פרוייקט סיום רשתות תקשורת

איך להתקין

כדי להוריד את הפרוייקט נצטרך להשתמש בגיט אז נוריד
sudo apt-get install git

נוריד
git clone https://github.com/eytan1998/NET6.git

נכנס אל התקייה
cd NET6

כדי להריץ את התוכנית נצטאך להוריד עוד כמה דברים וכדי להוריד נשתמש בpip נתקין ע"י

sudo apt install python3-pip

עכשיו נתקין את scapy
pip install --pre scapy[complete]

ונתקין את jsonoickle
sudo pip install jsonpickle

אחרי כל ההתקנות נוכל להריץ את הקבצים (נזכור שאנחנו בתקייה NET6 שעכשיו הורדנו)

מבנה קבצים

NET6

BACKEND

DATA שומר את המידע של השרת

HELP כלים להרצה כמו בתי כנסת,גבאים,עיבוד בקשות וכו

RUDP את הקבצים הנחוצים לקשורת RUDP

TCP את הקבצים הנחוצים לקשורת TCP

DHCP השרת והקלינט של DHCP עם קבצים נוספים לשמירות

DNS השרת והקלינט של DNS, עם קבצים נוספים לשמירות

FRONTEND כאן יש את כל GUI

TEST

server.py

App.py מחבר את הכלDHCP, DNS,server כולם אחד אחרי השני

איך להריץ

DHCPserver

כדי להריץ את השרת נצטרך לכתוב את השורה הזאת

sudo python3 DHCP/DHCPserver.py -H <host>

<host> איזה כתובת לשרת DNS (שרת הDHCP נותן את הכתובת הזאת ללקוח), בברת מחדל "127.0.0.1"

בקובץ dhcpclient שמור הIP,lease_time,dns ששמורים ללקוח.

DNSserver

כדי להריץ את השרת נצטרך לכתוב את השורה הזאת

<sudo python3 DNS/DNSserver.py -i <iface> -H <host

כאשר **חייב** להשתמש בSUDO.

<iface> על מה לעשה את sniffing, בברת מחדל "lo"

<host> איזה כתובת לשרת, בברת מחדל "127.0.0.1"

אפשר לראות מדריך ע"י

sudo python3 DNS/DNSserver.py -h

לעדכן את הרשומה לשרת

כדי לקבל תשובה DNSserver צריך לערוך את הרשומה כאילו יש תשובה על שם האפליקציה. את הרשומות ניתן נמצא בקובץ DNS/records.json

```
}
"domain": "theBestApp.com",
"address": "127.0.0.1",
"ttl": 1678568841.1665707"
{
```

כאשר domain זה מה שנכניס בשדה שבפתיחת האפליקציה, address זה הכתובת שרת שנרצה שידבר.
TTL הוא לא משמעותי כי כמובן לא יקבל תשובה על זה.

לאפס את הנתונים של השרת

במידה וצריך לאפס את נתוני השרת צריך להריץ את הקובץ

python3 Backend/Data/reserServer.py

אחרי איפס הזה לא יישאר שום מידע חוץ מאשר האדמין כאשר ה ID =1, PASSWORD = 1243

SERVER

כדי להריץ את השרת כדי לעשות חיבור RUDP נשתמש בפקודה הבאה:

sudo python3 server.py

כדי להריץ את השרת כדי לעשות חיבור TCP נשתמש בפקודה הבאה:

sudo python3 server.py --tcp

אפשר לראות מדריך ע"י

sudo python3 server.py -h

APP

כאן החלק החשוב, האפליקציה היא בנוי ה על גבי כל השאר, קודם כל DHCP שממנו מקבל שרת DNS שממנו מקבלת את שרת האפליקציה, לכן לראות שכולם עובדים לפני שמריצים את האפיקציה. אחרת זה לא יעבוד.

שוב בגלל שאפליקציה יש קשר לDNS אז נצטרך SUDO, כדי
כדי להריץ את האפליקציה עם חיבור RUDP נשתמש בפקודה הבאה:

```
sudo python3 App.py
```

כדי להריץ את האפליקציה עם חיבור TCP נשתמש בפקודה הבאה:

```
sudo python3 App.py --tcp
```

אפשר לראות מדריך ע"י

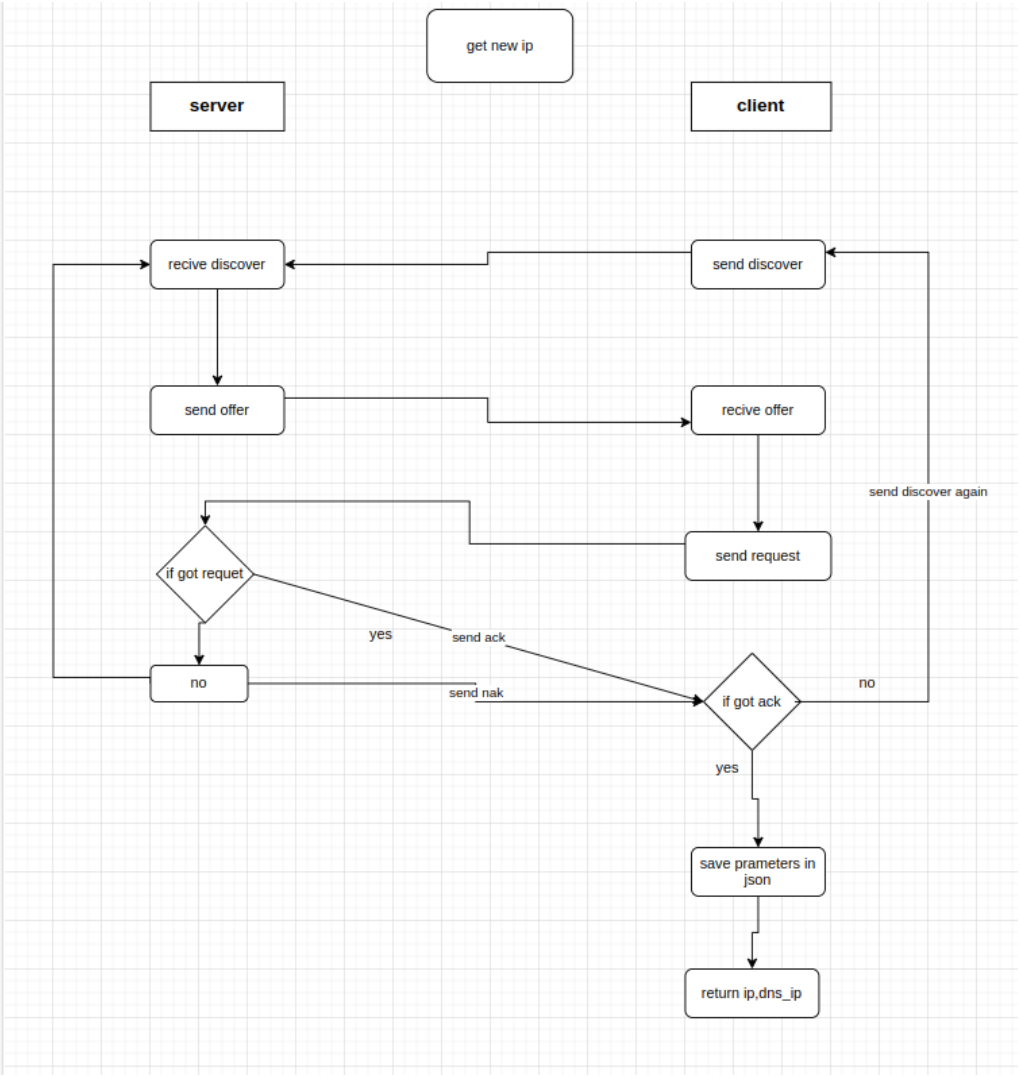
```
sudo python3 App.py -h
```

תיאור התוכנית

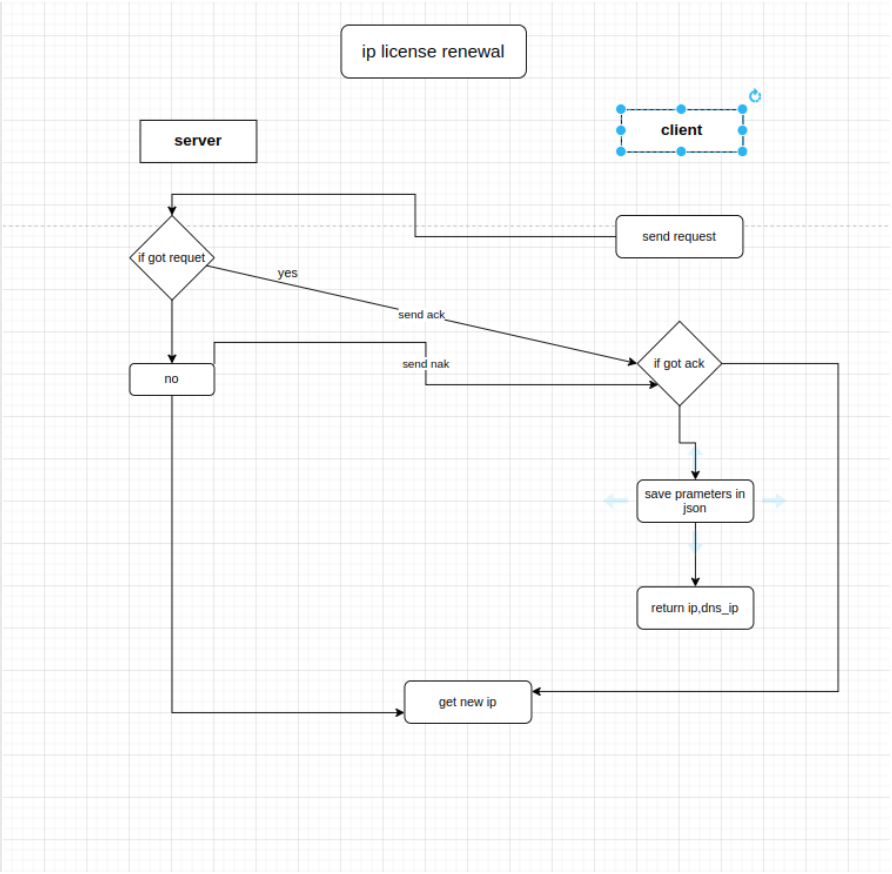
DHCP

מטרת שרת DHCP היא לחלק כתובות IP מהמאגר שברשותו למכשירים שמבקשים ממנו השרת מקצה בד"כ את הכתובות לזמן מוגבל.השרת פועל כל הזמן ומוכן לקבל בקשות.
נחלק את פעולת השרת לשני מקרים 1:המכשיר מבקש כתובת IP חדשה 2המכשיר מבקש לחדש רישיון לכתובת

1:המכשיר מבקש כתובת IP חדשה:
המכשיר יבקש כתובת חדשה כאשר זה הפעם הראשונה שהוא התחבר לרשת ואז אין לו בכלל כתובת או כאשר פג תוקף הרישיון שלו או כאשר בקשה קודמת(חידוש או כתובת חדשה)לא הצליחה.כל הפאקטות ישלחו בברודקאס. הלקוח שולח בקשת discover השרת מחזיר offer עם הצעה של כתובת IP הלקוח מחזיר request שבה הוא אומר מה הוא לקח ומקבל מהשרת ACK ועכשיו הוא יכול להשתמש בנתונים אם קיבל NAK אז הוא מתחיל מחדש

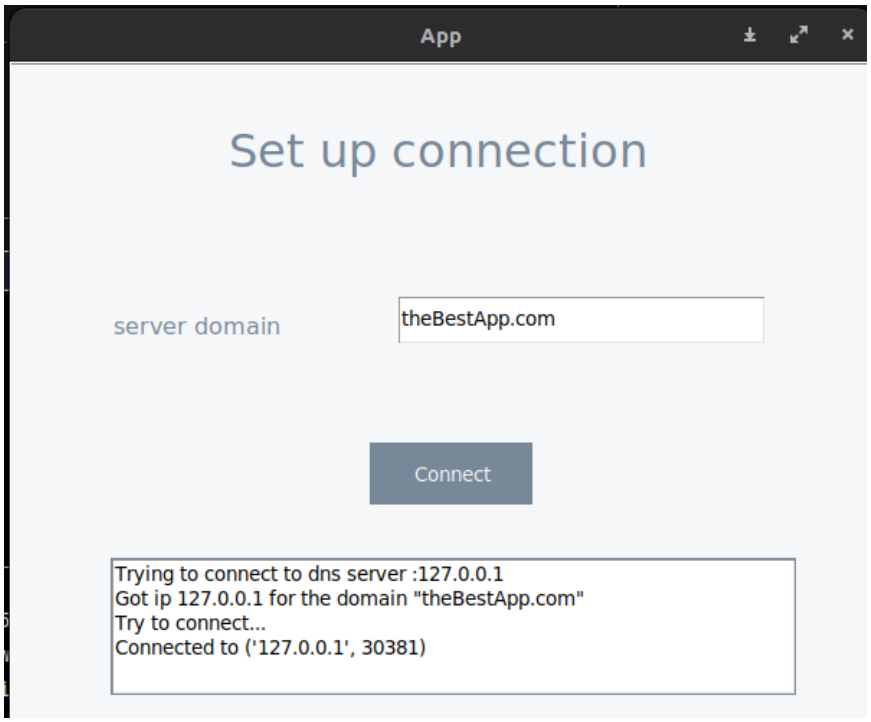


2. המכשיר מבקש לחדש רישיון לכתובת:
כאשר קידל כבר כתובת ולא עבר זמן ההחכירה לכן מספיק רק את החלק האחרון request and ack אבל אם יקבל nak אז יבקש כתובת חדשה.



DNS

תיאור



נתחיל מהאפליקציה כי היא מחוברת את DNS בכך ששואלת לגבי הDOMAIN של השרת ומקבל את הIP ואיתו היא מנסה לפתוח קשר.

נראה את החלונית של server domain, שם נכתוב את domain.

עכשיו אם נלחץ על connect אז ישלח בקשת DNS אל השרת DNS שהראנו איך להפעיל.

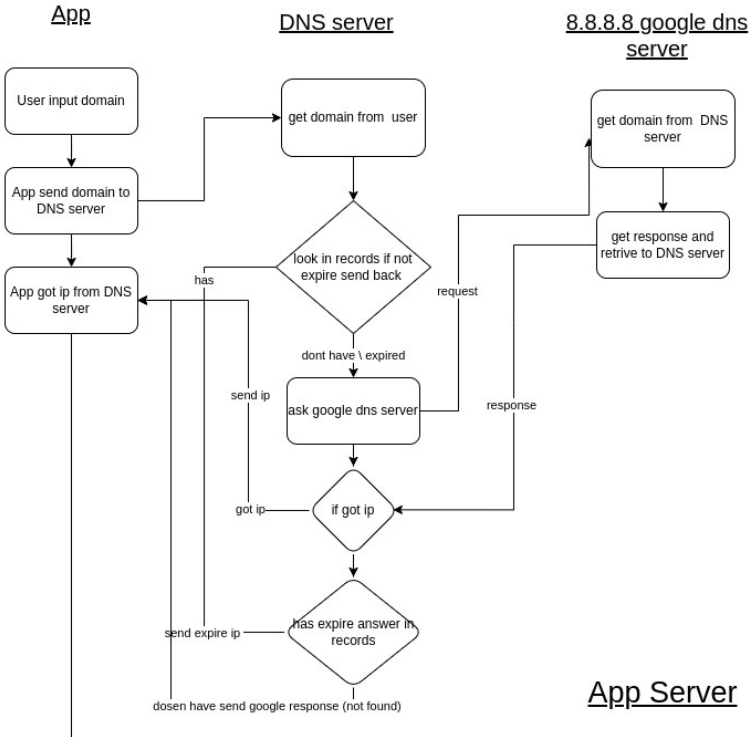
השרת מקבל את הבקשה ובודק אם יש לו רשומה שלא פגה תוקף אם כן מחזיר תשובה.

אם אין רשומה כזאת אז שואל את 8.8.8.8 אם קיבל תשובה מחזיר ושומר רשומה חדשה, עם זמן פגות תוקף.

אם אין תשובה אז בודק אם יש לו רשומה פגת תוקף אז מחזיר ממנה תשובה, אם אין בכלל אז מחזיר שאין תשובה.

נשים לב למלבן הגדול שם יהיו כל ה log.

דיאגרמא



APP

סקירה של GUI

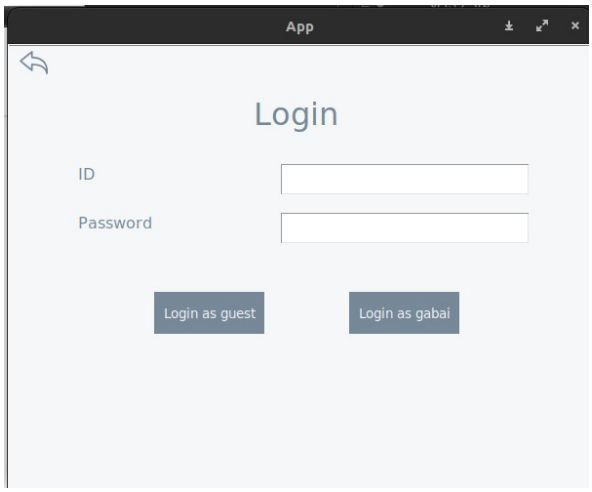
האפליקציה היא מערכת שיתופית של מידע על בתי כנסת, ככה שכוח של הציבור הוא זה שמייצר ומעדכן את המידע שאותו כולם יכולים לראות. כדי לייצר את זה יש שלושה סוגי חשבונות כאשר לכל אחד יש יותר הרשאות

לקוח: אדם פשוט שרק רוצה לראות בתי כנסת לעשות חיפושים ולצרוך את המידע.

גבאי: זה שם קוד לאדם שאחראי על בתי כנסת, הוא יכול להוסיף לשנות ולמחוק בתי כנסת משלו. כדי להיות גבאי יש ליצור קשר עם אדמין כדי לקבל משתמש של גבאי.

אדמין: רק חשבון אחד שהוא אחראי על הגבאים ועושה איתם סדר ככה שלא על אחד יכול להיות גבאי והוא משנה דברים שמשפיעים על כולם.

כדי להבדיל בין המשתמשים נגיע לחלון הבא

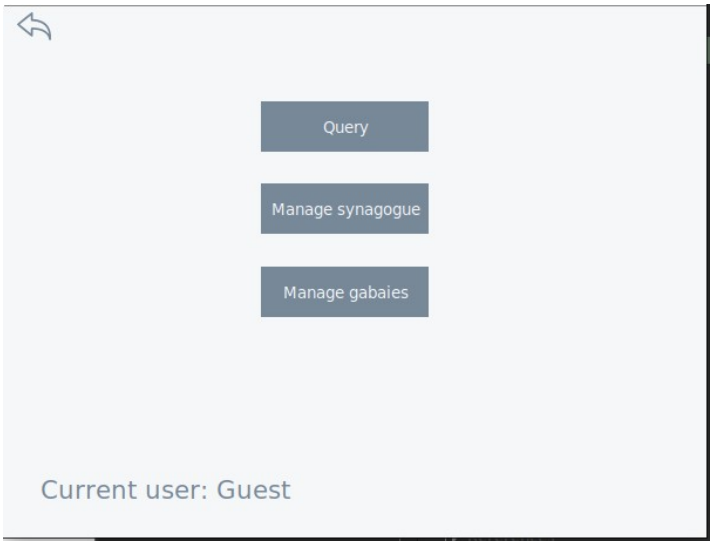


בחלון הזה נוכל או להיכנס כאורח, או לשים תז וסיסמא כדי להיכנס לחשבון שלך.

נדגיש שוב כמו שאמרנו ב"לאפס את הנתונים של השרת" חשבון האדמין הוא

ID =1, PASSWORD = 1243

אם כתבנו משהו לא נכון נקבל



עכשיו אחרי שהתחברנו נוכל להגיע למסך הראשי כאן יש את כל הפעולות שאפשר לעשות לחלקם יש הראשאות מסויימות ולחלקם לא.

כאן גם נוכל לראות איך התחברנו

Current user: Guest

Current user: oz

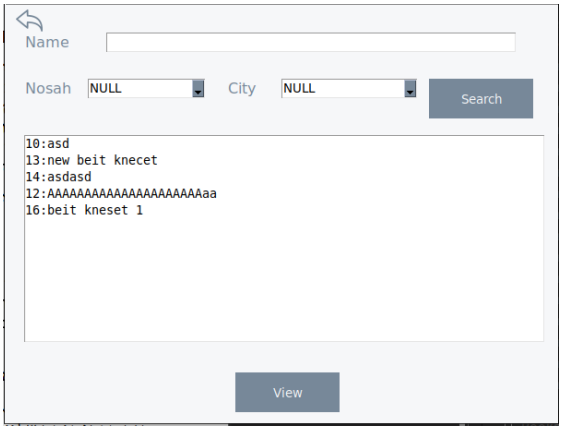
Current user: admin

לכל אחד הרשאות שונות, כמו שאמרנו. אדמין יש הרשאה להכל.

לגבאי אין הרשאה לשינוי גבאים

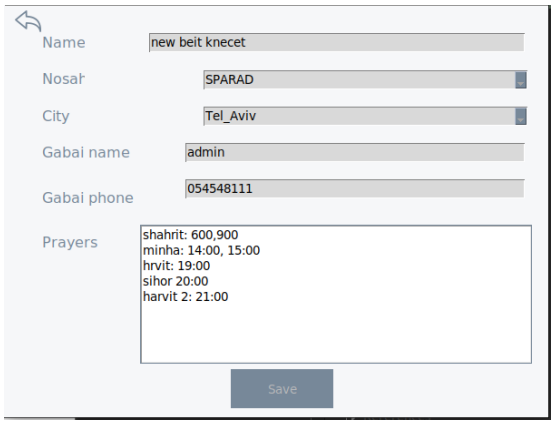
ואורח אין הרשאה גם לשיוני גבאים וגם לשינוי בתי כנסת

בחלון הראשון יש את QUERY



כאן נוכל לשלוח שאילתא שתעשה חיפוש בשרת ותחזיר רק את התשבות שרצינו. נוכל להגדיר סוג של נוסח שאנחנו רוצים, איזה עיר וגם אפשר לכתוב טקסט כאשר בית כנסת מכיל אותו אז הוא יבחר.בתי כנסת יוצגו במספר סידורי ואחריו שם הבית כנסת (כדי לא להתבלבל עם בתי כנסת עם אותו שם)

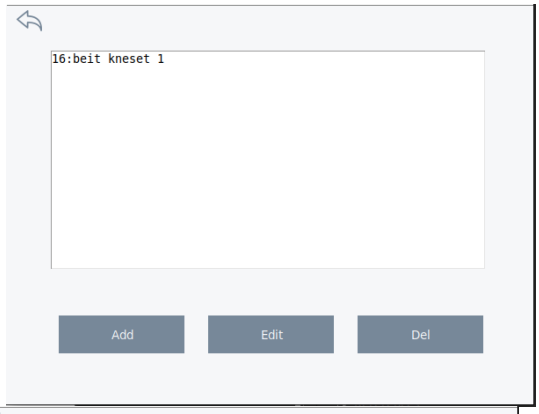
כמו שאפשר לראות אם לא נגדיר כלום הוא פשוט יביא את כולם.



אם נלחץ על הכפתור VIEW רואים את כל המידע על הבית כנסת כולל מידע על הגבאי שלו. נראה שהכל אור כי אנחנו אורחים ולא אחראים על הבית כנסת ולכן אי אפשר לערוך בו כלום

עכשיו נכנס כOZ גבאי וככה נוכל גם לערוך בתי כנסת שלנו.

נכנס בעמוד הראשי אל MANAGE SYNG ושם נוכל לנהל את הבתי כנסת שלנו.



כאן נראה שלושה אפשרויות אחת להוסיף בית כנסת חדש שמקושר אלינו ורק אנחנו יכולים לשנות אותו.

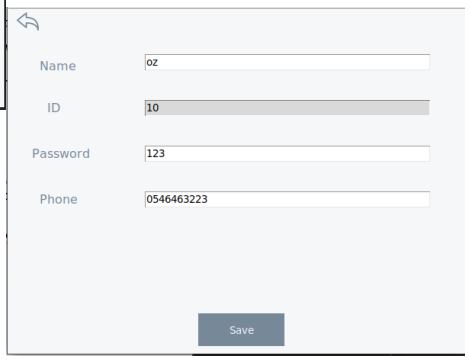
גם נוכל לערוך אחד קיים, וגם למחוק.

כל שינויים לבתי כנסת יהיו בחלון דומה למה שראינו מקודם, שרואים את כל פרטי הבית כנסת. נראה שלא משנים את הID כי זה מגיע משרת

אם נתחבר עכשיו במנהל נקבל עוד פונקצנליות של ניהול של הגבאים

כאן נוכל לראות את OZ שהתחברנו מקודם דרכו, נוכל להוסיף לשנות ואפילו למחוק שיגרום למחיקת כל הבתי כנסת שלו.

מסך העריכה יראה לנו עוד פרטים שנוכל לשנות חוץ מID שכמו מקודם מגיע מהשרת וחשוב אצלו הסדר. כמו כן ליד כל שם מופיע הID כי יכול להיות כמה עם אותו שם.



מאחורי קלעים אפליקציה

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Status_code          | kind|  nosah|       city      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                              data                             |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

כדי לעשות את הבקשות ותגובות הקשורות לאפליקציה בנינו פקאטה מתואמת ככה נוכל לכתוב מידע בצורה מדוייקת (יותר מגניב) ולא פשוא הכל בטקסט הפקטאה נראית ככה:

הכל כתוב בקובץ Backend/Help/app_packet.py

הסטטוס הוא עוזר באם הכל בסדר או יש בעיה כל שהיא ואם כן אז מה הבעיה הספציפית
גם מתפקד כריפוד

אחר כך יש את KIND שהוא מגדיר לנו איזה סוג של בקשה זאת, או שזה בכלל תגובה

יש את nosah ואת city שכמו שאמרנו קודם בQUERY נרצה לשלוח שאילתא עם נוסח או עיר מסויימים וככה נוכל לשלוח.
ואז DATA שהוא המקום הדינאמי שם נשלח דברים רבים שמשתנים בהתאם אל KIND

REQUEST_LOGIN = 0 -> id,password
REQUEST_BY_QUERY = 1 -> str to search
REQUEST_SYNG_BY_ID = 2 -> id
REQUEST_ALL_GABAI = 3 -> None
REQUEST_GABAI_BY_ID = 4 -> id
SET_SYNAGOGUE = 5 -> Synagogue
SET_GABAI = 6 -> Gabai
RESPONSE = 7 -->| Synagogue| Gabai| ids| None

לדוגמא אם נרצה לשלוח בקשה להתחברות אז נשלח את הפקאטה:

```
AppHeader(0, Kind.REQUEST_LOGIN.value, Nosah.NULL, City.NULL,(mId + ',' +password).encode())
```

אם נרצה לדוגמא לערוך בית כנסת אז נשלח
```
AppHeader(0, Kind.SET_SYNAGOGUE.value, Nosah.NULL, City.NULL, str(syng).encode())
```

אז כדי לטפל בכל הבקשות והתגובות יש שתי מחלקות אחת Backend/Help/Handleclient.py שהיא אחראית על כל בקשות של הלקוח ולכתוב אותה כפקאטה

ויש את מחלקה Backend/Help/Handelserver.py שהוא אחרי לקבל את הבקשה ולהחזיר את התשובה המתאימה, הוא גם אחראי לקחת את המידע על בתי כנסת וגבאים לקחת ממנו מידע \ לשנו אותו.

כמובן כל זה מדבר רק על האפליקציה ועל העברת המידע שלה אבל איך אנחנו מעבירים את המידע זה יגיע בפרק הבא RUDP

RUDP

כדי ליצור חיבור אמין בין שרת לאפליקציה אז צריך (TCP :( ), סתם נצטרך להתמודד עם **איבוד** פקאטות וגם עם **השחתה** של פקאטות. לכן בנינו את הפקאטה הבאה שנמצאת בקובץ Backend/RUDP/rudp_packet.py, קודם כל נעבור על שדה שדה ונסביר אותו.

Seq זה הבית הראשון שנרצה לשלוח, אנחנו מעדכן אותו בכל שליחה seq = seq + len(data) אפשר לראות שפקאטות שהם רק ACK לא ישנו את seq

Ack זה הבית הבא שאתה מצפה לקבל אנחנו נעדכן את זה אחרי כל קבלה נכולה של מידע ack = is.seq + len(data) כאן משתמש בפקאטה שהגיע

length זה האורך של הפקאטה כולל הDATA כאשר מינימאלי זה 16 בתים

Checksum כאן נשמור את הChecksum של הפקאטה באופן הבא. קודם נגדיר חבילה לשלוח כאשר checksum =0 עכשיו נפעיל את הפונקציה על הפקאטה בצורת בתים, עכשיו נשים את הערך שקיבלנו ונוכל לשלוח את הפקאטה. כאשר נקבל את הפקאטה נפעיל את הפונקציה ללא checksum ונשווה אליו.

SYN זה נועד לפתיחת קשר, בתחילת האפליקציה היא תשלח בקשה SYN השרת יחזיר SYN|ACK ואז ACK (נשמע מוכר)

ACK זה נועד להגדיר שהפקאטה היא מכירה בפקאטה אחרת שהתקבלה

PUSH זה נועד להגדיר סוף של סגמנתים, שאם נשלח מידע בכמה סגמנטים אז כאשר רואה את זה הוא יודע שזה הסוף ומאחד את המידע למשהו קריא.

FIN זה נועד לסגירת קשר, גם סגירה של החלון האפליקציה או סגירה עם C^ ישלח FIN ויכחה ל FIN ACK מהשרת ויחזיר ACK יש טיימר על זה ככה שהשרת \ לקוח לא ישראו לעד מחכים אלא יצאו

WIN_SIZE זה נועד בשביל FLOW CONTROL ככה כל צד שולח כמה הוא יכול לקבל ולפי זה נשלח.

DATA כאן בפועל תיהיה בפקאטה של האפליקציה או לפחות חלק ממנה אם נראה לנכון

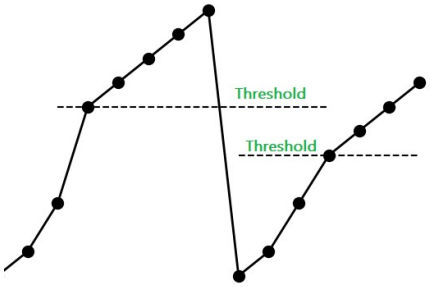
איבוד

כדי לטפל באיבוד נשתמש בACK ככה שכל פעם שנשלח משהו אנחנו נחכה זמן מוגדר על TIME OUT ברגע שקורה אז נשלח שוב עד שבסופו של דבר נקבל פקאטה שהיא המתאימה ואז נוכל להמשיך לשליחה הבאה

השחתה

כמו שראינו בהסבר על Checksum אנחנו מגדירים לכל פקאטה ששולחים ואז בקבלה אנחנו יכולים לראות אם השתבש במהלך התעבורה שלה.

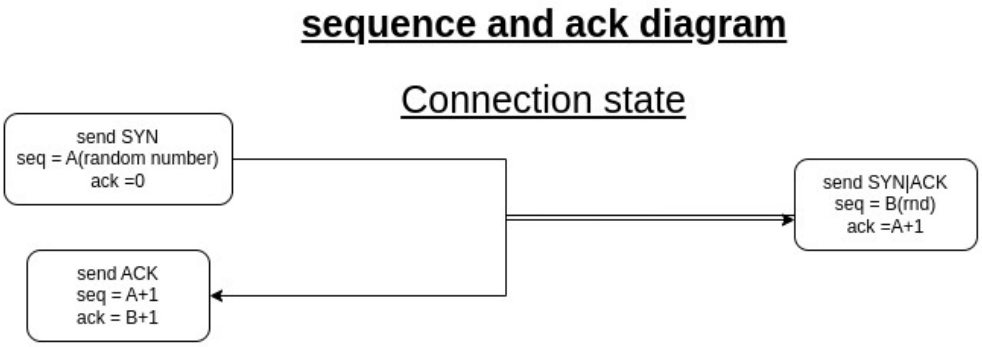
CC



כדי לטפל בCC יצרנו את המחלקה Backend/RUDP/CC.py לכל צד יש כזה והוא מתפעל את שינוי של CC, האלגוריתם פשוט יש לו שתי מצבים slow start כל קבלה אז מכפיל ב2 וaimd זה מעלה ב 1 , מוגדר לנו ערך התחלה שממנו מתחילים ואז עולים עד שמגיע TIMEOUT ברגע זה נוריד לערך התחילתי ואז נגדיר את הסף להיות חצי ממה שהיה עכשיו. הוא מתחיל ב SLOW START עד שיגיע לסף ומשם AIMD וחוזר חלילה

דיאגרמת שליחת מידע פתיחת קשר

בהתחלה פתיחת קשר כמו שאמרנו, נראה גם את העדכונים של SEQ ו ACK




```
client syn:RUDP_Header( seq_num=958449, ack_num=0, total_length=16, checksum=24550, SYN=True,ACK=False, PUSH=True, FIN=False, win_size=65535)
server syn|ack : RUDP_Header( seq_num=589219, ack_num=958450, total_length=16, checksum=25141, SYN=True,ACK=True, PUSH=True, FIN=False, win_size=65535)
client ack:RUDP_Header( seq_num=958450, ack_num=589220, total_length=16, checksum=25140, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
```

20215	30381	UDP	58	20215 → 30381	Len=16
30381	20215	UDP	58	30381 → 20215	Len=16
20215	30381	UDP	58	20215 → 30381	Len=16

אם קורה שהשרת לא מגיב אל SYN אז הלקוח ישלח שוב ושוב עד שיעברו 10 שניות ואז יפסיק אפשר לראות ע"י הזמן שבצד

12	5.992810033	127.0.0.1	127.0.0.1	20215	30381	UDP
13	5.992840296	127.0.0.1	127.0.0.1	20215	30381	ICMP
14	6.493496671	127.0.0.1	127.0.0.1	20215	30381	UDP
15	6.493506253	127.0.0.1	127.0.0.1	20215	30381	ICMP
16	6.994069831	127.0.0.1	127.0.0.1	20215	30381	UDP
44	14.004029863	127.0.0.1	127.0.0.1	20215	30381	UDP
45	14.004062110	127.0.0.1	127.0.0.1	20215	30381	ICMP
46	14.504673746	127.0.0.1	127.0.0.1	20215	30381	UDP
47	14.504684095	127.0.0.1	127.0.0.1	20215	30381	ICMP
48	15.005492424	127.0.0.1	127.0.0.1	20215	30381	UDP
49	15.005526034	127.0.0.1	127.0.0.1	20215	30381	ICMP

אם קרה ששלח SYN וחזר SYN ACK אז הלקוח יחזיק ACK לא משנה אם יגיע או לא עדיין הלקוח יניח שהשרת פתח את הקשר ומקשיב לו וישלח בקשות והשרת יניח שקיבל את SYN ACK אחרת היה שולח שוב

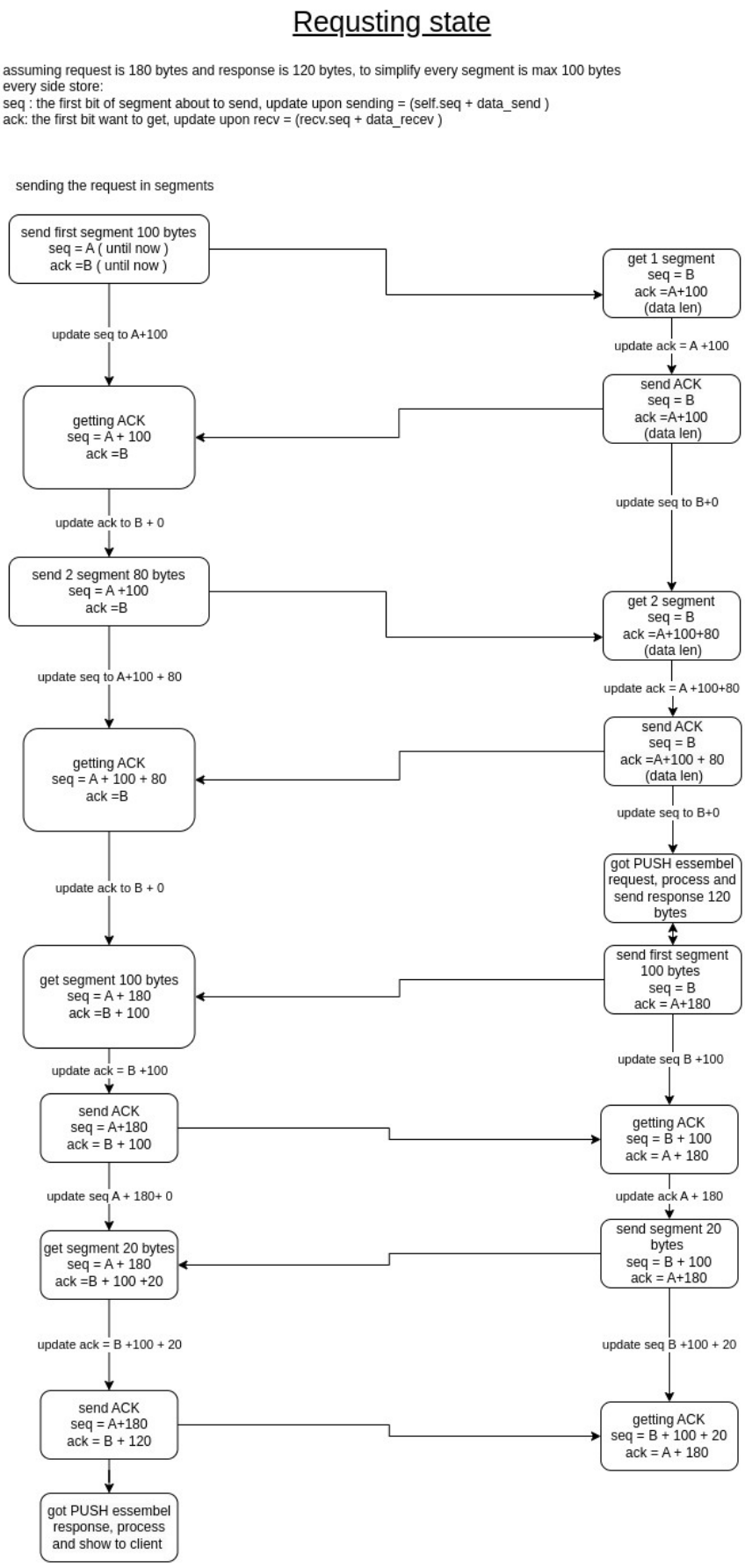
דיאגרמת שליחת מידע בקשה

כאן נוכל לראות דיאגמאה של שליחה של בקשה בשתי חלקים ולקבל תשובה בשתי חלקים.

נראה בדיאגרמה שעל כל שליחה יש חזרה של ACK בתצורה של STOP AND WAIT ככה שמחכה ושולח שוב עד שיקבל תשובה טובה.

אז לא משנה איזה בעיה קפצה פקאטה לא הגיעה הגיעה באיחור הגיעה מושחתת עדיין קורה אותו דבר שולחים ומחכים לאישור נכון של השליחה ואז שולחים את הבא בתור

כמו שרואים בדיאגמרה כדי לזהות פקאטה ישנה אז אפשר לזהות לפי seq,ack כאשר יוצא ש my.seq > is.ack כלומר אם בפעם הראשונה ACK ישן של A ולא A+100 אז האפלקציה תדחה אותו




```
client: 10 : RUDP_Header( seq_num=60969, ack_num=270513, total_length=26, checksum=23151, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
server ack: 0 : RUDP_Header( seq_num=270513, ack_num=60979, total_length=16, checksum=6170B, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65525)
server : 80 : RUDP_Header( seq_num=270513, ack_num=60979, total_length=96, checksum=15732, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65535)
client ack: 0 : RUDP_Header( seq_num=60979, ack_num=270593, total_length=16, checksum=61698, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65455)
server : 70 : RUDP_Header( seq_num=270593, ack_num=60979, total_length=86, checksum=44547, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
client ack: 0 : RUDP_Header( seq_num=60979, ack_num=270663, total_length=16, checksum=61698, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65385)
ensemble segment total length : 150
```

איך מחלקים לסגמנטים

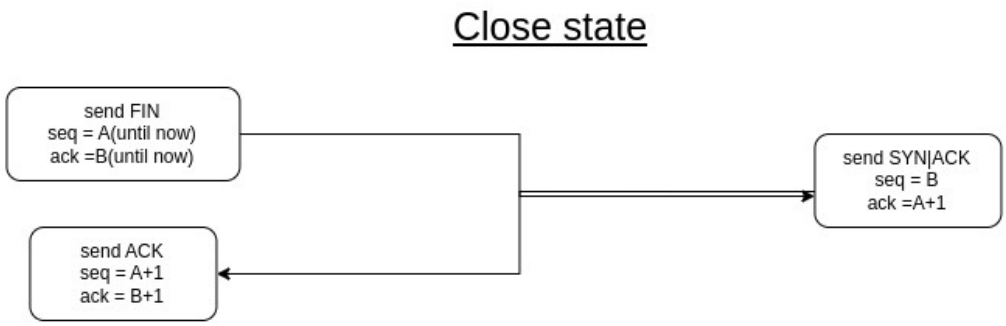
הראנו בכמה מקומות ששליחה של מידע יכול להיות בחתיחות משתנות, אז איך בפועל הם מתחלקות?

כמו שאמרנו קודם התצורה היא של STOP AND WAIT אז הוא שולח כמות בתים מחכה לתשובה ואז מחשב שוב כמה לשלוח ושולח שוב. אז בפועל זה מתחלק בין 4 פרמטרים שונים.פרמטר קבוע שהוא גודל פקאטה מקסימאלית,כמות המידע שהמקבל יכול לקבל, בקרת עומסים, וגודל של המידע שנשאר לשלוח.

ככה נגדיר את הסגמנט נשלח אותו ונחכה לשובה כמו שכבר עברנו הרבה פעמים

דיאגרמת שליחת מידע סגירת קשר

מצב זה מגיע מסגירת באפליקציה.




```
client fin:RUDP_Header( seq_num=157208, ack_num=196021, total_length=16, checksum=39958, SYN=False,ACK=True, PUSH=True, FIN=True, win_size=65535)
server fin|ack : RUDP_Header( seq_num=196021, ack_num=157209, total_length=16, checksum=39957, SYN=False,ACK=True, PUSH=True, FIN=True, win_size=65535)
client ack:RUDP_Header( seq_num=157209, ack_num=196022, total_length=16, checksum=39957, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
App closed
```

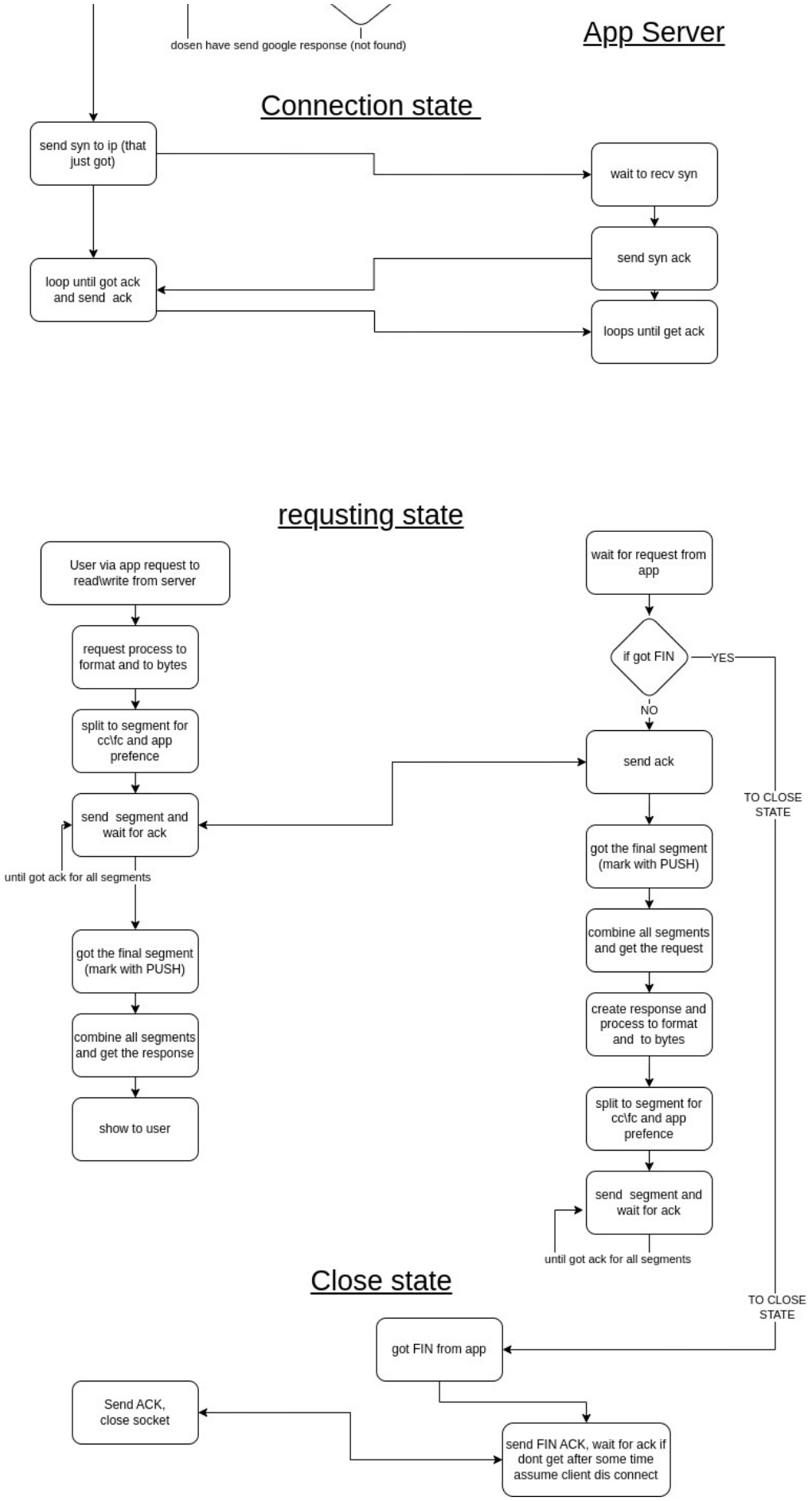
20215	30381	UDP	58	20215 → 30381 Len=16
30381	20215	UDP	58	30381 → 20215 Len=16
20215	30381	UDP	58	20215 → 30381 Len=16

נראה את הסגירה המשולשת שליחת FIN ואז מחזיר FIN ACK והוא מחזיר ACK.
גם פה אם הלקוח שולח FIN ואם אין תשובה הוא פשוט יסגור את הקשר

35	346.468880911	127.0.0.1	127.0.0.1	20215	30381	UDP	58	20215 → 30381 Len=16
36	346.468916432	127.0.0.1	127.0.0.1	20215	30381	ICMP	86	Destination unreachable (Port unreachable)
37	346.969763095	127.0.0.1	127.0.0.1	20215	30381	UDP	58	20215 → 30381 Len=16
38	346.969793462	127.0.0.1	127.0.0.1	20215	30381	ICMP	86	Destination unreachable (Port unreachable)
39	347.470476830	127.0.0.1	127.0.0.1	20215	30381	UDP	58	20215 → 30381 Len=16
40	347.470506694	127.0.0.1	127.0.0.1	20215	30381	ICMP	86	Destination unreachable (Port unreachable)

אם קרה שהשרת קיבל את FIN ושלח את FIN ACK ולא קיבל תשובה ישלח שוב ואחרי 3 פעמים הוא יתיאש ויניח שהלוח פשוט סגר את הקשר.

דיאגרמאת סיכום של אפליקציה



ווירשרק

DHCP בקשת IP חדש

1	0.0000000…	0.0.0.0	255.255.255.254	DHCP	286	DHCP Discover	- Transaction ID
2	0.7354178…	0.0.0.0	255.255.255.254	DHCP	295	DHCP Offer	- Transaction ID
3	1.1079067…	0.0.0.0	255.255.255.254	DHCP	286	DHCP Request	- Transaction ID
4	1.8400621…	0.0.0.0	255.255.255.254	DHCP	313	DHCP ACK	- Transaction ID

דו שיח רגיל של DHCP

DHCP חידוש IP

17.861690…	0.0.0.0	255.255.255.254	DHCP	286	DHCP Request	- Transaction ID
18.213789…	0.0.0.0	255.255.255.254	DHCP	311	DHCP ACK	- Transaction ID

רואים שנשלח רק בקשת request ו ack כי discover לא נצרך

שליחה DNS UPTODATE

1	0.000000000	127…	127…	7654	53	DNS	74	Standard query 0x0000 A theBestApp.com
2	0.028307638	127…	127…	53	7654	DNS	109	Standard query response 0x0000 Name exists URI theBestApp.com UR

```
▸ Frame 2: 109 bytes on wire (872 bits), 109 bytes captured (872 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 53, Dst Port: 7654
▾ Domain Name System (response)
    Transaction ID: 0x0000
  ▸ Flags: 0x8106 Standard query response, Name exists
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ theBestApp.com: type URI, class Unknown
        Name: theBestApp.com
        [Name Length: 14]
        [Label Count: 2]
        Type: URI (256)
        Class: Unknown (0x000e)
  ▾ Answers
    ▾ theBestApp.com: type URI, class Unknown
        Name: theBestApp.com
        Type: URI (256)
        Class: Unknown (0x000e)
        Time to live: 206 (3 minutes, 26 seconds)
```

כאן אפשר לראות שליחה אחת וחזרה משרת DNS נראה שקיבלנו תשובה אחת לא מהימנת ויש TTL שהוא קשור לזמן שלו בטבלה. מתחת 5 דקות ומעל 0 ולכן UP_TO_DATE

שליחה DNS NOT UP_TO_DATE

נשלח משהו ש8.8.8.8 יחזיר לנו תשובה

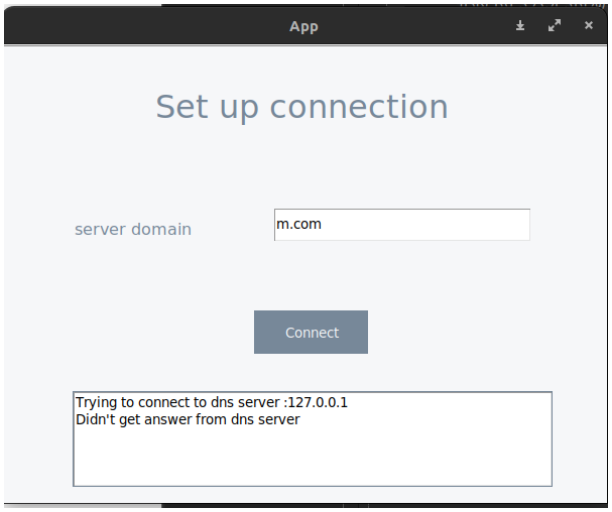
1	0.000000000	7654	53	127.0.0.1	127.0.0.1	Standard query 0x0000 A play.google.com
2	0.101358930	20580	53	10.9.8.108	8.8.8.8	Standard query 0x0000 A play.google.com
3	0.107241657	53	20580	8.8.8.8	10.9.8.108	Standard query response 0x0000 A play.google.com A 172.217.22.110
4	0.169393612	53	7654	127.0.0.1	127.0.0.1	Standard query response 0x0000 A play.google.com A 172.217.22.110

נראה קבלה של DOMAIN לא מצא אז שואל את גוגל ואז מקבל תשובה ואותה מחזיר

שליחה ואין תשובה בכלל

1	0.000000000	7654	53	127.0.0.1	127.0.0.1	Standard query 0x0000 A m.com
2	0.117720014	21071	53	10.9.8.108	8.8.8.8	Standard query 0x0000 A m.com
3	0.188363582	53	21071	8.8.8.8	10.9.8.108	Standard query response 0x0000 No such name A m.com SOA a.gtld-ser
4	0.242495474	53	7654	127.0.0.1	127.0.0.1	Standard query response 0x0000 No such name A m.com SOA a.gtld-ser

רואים ששולח לגוגל וגוגל אומר לו שאין תשובה ואת חוסר את התשובה הזאת מחזיר את האפליקציה



שליחה וקבלה הכל טוב

1	0.000000000	127…	127…	7654	53	DNS	74	Standard query 0x0000 A theBestApp.com
2	0.028307638	127…	127…	53	7654	DNS	109	Standard query response 0x0000 Name exists URI theBestAp
3	0.059923611	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16
4	0.060069757	127…	127…	30381	20215	UDP	58	30381 → 20215 Len=16
5	0.060234856	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16
6	0.091361956	127…	127…	53	7654	DNS	109	Standard query response 0x0000 Name exists URI theBestAp
7	4.602956333	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16
8	4.603227326	127…	127…	30381	20215	UDP	58	30381 → 20215 Len=16
9	4.603546809	127…	127…	20215	30381	UDP	68	20215 → 30381 Len=26
10	4.603749042	127…	127…	30381	20215	UDP	58	30381 → 20215 Len=16
11	4.604200098	127…	127…	30381	20215	UDP	232	30381 → 20215 Len=190
12	4.604527510	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16
13	6.894946642	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16
14	6.895212637	127…	127…	30381	20215	UDP	58	30381 → 20215 Len=16
15	6.895564830	127…	127…	20215	30381	UDP	58	20215 → 30381 Len=16

נוכל לראות את שליחת בקשה של DNS של theBestApp.com וקבלת תשובה מיידית גי נמצא ברשומות ולא עבר זמנו נוכל לדעת לפי זה שלא דיבר בכלל עם 8.8.8.8 , נתעלם בהכפול זה בגלל שמהזין על LO.

אחרי קבלה של כתובת אפשר לראות את הפתיחת קשר ואז מעבר של 6 פקאטות ואז סגירת קשר

```
----------------------------connection state----------------------------
client syn:RUDP_Header( seq_nun=226189, ack_num=0, total_length=16, checksum=35925, SYN=True,ACK=False, PUSH=True, FIN=False, win_size=65535)
server syn|ack : RUDP_Header( seq_nun=678551, ack_num=226190, total_length=16, checksum=12719, SYN=True,ACK=True, PUSH=True, FIN=False, win_size=65535)
client ack:RUDP_Header( seq_nun=226190, ack_num=678552, total_length=16, checksum=12726, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
----------------------------request state----------------------------
client: 0 : RUDP_Header( seq_nun=226191, ack_num=678552, total_length=16, checksum=12727, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65535)
server ack: 0 : RUDP_Header( seq_nun=678552, ack_num=226191, total_length=16, checksum=12727, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65535)
client: 10 : RUDP_Header( seq_nun=226191, ack_num=678552, total_length=26, checksum=39705, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
server ack: 0 : RUDP_Header( seq_nun=678552, ack_num=226201, total_length=16, checksum=12727, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65525)
server : 174 : RUDP_Header( seq_nun=678552, ack_num=226201, total_length=190, checksum=32730, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
client ack: 0 : RUDP_Header( seq_nun=226201, ack_num=678726, total_length=16, checksum=12717, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65361)
ensemble segment total length : 174
----------------------------close state----------------------------
client fin:RUDP_Header( seq_nun=226201, ack_num=678726, total_length=16, checksum=12714, SYN=False,ACK=True, PUSH=True, FIN=True, win_size=65361)
server fin|ack : RUDP_Header( seq_nun=678726, ack_num=226202, total_length=16, checksum=12539, SYN=False,ACK=True, PUSH=True, FIN=True, win_size=65535)
client ack:RUDP_Header( seq_nun=226202, ack_num=678727, total_length=16, checksum=12713, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65361)
App closed
```

שליחה וקבלה עם איבוד

שלחנו בקשה לא קיבלנו תשובה, שלחנו שוב ואז הכל עבד טוב. כמובן פלט האפליקציה הוא על מה שעובר בפועל

נראה שיש להם אותו מידע בדיוק

1	0.000000000	20215	30381	127.0.0.1	127.0.0.1	20215 → 30381	Len=26
2	0.000351952	20215	30381	127.0.0.1	127.0.0.1	20215 → 30381	Len=26
3	0.000636718	30381	20215	127.0.0.1	127.0.0.1	30381 → 20215	Len=16
4	0.502059759	30381	20215	127.0.0.1	127.0.0.1	30381 → 20215	Len=190
5	0.502505024	20215	30381	127.0.0.1	127.0.0.1	20215 → 30381	Len=16

```
Data (26 bytes)
    Data: 000eaf250003c06f001afc5f0006fd4700000000312c31323433
    [Length: 26]
```

```
Data (26 bytes)
    Data: 000eaf250003c06f001afc5f0006fd4700000000312c31323433
    [Length: 26]
```

```
client: 10 : RUDP_Header( seq_nun=962341, ack_num=245871, total_length=26, checksum=64607, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=64839)
server ack: 0 : RUDP_Header( seq_nun=245871, ack_num=962351, total_length=16, checksum=36923, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=65535)
server : 174 : RUDP_Header( seq_nun=245871, ack_num=962351, total_length=190, checksum=56936, SYN=False,ACK=True, PUSH=True, FIN=False, win_size=65535)
client ack: 0 : RUDP_Header( seq_nun=962351, ack_num=246045, total_length=16, checksum=37619, SYN=False,ACK=True, PUSH=False, FIN=False, win_size=64665)
ensemble segment total length : 174
```

שאלות

1.הבדלים בין QUIC ל TCP

א. TCP דורש לחיצת יד משולשת לעומת ה QUIC שבנוי על udp אז אינו צריך לחיצה משולשת ואז זה חוסך זמן ותעבורה.

ב.QUIC גם משפר את הביצועים במהלך אירועי החלפת רשת, כגון כאשר משתמש במכשיר נייד עובר מרשת Wi-Fi לרשת סלולרית. כאשר אותו דבר קורה ב-TCP, מתבצע תהליך ארוך שבו כל חיבור קיים מנותק אחד בכל פעם ולאחר מכן נוצר מחדש לפי דרישה. כדי לפתור בעיה זו והשלכותיה כאשר מבחינת ביצועים, QUIC כולל מזהה חיבור למקלט ללא קשר למקור. זה מאפשר ליצור מחדש את החיבור פשוט על ידי שליחת חבילה אחת מחדש, שתמיד מכילה את המזהה הזה ושתיחשב תקפה על ידי המקבל גם אם כתובת ה-IP של השולח השתנתה.

ג.אבטחה: עם TCP+TLS, כל HEADER ה-TCP חשוף וניתן למעשה לשנות (ולפגוע),בהשוואה ל-QUIC שמצפין כמעט את כל הHEADER פרט לכמה פריטים (פורטים וכו,).

ד.Multiplexing:ריבוי חשוב מאוד מכיוון שהוא מונע חסימת ראש קו . חסימה קורה היא כאשר אתה מבקש מספר אובייקטים, ואובייקט קטן נתקע בגלל שאובייקט גדול קודם התעכב. על ידי שימוש במספר זרמים, מנות אבודות משפיעות רק על הזרם הספציפי הזה. לכן, QUIC אכן מקטין באופן משמעותי את החסימות אך לא לחלוטין.הבדלים בין

2.הבדלים בין Vegas לבין CUBIC

א.Vegas מדגישה עיכוב פקטות, ולא אובדן פקטות, כדרך לקביעת הקצב שבו לשלוח פקטות. בניגוד ל-CUBIC שמזהה עומס רק באמצעות נפילת מנות,Vegas מזהה עומס בעזרת עליה בערכי RTT . לפיכך, וגאס מודעת לעומס ברשת לפני שמתרחש אובדן מנות

ב.Vegas העליה שלו היא לינארית לעומת cubic שהעליה שלו אינה לינארית ומשתנה בהתאם לאם עברנו את הרף הקודם או לא
.

3 פרוטוקול BGP וההבדל בינו לבין OSPF
לכל AS לניהול רשת באינטרנט מקוצה מספר יחודי. כל AS מהווה צומת לניתוב תעבורה שמשתמש בפרוטוקול BGP כדי לבנות מסלולי ניתוב וכדי להתממשק מול AS-ים אחרים.

כדי לבנות מסלולים, כל AS מפרסם את המספר שלו ורשימת הכתובות שבאחריותו שהוא יכול להעביר אליהם ישירות או באמצעות שכניו.כל שכן מקבל את ההודעה,שומר את המידע ומשקלל אותו עם המסלולים ששמורים אצלו. וכן הלאה. .כדי להימנע ממעגלים, AS-ים מפילים הודעות BGP שמכילות את המספר שלהם של עצמם.

בחירת המסלול המיטבי נעשית באופן נפרד בכל AS, ורק המסלול המיטבי מפורסם לשכנים. תהליך זה נעשה באמצעות התחשבות בהרבה גורמים כמו כמות הAS-ים בדרך ליעד מסוים או מהירות התקשורת בין שכנים שונים. עם זאת, ההחלטה נקבעת בדרך אחרת בכל AS בפני עצמו. דבר זה יכול לגרום לכך שמסלול השליחה ומסלול הקבלה של חבילת מידע לא תמיד יהיו זהים. ולכן הוא לא הולך דווקא לפי מסלולים הכי קצרים

ההבדל בין OSPF לבין BGP זה שB הוא בין AS לעומת OSPF שהוא פרוטוקול ניתוב בתוך AS.

.4
נניח את השורה הראשונה

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
שליחת פתיחת קשר	20215	30381	10.0.0.10	10.0.0.11	:5098:fa:9b:ceb:71:13:e5:20	e:78:48
שליחת קבלת פתיחת קשר	30381	20215	10.0.0.11	10.0.0.10	98:fa:9b:ce:78:48eb:71:13:e5:20	:50

כתובות המק משתנות בין כל NODE ככה שהערכים משתנים בין כלnode

nat
אם השרת והסרבר באותו מקום אז אין בעיה ולא צריך NAT אבל אם הם לא באותה סרבר היינו צריכים NAT כדי להמיר את הכותבת I הפנימיות לחיצוני.אם נשתמש בQUIC גם אם נשנה את הip שלנו השרת עדיין יזהה אותנו אבל הערכים בטבלה לא ישתנו

.5
ARP הוא פרוטוקול שמטרתו לאתר כתובת MAC לכתובת IP המתאימה הפרוטוקול הוא ברמת הלינק.

DNS הוא פרוטוקול המאפשר לתרגם דומיינס לכתובות IP עוזר למשתמשי האינטרנט להגיע לכתובות אינטרנט בקלות כי קשה לאנשים לזכור מספרים אבל שמות קל יותר, DNS הוא ברמת התעבורה,DNS הוא מבוזר אז אם לא יהיה לו במטמון הוא יפנה אותך לשרת אחר.