רות גולדברג / 322631235
בנימין טיבי / 208434290

פרויקט סיום - רשתות תקשורת

הרצת התוכנית:

יש לפתוח 5 טרמינלים מתוך תיקיית הפרויקט ולהפעיל קודם את השרתים ואז את הלקוח בסדר הבא:

```
sudo python3 DHCP.py
sudo python3 DNS.py
sudo python3 BRApp.py
sudo python3 RDServer.py
sudo python3 client.py
```

שימו לב! אצל הלקוח תצטרכו להכניס מס' בין 1 ל6 במהלך הריצה, יש להחזיק את קבצי הPDF הנתונים בתוך תקיית הפרויקט בהרצה, רצוי לכבות את שרת הDHCP של המחשב.

רות גולדברג / 322631235
בנימין טיבי / 208434290

DHCP

את שרת ה DHCP יצרנו בעזרת ספריית SCAPY של PYTHON בעזרתה הסנפנו פקטות DHCP וידענו כיצד לטפל בכל סוג פקטה.

1	0.0000000…	0.0.0.0	255.255.255.255	DHCP	286	DHCP Discover - Transaction ID 0x0
2	1.0239111…	192.168.1.190	255.255.255.255	DHCP	292	DHCP Offer - Transaction ID 0x0
3	2.5523660…	0.0.0.0	255.255.255.255	DHCP	286	DHCP Request - Transaction ID 0x0
4	3.5763557…	192.168.1.190	255.255.255.255	DHCP	292	DHCP ACK - Transaction ID 0x0
5	3.7778543…	192.168.1.1	255.255.255.255	DHCP	342	DHCP NAK - Transaction ID 0x0

בצילום המסך ניתן לראות כי מתבצעת שליחה של חבילת discover מהלקוח לכלל השרתים (broadcast), בקוד שלנו אנחנו מסניפים פקטות DHCP ובודקים את הסטטוס שלה וכך אנחנו יודעים איך להגיב לחבילה שהגיעה.
כתובת IP שנתנו לשרת ה DHCP שלנו הינה '192.168.1.190' ולכן אנחנו רואים שהשרת שאנחנו יצרו "הרים" את הבקשת discover ונענה לה ע"י שליחת חבילת תגובה, offer , חבילה זו מציעה ללקוח שפנה לשרת כתובת IP אופציונלית, במקרה שלנו אנחנו מציעים ללקוח כתובת IP יחידה ולכן הלקוח גם "בוחר" בה, כלומר הלקוח שולח חבילת request שמבקשת מהשרת את הכתובת הנ"ל שהציעו לו, בשלב האחרון השרת שולח חבילת ACK בחזרה ללקוח ורק לאחר מכן כתובת הIP שהלקוח ביקש הופכת להיות כתובת הIP שלו.

- ניתן לראות שקיבלנו גם חבילת NAK מכתובת '192.168.1.1' שהינה כתובת השרת DHCP האמיתי של המחשב עליו הרצנו את הקוד, זה מראה שהשרת המקומי של המחשב מתנגד להצעה של כתובת הIP שהבאנו ככל הנראה כי היא לא פנויה.

DNS

את שרת ה DNS יצרנו בעזרת ספריית SCAPY של PYTHON בעזרתה הסנפנו פקטות DNS וידענו כיצד לטפל בכל סוג פקטה.

1	0.0000000…	192.168.1.0	192.168.1.180	DNS	69	Standard query 0x0001 A BRApp.com
2	4.0918734…	192.168.1.180	192.168.1.0	DNS	94	Standard query response 0x0001 A BRApp.com A 127.0.0.200

את כתובת הIP של השרת - '192.168.1.180' הלקוח קיבל ביחד עם כתובת הIP שלו - '192.168.1.0' בחבילת ה ACK משרת הDHCP.
בצילום מסך זה ניתן לראות את חבילת ה request מהלקוח לשרת ה DNS, הלקוח ביקש את כתובת הIP של שרת האפליקציה שלנו לו נתנו את הדומיין BRApp.com , הבקשה היא מסוג A עבור כתובת IP גרסה 4.
לאחר מכן הלקוח מקבל את חבילת ה response מהשרת עם כתובת הIP של שרת האפליקציה והוא '127.0.0.200'.

רות גולדברג / 322631235
בנימין טיבי / 208434290

APP

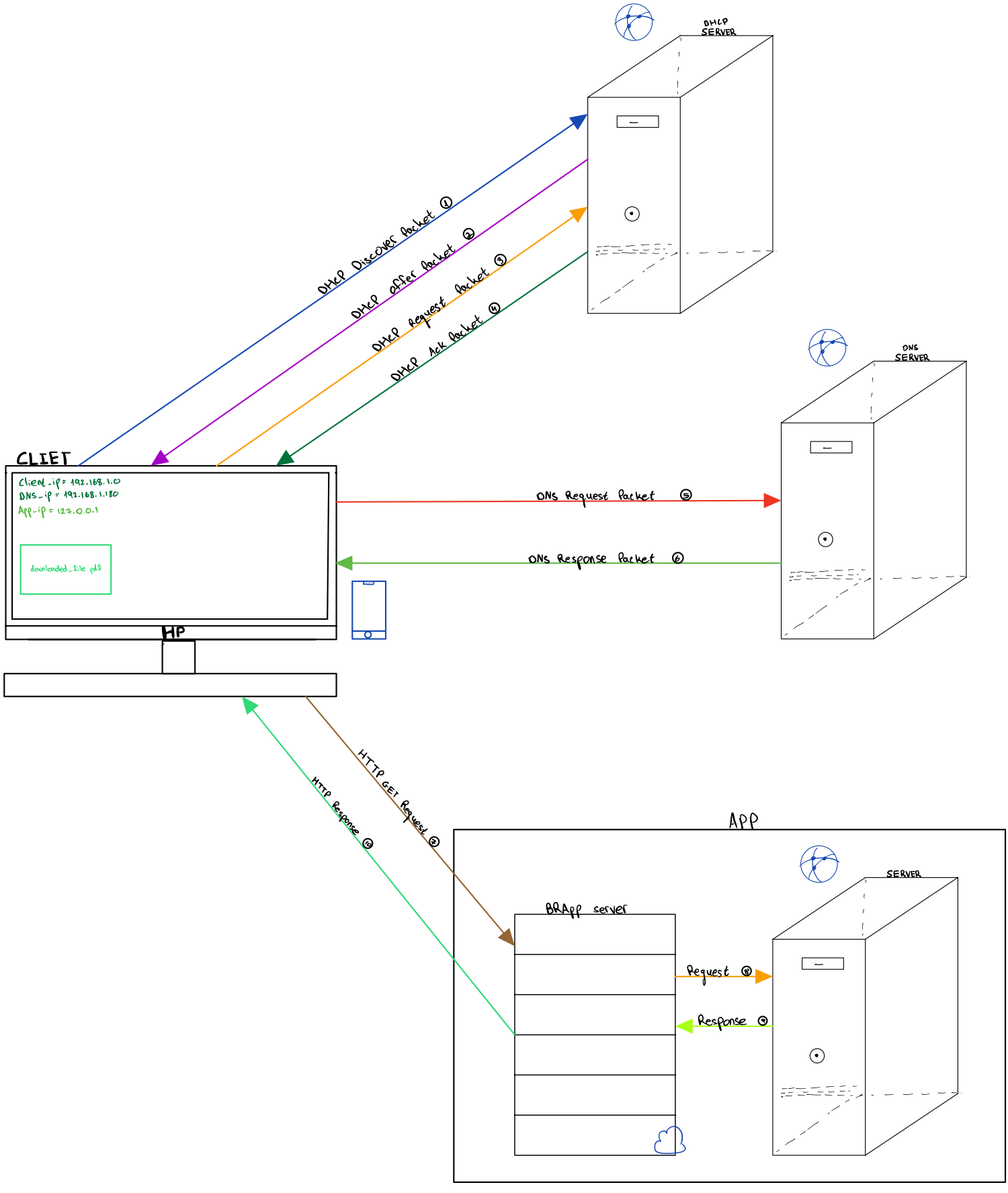
האפליקציה אותה בחרנו הינה מספר 3 ברשימה - http.
ביצענו את הredirect באופן הבא:
יצרנו שני שרתים : שרת האפליקציה - BRApp ושרת נוסף - RDServer.
כאשר הלקוח שולח את בקשת ה http הוא שולח אותה ישירות לשרת האפליקציה BRApp.
שליחת הבקשה מהלקוח מתבצעת ע"י ספריית requests של python עם פעולת get אשר מחזירה אובייקט מסוג response.
מאחורי הקלעים הפונקציה get פותחת סוקט חיבור עם השרת בכתובת הIP והPORT שנתונים בכתובת ה URL שמכניסים לפונקציה ולכן בצד השרת BRAPP קיים סוקט מסוג TCP כדי לקבל את הבקשה של הלקוח.
כאשר השרת BRApp מקבל את בקשת ה GET מהלקוח הוא פותח סוקט אל מול השרת RDServer ומעביר לו את בקשת ה GET של הלקוח.
בשלב זה השרת RDServer מקבל את הבקשה מהשרת BRApp , בודק את תקינותה ובמידה ותקינה (התבקש קובץ שנמצא אצל השרת) הוא פותח את הקובץ ומעביר את תוכנו לשרת הBRApp.
שרת הBRApp מקבל את התשובה, סוגר את הסוקט אצל מול שרת הRDServer ומעביר את התשובה שקיבל בחזרה אל הלקוח.
כאשר הלקוח קיבל את התשובה - אם סטטוס התגובה הינו 200 OK אז אצל הלקוח נוצר קובץ חדש עם השם downloaded_file עם תוכנו של הקובץ אותו ביקש מהשרת.
אם הסטטוס הינו 404 אז תודפס הודעה בהתאם.
הערות: - את החיבור בין הלקוח לשרת ובין השרתים עצמם ביצענו באמצעות TCP בלבד בשל קוצר הזמן.
-בעקבות השימוש בספריית requests לא ניתנה לנו האפשרות להגדיר את כתובת הIP של הלקוח ואת פורט המקור שלו ולכן כתובת הIP הינה '127.0.0.1' localhost והפורט משתנה בכל הרצה של התוכנית, הפתרון היה לשנות את הקוד ולא להשתמש בספרייה אלא רק בסוקט.

רות גולדברג / 322631235
בנימין טיבי / 208434290

1	0.0000000…	127.0.0.1	127.0.0.200	TCP	74	48214 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2144743621 TSecr=0 WS=128
2	0.0000170…	127.0.0.200	127.0.0.1	TCP	74	80 → 48214 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1030793640 TSecr
3	0.0000313…	127.0.0.1	127.0.0.200	TCP	66	48214 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2144743621 TSecr=1030793640
4	0.0000778…	127.0.0.1	127.0.0.200	HTTP	223	GET /ProxyServer.pdf HTTP/1.1
5	0.0000854…	127.0.0.200	127.0.0.1	TCP	66	80 → 48214 [ACK] Seq=1 Ack=158 Win=65408 Len=0 TSval=1030793640 TSecr=2144743621
6	0.0002608…	127.0.0.200	127.0.0.255	TCP	74	20235 → 30290 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1155005418 TSecr=0 WS=12
7	0.0002708…	127.0.0.255	127.0.0.200	TCP	74	30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3443792554 TS
8	0.0002797…	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1155005418 TSecr=3443792554
9	0.0002997…	127.0.0.200	127.0.0.255	HTTP	223	GET /ProxyServer.pdf HTTP/1.1
10	0.0003044…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=1 Ack=158 Win=65408 Len=0 TSval=3443792554 TSecr=1155005418
11	0.0005721…	127.0.0.255	127.0.0.200	HTTP	25534	HTTP/1.1 200 OK (application/pdf)
12	0.0005852…	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [ACK] Seq=158 Ack=25469 Win=52224 Len=0 TSval=1155005419 TSecr=3443792555
13	0.0006043…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [FIN, ACK] Seq=25469 Ack=158 Win=65536 Len=0 TSval=3443792555 TSecr=1155005419
14	0.0007726…	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [FIN, ACK] Seq=158 Ack=25470 Win=65536 Len=0 TSval=1155005419 TSecr=3443792555
15	0.0007872…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=25470 Ack=159 Win=65536 Len=0 TSval=3443792555 TSecr=1155005419
16	0.0008204…	127.0.0.200	127.0.0.1	HTTP	25534	HTTP/1.1 200 OK (application/pdf)
17	0.0008306…	127.0.0.1	127.0.0.200	TCP	66	48214 → 80 [ACK] Seq=158 Ack=25469 Win=52224 Len=0 TSval=2144743622 TSecr=1030793641
18	0.0008451…	127.0.0.200	127.0.0.1	TCP	66	80 → 48214 [FIN, ACK] Seq=25469 Ack=158 Win=65536 Len=0 TSval=1030793641 TSecr=2144743622
19	0.0017704…	127.0.0.1	127.0.0.200	TCP	66	48214 → 80 [FIN, ACK] Seq=158 Ack=25470 Win=65536 Len=0 TSval=2144743623 TSecr=1030793641
20	0.0017852…	127.0.0.200	127.0.0.1	TCP	66	80 → 48214 [ACK] Seq=25470 Ack=159 Win=65536 Len=0 TSval=1030793642 TSecr=2144743623

בצילום מסך זה רואים את כל ההרצה של חלק האפליקציה.
בשורות 1-3 : מתבצע חיבור הקשר TCP בין הלקוח לשרת BRApp.
בשורות 4-5 : מתבצעת שליחת בקשת הhttp מהלקוח ואישורה ע"י השרת.
בשורות 6-8: מתבצע חיבור הקשר TCP בין שרת BRApp לבין שרת ה RDServer.
בשורות 9-15: מתבצעת שליחת הבקשה משרת BRApp לשרת RDServer , השרת השני מאשר את הבקשה שולח את התגובה עם OK 200 ונסגר הקשר בין שני השרתים.
בשורות 16-20: מתבצעת קבלת התגובה משרת BRApp ללקוח ונסגר הקשר ביניהם.

BRApp - כתובת '127.0.0.200' פורט (80 קבלת הבקשה 20235 שליחת הבקשה)
RDServer - כתובת '127.0.0.255' פורט 30390


DHCP SERVER
DHCP Discover Packet ①
DHCP offer Packet ②
DHCP Request Packet ③
DHCP Ack Packet ④
DNS SERVER
CLIET
Client_ip = 192.168.1.0
DNS_ip = 192.168.1.180
App-ip = 127.0.0.1
downloaded_file.pdf
DNS Request Packet ⑤
DNS Response Packet ⑥
HP
HTTP GET Request ⑦
HTTP Response ⑩
APP
BRApp server
SERVER
Request ⑧
Response ⑨

רות גולדברג / 322631235
בנימין טיבי / 208434290

הסבר על הדיאגרמה:

בדיאגרמה ציירנו את הלקוח , שני שרתים DHCP ו DNS ואת חלק האפליקציה שם יש את BRApp ו RDServer.
עבור כל חץ יש מספר המציין מתי הפעולה מתבצעת ביחס לאחרות.
אז ניתן לראות שהסדר הוא כזה:
- הלקוח פונה לשרת DHCP על מנת לקבל את כתובת ה IP שלו ואת כתובת ה IP של שרת Local DNS שלו.
- הלקוח פונה לשרת ה DNS שלו על מנת לקבל את כתובת ה IP של שרת האפליקציה אליו הוא רוצה לפנות.
- הלקוח פונה לשרת האפליקציה BRApp ומבקש ממנו את הקובץ אותו הוא רוצה להוריד.
- שרת האפליקציה פונה לשרת RDServer על מנת לקבל ממנו את הקובץ המבוקש והוא נענה לבקשתו.
- שרת האפליקציה מחזיר את התגובה אותה קיבל מהשרת הנוסף אל הלקוח.

איבוד של 10% פקטות:

הרצנו את התוכנית עם הפעלה של 10% איבוד פקטות.
בפרויקט שלנו מאחר והשתמשנו רק ב TCP אז הטיפול באיבוד הפקטות מתבצע ע"י TCP ולא על ידינו בקוד.
בכל זאת, נסביר כיצד TCP מתמודד עם איבוד פקטות וזאת גם הצורה בה היה צריך לבנות את RUDP על מנת להתמודד עם איבוד פקטות בדומה ל TCP.
ראשית, TCP מזהה איבוד חבילות ע"י כך שכאשר חבילה נשלחת השולח מחכה לאישור - ACK - כאשר זה לא מגיע לאחר זמן מסוים הוא מניח שיש איבוד פקטה ולכן משתמש במנגנון שנקרא שידור חוזר / מהיר ובכך משדר את החבילה המדובר פעם נוספת.
כפי שלמדנו TCP משתמש בנוסף ב- CC אלגוריתם על מנת לא להעמיס על הרשת ובכך להבטיח שהמידע המועבר יגיע בצורה מהימנה.
ישנם 4 אלגוריתמים בהם TCP יכול להשתמש בשביל בקרת הגודש - congestion control:
התחלה איטית : מגביר את קצב השידור בהתאם כאשר נוצר חיבור חדש.
מניעת עומס : כאשר מזהה עומס ברשת הוא מגביר את קצב השידור לאט יותר.
שידור חוזר/מהיר : מזהה אובדן של חבילות ויכול לשדר אותן מחדש מהר יותר.
הודעת עומס מפורשת : משתמש במשוב המנתבים כדי לזהות עומס ולהתאים את קצב השידור.

1	0.0000000…	127.0.0.1	127.0.0.200	TCP	74	32800 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2149932592 TSecr=0 WS=128
2	0.0000099…	127.0.0.200	127.0.0.1	TCP	74	80 → 32800 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1035982611 TSec
3	0.0000176…	127.0.0.1	127.0.0.200	TCP	66	32800 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2149932592 TSecr=1035982611
4	0.2008485…	127.0.0.1	127.0.0.200	HTTP	216	GET /Ping.pdf HTTP/1.1
5	0.2008698…	127.0.0.200	127.0.0.1	TCP	66	80 → 32800 [ACK] Seq=1 Ack=151 Win=65408 Len=0 TSval=1035982812 TSecr=2149932793
6	0.2010246…	127.0.0.200	127.0.0.255	TCP	74	20235 → 30290 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1160194590 TSecr=0 WS=1
7	1.2048528…	127.0.0.200	127.0.0.255	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 20235 → 30290 [SYN] Seq=0 Win=65495 Len=0 MSS=
8	1.2048699…	127.0.0.255	127.0.0.200	TCP	74	30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3448982730 T
9	1.2048925…	127.0.0.255	127.0.0.200	TCP	74	[TCP Out-Of-Order] 30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1
10	1.2049099…	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1160195594 TSecr=3448982730
11	1.2049146…	127.0.0.200	127.0.0.255	TCP	66	[TCP Dup ACK 10#1] 20235 → 30290 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1160195594 TSecr=3448
12	1.2049602…	127.0.0.200	127.0.0.255	HTTP	216	GET /Ping.pdf HTTP/1.1
13	1.2049713…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=1 Ack=151 Win=65408 Len=0 TSval=3448982730 TSecr=1160195594
14	1.2052430…	127.0.0.255	127.0.0.200	HTTP	12060	HTTP/1.1 200 OK (application/pdf)
15	1.2052722…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [FIN, ACK] Seq=11995 Ack=151 Win=65536 Len=0 TSval=3448982731 TSecr=1160195594
16	1.2053548…	127.0.0.200	127.0.0.1	HTTP	12060	HTTP/1.1 200 OK (application/pdf)
17	1.2053632…	127.0.0.1	127.0.0.200	TCP	66	32800 → 80 [ACK] Seq=151 Ack=11995 Win=59008 Len=0 TSval=2149933798 TSecr=1035983817
18	1.2053729…	127.0.0.200	127.0.0.1	TCP	66	80 → 32800 [FIN, ACK] Seq=11995 Ack=151 Win=65536 Len=0 TSval=1035983817 TSecr=2149933798
19	1.2064169…	127.0.0.1	127.0.0.200	TCP	66	32800 → 80 [FIN, ACK] Seq=151 Ack=11996 Win=65536 Len=0 TSval=2149933799 TSecr=1035983817
20	1.2064361…	127.0.0.200	127.0.0.1	TCP	66	80 → 32800 [ACK] Seq=11996 Ack=152 Win=65536 Len=0 TSval=1035983818 TSecr=2149933799
21	1.2248598…	127.0.0.255	127.0.0.200	TCP	66	[TCP Retransmission] 30290 → 20235 [FIN, ACK] Seq=11995 Ack=151 Win=65536 Len=0 TSval=3448982
22	1.4368519…	127.0.0.255	127.0.0.200	TCP	12060	[TCP Retransmission] 30290 → 20235 [FIN, PSH, ACK] Seq=1 Ack=151 Win=65536 Len=11994 TSval=34
23	1.4368894…	127.0.0.200	127.0.0.255	TCP	78	[TCP Previous segment not captured] 20235 → 30290 [ACK] Seq=152 Ack=11996 Win=65536 Len=0 TSv
24	3.1888620…	127.0.0.200	127.0.0.255	TCP	66	[TCP Retransmission] 20235 → 30290 [FIN, ACK] Seq=151 Ack=11996 Win=65536 Len=0 TSval=1160197
25	12.344843…	127.0.0.200	127.0.0.255	TCP	66	[TCP Retransmission] 20235 → 30290 [FIN, ACK] Seq=151 Ack=11996 Win=65536 Len=0 TSval=1160206
26	12.344880…	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=11996 Ack=152 Win=65536 Len=0 TSval=3448993870 TSecr=1160197578

רות גולדברג / 322631235
בנימין טיבי / 208434290

1000ms latency :

גם פה נסביר כיצד TCP מתמודד עם איחורי פקטות.
אחת הדרכים היא על ידי חלון הזזה.
לשולח יש חלון בו נקבעים מספר החבילות שיכולות להיות משודרות בכל רגע נתון.
בכל רגע בחלון יש חבילות ששודרו וטרם אושרו על ידי המקבל, כך אם חבילה כלשהי נאבדת או מתעכבת השולח יכול לשלוח אותה מחדש לאחר פרק זמן מסוים הנקבע בהתאם לצפיפות הרשת וזמן משוער של קבלת אישור מרגע שליחת החבילה.
הדרכים הנוספות של בקרת הגודש שהצגנו לעיל גם יכולות לעזות להתמודד עם איחורי פקטות.

1	0.000000000	127.0.0.1	127.0.0.200	TCP	74	52622 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2153437352 TSecr=0 WS=128
2	1.000078607	127.0.0.200	127.0.0.1	TCP	74	80 → 52622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1039488371 TSecr=2153
3	1.020911806	127.0.0.1	127.0.0.200	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 52622 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SAC
4	2.000192135	127.0.0.1	127.0.0.200	TCP	66	52622 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2153439352 TSecr=1039488371
5	2.000256010	127.0.0.1	127.0.0.200	HTTP	215	GET /TCP.pdf HTTP/1.1
6	2.016902659	127.0.0.200	127.0.0.1	TCP	74	[TCP Retransmission] 80 → 52622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=
7	2.020958828	127.0.0.200	127.0.0.1	TCP	74	[TCP Retransmission] 80 → 52622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=
8	3.000317878	127.0.0.200	127.0.0.1	TCP	66	80 → 52622 [ACK] Seq=1 Ack=150 Win=65408 Len=0 TSval=1039490372 TSecr=2153439353
9	3.000601743	127.0.0.200	127.0.0.255	TCP	74	20235 → 30290 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1163702150 TSecr=0 WS=128
10	3.016965637	127.0.0.1	127.0.0.200	TCP	66	[TCP Dup ACK 4#1] 52622 → 80 [ACK] Seq=150 Ack=1 Win=65536 Len=0 TSval=2153440369 TSecr=1039488371
11	4.000679542	127.0.0.255	127.0.0.200	TCP	74	30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3452490286 TSecr=1
12	4.028920401	127.0.0.200	127.0.0.255	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 20235 → 30290 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
13	5.000740703	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1163704150 TSecr=3452490286
14	5.000793695	127.0.0.200	127.0.0.255	HTTP	215	GET /TCP.pdf HTTP/1.1
15	5.020921644	127.0.0.255	127.0.0.200	TCP	74	[TCP Retransmission] 30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSv
16	5.028997355	127.0.0.255	127.0.0.200	TCP	74	[TCP Retransmission] 30290 → 20235 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSv
17	6.000854936	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=1 Ack=150 Win=65408 Len=0 TSval=3452492286 TSecr=1163704150
18	6.001036412	127.0.0.255	127.0.0.200	HTTP	6618	HTTP/1.1 200 OK (application/pdf)
19	6.001072902	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [FIN, ACK] Seq=6553 Ack=150 Win=65536 Len=0 TSval=3452492286 TSecr=1163704150
20	6.020973954	127.0.0.200	127.0.0.255	TCP	66	[TCP Dup ACK 13#1] 20235 → 30290 [ACK] Seq=150 Ack=1 Win=65536 Len=0 TSval=1163705170 TSecr=3452496
21	7.001090402	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [ACK] Seq=150 Ack=6553 Win=61696 Len=0 TSval=1163706150 TSecr=3452492286
22	7.001282287	127.0.0.200	127.0.0.255	TCP	66	20235 → 30290 [FIN, ACK] Seq=150 Ack=6554 Win=65536 Len=0 TSval=1163706151 TSecr=3452492286
23	7.001328848	127.0.0.200	127.0.0.1	HTTP	6618	HTTP/1.1 200 OK (application/pdf)
24	7.001330517	127.0.0.200	127.0.0.1	TCP	66	80 → 52622 [FIN, ACK] Seq=6553 Ack=150 Win=65536 Len=0 TSval=1039494373 TSecr=2153440369
25	8.001346626	127.0.0.255	127.0.0.200	TCP	66	30290 → 20235 [ACK] Seq=6554 Ack=151 Win=65536 Len=0 TSval=3452494287 TSecr=1163706151
26	8.001403668	127.0.0.1	127.0.0.200	TCP	66	52622 → 80 [ACK] Seq=150 Ack=6553 Win=61696 Len=0 TSval=2153445354 TSecr=1039494373
27	8.002770581	127.0.0.1	127.0.0.200	TCP	66	52622 → 80 [FIN, ACK] Seq=150 Ack=6554 Win=65536 Len=0 TSval=2153445355 TSecr=1039494373
28	9.002875936	127.0.0.200	127.0.0.1	TCP	66	80 → 52622 [ACK] Seq=6554 Ack=151 Win=65536 Len=0 TSval=1039496374 TSecr=2153445355
29	26.1097204…	127.0.0.1	127.0.0.1	TCP	74	51622 → 35575 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=365418629 TSecr=0 WS=128
30	27.1098069…	127.0.0.1	127.0.0.1	TCP	74	35575 → 51622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=365419629 TSecr=36
31	27.1329315…	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 51622 → 35575 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
32	28.1098942…	127.0.0.1	127.0.0.1	TCP	66	51622 → 35575 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=365420629 TSecr=365419629
33	28.1289241…	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] 35575 → 51622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSv
34	28.1329687…	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] 35575 → 51622 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSv
35	29.1289520…	127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 32#1] 51622 → 35575 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=365421648 TSecr=365419629
36	31.1189031…	127.0.0.1	127.0.0.1	TCP	67	35575 → 51622 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=365423638 TSecr=365421648
37	31.2340551…	127.0.0.1	127.0.0.1	TCP	66	35575 → 51622 [FIN, ACK] Seq=2 Ack=1 Win=65536 Len=0 TSval=365423753 TSecr=365421648
38	31.2341175…	127.0.0.1	127.0.0.1	TCP	66	51622 → 35575 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=365423753 TSecr=365419629

ניתן לראות שחבילות נשלחו שוב וגם נשלחה הודעה של ACKים כפולים כי כל עוד לא הגיע ACK מעודכן נשלחת הודעה של אותו ACK שנשלח כבר.

הבדלים עיקריים בין פרוטוקול TCP לבין פרוטוקול QUIC:
QUIC (Quick UDP Internet Connections)-ו TCP (Transmission Control Protocol)

הם שניהם פרוטוקולי שכבת תעבורה המשמשים להעברת נתונים דרך האינטרנט. עם זאת, ישנם כמה הבדלים מרכזיים בין השניים:

1)
TCP הוא פרוטוקול חיבור מכוון כלומר הוא יוצר חיבור ייעודי בין שני מכשירים לפני העברת נתונים.
QUIC .לעומת זאת הוא פרוטקול ללא חיבור כלומר לא נדרש חיבור ייעודי לפני העברת נתונים

2)
:אמינות
TCP
מספקת העברת נתונים בצורה אמינה, למעשה היא מבטיחה שכל החבילות יעברו כמו שצריך וכל החבילות .שנאבדו יועברו מחדש
QUIC
.מספקת גם אמינות אך עושה זאת באמצעות מנגנון אחר שהוא יעיל יותר ומהיר יותר

3)
QUIC כולל הצפנה כברירת מחדל בעוד ש TCP דורש פרוטוקולים נוספים כגון TLS .לתקשורת מאובטחת

4)
QUIC
נועד להפחית את זמן ההשהייה ולשפר את הביצועים על ידי צמצום מספר הפעמים שמנסים ליצור חיבור .ולהעביר נתונים
דבר זה הופך את QUIC למהיר יותר מTCP במצבים מסויימים
.במיוחד עבור יישומים שדורשים מידע בזמן אמת כגון הזרמת ווידאו ומשחקים

5)
:שחזור אובדן פאקטות
TCP
משתמש באלגוריתם של התחלה איטית כדי להתאושש מאובדן פאקטות, דבר שעלול להוביל להעברת .נתונים איטית יותר
QUIC
משתמש במנגנון שחזור אובדן פאקטות מהיר ויעיל יותר

לסיכום:
QUIC מהיר ויעיל יותר מ-TCP בשל תהליך יצירת החיבור היעיל שלו, ומנגנוני שחזור אובדן מנות ותכונות אבטחה חזקות יותר. עם זאת, TCP עדיין נמצא בשימוש נרחב ונשאר הפרוטוקול הדומיננטי עבור סוגים רבים של העברת נתונים דרך האינטרנט.

הבדלים עיקריים בין אלגוריתם Vegas לבין אלגוריתם Cubic:
VEGAS (Variable Estimated Gain Adaptive Sizing)

"Cubic" ו-"VEGAS" הם אלגוריתמים לבקרת צפיפות (congestion control) המשמשים בפרוטוקול בקרת שידור (TCP) לניהול זרימת הנתונים ברחבי הרשת.

1) בקרת צפיפות(**congestion control**):
Cubic הוא אלגוריתם בקרת צפיפותTCP (**congestion control**) שמטרתו היא למקסם את ניצול הרשת.
הוא משתמש בcubic function כדי לקבוע את חלון הצפיפות , שהיא למעשה כמות הנתונים שהשולח יכול לשלוח לפני המתנה לאישור מהמקבל.
cubic function מתאימה את גודל החלון בהתבסס על רמת הגודש של הרשת ומטרתה לשמור על זרימה קבועה של נתונים תוך כדי הימנעות מקריסה בשל עומס.
Vegas הוא אלגוריתם אחר של בקרת צפיפות שנועד לשפר את הדיוק של זיהוי הגודש ברשת והתגובה.
Vegas מודד את הזמן שלוקח לחבילה לחצות את הרשת ומשתמש במידע זה כדי להעריך את רמת העומס על הרשת.
לאחר מכן הוא מתאים את גודל החלון בהתבסס על הערכה הנ"ל במטרה לשמור על רמה יציבה של עומס תוך כדי ניצול מקסימלי של הרשת.

2) מימושים:
Cubic וVegas ממומשים בצורות שונות.
Cubic ממומש "כמכונת מצב" השומרת על גודל חלון הצפיפות הנוכחי ומתאים אותו על סמך cubic function .
Vegas לעומת זאת ממומשת באמצעות שני אלגוריתמים שונים:
אלגוריתם Vegas שמעריך את זמן ההעברה ומתאים בהתאם לכך את גודל החלון
ואלגוריתם Vegas2 המשתמש בגישה שונה לזיהוי ותגובה של הצפיפות ברשת.

3) היענות:
וגאס מגיבה יותר לשינויים בתנאי הרשת מאשר ב- Cubic . המשמעות היא שוגאס טובה יותר באיתור מהיר של עומס ברשת ותגובה להן, מה שמביא לביצועים טובים יותר בסביבות רשת דינאמיות. קוביק פחות מגיב, אבל הוא יכול להתמודד עם רמות גבוהות יותר של עומס בצורה יעילה יותר מאשר וגאס.

BGP-Border Gateway Protocol Vs OSPF- Open Shortest Path First:

BGP (Border Gateway Protocol) ו-OSPF (Open Shortest Path First) שניהם פרוטוקולי ניתוב המשמשים ברשתות תקשורות, אך יש להם תפקידים שונים והם פועלים בדרכים שונות.

BGP:
הוא פרוטוקול ניתוב המשמש להחלפת מידע וניתוב בין מערכות אוטונומיותASes)) שונות באינטרנט.
(מערכת אוטונומית היא אוסף של רשתות המופעלות על ידי גורם יחיד)
BGP פועלת על ידי החלפת מידע בין ראוטרים (נתבים) בגבולות של מערכות אוטונומיות, כל ראוטר שולח עדכונים לגבי המסלולים הוא מכיר לראוטרים שקרובים אליו(השכנים שלו) ועדכונים אלו מופצים ברחבי הרשת.
BGP הוא פרוטוקול חשוב לתפקוד האינטרנט מכיוון שהוא מאפשר למערכות האוטונומיות השונות לעבוד יחד כדי לספק למשתמשים ברחבי העולם דרך תקשור טובה.
יש לחדד שBGP הוא פרוטוקול מורכב והוא דורש ניהול קפדני כדי להבטיח שהוא פועל בצורה נכונה ומאובטחת.

OSPF:
הוא פרוטוקול שער פנימיIGP)) והוא משמש להפצת מידע ברחבי האינטרנט בתוך מערכת אוטונומית אחת.
בדרך כלל משתמשים בו בארגונים כדי לנתב תעבורה בין רשתות משנה או רשתות שונות.

ההבדלים העיקריים ביניהם הוא שOSPF משמש להפצת מידע בתוך רשת או ארגונים בודדים, בעוד שBGP משמש להחלפת מידע בין מערכות אוטונומיות שונות באינטרנט.
OSPF הוא פרוטוקול שער פנימי בעוד שBGP הוא פרוטוקול שער חיצוני.
לסיכום שניהם חשובים מאוד לתפקוד של רשתות האינטרנט אך הם ממשים פונקציות שונות ופועלים ברמות שונות בהיררכיית הרשת.

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
DHCP	67	68	192.168.1.190	255.255.255.255	e8:84:a5:b9:17:d4	e8.84.a5.b9.17.d4

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
DNS	53	20255	192.168.1.180	192.168.1.0	e8:84:a5:b9:17:d4	e8.84.a5.b9.17.d4

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
BRApp	20235	30290	127.0.0.200	127.0.0.255	e8.84.a5.b9.17.d4	e8.84.a5.b9.17.d4

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
RDServer	30290	20235	127.0.0.255	127.0.0.200	e8.84.a5.b9.17.d4	e8.84.a5.b9.17.d4

אם היה NAT בין הלקוח והשרת אז הוא ישנה את כתובת IP של המקור בכתובת ג'ברית משלו ואילו את כתובת ה-IP של היעד הוא יחליף בכתובת ה-IP הפרטית של השרת.

לא נכון להשתמש בפרוטוקול QUIC אחרת ה port src של ה- tcp היה משתנה באופן דינמי ל port שם 443.

ARP-(Address Resolution Protocol) VS DNS(Domain Name System)

שני פרוטוקולים שונים שמשתמשים בהם ברשתות תקשורת, שניהם משרתים פונקציות שונות ופועלים בשכבות שונות בהיררכיית הרשת.

ARP הוא פרוטוקול בשכבת הרשת של מודל השכבות OSIלמיפוי כתובות רשת(כתובות IP) לכתובת פיזית(כתובת MAC).
ARP משמש את התקני הרשת כדי למצוא כתובת MAC של מכשיר באותו אזור של רשת כך שהם יכולים לשלוח נתונים לאותו מכשיר.
כאשר מכשיר רוצה לשלוח נתונים למכשיר אחר באותו מקטע של רשת, הוא שולח בקשת ARP כדי למצוא את כתובת הMAC של מכשיר היעד.
מכשיר היעד מגיב עם כתובת MAC משלו והשולח יכול להשתמש בכתובת זו על מנת לשלוח נתונים למכשיר היעד.

DNS הוא פרוטוקול בשכבת האפליקציה של מודל השכבות OSI למיפוי שמות המתחם (domain names) לכתובות IP.
על מנת להקל על המשתמשים להגיע ממחשב למחשב ולאתר את המיקום של כל אחד ברשת, נבנתה מערכת שנועדה לעזור "לאתר" אחד את השני.
זוהי מערכת שם המתחם.
תפקיד המערכת הוא לתרגם את המספרים הייחודיים (כתובות כמו לדוגמא 192.112.112.11) לשם שבני האדם יהיו יכולים להבין כמו למשל.
www.walla.com

ההבדלים העיקריים בינהם הם: ARP הוא פרוטוקול המשמש למיפוי כתובת רשת לכתובת פיזית באותו קטע של רשת, בעוד שDNS הוא פרוטוקול שמשמש למיפוי שמות מתחם לכתובות IP.
ARP פועל בשכבת הרשת בעוד ש DNS פועל בשכבת האפליקציה.
שני הפרוטוקולים חשובים מאוד לתפקוד הרשת שכן הם מאפשרים למכשירים לתקשר זה עם זה ולאפליקציות להתחבר לשרתים ברחבי האינטרנט.

רות גולדברג / 322631235
בנימין טיבי / 208434290

ביבליוגרפיה

DHCP

https://data.cyber.org.il/networks/networks_py27.pdf

DNS

https://thepacketgeek.com/scapy/building-network-tools/part-09/

APP

https://www.geeksforgeeks.org/socket-programming-python/
https://reqbin.com/code/python/ixa9gi7o/python-http-request-example
https://www.geeksforgeeks.org/get-post-requests-using-python/
https://iximiuz.com/en/posts/writing-web-server-in-python-sockets/
https://reqbin.com/code/python/6mwlgbqa/python-requests-download-file-example
https://www.datacamp.com/tutorial/making-http-requests-in-python