	Non-VPN	VPN	Tor
	Non-VPN	**99.9**	98.8	83.8
	VPN	54.0	**99.9**	57.8
	Tor	55.3	86.1	**99.9**
File Transfer	Training/Test	Non-VPN	VPN	Tor
	Non-VPN	**98.8**	79.9	60.6
	VPN	65.1	**99.9**	54.5
	Tor	63.1	35.8	**55.8**
Chat	Training/Test	Non-VPN	VPN	Tor
	Non-VPN	**96.2**	78.9	70.3
	VPN	71.7	**99.2**	69.4
	Tor	85.8	93.1	**89.0**
Browsing	Training/Test	Non-VPN	VPN	Tor
	Non-VPN	**90.6**	-	57.2
	VPN	-	-	-
	Tor	76.1	-	**90.6**



(a) Non-VPN (b) VPN (c) Tor

Figure 4: Confusion matrices of the 3 traffic categorization problems; over non-VPN, over VPN, and over Tor.

בס"ד

206847584_322772880_325612695_328144951_325202158

תוצאות המאמר מגלות דיוק גבוה מאוד עבור השיטה המתוארת (זיהוי תעבורה באמצעות flow pic) , בשיטה זו נזהה ב99.7% מהמקרים את היישום הנכון לעומת שיטות קודמות שהגיעו לכל היותר ל93.9% .

המודל התאמן על 5 סוגי קטגוריות voip , צ'אט, העברת קבצים, וידאו וגלישה. וכל בדיקה התבצעה על 3 סוגי תעבורה עם VPN , ללא VPN ו TOR .

נפרט אותן כאן :

- כאשר המודל אומן על non-vpn הוא הצליח לזהות ביותר מ96.2% את סוג התעבורה למעט במקרה של גלישה שם הוא הצליח רק ב90.6% מהמקרים.
- כאשר המודל אומן על vpn הוא הצליח לזהות המעל 99.2 מהמקרים את סוג התעבורה.
- כאשר המודל אומן על TOR הוא הצליח לזהות במעל 89% את סוג התעבורה למעט העברת קבצים שם הצליח רק ב 55.8%

נשים לב שהמודל זיהה בחלק גדול מהמקרים גם סוגי תעבורה שלא אומן עליהם למשל :

- כאשר התאמן על non-vpn המודל זיהה vpn במעל מ 78.9% מהמקרים וזיהה tor בממוצע כ-60% מהמקרים.
- כאשר התאמן על vpn המודל זיהה non-vpn במעל 54% מהמקרים ו tor מעל 54.5% מהמקרים.
- כאשר התאמן על tor זיהה non-vpn במעל 52.1% מהמקרים ו vpn במעל 35.8% מהמקרים.

מסקנות ותובנות מהתוצאות :

- שימוש ברשת נוירונים CNN מאפשר זיהוי מדויק ביותר מאשר השיטות המוכרות עד היום
- Flow pic מזהה תעבורה מוצפנת בצורה כמעט מושלמת מה במראה שהמרת התעבורה לתמונות הינה שיטה מוצלחת
- בנוגה לזיהוי תעבורה בtor המודל מזהה תעבורה באופן סביר, מה שמצביע על כל שהתעבורה בtor נראית באופן שונה מאשר vpn ו non-vpn עבור flowpic ושהמודל שלנו לא מצליח לזהות את הדפוס של תעבורה זו בצורה מצויינת
- יכולת הזיהוי של flow pic לא תלויה באף יישום אלא היא גנרית לחלוטין מכיוון שהיא משתמשת בגדלי וזמני החבילות ולא במידע פנימי, לכן היא מסוגלת לזהות בדיוק מירבי סוגי תעבורה אף שלא אומנה עליהם.
- קל להבדיל בין סוגי התעבורה של vpn ל non-vpn אך עבור tor הרבה יותר קשה.
- העברת זרם המידע כגודל וזמני השליחה אל תמונה חוסך את הצורך בחילוץ ידני של התכונות הרלוונטיות לזיהוי סוג התעבורה, מה שחוסך זמן רב.
- השיטה פועלת היטב גם עבור חלונות זמן קצרים של זרמי תעבורה חד כיווניים.

סיכום : שיטת flow pic מספקת מודל חדשני לזיהוי תעבורה מוצפנת ברשת בדיוק גבוה מאוד, תוך שימוש בשיטה גנרית ולא תלויה באפליקציות ומאפיינים שחולצו ידנית.

בס"ד

206847584_322772880_325612695_328144951_325202158

שם המאמר:

Early Traffic Classification With Encrypted

ClientHello: A Multi-Country Study

3. מה התרומה העיקרית של המאמר?
תרומתו העיקרית של המאמר היא הצגת שיטה חדשנית לזיהוי סוג תעבורה ברשת עוד לפני שליחת מידע כל האפליקציה. המאמר מציג דרך שונה מהדרך המסורתית TC (traffic classification) שהייתה תלויה בעיקר ב SNI (Server name indication) ונכשלה כאשר משתמשים ב ECH (Encrypted client hello). השיטה החדשה נקראת: hRFTC (hybrid Random Forest Traffic Classification). שיטה זאת בעזרת שילוב מידע על מאפייני tls שכלל אינם מוצפנים בנוסף אל תכונות תעבורה סטטיסטיות מגיעו לזיהוי מוקדם של עד 94.6% מהתעבורה. בעוד שהשיטות המסורתיות משיגות 38.4% .

4. באיזה פיצ'רים המאמר משתמש ואיזה מהם חדשניים?

א. פיצ'רים מבוססים חבילות – packet based features:

- שימוש בפרמטרים של tls handshake , cipher suite , key share group - קבוצת פרמטרים שבעזרתם הלקוח והשרת מחליטים כיצד לשנות מפתחות הצפנה, וסדר התוספות שיש בשדה של client hello.
- שימוש ברשת נוירונים NN לטובת למידה תכונות של תעבורה מוצפנת מהחבילות הראשונות.
- שימוש ב random forest על מנת לחזות את סוג התעבורה שאמורה להגיע.

ב. פיצ'רים מבוססים זרימה – flow based features:

- זיהוי סדרות של גדלי חבילות על פי זמנים
- זיהוי סדרות של הפרשי זמנים בין חבילות.
- ניתוח נתונים סטטיסטיים של גדלי והפרשי החבילות כמו: מינימום, מקסימום, ממוצע, חציון, ומידע נוסף על חלקים שונים מהמידע (25% , 30% וכו' – Percentiles)
- שימוש ברשת נוירונים לעיבוד המידע.

ג. פיצ'רים היברידיים:

- שילוב של tls handshake וניתוח סטטיסטי של חבילות ראשוניות.

ד. פיצ'רים חדשניים בהם משתמשת השיטה hRFTC:

- שימוש בנתוני tls handshake הלא מוצפנים המצוינים בפיצ'רים מבוססי חבילות והתאמתם אל פרוטוקול quic בנוסף.
- בחירת החבילות לבדיקה, בשיטה זו נבדוק עד החבילה הראשונה של האפליקציה.
- שימוש עדכני בשיטת RB-RF שמבצעת מיקום מחדש של ביטים בפרוטוקול tls והופכת אותם לווקטור קבוע שנכנס כקלט לrandom forest. העדכון הוא הרחבה לquic ושימוש במאפייני זרימה בנוסף למאפייני גודל חבילה.

בס"ד

206847584_322772880_325612695_328144951_325202158

5. מהם התוצאות העיקרית מהמאמר ומהם התובנות מהתוצאות?

- התוצאה העיקרית מהמאמר היא היכולת לבצע eTC (early traffic classification) לתעבורה הכוללת ECH (encrypted client hello) ל94.6% מהתעבורה בהצלחה באמצעות השיטה hRFTC .
- בשיטות מבוססות חבילה (RB-RF) הצליחו לזהות לכל היותר 38.4% מהתעבורה נכונה.
- ובאלגוריתמים מבוססי זרימה אמנם הגיעו לתוצאות טובות יותר (88.9% עבור CECNET) אבל לא טובות כמו hRFTC .

Class	F-score [%]						
	Hybrid Classifiers			Flow-based Classifier	Packet-based Classifiers		
	hRFTC [proposed]	**UW** [35]	**hC4.5** [34]	**CESNET** [63]	**RB-RF** [24]	**MATEC** [33]	**BGRUA** [32]
BA-AppleMusic	**92.1**	89.5	80.2	89.2	25.5	13.1	14.5
BA-SoundCloud	**99.6**	98.9	97.8	98.7	84.4	81.8	82.0
BA-Spotify	**93.6**	90.8	89.0	88.5	16.3	0.0	3.6
BA-VkMusic	**95.7**	89.7	88.5	91.8	2.6	2.1	3.2
BA-YandexMusic	**98.5**	93.2	93.7	92.5	1.8	0.2	0.1
LV-Facebook	**100.0**	99.7	99.8	99.8	**100.0**	**100.0**	**100.0**
LV-YouTube	**100.0**	**100.0**	99.9	**100.0**	**100.0**	99.0	98.4
SBV-Instagram	**89.7**	74.7	76.5	78.8	10.0	6.3	6.4
SBV-TikTok	**93.3**	81.8	81.8	76.3	38.3	34.3	34.5
SBV-VkClips	**95.7**	94.0	91.3	92.4	53.2	37.7	46.0
SBV-YouTube	**98.2**	96.6	94.7	96.4	1.1	0.2	0.2
BV-Facebook	**87.7**	78.2	79.7	77.6	5.6	3.2	3.8
BV-Kinopoisk	**94.1**	84.1	85.8	89.8	5.4	4.0	4.1
BV-Netflix	**98.5**	97.2	95.2	93.7	50.7	52.3	56.1
BV-PrimeVideo	**91.3**	86.7	84.1	84.7	32.5	24.7	26.8
BV-Vimeo	**94.8**	90.5	90.2	81.4	72.0	19.5	68.6
BV-VkVideo	**88.6**	80.5	80.4	79.7	10.5	0.0	0.1
BV-YouTube	**85.9**	84.3	77.0	78.5	22.3	19.6	20.2
Web (known)	**99.7**	99.5	99.4	99.4	98.0	98.0	98.0
Macro-F-score (average)	**94.6**	89.9	88.7	88.9	38.4	31.4	35.1

Test Country	Share in Dataset	Training Country	Classifier Macro F-score [%]		
			hRFTC	hC4.5	UW
Germany	18.8%	Others	38.4	26.9	19.5
Kazakhstan	3.0%	Others	57.3	32.3	27.5
Russia	29.2%	Others	49.8	35.6	20.9
Spain	16.3%	Others	38.5	34.4	12.6
Turkey	25.2%	Others	35.1	26.0	16.4
USA	7.5%	Others	49.2	41.4	21.3

- כל האלגוריתמים הדינמיים הגיעו לתוצאות נמוכות מאוד בבדיקתם על אזורים גאוגרפיים בהם לא אומנו.
- קיים קושי בזיהוי התעבורה עם שימוש בquic באלגוריתמים הדינאמיים כאשר משתמשים ב padding .

<u>תובנות מהתוצאות:</u>

- אף אחד מהמחקרים המוצגים במאמר אינו מתייחס אל בעיית ECH למעט אחד שייחס קשר בין גודל חבילת CH אל SNI ובכך ניתן לבצע זיהוי עם flow based.
- התוצאות הלא טובות במודלים היברידיים על תעבורה מאזורים גאוגרפיים אחרים מגיעות מכיוון שבמדינות שונות אופן פיצול החבילה מתבצע אחרת. מסיבה זו ישנו צורך לאמן את המודל בכל אזור גאוגרפי בנפרד.
- מודלים מבוססי החבילה כשלו בעיקר מכיוון שהסתמכו על זיהוי SNI וכאשר הוא הפך למוצפן בעזרת ECH הם הפכו ללא רלוונטיים.
- מודלים מבוססי זרימה אמנם לא ניתן לומר שהם כשלו אך הם הצליחות במידה, אחת הבעיות העיקריות שלהם נבעה מהעובדה שהם היו צריכים מידע רב על מנת לזהות את סוג התעבורה מה שלא מתאים ל eTC .
- הצלחת המודלים ההיברידיים בכלל hRFTC בפרט נבעה ממספר סיבות:
- שילוב של מאפייני tls שאינם מוצפנים עם מאפייני הזרימה מגדיל משמעותית את דיוק זיהוי התעבורה, בעיקר בסביבה בה SNI מוצפן (ECH).
- התאמת המידע הרלוונטי אל quic ושימוש במודל rb-rf גם עבור quic , משום שהמודל ההיברידי משתמש במאפייני tls נוספים ולא רק גודל החבילה הוא יודע להתגבר על quic padding .

בס"ד

206847584_322772880_325612695_328144951_325202158

שם המאמר:

Analyzing HTTPS Encrypted Traffic to Identify

User's Operating System, Browser and Application

6. מה התרומה העיקרית של המאמר?
התרומה העיקרית של מחקר זה היא הצגת שיטה חדשנית בה מתואר כיצד תוקף יכול לזהות פרטים על המותקף כמו, מערכת הפעלה, דפדפן והאפליקציה של המשתמש מתוך תעבורת https מוצפנת.
תרומתו העיקרית של המחקר:
   א. מחקר זה מוכיח כי גם תעבורה מוצפנת אינה מספיקה לחלוטין על מנת להגן על פרטיות המשתמש.
   ב. מחקר זה הוכיח כי בעזרת שיטה חדשנית ניתן להגיע לדיוק רב בזיהוי השלשה (מערכת הפעלה, דפדפן ואפליקציה) של 96.06% , מהווה שיפור רב לעומת שיטות קודמות במיוחד בתעבורה מוצפנת.
   ג. השיטה החדשנית נעזרה בטכניקות מסורתיות (גודל חבילה והפרש זמנים) וטכניקות חדשניות כמו התנהגות מתפרצת של דפדפנים, מדדי SSL כמו מספר הרחבות ואורך session id, מדדי tcp כמו גודל חלון ומספר חבילות keepalive . שילוב המאפיינים החדשים הביא את הביצועים ל96.06% מ 93.51% .

7. באיזה פיצ'רים המאמר משתמש ואיזה מהם חדשניים?
באופן פורמלי -
התוספות הבסיסיות שהמאמר השתמש בהם:
   1. Forward packets #
   2. Forward total Bytes #
   3. Min forward inter arrival time difference
   4. Max forward inter arrival time difference
   5. Mean forward inter arrival time difference
   6. STD forward inter arrival time difference
   7. Mean forward packets
   8. STD forward packets
   9. Backward packets #
   10. Backward total Bytes #
   11. Min backward inter arrival time difference
   12. Max backward inter arrival time difference
   13. Mean backward inter arrival time difference
   14. STD backward inter arrival time difference
   15. Mean backward packets
   16. STD backward packets
   17. Mean forward TTL value
   18. Minimum forward packet
   19. Minimum backward packet
   20. Maximum forward packet
   21. Maximum backward packet
   22. Total packets #
   23. Minimum packet size
   24. Maximum packet size
   25. Mean packet size
   26. Packet size variance

בס"ד

206847584_322772880_325612695_328144951_325202158

התוספות החדשניות הם :

1. TCP initial window size
2. TCP window scaling factor
3. SSL compression methods #
4. SSL extension count #
5. SSL cipher methods #
6. SSL session ID length
7. Forward peak MAX throughput
8. Mean throughput of backward peaks
9. Max throughput of backward peaks
10. Backward min peak throughput
11. Backward STD peak throughput
12. Forward number of bursts
13. Backward number of bursts
14. Forward min peak throughput
15. Mean throughput of forward peaks
16. Forward STD peak throughput
17. Mean backward peak inter arrival time diff
18. Minimum backward peak inter arrival time diff
19. Maximum backward peak inter arrival time diff
20. STD backward peak inter arrival time diff
21. Mean forward peak inter arrival time diff
22. Minimum forward peak inter arrival time diff
23. Maximum forward peak inter arrival time diff
24. STD forward peak inter arrival time diff
25. Keep alive packets #
26. TCP Max Segment Size
27. Forward SSL Version

ננתח את התוספות באופן קצת ברור יותר :

השיטות הבסיסיות :

- מידע שמתבסס על גודל חבילה.
- מידע שמתבסס על הפרשי זמני הגעה בין חבילות.

השיטות החדשניות :

- בדיקת גודל החלון ב tcp header .
- מבט על מאפייני ssl כמו כמות תוספות, מספר אפשרויות ההצפנה, אורך id , גרסה (עוזר לזיהוי מערכת ההפעלה) וכו'
- מאפיינים נוספים על גדלי החבילות מול הפרשי הזמנים כמו שיא קצב העברת הנתונים. מידע מסוג זה יעזור לנו להבחין בהתנהגות מתפרצת של דפדפן.
- מספר חבילות keep alive , דפדפנים שונים משתמשים בחבילות שונות.

בס"ד

206847584_322772880_325612695_328144951_325202158

8. מהם התוצאות העיקריות מהמאמר ומהם התובנות מהתוצאות?

תוצאות:

התוצאה העיקרית מהמאמר היא שבעזרת השיטה החדשנית המתוארת ניתן לזהות בדיוק גבוה מאוד את מערכת ההפעלה, הדפדפן והיישומים שאותם מפעיל המשתמש.

הסבר על התוצאות, ניתן לשים לב כי עבור זיהוי שלושת האלמנטים (מערכת ההפעלה, הדפדפן והיישום) עם הפיצ'רים החדשים מוגבר משמעותית ל96%.

עבור מערכת ההפעלה השימוש עם הפיצ'רים החדשים העלה את הזיהוי ל כ – 100% נשים לב כי גם עבור התוספות החדשות בלבד ללא תוספות הבסיס היינו באותו אחוז זיהוי.

עבור זיהוי דפדפן עם הפיצ'רים החדשים הגענו אל 99% לעומת פחות מ96% עבר רק הבסיס.

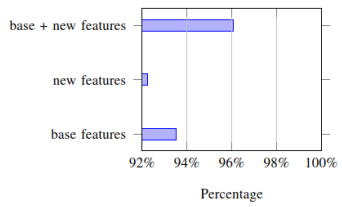
עבור זיהוי היישום עם הפיצ'רים נגיע אל יותר מ96% לעומת פחות מ96% עבור רק הבסיס.

נתאר כעת את טבלת הטעויות לזיהוי שגוי:

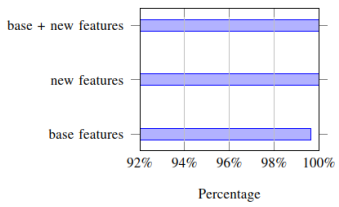
Predicted labels

Real labels	Windows IExplorer Twitter	Ubuntu Firefox Google-Background	Windows Non-Browser Microsoft-Background	Windows Chrome Twitter	Windows Firefox Twitter	OSX Safari Google-Background	OSX Safari Youtube	Ubuntu Chrome Unknown	Windows Chrome Google-Background	Ubuntu Firefox Twitter	OSX Safari Unknown	Ubuntu Firefox Unknown	Ubuntu Chrome Google-Background	Ubuntu Chrome Twitter	Windows Firefox Google-Background	OSX Safari Twitter	Ubuntu Firefox Youtube	Windows Non-Browser Teamviewer	Ubuntu Chrome Youtube	Windows Non-Browser Dropbox	Windows Chrome Unknown	Ubuntu Chrome Facebook	Windows Firefox Unknown	Ubuntu Firefox Facebook	OSX Chrome Twitter	Windows IExplorer Unknown	Ubuntu Non-Browser Microsoft-Background	Windows IExplorer Google-Background	OSX Chrome Google-Background	OSX Chrome Unknown
Windows IExplorer Twitter	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Firefox Google-Background	0	.97	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Windows Non-Browser Microsoft-Background	0	0	.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Windows Chrome Twitter	0	0	0	.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0
Windows Firefox Twitter	0	0	0	0	.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.02	0	0	0	0	0	0	0
OSX Safari Google-Background	0	0	0	0	0	.92	.04	0	0	0	.02	0	0	0	0	.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OSX Safari Youtube	0	0	0	0	0	.02	.97	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Chrome Unknown	0	0	0	0	0	0	0	.84	0	0	0	0	.07	.04	0	0	0	0	.01	0	0	.03	0	0	0	0	0	0	0	0
Windows Chrome Google-Background	0	0	.01	.03	0	0	0	0	.94	0	0	0	0	0	.02	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0
Ubuntu Firefox Twitter	0	0	0	0	0	0	0	0	0	.95	0	.03	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0
OSX Safari Unknown	0	0	0	0	0	.06	.01	0	0	0	.91	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Firefox Unknown	0	.02	0	0	0	0	0	0	0	.08	0	.87	0	0	0	0	.01	0	0	0	0	0	0	.03	0	0	0	0	0	0
Ubuntu Chrome Google-Background	0	.07	0	0	0	0	0	.18	0	0	0	0	.73	0	0	0	0	0	.02	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Chrome Twitter	0	.02	0	0	0	0	0	.08	0	0	0	0	.03	.84	0	0	0	0	.01	0	0	.01	0	0	0	0	0	0	0	0
Windows Firefox Google-Background	0	0	0	.01	0	0	0	0	.01	0	0	0	0	0	.97	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0
OSX Safari Twitter	0	0	0	0	0	0	.06	0	0	0	.03	0	0	0	0	.91	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Firefox Youtube	0	.02	0	0	0	0	0	0	0	.02	0	.02	0	0	0	0	.93	0	0	0	0	0	0	0	0	0	0	0	0	0
Windows Non-Browser Teamviewer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Ubuntu Chrome Youtube	0	0	0	0	0	0	0	.07	0	0	0	0	.13	.04	0	0	0	0	.74	0	0	.02	0	0	0	0	0	0	0	0
Windows Non-Browser Dropbox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Windows Chrome Unknown	0	0	.02	.09	0	0	0	0	.02	0	0	0	0	0	0	0	0	0	0	0	.86	0	0	0	0	0	0	0	0	0
Ubuntu Chrome Facebook	0	0	0	0	0	0	0	.3	0	0	0	0	.04	0	0	0	0	0	0	0	0	.67	0	0	0	0	0	0	0	0
Windows Firefox Unknown	0	0	.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.94	0	0	0	0	0	0	0
Ubuntu Firefox Facebook	0	.06	0	0	0	0	0	0	0	.11	0	.28	0	0	0	0	0	0	0	0	0	0	0	.56	0	0	0	0	0	0
OSX Chrome Twitter	0	0	0	0	0	0	0	.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.75	0	0	0	.06	.06
Windows IExplorer Unknown	.71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.29	0	0	0	0
Ubuntu Non-Browser Microsoft-Background	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Windows IExplorer Google-Background	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
OSX Chrome Google-Background	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
OSX Chrome Unknown	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

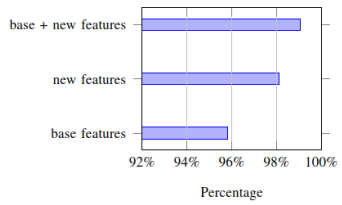
(a) Tuple Confusion Matrix



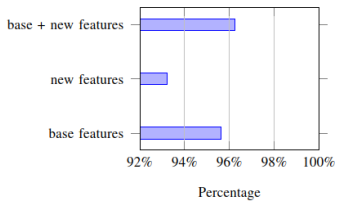
(a) Tuple Accuracy Results



(b) OS Accuracy Results



(c) Browser Accuracy Results



(d) Application Accuracy Results

בס"ד

206847584_322772880_325612695_328144951_325202158

א. בזיהוי מערכת ההפעלה כמעט ואין טעויות לכן הדיוק כמעט מושלם, אם ישנה טעות היא בדרך כלל ביישום או דפדפן אך מערכת ההפעלה מזוהה במדויק.
ב. ניתן לראות שבזיהוי ישנם מעט שגיאות, עיקר הטעויות הן בין הדפדפנים chrome firefox ו safari .
ג. בזיהוי יישומים ניתן לראות כי יש אחוז זיהוי שגוי גבוה לfacebook להיות אפליקציה לא ידועה ב 28% מהמקרים, מכאן ניתן להסיק כי יש קושי רב לזהות את facebook .
ד. כמו כן ניתן לראות שב18 אחוז מהמקרים את google-backround זיהינו בתור אפליקציה לא ידועה.
ה. כאשר התעבורה לא מזוהה מראש המודל מתקשה לזהות, למשל ב71% מהמקרים המודל מזהה את windows – explorer – unknown כ windows – explorer – twitter
ו. כאשר התעבורה הייתה ubunto ללא דפדפן ורק עדכוני רקע של מיקרוסופסט המודל מזהה את התעבורה כ – ubuntu – chrome – unknown .

<u>תובנות מהמאמר:</u>

1. זיהוי מערכת ההפעלה הוא הקל ביותר לאחר מכן קל יותר לזהות דפדפנים ואז הכי קשה לזהות זה יישומים.
2. ישנם אפליקציות מסוימות שקשות יותר לזיהוי מאחרות במיוחד כאשר הפרמטרים שלהן דומות מאוד לאחרות.
3. שימוש בזיהוי דפוסי תעבורה (Bursty Traffic Behavior) תורם מאוד בעיקר לזיהוי סוג הדפדפן ויישומים, ניתן לראות זאת במיוחד בטבלה של זיהויים נכונים עם התוספות החדשות.
4. כאשר התעבורה לא זוהתה בעבר (unknown) המודל מתקשה בזיהוי.
5. השיטה מאפשרת זיהוי מצוין אפילו כאשר התעבורה מוצפנת ולעומת המודל הנאיבי שמשער את הזיהוי ומצליח רק ב 32% שיטה חדשנית זו מזהה 96.06% .
6. נשים לב שלא חייב לנתח את התעבורה המוצפנת בשביל לזהות בדיוק גבוה מאוד את מערכת ההפעלה, בנוסף אפילו המודל ללא הפיצ׳רים החדשים הצליח בדיוק כמעט מלא.
7. נתונים אלו מעלים חשש גדול לפרטיות מכיוון שמוכח כי ניתן לדעת המון מידע על המשתמש מבלי לדעת מהו התוכן עצמו של המידע.

בס"ד

206847584_322772880_325612695_328144951_325202158

חלק 3 – ניתוח גרפים

הסבר כללי על הקוד ליצירת גרפים:

ספריות שהוספנו :

```
import pyshark
import matplotlib.pyplot as plt
import nest_asyncio
nest_asyncio.apply()
```

הקוד נכתב בפייתון עם הספריות לעיל, הקוד מצורף יחד עם קוד של התוקף רק בקובץ נפרד.

שם הקובץ ליצירת גרפים הוא main.py .

הקוד מכיל בתוכו התייחסות לכל מאפיין שהתבקשנו A-F :

A. IP header fields – חלוקת כל אפליקציה לגרף עוגה על פי הפרוטוקול בו היא משתמשת, בנוסף גרף אחד שמשווה לפי ממוצע ttl בכל אפליקציה.

B. TCP header fields – גרף 1 שמשווה בין ממוצעי גדלי החלונות של כל אפליקציה וגרף 2 שבודק כמה חבילות עברות בכל פורט מסך כל החבילות של אותה אפליקציה.

C. TLS header fields – גרף 1 המשווה בין כמות חיבורי הtls המקביליים בכל אפליקציה וגרף 2 שבודק את גרסאות החיבורים.

D. packet sizes – גרף 1 משווה בממוצע כמה בתים נשלחים בכל אפליקציה, גרף 2 משווה את גודל החבילה המקסימלית בכל אפליקציה. כמו כן הצגנו בנקודה זו את ההשווה של flowpic כפי שהוצג באחד המאמרים (מפורט בהמשך)

E. Packets inter-arrivals – גרף המשווה בין ממוצעי הפרשי הזמנים בין החבילות לכל אפליקציה.

G. Flow volume + F. Flow size – גרף אחד שמשווה מספר חבילות ממוצע לשנייה וגרף שני שמשווה כמות בתים ממוצעת לשנייה.

בס"ד

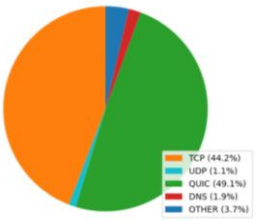
206847584_322772880_325612695_328144951_325202158

ניתוח גרפים

ניתוח גרף לפי Ip Header (A):

בחלק זה הפקנו חמישה גרפי עוגה שמחלקים את התעבורה לפי סוג הפרוטוקול בה עוברת החבילה, כמו כן פירטנו את המידע היבש מהגרפים בצמוד לגרף ובסוף הקטע כתבנו את סיכום המשמעויות. לאחר מכן הפקנו גרף עמודות שמשווה TTL בכל אפליקציה.

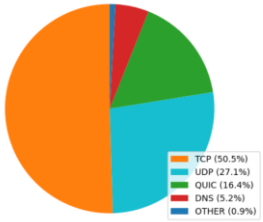
chrome



Chrome : רוב התעבורה (49.1%) עוברת דרך פרוטוקול quic, שהוא פרוטוקול מבוסס UDPשנועד לשיפור ביצועי רשת. tcp מהווה 44.2% מהתעבורה, כלומר חלק משמעותי מהתעבורה עדיין משתמש בפרוטוקול tcp. פרוטוקולים אחרים (dns וכו') משתמשים בחלק קטן יחסית מהתעבורה.

משמעות : chrome מנצל את quic בכמות גבוהה יחסית, מה שמשפר את מהירות טעינת האתרים לעומת שימוש ב tcp-בלבד.

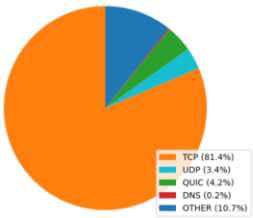
edge



Edge: בתעבורה של edge רוב החבילות עוברות מעל פרוטוקול tcp (50.5%) בעוד שמעל quic עוברות רק 16.4% .

משמעות : edge אמנם מאמץ את פרוטוקול quic אך על תווך שלא מתאים לquic יבחר בtcp.

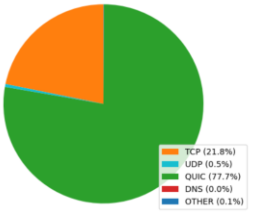
spotify



Spotify : בתעבורה של spotify כמעט כלל החבילות עוברות בtcp (81.4%) , spotify משתמש בudp רק ב3.4% מהתעבורה מעט לעומת youtube .

משמעות : מעיד על אופן הפעולה של spotify (הורדת buffer של מידע) לעומת youtube שמעדיפה מהירות על פני אמינות.

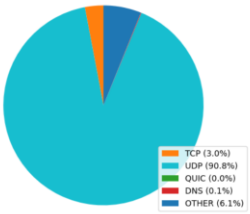
youtube



Youtube : בתעבורה של youtube ניתן לראות שימוש נרחב בפרוטוקול quic (77.7%) , ושימוש מסוים בtcp (21.8%).

משמעות: youtube דורש העברת מידע רב (סרטונים) באיכות טובה ומהירה . quic מאפשר זאת מכיוון שהוא עובד מעל udp (מכאן המהירות), חוסך שלבי handshake ומונע עיכוב בעת איבוד של חבילה בעזרת מנגנונים חכמים כמו יצירת מספר נתיבי תעבורה.

zoom



Zoom : בתעבורה של זום ניתן לראות כי כמעט כל התעבורה עוברת ב udp (90.8%), כמו כן חלק קטן מאוד עובר ב tcp (3%) .

משמעות : זום מעדיף מהירות גבוהה ומינימום השהיות מאשר אמינות במידע. בעזרת שימוש ב udp זום יודע להתמודד היטב עם אובדן של חבילות ועדיין לשמור על מינימום השהיות, לעומת spotify שתוודא שיש לה את כל המידע בbuffer ורק אחרי זה יהיה ניתן לנגן את השיר.

מסקנות:

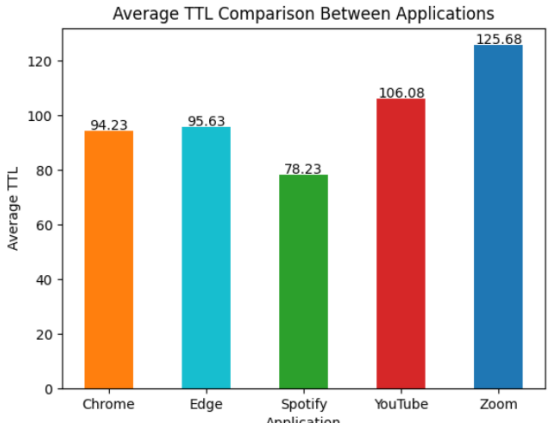
א. ניתן לראות כי הדפדפנים edge ו- chrome משתמשים בפרוטוקול quic אך chrome משתמש בו הרבה יותר לעומת edge .

ב. Youtubeמשתמש בפרוטוקול quic ב77% מהמקרים לעומת spotify שמשתמש בעיקר בtcp מה שמעיד על כך ש youtube מעדיף את מהירות העברת המידע ויתגבר על בעיית האמינות של udp בעזרת quic (שמסתדר בקלות כי quic שייכת לגוגל וגם יוטיוב) לעומת זאת spotify לא מתפשר על איכות השירים ומתוך כך משתמש ב tcp בעיקר (81%) .

ג. זום משתמש בעיקר בudp (90%) לעומת spotify שמשתמש בעיקר ב tcp (81%) השוואה זו מראה על בחירה של זום לביצוע שיחת הוידאו במינימום השהיות לעומת spotify שיש לו את הפריבילגיה להוריד את הbuffer שצפוי להגיע מכיוון שזו הקלטה של שיר ולא זמן אמת.

בס"ד

328144951_325202158



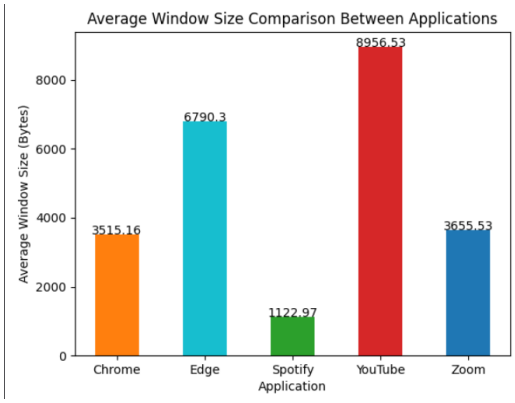
ממוצע TTL :

ניתן לראות כי ממוצע הttl של זום הוא הגבוה ביותר, משום שזום זו אפליקציה שמבצעת שיחות וידאו בכל רחבי העולם הגיוני שערך הttl ההתחלתי שלה יהיה יחסית גבוה מכיוון שחבילות בה עוברות מספר מרובה של נתבים ולא נרצה שיזרקו בדרך. כמו כן האפליקציה בעלת ה ttl הנמוך ביותר היא spotify מה שיכול להצביע על כל שהאפליקציה מצליחה למצוא מסלולים קצרים שלא דורשיttl גבוה.

ניתוח לפי TCP Header (B) :

בחלק זה הפקנו גרף שמכיל את גדלי החלונות הממוצעים בתעבורה של כל אפליקציה, לאחר מכן נציג טבלה שמסבירה באילו פורטים עיקריים כל אפליקציה משתמשת.

Window size :



Youtube : יוטיוב משתמשת בגודל חלון גדול במיוחד (בממוצע כ 9000 בתים) . גודל חלון זה מאפשר שליחת של המון מידע ללא צורך בהמתנה ל ack מה מאפשר זרימת וידאו חלקה ומהירה. נשים לב גם כי אפשרות זו מתחברת עם אפליקציה של יוטיוב שדורשת העברת מידע רב בקצב מהיר מאוד.

Spotify : אפליקציה זו בעלת גודל החלון הקטן ביותר (רק 1122 בתים בממוצע). ברור כי העברת אודיו דורשת הרבה פחות רוחב פס מאשר וידאו ולכן גודל החלון הנמוך יחסית אל יוטיוב. בנוסף גודל זה יכול להצביע על אסטרטגיה של spotify בניהול התעבורה, עצם הקטנת גודל החלון דורש המתנה לack על חבילות שלא הגיעו ובפועל מספק אמינות גבוהה ושליטה בעומס ברשת.

Chrome : אפליקציה זו הינה דפדפן ולכן צריכה לאזן בין זרימת מידע מהירה לבין מיתון העומס ברשת ולכן גודל החלון שלה בינוני יחסית ליוטיוב ו spotify. רוב התקשורת בchrome מתבססת על פרוטוקול quic שיודע לבצע אופטימיזציה למידע המועבר מה שלא דורש חלון גדול במיוחד בשביל מידע רב.

Edge – ניתן לראות כי רוחב החלון של edge יחסית גבוה למרות שהוא דפדפן בדיוק כמו chrome הוא עובד בצורה שונה. בתרשים העוגה הראשון ניתן לשים לב כי חלק גדול מהתעבורה של edge (כ 50%) עובר בtcp ולא ב quic ועובדה זו ביחד עם שילוב העובדה שדפדפן דורש מידע רב ממקורות שונים מהר מאוד מאלץ את הדפדפן להשתמש בגודל חלון יחסית גדול. סביר להניח שבמידה ו edge יאמץ יותר בquic (או ישתמש בתווך שתומך בquic) כל גם גודל החלון ירד ויהיה זהה אל החלון של chrome .

Zoom : גודל החלון של זום קרוב יותר אל chrome מאשר אל יוטיוב מתוך כך שרוב מוחלט של התעבורה בזום עוברת בudp ולכן אין לו שימוש רב במנגנון של tcp לכן גודל החלון שלו ישמו על איזון בין עומס ברשת למידע מהיר בדומה לדפדפנים.

בס"ד

206847584_322772880_325612695_328144951_325202158

מסקנות:

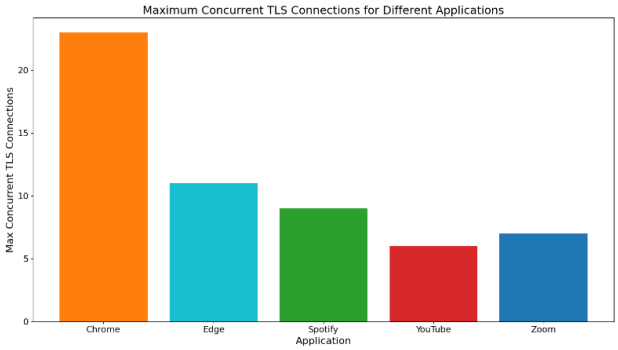
א. כדי לאפשר לאפליקציה כמו יוטיוב שמשתמשת לעיתים בעברת המידע שלה על tcp להעביר מידע רב נאלץ להגדיל את גודל החלון, כך נמנע השהיות.
ב. Youtube לעומת zoom משתמשת ב quic כשניתן ועוברת ל tcp כאשר לא ניתן, מכיוון שהתעבורה ביוטיוב דורשת העברת מידע רב ב tcp יהיה לו רוחב פס גדול. לעומת זאת זום משתמש באופן קבוע ב udp שלא דורש כמעט בכלל שימוש בגודל חלון לכן יהיה לו גודל סטנדרטי.
ג. Spotify אמנם משתמש לרוב בtcp אך גודל החבילות של אודיו קטן בהרבה מווידאו ולכן מספיק גודל החלון מינימלי כדי להזרים מידע זה. בנוסף לכל spotify מעדיפה אמינות על פני מהירות ומתוך כך המעיטה את גודל החלון אף יותר מהממוצע של הדפדפנים.

טבלת חלוקה של שימוש עיקרי בפורטים לכל אפליקציה: (כמות חבילות בפורט, סוננו רק פורטים שעברו דרכם יותר מ 1.2% מהתעבורה)

App/port	Chrome	Edge	Youtube	Spotify	Zoom
443	(2500) 42.4%	(1200) 48%	(400) 12.1%	(6500) 80.9%	(250) 3.1%
80	-	-	(300) 9%	-	-
53	-	-	-	-	-
סך כל החבילות	5892	2500	3299	8030	7820

ניתוח לפי TLS Header (C) :

בחלק זה נציג גרף שישווה בין כמות החיבורים המקביליים שיש בכל אפליקציה, כמו כן נשווה בין הגרסאות השונות של tls בכל אפליקציה.



חיבור מקביליים:

Chrome – Edge : אפליקציות אלה הם בעלות הכי הרבה חיבורי tls מקביליים (כ – 24 לכרום ו10 ל edge) . כמות זו היא הגיונית מכיוון שדפדפנים נדרשים לנהל מספר רב של חלוניות ומכך מספר רב של חיבורים ממקורות שונים. כמו כן כאשר נבצע חיפוש מסוים בדפדפן נטענים המון תמונות סרטונים או מודעות שדורשות חיבורי tls שונים. בכל זאת קיים הבדל גדול בין edge ל chrome וזה נובע מפרוטוקול התקשורת עליו הם עובדים, כאשר edge בעיקר משתמש ב tcp ו tls מסורתיים שעבורם מספיקים לו 10 חיבורים מקביליים. בעוד ש chrome משתמש בעיקר ב quic שמאפשר שימוש חוזר בחיבור יחיד עבור מספר בקשות ,(multiplexing) מה שמפחית את הצורך לפתוח חיבורי TLS חדשים עבור כל משאב. עם זאת, אם Chrome פותח חיבורים נוספים לשרתים שאינם תומכים ב quic ומשתמשים ב tls-מעל tcp הוא עשוי לפצות על כך על ידי הגדלת מספר החיבורים המקסימליים כדי לשפר ביצועים.

Spotify : אפליקציה זו מנהלת לכל היותר 10 חיבורים מקביליים מה שמצביע על העובדה שspotify לא דרושה בניהול מספר רב של תעבורה ממקורות שונים כמו דפדפן, אלא הורדת רצף השיר ממקורות בודדים. עם זאת spotify דואגת להזרמת המידע גם עבור השירים הבאים ברשימת ההשמעה ועדכונים שונים על רשימות השמעה שונות וכו' לכן יש לאפליקציה זו מעט יותר חיבורים מ – youtube .

Youtube : לאפליקציה זו יש הכי מעט חיבורי tls מקביליים (כ -5) , זאת מכיוון שהתעבורה ביוטיוב מתמקדת בעיקר בעברת סרטון בודד על ערוץ tls אחד או מספר ערוצים בודדים. בנוסף לכך אף את התמונות של הסרטונים הבאים יוטיוב מעדיפה להעביר על אותו ערוץ tls ובכך למטב את החיבור באופן מירבי. מכל הנ"ל מגיע מספר חיבורים מינימלי.

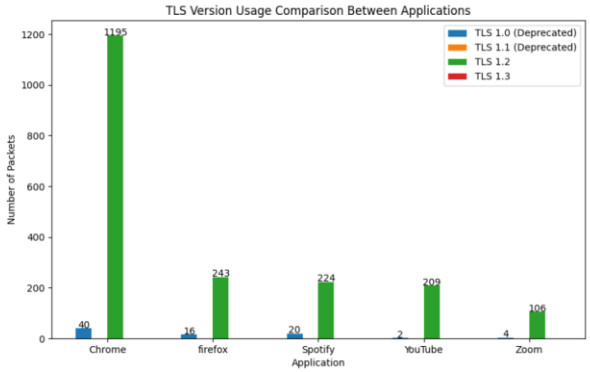
Zoom : אפליקציה זו מנהלת לכל היותר 5 חיבורי tls , זאת מכיוון שזום מעביר בעיקר 2 סוגים של מידע אודיו ווידאו ממקור יחיד, שלא כמו דפדפן שדורש מספר רב של חיבורים ממקורות שונים. אין צורך בחיבורים מקביליים נוספים למעט חיבור עבור שיתופי מסך וכו'.

בס"ד

206847584_322772880_325612695_328144951_325202158

פילוג לפי גרסאות tls :

ניתן לראות כי רוב מוחלט של התעבורה עובר תחת גרסה של tls 1.2 וזאת למרות שבהקלטות ניתן לראות שינה תמיכה מצד הלקוח ב1.3 אך השרת בוחר בכל זאת להשתמש ב 1.2 על מנת להתאים לשרתים ישנים שלא תומכים ב1.3. כמו כן הhost שביצע את ההקלטה יכול להיות שלא תמך ב1.3 .

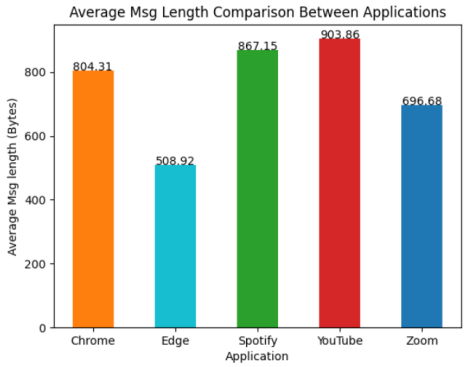


פילוג לפי גודל חבילה – Packet sizes (D):

בחלק זה נציג 2 גרפים המכילים את גודל החבילה הממוצע בכל אפליקציה ואת גודל החבילה המקסימלית.

גודל חבילה ממוצע:

בגרף זה ניתן לראות כי youtube בממוצע שולח את ההודעת הגדולות ביותר (903 בתים) לעומת spotify ששולח חבילות של אודיו בהם מספיק בממוצע 867 בתים לחבילה.



כמו כן ניתן לראות כי זום שולח את החבילות עם מעט בתים בממוצע (696 בלבד) למרות שהוא מעביר וידאו וזאת מכיוון שהוא מבצע דחיסה חזקה.

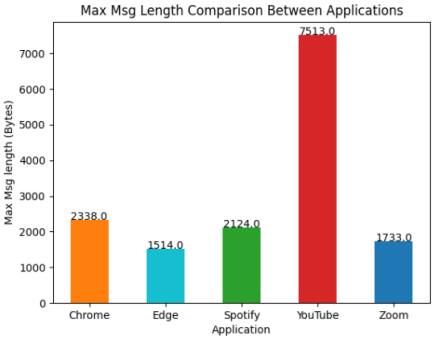
chrome מטפל בתעבורה מגוונת הכוללת טעינה של דפים סרטונים תמונות ועוד', לכן גודל הודעה ממוצע יחסית גבוה.

Edge גם כן דפדפן, מראה אורך הודעה ממוצע נמוך יותר כמעט מחצית מזה של Chrome זה עשוי להצביע על כך שהתעבורה שלו כוללת יותר בקשות קטנות (כגון טעינת דפים פשוטים או משאבים קלים) או ש Edge- מפצל את הנתונים לחבילות קטנות יותר, אולי כחלק מאסטרטגיה שמרנית יותר בניהול תעבורה.

גודל חבילה מקסימלית:

Youtube : שולח את החבילות הגדולות ביותר (7513 בתים) , נובע מכך שהתעבורה שלו כוללת סרטוני וידאו באיכות גבוהה מאוד ולכן דורש חבילות גדולות.

Spotify : לעומת יוטיוב החבילה המקסימלית בתעבורה זו בגודל 2124 בתים בלבד, זאת מכיוון שspotify מעביר שירותי אודיו בלבד ולא נדרש בחבילות גדולות כמו יוטיוב.



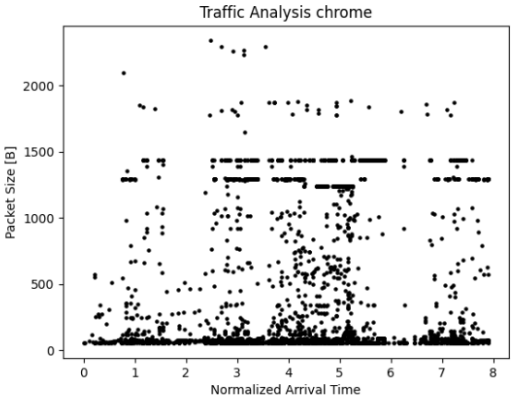
Chrome – Edge : אפליקציות אלו בעלות סוג תעבורה דומה, החבילה המקסימלית קטנה יותר ב edge ככל הנראה מאופטימיזציה שנעשתה או פשוט ממידע אחר שהתבקש.

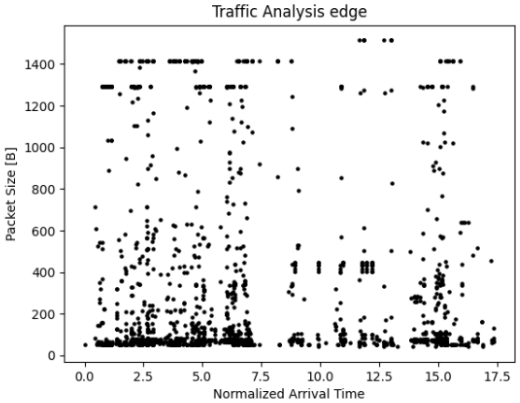
Zoom – ביחס לspotify ויוטיוב שולח את החבילות הקטנות ביותר, זאת מכיוון שזום עובד בudp ומתעדף תעבורה מהירה משום שזהו וידאו בזמן אמת. לכן זום מחלק את התעבורה לחבילות קטנות ותורף לשידור מהיר ורציף.

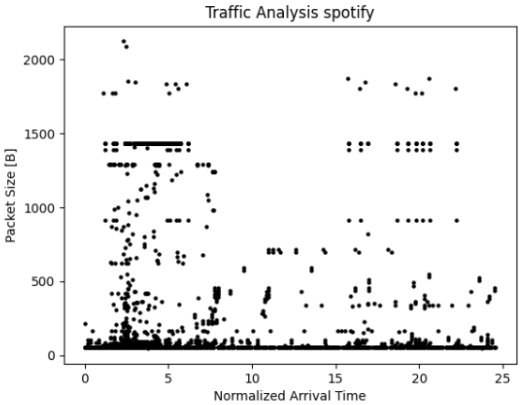
בס"ד

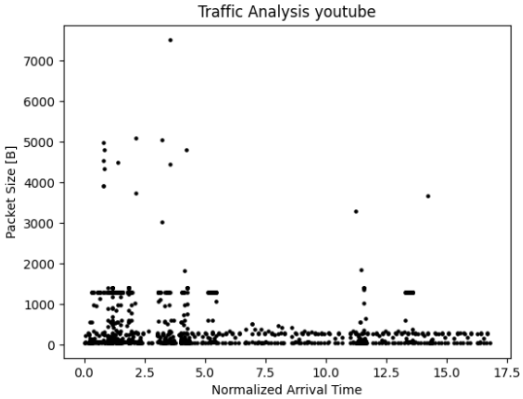
206847584_322772880_325612695_328144951_325202158

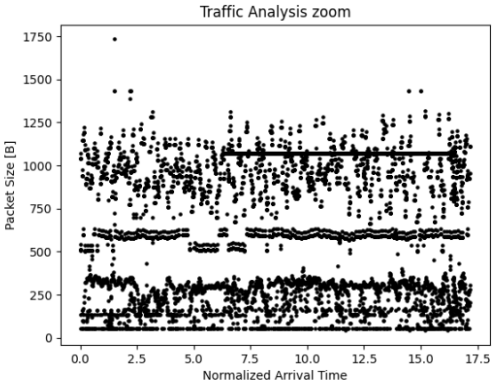
<u>Flow pic – גודל החבילה ביחס של זמן</u> : בהמשך לגרפים המתארים את גדלי החבילות נציג גרפים שמציגים את גדלי החבילות ביחס של זמן, בדומה גרפים שהוצגו במאמר "FlowPic_Encrypted_Internet_Traffic_Classification_is_as_Easy_as_Image_Recognition" גם אנחנו יצרנו גרפים דומים :











בגרפים הנ"ל ניתן ממש לזהות בעיניים הבדלים בזרימת התעבורה, ניתן לראות כי בתעבורה של זום יש חבילות בצפיפות רבה בגודל בסביבות 600 בתים.

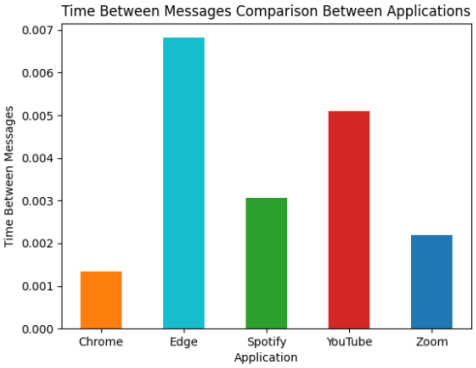
כמו כן ביוטיוב התעבורה פחות מפוזרת בגדלים מזום ויותר אחידה, ניתן לראות שישנם חבילות גדולות מאוד בתחילת התקשורת ולאחר מכן זה ירד.

ניתן לראות כי הגרפים של edge ו chrome יחסית דומים משום אופיים הדומה.

בס"ד

206847584_322772880_325612695_328144951_325202158

זמן ממוצע בין 2 חבילות - Packets inter-arrivals (E):



Edge – בעל זמן ההמתנה הארוכים ביותר בין חבילות (0.007 שניות), ככל הנראה בגלל אופטימיזציה שמתרחשת מאחורי הקלעים.

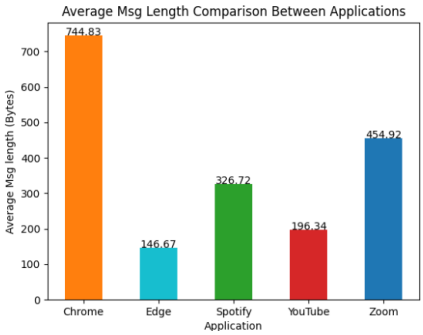
Youtube – ליוטיוב יש זמני המתנה ארוכים יחסים וזה ככל הנראה נובע מהעובדה שהחבילות שמגיעות גדולות מאוד ובמידה וצריך לאחזר אותם לוקח לזה זמן, כמו כן גודל החלון של יוטיוב גדול מאוד אז זיהוי החבילה שנפלה גם בפני עצמו ייקח יותר זמן מאפליקציות אחרות.

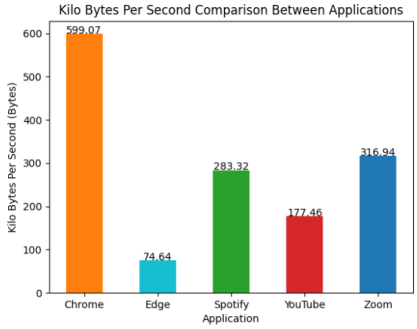
Spotify – נשים לב שהיינו מצפים שההמתנה של spotify תהיה אף קטנה יותר אך משום שעובד על tcp הוא דורש אמינות ואיכות ולכן "משלם" על זה כאן.

Zoom – יחסית נמוך להעברת וידאו וזה נובע מחלוקת המידע לחבילות קטנות ושימוש בudp שלא דורש המתנה לאישור.

Chrome – בעל זמן ההמתנה הנמוך ביותר, נובע ישירות משימוש הנרחב בפרוטוקול udp שמאפשר שימוש ממוקבל באותו תווך תקשורת ועל אותו חיבור tls .

מספר חבילות ממוצע לשנייה - Flow size (F) + גודל חבילות ממוצע לשנייה - Flow volume(G):





Chrome – צפיפות התעבורה ב Chrome-גבוהה מאוד, בזכות השימוש ב-QUIC, מה שמאפשר לו להעביר מידע רב בשנייה (744.83 חבילות לשנייה), בפער גדול לעומת Edge ושאר האפליקציות

Edge - הנתונים מן הגרף מצביעים כי בממוצע edge מעביר 146.67 חבילות לשנייה, נמוך מאוד יחסים לchrome וזאת משום שהוא משתמש ב tcp ונאלץ לבצע בדיקות אמינות בכל רוחב חלון. אימוץ של quic יעצים משמעותית את קצב השליחה.

Zoom – ובהתחשב בגרף ניתן לראות כי zoom מעביר 454.92 חבילות בשנייה וזאת משום הרצון לתרום לשידור רציף של מידע ללא השהיות, זום מחלק את המידע לחבילות יחסית קטנות ואחידות לטובת שידור רציף.

Youtube – בגרף של יוטיוב ניתן לראות כי הוא מעביר 196.34 חבילות בשנייה בממוצע. אמנם היינו מצפים לראות את יוטיוב עם הממוצע הגבוה ביותר משום חוסר התפשרות על איכות והעברת תעבורת וידאו אך בכל זאת הממוצע שלו נמוך מאוד (חצי מזום והרבה פחות מ spotify). זאת מכיוון שהתעבורה ביוטיוב משתמשת במנגנון שנקרא dash שבו הסרטון מחולק למקטעים קטנים אך חלק מן החבילות יכולות להכיל מידע רב.

Spotify – עולה מן הגרף כי spotify מעביר 326.72 חבילות לשנייה. Spotify מעביר איכות גבוהה של אודיו בתעבורת tcp בקצב אחיד, הוא לא מבצע חלוקה משתנה של המידע כמו youtube אך גם לא מתפשר על איכות בשביל רציפות כמו zoom , אלא מבצע חלוקה מאוזנת של הזרמת מידע אמין.

בס"ד

206847584_322772880_325612695_328144951_325202158

חלק 4 – הסבר קוד תוקף

חלק א – ניתן להשתמש רק בflowID , בגדלי החבילות וחותמת הזמן.

ספריות שהשתמשו בהם :

```
import pyshark
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
import nest_asyncio
nest_asyncio.apply()
```

הסבר כללי על הקוד:

בקוד זה נעזרנו במכונת למידה KNN על מנת לחזות שימוש באפליקציה מסוימת, ביצענו אימון על הקלטות ממידע ידוע (125 הקלטות) וניסינו לחזות הקלטות חדשות. תוצאות החיזוי לא רעות כלל 87.49% למרות שהאימון הוא על מעט הקלטות.

תוצאות אלו קיבלנו בממוצע עבור 10,000 בדיקות דיוק, על 25 חבילות pcap לכל אפליקציה (125 הקלטות סה"כ) בחלוקה של 80% אימון ו 20% בדיקה.

תהליך החיזוי כלל 2 שלבים האחד בניית הdata והשני אימון ובדיקת דיוק :

בניית הdata עצמו גם כלל מספר שלבים :

1. חילוץ המידע מההקלטה
2. מהמידע על התעבורה חילוץ פרמטרים נוספים כמו גודל ממוצע והפרשי זמנים
3. הוספת המידע לקובץ הcsv ויצירת המודל על פי המידע המעודכן

תהליך אימון ובדיקת דיוק :

1. יצירת המודל והזנת המידע מקובץ CSV ואימון המודל בעזרתם.
2. חיזוי התעבורה (כולל את חילוץ המידע ופרמטרים נוספים ולאחר מכן שימוש במודל לחיזוי)

```
i is: 0
i is: 1000
i is: 2000
i is: 3000
i is: 4000
i is: 5000
i is: 6000
i is: 7000
i is: 8000
i is: 9000
avg accuracy: 87.49%
```

תוצאות הרצת 10,000 פעמים בדיקת דיוק

בס"ד

206847584_322772880_325612695_328144951_325202158

2 חלקי המודל נעזרים באותם פונקציות לחילוץ המידע ופרמטרים נוספים :

<u>פונקציה א'</u> :

בעזרת פונקציה זו נחלץ את המידע מקובץ pcap נתון.

```python
def extract_features_from_pcap(pcap_file):  2 usages
    cap = pyshark.FileCapture(pcap_file) #Using pyshark function extracting all pcap packets
    flows = {}
    for packet in cap:
        try:
            # Extract source and dest ip
            src_ip = packet.ip.src
            dst_ip = packet.ip.dst
            src_port = packet[packet.transport_layer].srcport
            dst_port = packet[packet.transport_layer].dstport
            # Create flow id
            flow_id = f"{src_ip}:{src_port}-{dst_ip}:{dst_port}"
            # Extract packet size
            packet_size = int(packet.length)
            # Extract time stamp
            timestamp = float(packet.sniff_time.timestamp())
            # create list of dictionaries that contains flow id's
            if flow_id not in flows:
                flows[flow_id] = {'sizes': [], 'timestamps': [], 'src_ip': src_ip, 'dst_ip': dst_ip}
            flows[flow_id]['sizes'].append(packet_size)
            flows[flow_id]['timestamps'].append(timestamp)
        except AttributeError:
            continue
    top_flows = sorted(flows.items(), key=lambda x: len(x[1]['sizes']), reverse=True)[:5]
    # create one dictionary for top 5 flows
    merged_flows = {
        'sizes': [],
        'timestamps': [],
    }
```

```python
#create merged flow that contains all sizes and time stamps
for _, data in top_flows:
    merged_flows['sizes'].extend(data['sizes'])
    merged_flows['timestamps'].extend(data['timestamps'])
return merged_flows
```

נחלץ את כלל החבילות אל תוך cap שיכיל אותם כרשימה, ועבור כל חבילה ב cap נוציא את המאפיינים הבאים : Ip מקור ויעד , פורט מקור ויעד, גודל החבילה והזמן שלה. ניצור מחרוזת שכולל את tuple 4 שהתבקשנו.

לאחר מכן ניצור רשימה בשם flows שתכיל מילון , אותו מילון יכיל עבור כל flow ספציפי : רשימה של כל גדלי החבילות עם אותו flowid , רשימה של כל חותמות הזמן עם אותו flowid , את ip מקור ויעד.

מהרשימה flows נמיין הפוך לפי גודל הרשימה size של כל אחד (כלומר flow הראשון יהיה זה בעל המספר הגדול ביותר של גדלי חבילות זהים), לאחר המיון ניקח את החמישה הראשונים בלבד.

נאחד את 5 הרשימות של top_flow הממוינות אל רשימה אחת מוארכת עם שדות גודל וחותמת זמן ואותה נחזיר.

בס"ד

206847584_322772880_325612695_328144951_325202158

<u>פונקציה ב'</u>:

פונקציה לחילוץ פרמטרים נוספים

```python
'''
    Computes two key features for each flow:
    1. Average packet size.
    2. Average inter-arrival time between packets.
    This function is used for both training and prediction.
'''
def generate_features_from_one_flow(flows):  2 usages
    features = [] #will contain avg_packets_size and inter_arrival_time
    labels = []
    sizes = flows['sizes']
    timestamps = flows['timestamps']
    timestamps = sorted(timestamps)

    avg_packet_size = np.mean(sizes) # avg of top 5 sizes

    inter_arrival_times = np.diff(timestamps)
    avg_inter_arrival_time = np.mean(inter_arrival_times) if len(inter_arrival_times) > 0 else 0 #culc the avg of inter arrival time
    num_packets = len(sizes)
    label = 'edge' #tempaorary set a label for inserting new data
    features.append([avg_packet_size, avg_inter_arrival_time])
    labels.append(label)

    return features, labels
```

פונקציה זו תקבל את הרשימה flows (שזו בעצם הרשימה המאוחדת של top_5 שהשגנו בפונקציה הקודמת) ותחזיר רשימה של מאפיינים ותווית לכל רשימה.

תחילה נפצל את הרשימה לחלקים בחלוקה ל size ו timestamps .

נוציא ממוצע של גדלי החבילות, נחשב את סכום הפרשי הזמנים בין כל זוג חבילות ונוציא ממוצע שלו.

נסמן את label כedge כאשר נוסיף מידע אל המאגר להקלטה של edge ונשנה בהתאמה (כאשר אנחנו מבצעים חיזוי נתעלם מנתון זה).

נחזיר את features ואת label עבור ה flow המתקבל כקלט.

בס"ד

206847584_322772880_325612695_328144951_325202158

פונקציה ג׳ :

פונקציה שמוסיפה data לקובץ CSV

```
'''
    Reads a PCAP file, extracts network features, and updates the dataset.
    Trains a K-Nearest Neighbors (KNN) model using the updated data and prints accuracy.
'''
def analyze_traffic(pcap_file, output_csv):
    flows = extract_features_from_pcap(pcap_file)
    features, labels = generate_features_from_one_flow(flows)
    df = pd.read_csv(output_csv) #read the old database
    new_df = pd.DataFrame(features, columns=['avg_packet_size', 'avg_inter_arrival_time']) # create new df with the features
    new_df['label'] = labels # insert the labels
    df = pd.concat( objs: [df, new_df], ignore_index=True) # add the new df to the old one
    X = df[['avg_packet_size', 'avg_inter_arrival_time']] # extract to x avg_packet_size and avg_inter_arrival_time
    y = df['label'] # extract to y all labels

    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42) # split x and y to test and train
    scaler = StandardScaler() #normelized all values
    x_train_normal = scaler.fit_transform(X_train) # insert to x_train_normal the normal values of x_train
    x_test_normal = scaler.transform(X_test) # insert to x_test_normal the normal values of x_test
    model = KNeighborsClassifier(n_neighbors=1) #Create the KNN model
    model.fit(x_train_normal, y_train) #train the model
    y_pred = model.predict(x_test_normal) #get the prediction of the model after training
    accuracy = accuracy_score(y_test, y_pred) # check accuracy
    print(f"Model Accuracy: {accuracy * 100:.2f}%")

    for xt, yt, yp in zip(X_test.values, y_test.values, y_pred):
        print(f" ◆ stats: {xt} | real answer: {yt} | model answer: {yp}")

    df.to_csv(output_csv, index=False) #insert to csv
    return model, df
```

מטרת הפונקציה היא להוסיף מידע על תעבורה ידועה אל מאגר המידע , לאמן את המודל ולבצע בדיקת דיוק עם המידע החדש.

הפונקציה מקבלת קובץ pcap ו קובץ csv , ומכניסה אל flows את החילוץ של הפרטים שהסברנו בפונקציה א׳ ואל features ואל label את הפיצ׳רים המתקבלים מפונקציה ב׳.

נכניס אל df (dataframe) את כל המידע שכרגע נמצא בcsv כדי שנשתמש בו ונוסיף עליו במקום לדרוס.

ניצור new_df שיכיל בתוכו את המאפיינים של הflow החדש שחילצנו, ואת הlabel בהתאמה.

נאחד בין 2 ה dataframes בעזרת concat ונחלץ אל x את המידע המאוחד של features ואל y את המידע המאוחד של label.

נפצל את המידע אל אימון ובדיקה באופן רנדומלי כך שנשלח 80% לאימון ונבדוק על 20% את דיוק המודל.

נשים לב כי החל מרגע זה נשתמש בערכים מנורמלים על מנת לקבל תוצאות מתאימות בעזרת knn .

ניצור את המודל כך שיתבסס על שכן אחד, לאחר מכן נאמן אותו עם המאפיינים שהגדרנו לאימון. בשלב זה נקבל את החיזוי, נבחן את הדיוק ונדפיס.

KNN מקבל את הנתונים ומחשב בעזרתם את המרחק מהנתונים הקיימים, בחיזוי המודל יבדוק את השכן הקרוב ביותר ומי שיופיע הכי הרבה יבחר כחיזוי המתאים.

בס"ד

206847584_322772880_325612695_328144951_325202158

<u>פונקציה ד'</u>:

פונקציה היוצרת את המודל מקובץ csv נתון.

```
'''
This function create the model and use the known information from the csv.

'''
def make_modle(output_csv):  1 usage
    df = pd.read_csv(output_csv) #read the old database
    X = df[['avg_packet_size', 'avg_inter_arrival_time']] # create new df with the features
    y = df['label']  # insert the labels
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2) # add the new df to the old one
    scaler = StandardScaler() # add the new df to the old one
    x_train_normal = scaler.fit_transform(X_train)  # insert to x_train_normal the normal values of x_train
    x_test_normal = scaler.transform(X_test) # insert to x_test_normal the normal values of x_test
    model = KNeighborsClassifier(n_neighbors=1) #Create the KNN model
    model.fit(x_train_normal, y_train) #train the model
    y_pred = model.predict(x_test_normal) #get the prediction of the model after training
    accuracy = accuracy_score(y_test, y_pred) # check accuracy
    print(f"Model Accuracy: {accuracy * 100:.2f}%")
    for xt, yt, yp in zip(X_test.values, y_test.values, y_pred): #creates a tuple
        print(f" ◆ stats: {xt} | real answer: {yt} | model answer: {yp}")
    return model, df,scaler
```

מטרת פונקציה זו היא ליצור את המודל ולאמן אותו על קובץ csv נתון, לאחר מכן תבדוק נתוני חיזוי ותדפיס אותם. פונקציה זו תשמש לבניית המודל לחיזוי אמת של תעבורה.

תחילה נחלץ את המידע (features, label) מתוך קובץ csv אל data frame , נפריד בין החלק של המאפיינים ב x ואל החלק של label ב y .

נחלק את x ו y אל נתונים אימון ובדיקה ביחד של 20% לבדיקה ו80% לאימון, לאחר מכן ננרמל את הנתונים לטובת עבודה תקינה עם KNN .

ניצור מודל KNN עם שכן אחד ונאמן אותו עם הנתונים המנורמלים שחילקנו לבדיקה.

נקבל את החיזוי עבור החלק שנותר לחיזוי ונבדוק דיוק, לאחר מכן נדפיס את התוצאות.

נחזיר את המודל המאומן, את הdataframe עם כל המידע, ואת ה scalar המנרמל.

בס"ד

206847584_322772880_325612695_328144951_325202158

<u>פונקציה ה</u> :

פונקציה לביצוע החיזוי.

```
'''
This function get as input a model (KNN) , file to predict and scalar for normalized,
 and predict the application that being used in the pcap file.
'''
def print_prediction(model, predict_file,scaler):  1 usage
    flows = extract_features_from_pcap(predict_file)
    features, labels = generate_features_from_one_flow(flows)
    X_predict = pd.DataFrame(features, columns=['avg_packet_size', 'avg_inter_arrival_time'])
    x_predict_normal = scaler.transform(X_predict)
    predictions = model.predict(x_predict_normal)
    print(f'my prediction: {predictions}',end='\t')
```

מטרת הפונקציה היא לבצע חיזוי של התעבורה כאשר היא מקבלת מודל (מאומן) , קובץ שצריכה לחזות את התעבורה שלו (pcap) וscalar מנרמל.

נחלץ אל flows את המידע שצריך לחלץ בעזרת פונק א׳ , נשלח את flows אל פונק ב׳ ונקבל אל features את המאפיינים המתאימים של התעבורה החדשה.

נכניס אל x_predict data frame שמכיל את הfeatures , לאחר מכן ננרמל את x_predict לטובת שימוש בKNN בצורה תקינה.

נפעיל את חיזוי המודל על x_predict_normal ונציג את התוצאה.

הערה : קיימות פונקציות נוספות בקוד שהן עזר ולכן לא צילמנו והסברנו עליהם.

בס"ד

206847584_322772880_325612695_328144951_325202158

חלק ב – ניתן להשתמש רק בגדלי החבילות ובחתימות הזמן.

הספריות שהשתמשנו בהם הם:

```
import pyshark
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
import nest_asyncio
nest_asyncio.apply()
```

הסבר כללי על הקוד:

עבור החלק השני נימנע מחילוץ של המידע שקשור לflowid ונתמקד אך ורק במידע של גדלי החבילות וחתימות הזמן. רעיון הזיהוי ייעזר בKNN , בדומה לפתרון הראשון נחלץ את המידע המתאים (אך ורק גודל וחותמת זמן) , ננתח את המידע. לאחר שניתחתנו נאמן את המודל עם מידע זה ונבדוק את הדיוק שלו.

תוצאות מודל זה יצאו כ- 70% הצלחה.

תוצאות אלו קיבלנו בממוצע עבור 10,000 בדיקות דיוק, על 25 חבילות pcap לכל אפליקציה (125 הקלטות סה"כ) בחלוקה של 80% אימון ו 20% בדיקה.

חשוב לציין, מודל זה נותן משקל רב לגודל החבילה והפרשי הזמנים בין החבילות, לכן שינוי כלשהו בנתונים אלה יפגע משמעותית באחוזי ההצלחה (בעצם זו הדרך לענות על השאלה שהוצגה בסעיף זה – נענה בסוף ההסבר)

פונקציה א – חילוץ המידע

```
def extract_features_from_pcap(pcap_file):  2 usages
    cap = pyshark.FileCapture(pcap_file)
    sizes = []
    timestamps=[]
    for packet in cap:
        try:
            packet_size = int(packet.length)
            timestamp = float(packet.sniff_time.timestamp())
            sizes.append(packet_size)
            timestamps.append(timestamp)
        except AttributeError:
            continue
    return sizes, timestamps
```

פונקציה זו מחלצת מכל קובץ pcap את גודל כל חבילה ואת חותמת הזמן שלה וכניסה אל רשימות בשם sizes ו timestamps ומחזירה ערכים אלו.

בס"ד

206847584_322772880_325612695_328144951_325202158

פונקציה ב – ניתוח המידע

```
def generate_features_from_sizes_and_times(sizes,timestamps):  2 usages
    features = []
    labels = []
    timestamps = sorted(timestamps)
    avg_packet_size = np.mean(sizes)
    inter_arrival_times = np.diff(timestamps)
    avg_inter_arrival_time = np.mean(inter_arrival_times) if len(inter_arrival_times) > 0 else 0
    features.append([avg_packet_size, avg_inter_arrival_time])
    label = 'youtube'
    labels.append(label)
    return features,labels
```

פונקציה זו מחשבת את ממוצע גדלי החבילות שהתקבלו ואת ממוצע הפרשי הזמנים שהתקבלו, נעזר בפונקציה זו גם כדי לאמן את המודל וגם בשביל לנתח תעבורת אמת.

כאשר נאמן את המודל נתייחס אל תווית label ונשנה אותה בהתאמה לתעבורה הידועה וכאשר נרצה בדיקה לpcap לא ידוע פשוט נתעלם מתווית זו כי לא תהייה רלוונטית.

פונקציה ג – הוספת המידע אל קובץ הCSV , אימון המודל ובדיקת דיוק על המידע החדש.

```
def analyze_traffic(pcap_file, output_csv):
    sizes,timestamps = extract_features_from_pcap(pcap_file)
    features,labels = generate_features_from_sizes_and_times(sizes,timestamps)
    df = pd.read_csv(output_csv)
    new_df = pd.DataFrame(features, columns=['avg_packet_size', 'avg_inter_arrival_time'])
    new_df['label'] = labels
    df = pd.concat( objs: [df, new_df], ignore_index=True)
    X = df[['avg_packet_size', 'avg_inter_arrival_time']]
    y = df['label']
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2)
    scaler = StandardScaler()
    x_train_normal = scaler.fit_transform(X_train)
    x_test_normal = scaler.transform(X_test)
    model = KNeighborsClassifier(n_neighbors=1)
    model.fit(x_train_normal, y_train)
    y_pred = model.predict(x_test_normal)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model Accuracy: {accuracy * 100:.2f}%")
    for xt, yt, yp in zip(X_test.values, y_test.values, y_pred):
        print(f" ◆ stats: {xt} | real answer: {yt} | model answer: {yp}")
    df.to_csv(output_csv, index=False)
    return model, df
```

מטרת פונקציה זו היא לאמן את המודל על קובץ pcap שהתעבורה שלו ידועה ולהכניס את המידע אל קובץ CSV לטובת חיזוי עתידי.

נעזר בפונקציה א' ופונקציה ב' לטובת חילוץ המידע הנחוץ וניתוח שלו כך שלאחר סיום הריצה של פונקציות אלו נקבל במשתנה features את ממוצע גדלי החבילות וממוצע הפרשי הזמנים עבור הpcap הנתון.

נקרא את הנתונים הישנים מקובץ הcsv ונאחד אותם אל data frame אחד, לאחר מכן נפצל את המידע אל label במשתנה Y ו features במשתנה X . נחלק את הנתונים ל 80% אימון ו 20% בדיקה לצורך בדיקת דיוק המודל.

נשים לב כי חייב לנרמל את הנתונים משום שמשתמשים ב KNN , ניצור את המודל כך שיסתכל רק על שכן אחד ונאמן אותו עם הנתונים לאימון. לאחר מכן נבצע חיזוי ל20% שהקצנו ונבדוק דיוק עבורם. את כל המידע שצברנו נכניס חזרה אל קובץ CSV לטובת המשך אימון וחיזוי עתידי.

בס"ד

206847584_322772880_325612695_328144951_325202158

פונקציה ד – יצירת המודל על הנתונים הקיימים ובדיקת דיוק

```
def make_modle(output_csv):
    df = pd.read_csv(output_csv)
    X = df[['avg_packet_size', 'avg_inter_arrival_time']]
    y = df['label']
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2)
    scaler = StandardScaler()
    x_train_normal = scaler.fit_transform(X_train)
    x_test_normal = scaler.transform(X_test)
    model = KNeighborsClassifier(n_neighbors=1)
    model.fit(x_train_normal, y_train)
    y_pred = model.predict(x_test_normal)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model Accuracy: {accuracy * 100:.2f}%")
    for xt, yt, yp in zip(X_test.values, y_test.values, y_pred):
        print(f" ◆ stats: {xt} | real answer: {yt} | model answer: {yp}")
    return model,scaler
```

פונקציה זו יוצרת את המודל מקובץ CSV נתון, מאמנת אותו על 80% מה מהמידע ובודקת דיוק על 20% .

הפונקציה מחזירה את המודל המאומן עם הסקלר לנרמול.

פונקציה ה – ביצוע חיזוי

```
def take_predict(model, predict_file,scaler):
    sizes, timestamps = extract_features_from_pcap(predict_file)
    features, labels = generate_features_from_sizes_and_times(sizes, timestamps)
    X_predict = pd.DataFrame(features, columns=['avg_packet_size', 'avg_inter_arrival_time'])
    x_predict_normal = scaler.transform(X_predict)
    predictions = model.predict(x_predict_normal)
    return predictions[0]
```

פונקציה זו מבצעת את החיזוי עצמו על קובץ pcap שתעבורתו לא ידועה מראש.

נעזר בפונקציות א׳ וב׳ לחילוץ וניתוח המידע כך שנקבל ב features את הגודל הממוצע לחבילה ואת ממוצע הפרשי הזמנים . נמיר מידע זה אל data frame וננרמל אותו.

לאחר מכן נשתמש במודל המאומן על הנתונים מפונקציה ד׳ ונבצע חיזוי עבור המידע הנתון מקובץ הpcap.

הערה : קיימות פונקציות נוספות בקוד שהן עזר ולכן לא צילמנו והסברנו עליהם.

בס"ד

206847584_322772880_325612695_328144951_325202158

מדוע התוקף יכול (באופן חלקי) לזהות את אפליקציה?

בחלק זה בקוד של התוקף הוכחנו כי במידה מסוימת התוקף יכול לזהות את התעבורה ברשת, זאת בעזרת ניתוח של גדלי החבילות והפרשי הזמנים בלבד. כמו כן הוכח במאמר flowpic כי ניתן לזהות תעבורה בעזרת מאפיינים אלו במעל מ90% .אצלנו בקוד נשלח את המידע אל KNN ואחרי אימון המודל ילמד את התעבורה וידע לזהות אותה בכ-70%.

מכיוון שיש ברשותנו פחות מידע אז אחוזי ההצלחה ירדו, במאמר flowpic נעזרו בשיטה חדשנית עם CNN והמרת המידע אל תמונה ובכך הצליחו לזהות בשיעור הצלחה הרבה יותר גדול.

כיצד ניתן לצמצם הצלחה זו?

כדי לענות על שאלה זו נעזר במידע שצברנו במטלה זו, נבחן את תוצאות מאמר flow pic .

במאמר זה ניתחו תעבורה בעזרת גדלי חבילות וחותמות זמן והעבירו מידע זה אל גרפים דו ממדיים לטובת שימוש ברשת נוירונים בזיהוי התמונה. במאמר התוצאות הראו ירידה בזיהוי בnon-vpn לעומת vpn ואף ירידה משמעותית יותר בשימוש בtor . וזאת מכיוון שבתהליך ההצפנה בtor מתבצע padding שמשנה את גודל החבילות לגודל אחיד, כמו כן גם מתערבבים מספר זרמי תעבורה בנתיב יחיד ומשתי סיבות אלו קשה מאוד למודל ליצור תבנית אחידה לכל אפליקציה.

לכן על מנת לצמצם את הצלחת התוקף בזיהוי האפליקציה נשתמש ברשת האנונימית tor או במנגנון דומה לשינוי גודל החבילה.

<u>בונוס:</u>

בעזרת למידת המכונה, המודל KNN שלנו יודע גם לזהות תעבורה של אפליקציות משלובות. ביצענו הקלטה של הרצת שיר בspotify ובמקביל ביצענו שליחת 3 מיילים דרך gmail .

ביצענו את הבונוס עבור 2 חלקי הקוד (כולל flowid וללא) וקיבלנו תוצאת חיזוי נכונה – SPOTIFY .

שם הקובץ לבדיקה הוא spotify_and_gmail.pcap והוא מצורף.

```
C:\Users\itama\AppData\Local\Programs\Python\Python39\python.exe C:\Users\itama\Downloads\Final_project_communicatio_networks-master\attack_part_b.py
Enter the number that you want
    1 for testing model accuracy
    2 for testing bonus prediction
    3 for testing prediction
2
if you want your pcap enter 1 else enter any other number
2
my prediction is spotify

Process finished with exit code 0
```

```
def print_Bonus_prediction(pcap_file,data_file_name):  1 usage
    model,df,scaler = make_modle(data_file_name)
    print_prediction(model,pcap_file,scaler)
```