



## תיק פרויקט

שם התלמיד: אביתר דמרי  
ת.ז. תלמיד: 206833113  
בית ספר: אורט חולון למדעים  
שם המנחה: יעל שחורי  
עוזר הוראה: ליאור גרנשטיין  
מועד הגשה:

נושא הפרויקט: פרוטוקול/תוכנה לשיתוף קבצים  
שם הפרויקט: Glomp



## תוכן עניינים

1.....	תיק פרויקט
2.....	תוכן עניינים
3.....	מבוא ורקע כללי
3.....	מבוא
3.....	שאלות שהמערכת עונה עליהן
4.....	תיאור המוצר המגומר
4.....	שם המוצר
4.....	קהל היעד
4.....	תיאור מפורט
4.....	השוואת העבודה עם פתרונות ויישומים קיימים
4.....	השוואה מול טורנט
4.....	השוואה מול אימיוול
4.....	השוואה מול Soulseek
5.....	מבט אישי על העבודה ועל תהליך הפיתוח
30.....	תיאור הממשק למשתמש
31.....	סביבת עבודה
31.....	שפת התכנות
31.....	פירוט סביבת העבודה והכלים הנדרשים לפיתוח
32.....	הוראות התקנה של הספריות החיצוניות וכיצד להריץ את הפרויקט
32.....	ספריות חיצוניות
32.....	הוראות התקנה
32.....	הוראות הרצה ותפעול
33.....	תיאור ארכיטקטורה
33.....	סקירת מודולים
34.....	פירוט המחלקות והתהליכים
34.....	צד לקוח
37.....	דיאגרמה כרונולוגית
38.....	תיאור היבטי הסייבר בפרויקט
38.....	Socket
38.....	Threads
39.....	רכיבי המערכת
39.....	צד לקוח
39.....	צד שרת
40.....	פרוטוקול תקשורת
41.....	האלגוריתמים המרכזיים בפרויקט
42.....	ביבליוגרפיה



## מבוא ורקע כללי

### מבוא

הפרויקט שבחרתי לעבוד בו הוא מערכת המאפשרת שיתוף קבצים גדולים בין משתמשים באופן יעיל. המערכת בנויה בדומה ל Torrent, ומאפשרת הורדה מקבילה ממספר מקורות הורדה שונים כדי להשיג מהירויות גבוהות.

במערכת אין התערבות בתקשורת מצד השרת, אשר אחראי על איסוף וחיפוש התכנים.

הפרויקט הוא מעיין שילוב של מערכות קיימות כגון torrent, soulseek, emule וכדומה. בדומה למערכות דומות הוא מאפשר שליחה ושיתוף של קבצים ללא הגבלה בין משתמשים, אך גם מאפשר לחפש קבצים בשרת על מנת לשתף אותם.

### שאלות שהמערכת עונה עליהן

- איך ניתן לשלוח תוכן לאדם אחר מבלי התערבות של שרת חיצוני?
- איך ניתן לשלוח תוכן לאדם אחר ללא הגבלת גודל או סוג התוכן?



## תיאור המוצר המגומר

### שם המוצר

Glomp

### קהל היעד

אנשים שרוצים פלטפורמה לשיתוף היצירות שלהם/שיתוף קבצים בכללי.

### תיאור מפורט

התכנה תאפשר למשתמשים לחפש קבצים (דרך שרת) ולהוריד אותם ממשתמשים אחרים. כל קובץ שהורד מאפשר לאחרים להוריד דרך אותו אדם את אותו הקובץ.

השרת לא יתערב בהורדה עצמה ולמעשה רק ישמש כדי ליצור מיפוי בין משתמשים המחזיקים בקבצים לתיאור של הקובץ(שם, גודל, תגיות וכ"ו), וכמו כן יאפשר את פונקציונליות החיפוש של הקבצים בין משתמשים.

ברגע שמשתמש סוגר את התכונה, השרת יפסיק למפות אליו בקשות ממשתמשים.

ברגע שמשתמש פותח מחדש את התכונה, ההעלאות שהיו לו(במידה והופסקו בסגירת התוכנית) ימשיכו, והוא יחזור להעלות קבצים כרגיל. כמו כן, כל הורדה שנעצרה תחזור לעבוד.

## השוואת העבודה עם פתרונות ויישומים קיימים

### השוואה מול טורנט

- טורנט לא מאפשר חיפוש גלובלי של קבצים אלא מסתמך על טראקרים ומנועי חיפוש שונים לכך, בניגוד לפרויקט זה.
- טורנט מאפשר הוספה דינאמית של משתפים להעלאה, וזאת בגלל המעורבות של הטראקר בתהליך שיתוף הקובץ.

### השוואה מול אימיול

- אימיול, בדומה לפרויקט, מאפשר חיפוש גלובלי של קבצים על פני כל המשתמשים.
- אימיול אינו מאפשר הורדה במקביל ממספר מקורות, דבר שהפרויקט הזה מאפשר.

### השוואה מול Soulseek

- בדומה לאימיול, Soulseek אינו מאפשר הורדה במקביל ממספר מקורות.



## מבט אישי על העבודה ועל תהליך הפיתוח

### דוח התקדמות 1 – 28.10.17

בתור התחלה רציתי קודם כל להוציא קוד פועל, שליחת קבצים בסיסית בין לקוח לשרת. הרעיון היה שבעתיד השרת אותו אני יוצר עכשיו יהווה חלק מתוך מנגנון ה-P2P. למעשה אין לו שום קשר לשרת אשר מתואר בהצעת הפרויקט.

מצאתי מספר ספריות שעזרו לי בתור התחלה בתוך הספריות המובנות בפיתון, ביניהם ספריה בשם socketserver, אשר מאפשרת ליצור שרתי תקשורת בצורה נורא פשוטה, ואפילו מאפשרים לעשות זאת במקביל עם thread-ים או עם תהליכים. זה נראה לי כמו הפתרון הפשוט ביותר, אז זה הכיוון שהלכתי אליו.

במחלקה המדוברת, ה"שרת" עצמו הוא בסך הכל אובייקט של מחלקה היורשת מ-2 מחלקות שונות. מחלקה אחת קשורה לסוג החיבור – TCP/UDP, והמחלקה השנייה קשורה להתנהגות של התקשורת(מקבילית/לא מקבילית ואיך זה מתנהל בהתאם לדרישות המערכת).

בנוסף, צריך להגדיר מחלקת Handler, אשר השרת שהוגדר לעיל משתמש בו כדי לנתח כל בקשה. כל מחלקת Handler צריכה להגדיר פעולת Handle אשר פועלת באופן אוטומטי בעת התחברות חדשה.

יצירת השרת פשוטה כמו `server = Server((addr, port), ServerHandler)`.

יצירת ה Handler היוותה למעשה את רוב העבודה בצד השרת, והגדרת פרוטוקול התקשורת לקח קצת זמן, אך בסופו של דבר התקבעתי על הפרוטוקול הבא:

בקשה מלקוח:

```
<title> <filename>\n
<header>: <value>\n
<header2>: <value2>
...
\n\n
```

כאשר `\n` היא שורה חדשה המפרידה בין שורות הבקשה. ההודעה מסתיימת עם `\n`. דומה מאוד ל HTTP רק עם פחות זבל מיותר כמו תו ה `\r` שקיים מסיבות היסטוריות וציון הגרסה שאין בו טעם בפרויקט הזה.

תגובה משרת:

```
<title>\n
<header>: <value>\n
<header2>: <value2>
...
\n\n
```

אחת ההתלבטויות הגדולות ביותר בהגדרת הפרוטוקול הייתה הדרך שבה יקבע סיום ההודעה. לצורך העניין, יש פרוטוקול אשר שולחים קודם את גודל ההודעה כדי לדעת כמה לקבל בצד השני, יש פרוטוקולים



ששולחים EOD(end of data) כדי לסמן מתי נגמרת שליחת ההודעה, וכן הלאה. בסופו של דבר התקבעתי על ה EOD כי זה נראה לי קצת מיותר לסבך את הפרוטוקול על ידי שליחת גודל גם לבקשות קטנות. מנגד, כאשר שולחים קובץ, יש חשש שה EOD יהיה בתוך הקובץ עצמו וזה יהרוס קצת דברים, אז במקרים כאלו יש צורך לשלוח את הגודל קודם כל ואז את המידע.

מאחר ורק השרת שולח קבצים, ההחלטה שלי הייתה להוסיף header להודעת התגובה של השרת המציין את גודל הקובץ שנשלח לאחר ההודעה. אז הפורמט הסופי של תגובה מהשרת הוא:

```
<title>\n
size: <data-size>\n
<header2>: <value2>
...
\n\n
<data>
```

למעשה זאת הדרך שבה השרת שולח מידע ללקוח בפרוטוקול HTTP, בתור "content-length". זאת נראית לי כמו הדרך הכי אפקטיבית להשיג את המטרה הרצויה.

פענוח ההודעה בצד השרת לא היה מאתגר, גיליתי לגבי פעולה במודול socket בפיתון הנקראת makefile. היא הופכת socket לדמוי קובץ – זה מאפשר לעשות בקלות פעולות כמו קריאת שורה. למרות היתרונות של דבר כזה, הבנתי בסוף שזה די מיותר ולא נחוץ. יותר קל פשוט לחפש את ה EOD בתוך ההודעה. בצד הלקוח העבודה הייתה קצת יותר מורכבת. היו לי מספר רב של משימות בתור התחלה, וזה הקשה עלי.

הייתי צריך לקבל קובץ מהשרת, לרשום את מהירות ההורדה שלו והזמן בו הוא צפוי לרדת. כמו כן היה עלי לדאוג למקרי קצה בהם הלקוח מתנתק בזמן קבלת הקובץ.

לגבי רשימת הפרטים על מהלך ההורדה אני הצלחתי למצוא שיטה פרימיטיבית יחסית להערכת זמן משוער לסיום ההורדה(לא ממש טובה, אבל עובדת בערך), והדפסת המהירות לא הייתה מעניינת מספיק כדי שאכתוב עליה. מה שכן היה מעניין הוא איך להדפיס לקונסול את הפרטים בצורה קריאה. בהנחה ואני אדפיס עבור כל chunk שמגיע מהשרת, זה ימלא את הקונסול במידע מבולגן ועמוס, ויהיה בלתי קריא. מצאתי דרך לעקוף את הבעיה הזאת. התו \n מחזיר את המצביע לתחילת השורה בקונסול, ולכן אני יכול פשוט להשחיל את התו הזה בסוף כל הדפסה כך שזה ימחק את השורה הקודמת במקום להציף בהרבה שורות. זה היה מספק לראות את זה עובד.

לגבי המשכת הורדת הקובץ לאחר התנתקות – חשבתי על זה זמן מה, אבל הפתרון האידיאלי אליו הגעתי כולל יצירת תיקיית הורדות זמנית שתאחסן את קובץ ההורדה הלא גמור, ובעת סיום ההורדה תעביר אותו לתיקייה בשם completed. מכאן המעבר הוא מאוד פשוט – לסרוק את תיקיית ה tmp, ולבקש מהשרת את הקובץ מה byte האחרון שירד בקובץ.



כמובן זה לא נגמר שם, כי השרת עכשיו צריך לתמוך בהורדה של קובץ חלקי. למען האמת זה לא היה קשה – לפני קריאת הקובץ פשוט להשתמש בפעולה seek של הקובץ כדי להתחיל את הקריאה מ byte ספציפי.

אחרי שהיה לי קוד עובד, בערך, התחילה העבודה האמתית של לעשות לכלל הקוד refactor, דוקומנטציה עשירה יותר ואולי חילוק יותר טוב למחלקות.

החלטתי להוסיף מחלקה בשם message המגדירה את 2 סוגי ההודעות הקיימות – בקשה ותגובה, ומאפשרת בקלות לתרגם את ההודעה לפי שדות – title, headers-dict. לעשות את זה היה קצת מעיק. כעת היה ניתן ליצור הודעה בצורה פשוטה דרך הבנאי של סוג ההודעה.

בשלב זה הבנתי שיש בעיה עם הקוד, כיוון שניתן ליצור הודעה ב-2 דרכים. דרך אחת היא דרך העברת הפרמטרים (title, headers) לבנאי, ודרך נוספת היא על ידי תרגום הודעה טקסטואלית.

בפיתון אין העמסת פעולות. דרך פשוטה ואלגנטית שמצאתי לפתור את זה היא פשוט להגדיר פעולה סטטית בשם from\_message המחזירה אובייקט הודעה לאחר התרגום שלו.

## בביליוגרפיה

- <https://docs.python.org/3/library/socketserver.html>
- <http://code.activestate.com/recipes/408859-socketrecv-three-ways-to-turn-it-into-recvall>
- [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- <https://docs.python.org/3/library/socket.html>



## דוח התקדמות 2 – 04.11.17

לאחר תקופה יחסית ארוכה שלא נגעת בקוד (מאחר ולא התאפשר לי... הייתי די עסוק), אני חוזר חזרה ומוצא את עצמי די מבולבל בנוגע למה קורה בפנים. ממבט כעת אני רואה דברים שנראים לי לא טוב.

הפרוטוקול עצמו שהגדרתי עובד ותקין, אבל יש מספר בעיות שאני כעת צריך להתמודד איתן.

בתור התחלה, התבקשתי להוסיף פונקציונליות של בקשת קובץ ממספר מקורות שונים, וזה עורר בעיה: כדי לבקש קובץ כעת אני לא צריך לדעת מה הגודל הנוכחי שלו. אני יכול להכניס את המיקום ממנו להתחיל את הבקשה אבל אני לא יכול לדעת מה גודל הקובץ מתוך הלקוח. כלומר- אני יכול לבקש חלק מקובץ בלי בעיה, אבל אני לא יכול לחלק את הבקשה שלי בהתאם לכמות המקורות השונים שיש להם את הקובץ.

במקור זו לא הייתה בעיה כיוון שלא תכננתי לתמוך בכך שבקשת קובץ תוכל לגשת למידע ממספר מקורות שונים. התכנון המקורי היה שכל קובץ יתקבל ממקור אחד בלבד, ולכן לא היה צורך לחלק את העבודה.

פתרון כדי לפתור את זה יכול להיות להחזיק קובץ שמכיל מידע לגבי קבצים... זה יכול להיות נוח, אבל קצת מכוער. זה פתרון שמעלה שאלות כמו האם אני צריך קובץ אחד כזה בסך הכל או האם יהיה קובץ כזה לכל קובץ שיוּרד. וכך או כך, מה יהיה הפורמט בו אני מאחסן את המידע על הקובץ?

בעיה נוספת שצצה לי כשהסתכלתי על הקוד כרגע היא שאין לי הפרדה טובה בין האינטראקציה עם המשתמש והמחלקה הראשית, client.py. הכל נראה מבולגן – ההדפסות והקלט בתוך הפעולות. זה כנראה יעשה לי חיים קשים כשאעשה מעבר ל GUI.

לצורך העניין – כתבתי את פעולת בקשת הקובץ כך שתדפיס את הפרטים על ההורדה (מהירות, זמן משוער). ההדפסה הזאת לא יכולה להתקיים מחוץ לפעולה כיוון שהיא צריכה מידע שקיים רק מקומית בתוך הפונקציה (גודל החלק שהגיע והזמן שעבר עד שהגיע).

פתרון לבעיה הזאת יכול להיות שפעולת הבקשה תקבל בנוסף לכך בתור פרמטר פונקציית הדפסה (או הצגה דרך GUI) ותשלח אליה את הפרטים הרלוונטיים להצגה. ממימוש לכך יכול להיעשות בעזרת coroutines – מעין גנרטור שמקבל אליו ערכים ומעבד אותם (בניגוד לאחד קלאסי שמחזיר "שרשרת ערכים").

דוגמה למימוש של משהו כזה יכול להיות בצורה נורא סקצ'ית ככה:

```
def print_data():
    ...
    while True:
        ...
        speed, tleft = (yield), (yield)

        print('dl speed: ' + str(speed))
        print('time left: ' + str(tleft))

def receive_file(..., out)
    ...
    for chunk in ...:
        out.send(speed)
        out.send(tleft)
```





בצורה כזאת אני יכול לשלוח את כל החלקים הרלוונטיים להדפסה/הצגה גרפית בצורה מרוחקת מהמחלקה החשובה.

עם כמה שזה מגניב ומרתק, נראה שזה גם בעייתי מאוד כיוון שלא תמיד צריך אני ארצה להציג את הפרטים של ההורדה. אולי אני אבדוק אם סופק out, ואם לא אז אין צורך לשלוח... בכל אופן אני צריך לחשוב על פתרון רציני לזה. אולי לנבור בקוד מקור של תוכנות אחרות.

לעת עתה אני אצור לי מעין GUI פשוט (CLI) רק כדי לעודד את עצמי לפתור את הבעיה.

מחשבה נוספת שיכולה להיות בעלת ערך היא הרעיון של מחלקת "בקשה". למעשה, הלקוח יהיה רק מסגרת שיוצרת בקשות חד פעמיות, ואותה בקשה תישלח רק לשרת ספציפי. דבר כזה יכול ליצור הפרדה יותר ברורה בין בקשה ממקור מידע אחד לאחר. למעשה בצורה כזאת, אפקטיבית, לא תהיה לי בקשת קובץ מלאה בשום מקום מאחר וקובץ מלא יתקבל ממספר מקורות שונים.

למען האמת אולי לקבצים קטנים אני אאפשר לקבל את הקובץ ממישהו אחד ספציפית(ראיתי את האופציה הזאת מופיעה בעקיפין בתוכנה בשם Soulseek לשיתוף קבצים – קבצים קטנים מקבלים העדפה אוטומטית ונשלחים ללקוח ורק אחר כך קבצים גדולים).

בכל אופן, אני כנראה אצטרך לנסח את המחלקה הזאת בצורה יותר אפקטיבית. היא יכולה אפילו להיות הרחבה של מודול ה message.py שיצרתי האחראי על תרגום טקסט בקשה למשהו שניתן לעבוד אתו(שדות מוגדרים בתוך אובייקט). כל שאצטרך לעשות זה להוסיף לזה קצת יותר פעולות.(יש לי תחושה שזה לא יעבוד טוב. בקשה היא לא הודעה, אבל יש לה הודעה בתוכה)

כל או כך, נראה שאני חייב לשכתב את כל הלקוח רק כדי שזה יעבוד בצורה יותר ברורה.

כמובן, קצת זמן במקלחת תמיד עוזר כדי לחשוב על תובנות. ואכן הצלחתי לפתור את אחת הבעיות העיקריות שלי: קבלת ההודעה במלואה מהשרת ללא חלק ההמשך שלה(הקובץ)!

הרעיון הוא להפוך את ה socket לדמוי קובץ. הכוונה היא שפעולות סטנדרטיות על קבצים יעבדו עליה, כמו readline. זה לא משהו שחשבתי עליו עכשיו, אבל כשחשבתי עליו טרם זה הגיע למבוי סתום כיוון שהפעולה readline קוראת עד שורה חדשה, אבל אני צריך לקרוא עד 2 שורות חדשות.

הפתרון שהגעתי עליו הוא פשוט להוסיף לפעולה readline תו נוסף אחד בסוף – זה יבטיח לקבל את התו של השורה החדשה במידה והוא התו מיד אחרי ה\n. אם הוא לא, זה פשוט יוסיף אותו ל buffer וזה לא ישפיע על קבלת ההודעה.

החלטתי להפוך כמו כן את ה buffer לאובייקט io.BytesIO במקום למחרוזת פשוטה. הסיבה לכך היא שכדי להוסיף טקסט למחרוזת פשוטה כולל הצבה מחדש(לא mutable). לעומת זאת io.BytesIO יוצר אובייקט דמוי קובץ אשר תומך בפעולות קבצים מוכרות כמו write. קבלת המידע מהקובץ היא בעזרת getvalue.

זה אמור להיות מובנה ב C, כך שזה יותר אופטימלי בכל מקרה מהרבה בחינות. לפחות בפייתון 3. בפייתון 2 האימפלמנטציה ב C נעשית על ידי מודול בשם cBytesIO... משהו מכוער ולא יפה. בפייתון 3 הם סדרו את הספריות ולכן זה נמצא בתוך ספריית io.

הקטע נראה כך:

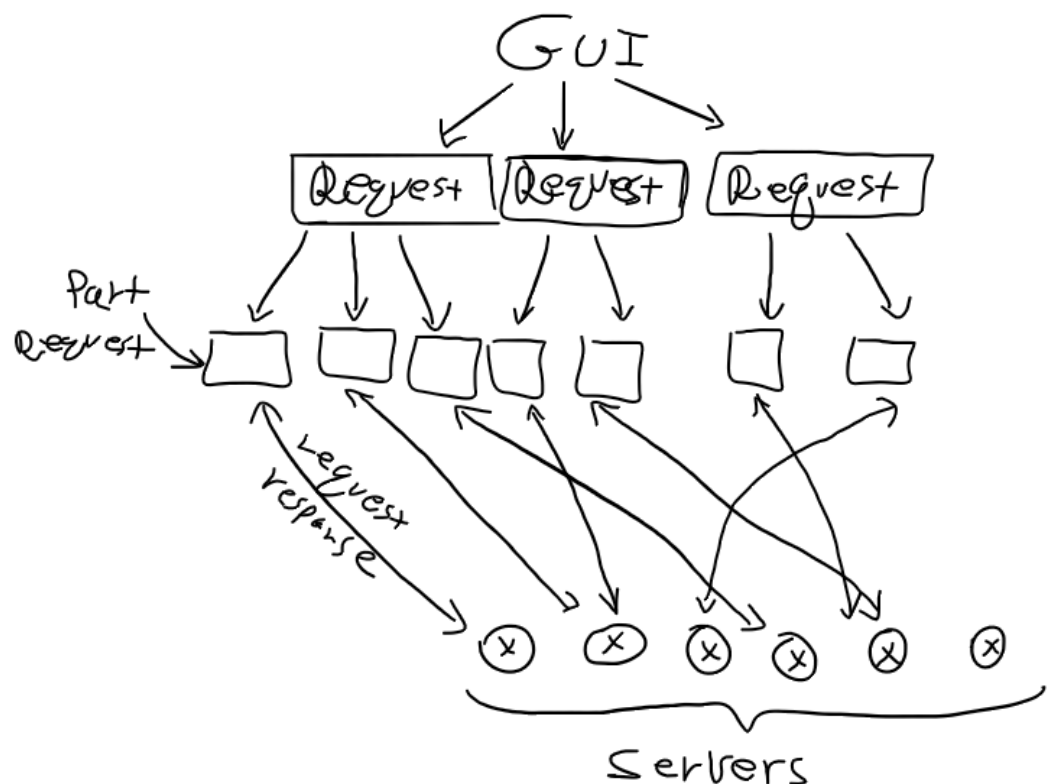
```
def receive_response(self):
    '''Receives the response from the server.'''
    with self.sock.makefile() as rfile, io.BytesIO() as response:
        while True:
            line = rfile.readline() + rfile.read(1) #to catch the extra \n
            response.write(line)
            if line.endswith(b'\n\n'):
                break # reached EOD
        return ResponseMessage.from_message(response.getvalue())
```

בסופו של דבר הוספתי למודול message (האחראי על פרוטוקול התקשורת בין השרת ללקוח) מחלקה חדשה: request, המגדירה את תהליך הבקשה והקבלה של הודעות מהשרת.

ל client שינתי חלק מהפעולות – אלו הרלוונטיות המרתי כך שישתמשו במחלקה החדשה בצורה אפקטיבית ויעילה יותר.

במחשבה שניה, אולי אפילו כדאי לשנות לחלוטין את כל המבנה של התכנית כדי ליצור הבדלה יותר ברורה בין הממשק הגרפי/CLI לקטע הבקשה. ולכן אני מציע את הניסוח הבא: ממשק המשתמש יוצר אובייקטי Request אשר מתארים את הבקשה של הלקוח, ואלו מייצרים אובייקטי PartRequest המתארים את בקשת החלקים הספציפיים ממספר שרתים שונים.

בצורה יותר טבעית זה נראה כך: (כן אני ציירתי את זה עם העכבר בצייר, התאמנתי הרבה)





הנקודה היא, בכל אופן, שאפשר ליצור אבסטרקציות ופירוט לוגי של תהליך הבקשה. הלקוח מבקש קובץ על ידי יצירת בקשה. בקשה יוצרת תתי בקשות לשרתים שונים ("מקורות מידע" שיש להם את הקובץ).

## בביליוגרפיה

- <https://docs.python.org/3.6/library/io.html>
- <https://docs.python.org/3/library/asyncio-task.html>
- <http://dabeaz.com/coroutines>
- [/http://www.slsknet.org](http://www.slsknet.org)
- <https://docs.python.org/3.6/library/socket.html#socket.socket.makefile>
- [https://developer.mozilla.org/en-US/docs/Web/HTTP/Range\\_requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests)



## דוח התקדמות 3 – 14.11.17

לאחר שסידרתי את כל הנושא של היחס בין ה-GUI ללקוח(על אף שאין לי עדיין GUI) אני התחלתי לעבוד על הקבלה של הקובץ כחלקים.

בתור התחלה, אני צריך לחשוב איך אני בכלל מחלק את הגודל של הקובץ לעבודות שונות. , כמו שצינתי טרם, הוטלה עלי המטלה לטפל לקבל את הקובץ בצורה מבוזרת ממספר מקורות שונים.

אם רמת המורכבות של הפרויקט שלי מאחד עד עשר הייתה 7 עכשיו היא בערך 9... הביזור של פרוטוקול ההודעה דורש ממני שיוניים דרסטיים של הקוד.

זה כולל בתוכו, אך לא מוגבל לפיצול של העבודה לחלקים ועבודה במקביל על כל אחד מחלקי העבודה האלה.

כל אחד מהם מהווה אתגר בפני עצמו.

ישנן 2 דרכים לפצל את העבודה, לפי מספר חלקים ולפי גודל:

פיצול לפי מספר חלקים הכוונה שיש לי מספר קבוע של חלקים,  $n$ , והגודל של העבודה עליו משתנה כדי להיות מותאם. לפי היחס הבא, בהינתן  $S$  זה גודל הקובץ:  $n_{zis} = \frac{S}{n}$ . השימוש של זה יכול להיות נוח

בהינתן ויש מספר קבוע של אנשים שיש להם את הקובץ ואני רוצה לחלק את העבודה עליהם. זה נראה יפה בתאוריה, אבל פרקטית זה רע, ואפילו מאוד! הסיבה לכך היא שזה אלגוריתם משתנה. זה לא מותאם לסביבה שיכולה להשתנות. לצורך העניין מה אני עושה אם מישהו מחליט להתנתק בזמן ההורדה? יש לי עכשיו גוש גדול של מידע שלא יכול להיות מנותב לשום מקום. ומה קורה אם מישהו חדש מתחבר? זה לא יהיה קל לפרק את העבודה מחדש על כולם. ולכן, החלטתי ששיטה זו לא מתאימה לפרויקט.

לפי השיטה השנייה, לעומת זאת, בהינתן גודל קבוע של כל "חלק", מספר החלקים הכולל יהיה:

$$n_{zis} = \frac{S}{n}$$

זה כמובן שקול מתמטית לשיטה הראשונה, אבל קצת הפוך... היישום של זה אמור להיות די פשוט. יש לי גודל קבוע כלשהו, נקרא לו `chunk_size`, וגודל של קובץ, ועלי לחלק את העבודה לחתיכות בגדלים שווים.

שני מצבים שאוטומטית צעקו לי הצילו הן מה קורה במידה וגודל הקובץ קטן מ-`chunk_size`, ומה קורה במידה והגודל לא מתחלק כראוי.

הייתי צריך אלגוריתם אחד כדי לטפל בשני אלו, מה שלקח הרבה יותר זמן מן הצפוי. מסתבר ששכחתי שנתיים שלמות של אלגוריתמיקה ונתקעתי על משהו מציק ו-"low level" כל כך...

כמובן, זו בעיה יותר מתמטית מאשר שקשורה למחשבים, מאחר וחילוק העבודה למעשה מדבר על חילוק שלמים (integer division) וטיפול ביתר (remainder).

בהתחלה ניסיתי לקחת את כל היתר ו"לפזר אותו" על כל החלקים הנוותרים. אחר כך הבנתי שזה באמת לא משמעותי כל כך ולא שווה את כל המאמץ, במיוחד כאשר נתקלתי בבעיה אחרת שקשורה לזה(אותה אפרט בהמשך).



הפתרון הפשוט יותר היה להוסיף את היתר ל-chunk האחרון – שינוי לא משמעותי במיוחד כשמסתכלים על העבודה ככלל.. לא היה צורך מייד ל-pצל כמה בייטים שסטו לכאן או לכאן, לא משמעותי.

אבל זה הוכיח את עצמו כפתרון לא טוב שוב, כי אז הייתי צריך לטפל במקרה שבו גודל הקובץ קטן מגודל ה-chunk.

זה לא הלך טוב כי הייתי צריך לטפל בשני מקרים מיוחדים כעת והקוד היה ארוך, מבולגן ועמוס. המסקנה הסופית הייתה שיש לשים את היתר על ה-chunk הראשון וזה פתר את רוב הבלגן.

התהליך הזה לקח זמן והרבה כאב ראש.

כמובן הייתי צריך גם להגדיר את הדרך שבה כל chunk מיוצג, וזה היה פשוט כמו לשים את המיקום ההתחלתי והסופי של אותו חלק בתוך tuple. השארתי את זה כמו שהוא לעת עתה וניסיתי לחשוב קדימה לגבי מה קורה לאחר שאני משיג את כל החלקים של הקובץ.

טוב, ההנחה שלי הייתה שאני אשים אותם בקבצים... אופס – אם אני שם אותם בקבצים ומנסה לחבר אותם יש צורך לשמור על הסדר של החלקים... אני לא אוכל להסתמך על הסדר שבו אני מחזיר את כל החלקים מאחר וזה לא בהכרח הסדר בו החלקים הולכים לחזור... ולכן עלי לשים אינדקס על כל טווח.

אם כך הפעולה לבסוף נראתה כך:

```
def generate_parts(self) -> list:
    """
    Divides the file of a given size to multiple different ranges.

    Returns
    -----
    list
        list of tuples of the form (index, (prepos, postpos)), where index
        is the index of the part, and prepos/postpos define the range
        of data.
    """
    nparts = self.file_size // self.CHUNK_SIZE
    remainder = self.file_size % self.CHUNK_SIZE
    parts = []

    # first part
    index = 0
    prepos = 0
    postpos = remainder + (self.CHUNK_SIZE if nparts else 0)
    part = (index, (prepos, postpos))
    parts.append(part)

    for index in range(1, nparts):
        prepos = index * self.CHUNK_SIZE + remainder
        postpos = prepos + self.CHUNK_SIZE
        part = (index, (prepos, postpos))
        parts.append(part)

    return parts
```



יפה, אבל יש בעיה, כמובן. אם הקובץ שאני מנסה להשיג הוא גדול הרבה יותר מגודל מ-`chunk_size`, יהיה לי הרבה מאוד חלקים... וזה כמובן בעיית זיכרון כיוון שאני שם את כולם בתוך רשימה אחת. הפתרון היה ליצור generator שיחזיר בכל פעם חתיכה אחרת. בשלב זה אנחנו חוזרים לטענה לגבי ה"פיזור" של כל השארית על החלקים – זה לא היה עובד גם אם הייתי מנסה כיוון שבשביל לפזר את השארית על החלקים אני צריך שיהיו לי כל החלקים, ולכן זה לא היה עובד בלי מאמץ מיותר.

הצורה הסופית של הקוד (לאחר מלא זמן של מחשבה) הייתה ככתוב למטה. אלגנטי ופונקציונלי, אבל לקח לי הרבה יותר זמן ממה שזה היה אמור.

```
def generate_parts(self):
    """
    Divides the file of a given size to multiple different ranges.

    Yields
    -----
    tuple
        A tuple of the form (index, (prepos, postpos)), where index
        is the index of the part, and prepos/postpos define the range
        of data.
    """
    nparts = self.file_size // self.CHUNK_SIZE
    remainder = self.file_size % self.CHUNK_SIZE

    # first part
    index = 0
    prepos = 0
    postpos = remainder + (self.CHUNK_SIZE if nparts else 0)
    part = (index, (prepos, postpos))
    yield part

    for index in range(1, nparts):
        prepos = index * self.CHUNK_SIZE + remainder
        postpos = prepos + self.CHUNK_SIZE
        part = (index, (prepos, postpos))
        yield part
```

## ביבליוגרפיה

לא השתמשתי במקורות מידע בחלק זה כיוון שזה כלל בעיקר אלגוריתמיקה...



## דוח התקדמות 4 – 23.11.17

חוזרים לעבודה, והפעם הייתי מאוד עסוק...

היה צורך לטפל במספר בעיות, לחזור חזרה לשרת ולטפל בכמה שיוניים קטנים וכמו כן לעבוד על הלקוח עוד.

ממבט עילי כרגע השרת היה זקוק לטיפול דחוף – כיוון שלא הייתה פונקציונליות קיימת כדי לטפל בבקשות של טווח ספציפי. הייתה האפשרות לבקש היסט מקובץ מסוים עד לסופו, אבל לא טווח ספציפי. החלטתי גם לעשות refactor נחמד לקוד שלו כדי שלא יבאס אותי. זה לא לקח יותר מדי זמן, אבל דרש עבודה מחדש על חלק מהכיעור.

בנוגע ללקוח – הייתי צריך להתחיל לחשוב לגבי הפיצול של העבודה בצורה הרבה יותר ברורה. וזה מתחיל מלבחור את הממשק שיאפשר לי את העבודה במקביל. בחרתי ב-thread-ים כי הם הכי פשוטים ולא ראיתי הרבה צורך בדברים אחרים... ישבתי הרבה מאוד זמן וקראתי בדיוק על איך מממשים דברים כאלה בעולם האמתי, ולא מצאתי הרבה מידע רלוונטי. התחלתי לחפש בעיקר ב-github – לעבור על קודי מקור של כל מיני מימושים לפרוטוקולים שונים כמו torrent, אבל זה כמו מחילת ארנב של קוד נוראי... הכל שם היה או מאוד מבולגן ולא ברור... ובסופו של דבר הקודים הרציניים היו כתובים או ב-C או בשפה דומה – אשר אין לי מושג איך לקרוא. כל הקודים בפיתוח היו ממש מוכועים ולא ברורים.

מה לעשות, השארתי את מלאכת חיפוש המידע לזמן אחר וניסיתי ליצור בעצמי את המערכת עם thread-ים פשוטים. הסקיצה הראשונית שלי הייתה ככה:

אני יוצר מספר כלשהו של thread-ים, אשר לוקחים host אקראי מתוך רשימה של host-ים, לוקחים איבר מתוך ה-generator של החלקים ומעבדים אותו. (את פעולת העיבוד טרם הגדרתי בשלב זה). זה אוטומטית לא נראה יפה במיוחד, בעיקר כשמוסיפים לזה את העובדה שהמטרה של generator זה לעבור עליו עם לולאת for...

אבל זו לא הסיבה שבחרתי אחרת... מסתבר ש-generator-ים הם לא thread-safe. משמע שאם 2-thread-ים לוקחים מידע מ-generator בו זמנית, הוא קורס. בעיה.

באותו הזמן נזכרתי במבנה נתונים שפותר לי את כל הבעיה הזאת-Queue. ובפיתוח נראה שהוא בנוי במיוחד בשביל עבודה עם thread-ים. מה נוח. מקריאה מעמיקה על הדוקומנטציה של מודול Queue בפיתוח גיליתי מספר פעולות מאוד יעילות, כמו כן דוגמאות נוחות מאוד לעבודה מרובת thread-ים. אז מימשתי את הדוגמאות בצורה מאוד פשוטה ומרופרת, בלי להתייחס לשגיאות ובעיות פוטנציאליות – רק רציתי קוד עובד.

אחר כך נותר לי לעבוד על הפעולה שמטפלת במידע מהשרת. הדרך שבה סידרתי את זה היא מאוד פשוטה – עבור כל חלק מידע מהשרת שאני מקבל אני כותב קובץ זמני שיכיל את המידע, ולשם הקובץ אני מוסיף את האינדקס של הטווח אשר הקובץ מייצג. כאשר כל הקבצים מוכנים וירדו אני משלב את כולם לקובץ אחד.

עד כאן זה בהנחה ולא היו שגיאות בקבלת הקובץ.



ובנקודה אחרת, ישנם עוד מספר שינויים שהחלטתי לעשות, בעיקר refactoring לקוד של הלקוח והאצלת סמכויות למחלקות אחרות... זה לקח את רוב הזמן של העבודה.





## ביבליוגרפיה

- <https://github.com/jefflovejapan/drench>
- <https://github.com/borzunov/bit-torrent>
- <https://stackoverflow.com/questions/1131430/are-generators-threadsafe>
- <http://www.dabeaz.com/generators/Generators.pdf>



## דוח התקדמות 5 – 05.12.17

שבועיים גדוש בשינויים – ועל כן גם העיכוב הרב.

מצאתי מקורות מידע! בניגוד ל-2 הדוחות האחרונים בהם לא הצגתי מקורות מידע, הפעם ישבתי וקראתי – והרבה... הנוחות שבמאמרים ומחקרים היא שכל אחד מציג את המקורות שלו בתחתית המחקר... ולכן אני יכול ללכת אחורה ולמצוא מכל מחקר אחד עוד 30 אחרים (בעיקר כאלו שהולכים אחורה עשרות שנים...)

באופן כללי, גיליתי שלסוג הפרוטוקול שאני מנסה לממש יש שם – segmented file transfer, או multisource file-transfer. ובדפים אקדמאים מתוארות דוגמאות לגבי המימוש הספציפי של ההורדה המבוזרת.

מסתבר שזה לא רחוק ממה שאני כתבתי מקודם ולכן לא הייתי צריך ללכת רחוק במיוחד ולערוך את כל הקוד מהתחלה, אלא רק ליישם את זה בצורות שונות. וזה עבד בדיוק כמו שציפיתי...

הבעיה התחילה רק מאוחר יותר, כשניסיתי לטפל בשגיאות.

כרגע הדרך שבה אני את העברת המידע לתוך ה-thread-ים היא על ידי Queue, והדרך שבה Queue ממומש מאפשרת לי להשתמש בפקודה בשם queue.join. הפקודה הזאת למעשה עוצרת את התוכנית (blocking) עד שעבור כל איבר אשד שמתו בתוך ה-queue (עבור כל queue.put שעשיתי) עשיתי גם queue.task\_done.

כלומר, בהנחה ולא עשיתי task\_done לכל איבר ששמתי ב-queue, אני אתקע שם עד אשר זה יסתיים. בצורה כזאת אני יכול לדעת מתי כל המידע הועבר.

אבל לאט לאט הבנתי כמה הדרך הזאת בעייתית. בצורה כזאת אין שום דרך שאוכל "לסיים" את העבודה מבלי שסיימתי לעבד את כל החלקים. זה נשמע לגיטימי בהתחלה – אבל בהינתן בעיה בקובץ שאינה מאפשרת לי להשלים את ההורדה, למשל כל ה-host-ים שיש בידיהם את הקובץ התנתקו במהלך ההורדה, ועל כן אין לי שום דרך לקבל את הקובץ, אני לא אוכל לצאת מההורדה. אני אתקע על ה-queue.join לנצח!

ניסיתי במשך שעות לתקן את זה, וככל שניסיתי יותר ככה הבנתי שזה יותר קשה ממה שחשבתי. הפתרון הסופי שלי כולל בתוכו גישה לתוך משתנים פנימיים של אובייקט ה-queue, ועריכה מכוערת של התנאים שמאפשרים לי לצאת מה-join... זו הייתה הדרך היחידה לגרום לזה לעבוד, אבל זה רק גורם לי לחשוב יותר ויותר כי להשתמש ב-queue.join הוא לא הפתרון הנכון לבעיה הספציפית הזאת.

משם לא הצלחתי להתקדם על אף שישבתי וערכתי את הקוד עשרות פעמים, לקח לי מלא זמן למצוא פתרון שמאפשר לי בכלל בדרך כלשהי לטפל בכל הסיפור הזה...

הוספתי גם קוד שמוריד מרשימת ה-host-ים הפוטנציאליים host פגום – כדי לא לנסות להוריד ממנו סתם – מעיין דרך להבטיח שההורדה תהיה יותר יציבה.

בנימה אחרת המשכתי לעשות refactoring לקוד, והפעם הצגתי מחלקה חדשה שנקראת connection אשר מגדירה את החיבור בין השרת ללקוח, ומגדירה פעולות בסיסיות כמו קבלת "הודעה" וקבלת "מידע". לא יודע למה לא עשיתי את זה טרם, אבל זה צמצם לי את הקוד בשרת והקוד בלקוח ועשה אותו להרבה יותר אבסטרקטי ומובן. לא הייתי צריך לראות את הקוד ה-"low level" בכל פעם שערכתי את הקוד וזה עשה לי את החיים קלים יותר.



כמו כן, טיפולתי במקרים קיצוניים של ניתוקים של הלקוח והשרת, והשתמשתי במודל נחמד שקיים בפיתון שנקרא context-manager כדי לוודא שאף socket לא יישאר פתוח בקוד החדש. במקום להשאיר את ה-socket פתוח בצורה שעלולה לגרום לבעיות או שיבושים במהלך התוכנית לאורך זמן, ה-context manager מאפשר לי "להשתמש" בקטע קוד מסוים תחת בלוק ספציפי אשר ביציאה שלו קוד מסוים מבוצע – בין אם היציאה היא על ידי שגיאה או על ידי סגירה סטנדרטית של הקוד.

פחות או יותר, context-manager נראה ככה:

```
class Something:
    def __enter__(self):
        print('inside the block')

    def __exit__(self, exc_type, exc_value, traceback):
        print('got outside of the block')

with Something() as s:
    do_some_stuff(s)
```

במקרה כזה, בין אם הפעולה do\_some\_stuff נסגרה בגלל שגיאה או בגלל שהיא הסתיימה, עדיין יודפס "got outside of the block" בסוף.

השימושים של זה הם בעיקר כדי "לנקות" אחרי אובייקטים שלא יפה להשאיר כפי שהם. בעיקר משתמשים בזה כשעובדים עם קבצים ו-socket-ים מאחר ובצורה כזאת אין סיכוי שישכחו אותם פתוחים... זה מאוד נוח לעבודה, וגם נראה די יפה.

בחזרה לבעיות עם thread-ים – ישנם גם כל מיני בעיות כלליות שעלו במהלך הבדיקות הפולשניות שערכתי על האלגוריתם:

- בהנחה ויש לי n עובדים שמנסים להוריד קובץ, ויש לי 1 או יותר host-ים בעייתיים, העובדים סטטיסטית ייקחו יותר host-ים בעייתיים! הכוונה בבעייתיים בקונטקסט הזה ל-host-ים שכנראה לא מחוברים בכלל. הם יכולים לגרום ל-timeout ארוך עד שיתנתקו. בזמן הזה כל עובד אחר שמנסה להוריד מ-host-ים תקינים יסיים ויבחר לו host חדש. משמעות הדבר היא שכל העובדים בסופו של דבר יתקעו הכי הרבה זמן עם ה-host-ים שעושים timeout... סטטיסטית זה פשוט יותר סביר שהם יתקעו – למרות שהסיכוי הוא שווה לכל host, ההתנהגות של כל host לא שווה זה לזה ולכן יש כאלו שיתקעו את הכל. ניסיתי לשחק עם הזמנים של ה-timeout אבל זה לא עזר.
- השרתים מגיבים ממש רע לריבוי הבקשות... ככל שאני מוסיף יותר עובדים ככה השרתים מגיבים פחות. כתוצאה מכך, באופן מאוד אנטי-אינטואיטיבי, יותר עובדים משמע פחות מהירות. זה נכלל גם כן בסדרת הבעיות שנגרמות כתוצאה מסטטיסטיקה – ככל שיש יותר עומס על שרת אחד, ככה יותר סביר שיבחרו בו כי הוא מעכב את כולם.



המסקנות המתבקשות מהבדיקות האלה מעלות חשד כי תהליך בחירת ה-host-ים מאפשר "אבולוציה הפוכה" של הקוד. ככל שהקוד רץ יותר זמן ככה הוא נהיה גרוע יותר במקום טוב יותר. זה משהו שאני צריך לטפל בו איכשהו.

בינתיים יש רוח של התקדמות באוויר, אני רק צריך למצוא את הכיוון הנכון לעבוד בו ומשם אפשר להיכנס לפיתוח הרבה יותר רציני. מה שטוב זה שברגע שאני אעבור את כל האזור הבעייתי – האזור שקשור ל-concurrency – הכל יהיה הרבה יותר פשוט.

## ביבליוגרפיה

- [https://en.wikipedia.org/wiki/Segmented\\_file\\_transfer](https://en.wikipedia.org/wiki/Segmented_file_transfer)
- <https://docs.python.org/3/library/queue.html>
- <https://docs.python.org/3/reference/datamodel.html#context-managers>
- <https://docs.python.org/3/library/stdtypes.html#typecontextmanager>
- <https://pdfs.semanticscholar.org/8f26/39212106215f20e32db49723a47c2813517a.pdf>
- <https://mm.aueb.gr/publications/2013-MMFTP-ICN.pdf>
- <http://ijcset.net/docs/Volumes/volume6issue12/ijcset2016061201.pdf>
- <http://www.cs.virginia.edu/~son/cs851/papers/dataMgmt.pdf>
- <https://www.cs.rutgers.edu/~rmartin/teaching/fall04/cs552/readings/by98.pdf>



## דוח התקדמות 6 – 12.12.17

קיבלתי משימה חדשה, ולכן עזבתי מעט את הבעיות אשר הטרופו את מחשבתי מהדוח הקודם... הפעם מדובר בהגבלת מהירות ההורדה/העלאה/ניצול CPU של המערכת. בפרט את אלו של השרת.

הסיבה לכך היא נורא פשוטה – אני לא רוצה להעמיס על השרת מאחר ובסופו של דבר הוא לא באמת שרת אלא תוכנה שאמורה לעבוד במקביל לשימושים גנריים אחרים של המחשב, ולכן יש צורך להגביל את המהירות שלו.

לאחר חשיבה בנושא גיליתי דרך פשוטה לטפל בבעיה הזאת – שיטה הנקראת token bucket. הרעיון הכללי ממש פשוט – יש לך "דלי", ולדלי הזה נכנסים כל מספר קבוע של יחידות זמן כלשהיא (X שניות למשל) מספר קבוע של מטבעות (tokens). לדלי יש קיבולת מקסימלית ועל כן כאשר הדלי מלא, כל ניסיון להוסיף מטבעות לדלי לא יעשו כלום.

כאשר אני רוצה לעשות חישוב מסוים הדורש משאב כלשהו שניתן למדוד – למשל כמות בייטים שאני מעוניין להעלות/להוריד, אני צריך להוציא כמות שווה של מטבעות מהדלי. אם אין לי מספיק מטבעות, אי אפשר לבצע את הפעולה כיוון שאני מוגבל במטבעות. במקרה כזה אני אחכה קצת ואנסה שנית.

פתרון פרקטי ופשוט למימוש. אצטרך לעשות אותו thread-safe כמובן כי יש גישה חופשית לאותו אובייקט bucket ואני לא רוצה ליצור בעיות.

כדי לממש את זה אני אצטרך thread אחד שכל מספר קבוע של שניות יוסיף כמות מסוימת של "מטבעות" לדלי.

את המספר הזה אפשר לייצג כקצב, זרם, יחס, איך שלא נקרא לזה של bytes/sec. כל שניה אני מעוניין להוסיף x בייטים לדלי, ולכן הזרם הכולל של המידע לא יעלה על היחס הזה. שיטה זו תגביל באופן גלובלי את קצב ההורדה.

האלגוריתם עובד בצורה כזאת:

1. בייט אחד מתווסף לדלי כל  $\frac{1}{r}$  שניות (יחס r).

2. בהנחה ובדלי יש את הגודל המקסימלי, b, מטבעות, לא יתווספו חדשים.

3. כאשר ניגשים לשלוח packet בגודל n בתים החוצה, n מטבעות יורדות מהדלי וה-packet נשלח.

4. אם יש פחות מ-n מטבעות בדלי, ה-packet לא נשלח והמטבעות נשארים.

ישנן מספר ווריאציות של אלגוריתם זה, הכוללות מנגנונים יותר טובים אשר לא מאפשרים ליצור "burst" של מידע שיוצא החוצה (למשל בהנחה ובקשה בגודל b כאשר הדלי מלא תתבצע במהירות המקסימלית תוך ניצול מרבי של המשאבים). למעשה האלגוריתם מבטיח כי בסך הכל, **בממוצע**, קצב תעבורת המידע לא יעלה על r.

הוא לא מבטיח כי בכל זמן נתון הקצב לא עולה על r.

מימושים שונים לאלגוריתם זה נוצרו כדי להבטיח את זה, אבל לעת עתה אני אנסה לממש את האלגוריתם הבסיסי הזה.



ובנקודה אחרת, מצאתי פתרון לבעיית ה-thread-ים. אני תשאלתי אנשים שמבינים קצת בתחום והם אמרו לי שהבעיה הרבה יותר עמוקה מ-"טיפול בבעיות דרך thread-ים" אלא קשורה למקביליות בכללי, ושהפתרון שאני צריך לשאוף אליו מצריך **תקשורת** בין כל thread למפעיל שלו. הכוונה היא שכל thread יצטרך לשלוח את המצב שלו בנוגע לכל "משימה" שהוא מנסה לבצע.

אני אדבר על הנושא הזה בהעמקה בהמשך מאחר וזה משהו שקיבלתי רק אתמול, אבל ממבט ראשוני נראה שזה עונה על הבעיה שלי בצורה ממש טובה. עקרונית זה משתמש ב-queue כדי להעביר מידע לגבי התפקוד של כל אחד מה-therad-ים.

## ביבליוגרפיה

- [http://www.ee.oulu.fi/~skidi/teaching/mobile\\_and\\_ubiquitous\\_multimedia\\_2002/Turner.pdf](http://www.ee.oulu.fi/~skidi/teaching/mobile_and_ubiquitous_multimedia_2002/Turner.pdf)
- <http://www.itu.int/rec/T-REC-I.371-200403-I/en>
- [https://en.wikipedia.org/wiki/Token\\_bucket](https://en.wikipedia.org/wiki/Token_bucket)
- [https://en.wikipedia.org/wiki/Leaky\\_bucket](https://en.wikipedia.org/wiki/Leaky_bucket)



## דוח התקדמות 7 – 23.12.17

בהמשך לדוח הקודם, אני הולך לדבר על האלגוריתם החדש לטיפול המקבילי בבקשת ההודעה. הקוד נראה כך:

```
def worker(host, q, reply_q):
    reply_q.put((host, None)) # ready for work
    while True:
        range_to_download = q.get()

        # no work left
        if range_to_download is None:
            return

        try:
            process_range(host, range_to_download)
        except BadHostError:
            reply_q.put((host, range_to_download))
            return
        finally:
            reply_q.put((host, None)) # done and ready for work

def give_work():
    workers = []
    recv_q = queue.Queue()
    queues = {}

    for host in hosts:
        q = queue.Queue()
        w = threading.Thread(target=worker, args=(host,q,recv_q))
        workers.append(w)
        queues[host] = q
        w.start()

    work = collections.deque([1,2,3,4,5]) # structure containing work
    while work and queues:
        (host, msg) = recv_q.get()
        if msg is None:
            queues[host].put(work.popleft())
        else:
            work.append(msg)
            del queues[host] # remove worker from pool

    if work:
        print('workers all died before work finished!')
    elif queues:
        print('No work left')
        print('killing workers...')
        for host in queues:
            queues[host].put(None)

    for w in workers:
```



w.join()

## הסבר

הפעולה worker:

מתארת את מהלך עבודתו של עובד יחיד. הרעיון הוא שכל עובד מטפל ב-host אחד ויחיד. בתכנון הקודם, עבור מספר host-ים כלשהוא (n) ועבור מספר עובדים כשהוא (m), כל עובד בתורו לוקח host אקראי מהרשימה של ה-host-ים, מוריד חלק, ומשיג host חדש.

זה יצר בעיות רבות שקשורות להסתברות לבחור host-ים בעייתיים, וכמו כן לבעיות שקשורות לחוסר היעילות שנוצר כאשר  $m > n$ . מיותר לבקש ביותר מ-thread אחד מספר בקשות במקביל לאותו ה-host. ההמלצות בחלק מהמקורות שפרסמתי בדוח הקודם מדברות על כך שמספר החיבורים המקסימלי לכל מכשיר דרך TCP לא אמור לעלות על 2. אני מצמצם את זה ל-1 מטעמי נוחות.

בנוסף, מעבר ל-queue של החלקים להורדה, אני מציג כעת queue חדש של תגובות. בשונה מה-queue הראשון אשר היה מסוג single producer, multiple consumers, ה-queue הזה הוא הפוך בתפקיד שלו – multiple producers, single consumer. ה-queue הזה (מסומן כ reply\_q בקוד) הוא queue של תגובות – דיווחים, על ביצועיהם של כל אחד מה-thread-ים. כל עובד שולח את ההודעה (host, None) במידה והוא מעוניין לקבל עוד חלקים להוריד, או את ההודעה (host, range\_to\_download) במידה והוא לא מעוניין לקבל עוד חלקים. (הוא רוצה להחזיר את החלק להורדה לנותן העבודה כדי שינתב אותה למישהו אחר.)

הפעולה give\_work:

מתארת את תהליך חלוקת העבודה לעובדים. ישנה רשימה של כל העובדים – workers, תור שמתאר את התקשורת בין העובדים בחזרה לנותן העבודה – reply\_q, וכמו כן מילון של כל queue יחיד של נתינת עבודה לכל עובד. (המילון הוא מיפוי בין ה-host של העובד ל-queue של התקשורת אתו.)

בלולאה – כל עוד יש עבודה לתת, וגם יש עובדים לתת להם את העבודה (התנאי החשוב שעשה לי בעיות טרם): מקבל הודעה מעובד (דרך ה-reply\_q), אם ההודעה היא בקשת עבודה (host, None), לשלוח לאותו העובד עבודה דרך המילון של המיפוי. אחרת, אם ההודעה היא דיווח על סיום עבודה או תקלה ב-host, למחוק את העובד מתוך המילון של העובדים.

בסופו של דבר, ישנם שני מקרים בהם הלולאה הזאת תסתיים (מקרים בהם רצף העבודה ייגמר):

- במידה ואין יותר עבודה לתת (כל העבודה הסתיימה בהצלחה)
- אין יותר host-ים לבקש מהם עבודה – קרתה תקלה. במקרה הזה יש לסגור את כל העובדים (על ידי שליחת None ל-queue שלהם), וכמו כן יש לדווח למשתמש על התקלה.

## ביבליוגרפיה

- <http://www.cs.virginia.edu/~son/cs851/papers/dataMgmt.pdf> – חיבורי TCP מקביליים זה יעיל.
- <http://e-libdigital.com/download/protocols-for-high-speed-networks-iv.pdf> – חיבורי TCP במקביל יוצרים כמו כן בעיות (עמודים 349-360) – אולי בעתיד TCP ישתפר וזה ישחק יפה.





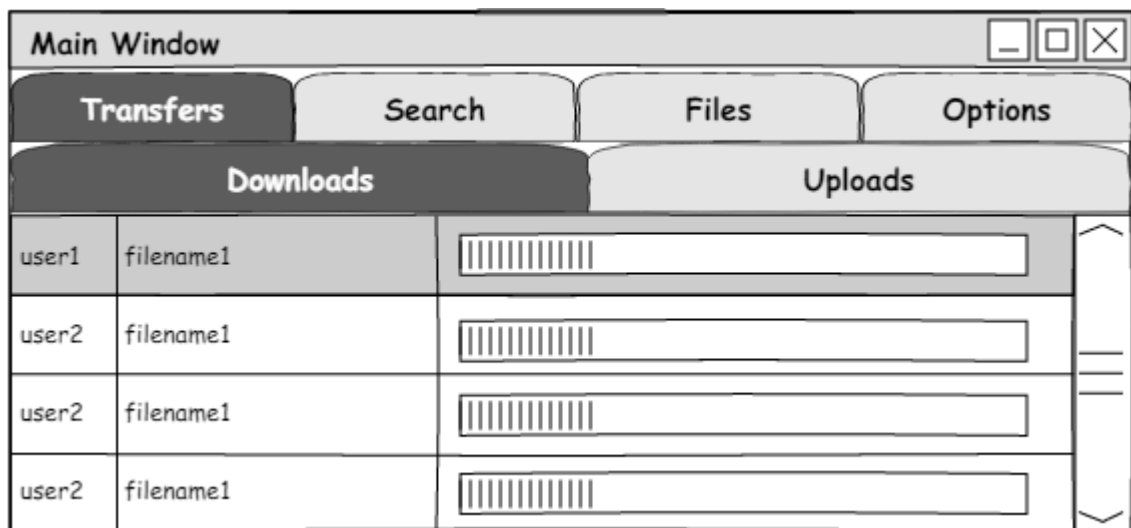
<https://pdfs.semanticscholar.org/8f26/39212106215f20e32db49723a47c2813517a.pdf> •

## דוח התקדמות 8 – 01.02.18

ניתנה לי משימה ליצור GUI לתוכנה, בערך במשך חודש אני הייתי תקוע על המטלה הזאת, ואיבדתי עניין בפרויקט. בלי להיכנס לפרטים, wxPython זו לא חבילה נחמדה במיוחד. הדוקומנטציה נוצרה באופן אוטומטי לחלוטין – כלומר אין אינטראקציה אנושית פרט למדריכים שונים. זה מקשה מאוד על תהליך העבודה.

עיצוב השלדה של התוכנה זה קל, ולקח לי בערך 20 דקות להבין את הקטע ולעשות את זה. אני חושב שהעיצוב הזה כמעט לחלוטין לא משנה באיזו ספריית GUI משתמשים (ניסיתי גם ב-Tkinter).

הבעיות התחילו כשניסיתי לעצב "דף הורדות". הצעת הפרויקט שלי הכילה את השרטוט הבא:



בהתאם לשרטוט הגנרי הזה, ניסיתי להכין את הממשק, אבל העיצוב הספציפי הזה מזכיר סוג של רשימה עם עמודות, או טבלה – דברים שקשה להציג אותם כמו שצריך דרך wxPython (אלוהים יודע שניסיתי).

אחרי מספר ימים של התחבטויות קשות בנוגע ליישום של מטלה זו, וויתרתי. לאחר דיבורים והתייעצות הם המורה, נראה שישנו פיצ'ר של wxPython הקרוי UltimateListCtrl שמסוגל לבצע את המטלה (אם כי בצורה די מכוערת ומגעילה).

זה לא נוח להתנהל עם UltimateListCtrl, כיוון שנראה שהתכנון שלו מפר את אחד מעקרונות התכנות – SRP, או Single Responsibility Principle. לפי עיקרון זה כל חלק בתוכנה אמור לעשות דבר אחד ויחיד. זה עוזר לנהל פרויקטים בצורה נכונה וברורה, וכמו כן עוזר לבחור שמות יותר משמעותיים לדברים כיוון שהמטרה של אותו הדבר ברורה.

במקרה הזה נראה שהשם מייצג בדיוק מה המחלקה הזו מטפלת בו – הכל. המחלקה הזו יכולה לייצג כל דבר למעשה. זה יכול ליצור רשימה רגילה, רשימת פריטים, תיבת בחירה, טבלה, או כל מבנה שמציג לך יותר מאיבר אחד. זה מבלבל, במיוחד כשמבנה פשוט יותר יכול להיות מספיק טוב לו היה נבנה אחרת (ListCtrl).



בכל אופן, לאחר מספר שעות של עבודה ביחד עם המורה, יצרנו משהו שיכול פוטנציאלית להיות מותאם כדי לעבוד עם מה שיש לי. הוא עדיין צריך הרבה עבודה והתאמה. אבל בשלב הזה אני מעדיף שלא לגעת ב-GUI אלא בזמן העבודה הכתית. אני באמת שונא לעבוד עם GUI.

לאחר הביאוס הלא פוסק שבא עם עבודה על ה-GUI, החלטתי להמשיך לעבוד על הפרויקט עצמו, כלומר על המחלקות הראשיות והעיקריות של הפרויקט. כרגע אני עובד על ליצור אינטגרציה בין מה שעד כה קראתי לו "שרת" (אני צריך לחשוב על שם אחר, אולי "מעלה"), והלקוח (אולי לזה אני אקרא "מוריד"). צריך שמות חדשים כיוון שבשלב הזה של הפרויקט הולך להיכנס בקרוב אלמנט נוסף של שרת (כמתואר בהצעת הפרויקט).

בכל אופן אין לי שמץ איך האינטגרציה בין המעלה למוריד צריכה להתקיים. אולי בשלב הזה אני פשוט צריך ליצור קובץ אחיד שמריץ את שתי התכנות במקביל ונותן ממשק למשתמש להריץ את הפקודות הרגילות.

רק אחד מהדברים שצריך לעבוד עליהם. נישאר ליצור את השרת המרכזי שאמור לאפשר לאנשים לחפש את הקבצים שהם רוצים להוריד ולהוריד אותם משם. זה נראה כמו מטרה מסובכת בשל עצמה בכל אופן...

## הערות נוספות

- השמות של כל הדברים בספריה wxPython לא עוקבים אחרי pep8.
- אני כנראה אצור דווחי התקדמות קצרים יותר כדי לא לבזבז זמן.



## דוח התקדמות 9 – 29.03.18

לאחר שקיבלתי את המטלות לעבוד עליהם פתחתי את הפרויקט ואת התיקיה של ה-GUI והתחלתי לראות איך אני גורם להכל לעבוד כמו שצריך. המטלות שלי כללו להוריד מספר דברים במקביל, וכמו כן להציג פרטים על ההורדה במקומות המתאימים – קצב ההורדה והזמן שנותר.

אתחיל ואומר שהיה באג לא ברור בספריית ה-Request שמנע ממני להריץ את הקוד כראוי, ולכן לקח לי הרבה מאוד זמן להתחיל לעבוד כמו שצריך. לאחר שפתרתי את הבאג (פשוט לשים try/except במקום המתאים שעושה blocking), עשיתי refactor לקוד של ה-GUI, בצורה כזאת עכשיו פתרתי בעיה נוספת שלא אפשרה לי לגשת ל-frame של ההורדות בממשק מבחוץ, וכמו כן לכל frame.

עשיתי כמו כן refactor לקוד של ה-DownloadsPanel, ועשיתי בו שינויים רבים כדי להקל על השימוש בו:

- פעולה כדי ליצור header.
- מילון שלפי השם של ה-header מחזיר את האינדקס שלו.
- מחיקת פעולות לא רלוונטיות מגרסאות קודמות.
- שינויי שמות לדברים מסוימים.
- קיצור הקוד והרחבתו בצורה ברורה.
- הוספת ממשק יותר ברור לניהול ההורדות מבחוץ (כדי שיהיה אפשר לעדכן בקלות כל הורדה)

כמו כן, אפשרתי הורדה במקביל מהלקוח. לצערי, לעומת זאת, זה שבר את ה-GUI עם שגיאה אותה אני עדיין מנסה לפתור. השגיאה ממש לא ברורה ואני מנסה לטפל בה כבר מלא זמן.

הבאג הזה כנראה קשור לפעולה update\_download בתוך DownloadsPanel. אבל מעבר לכך אין לי מושג מה הבעיה. השגיאה אומרת שה-gauge לא קיים ולכן לא ניתן לעדכן אותו, למרות שהוא אמור להיות קיים.

הפעולה היחידה שמוחקת את ה-gauge היא הפעולה populate\_downloads אשר מוחקת את כל העצמים ויוצרת אותם מחדש, ועל כן יוצרת מחדש גם את ה-gauge (לפחות אמורה...).

בעיה נוספת שככל הנראה קשורה ל-update\_download היא העובדה ששאר השדות לא מתעדכנים – ה-eta וה-download rate. אלו מקבלים את הערכים שלהם מאובייקט ה-Download שמתאים לשורה בתהליך היצירה מחדש שלהם ב-populate\_downloads. הערכים של אובייקט ה-Download נקבעים ב-update\_download אבל נראה שזה לא ממש משנה הרבה והאובייקט לא מקבל את הערכים החדשים שלו...



## דוח התקדמות 10 – 05.04.18

תיקנתי את כל הבאגים הנוכחיים בתוכנית ועשיתי קצת שיפורים נוחים.

הבאגים מהדוח הקודם נוצרו על ידי שימוש לא נכון ב `wx.CallAfter`. כמו כן, באגים תצוגתיים שנובעים מהשימוש בפעולה `populate_downloads` אשר יוצרת מחדש את כל רשימת ההורדות – וכן גם את ה-`gauge`, דבר הגורם לבאג וויזואלי לא ברור.

הפתרון היה להחליף את `populate_downloads` עם הפעולות `Append` ו-`DeleteItem`, אשר מתעסקות בשינוי של שורה אחת במקום בהכל.

התיקונים כללו מחיקת קודים לא רלוונטיים נוספים וכמו כן מניעת האפשרות להוריד את אותו הקובץ פעמיים במקביל – דבר הגורם לבאג כיוון שהתיקיה שמכילה את החלקים של הקובץ היא פומבית לכל ההורדות.

בתקווה, הפרויקט כרגע עונה על כל הדרישות שהוצבו בפני. (לא היה קל או כיף...)



## תיאור הממשק למשתמש

Glomp

Transfers Search Files Options

Downloads Uploads

File	Progress	Download Rate	ETA
007mb.mp3	<div style="width: 50%; background-color: green; height: 10px;"></div>	Speed: 2.51MiB/s	1.6sec

Glomp

Transfers Search Files Options

Downloads Uploads

File	Progress	Download Rate	ETA
007mb.mp3	<div style="width: 100%; background-color: green; height: 10px;"></div>	Completed	Completed



Glomp

Transfers Search Files Options

Downloads Uploads

File	Upload Rate	Uploaded
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\000mb.txt		
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\002mb.txt		
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\007mb.mp3	Speed: 0.28MiB/s	7.86MiB
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\024mb.mp3		
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\032mb.jpg		
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\200mb.mp4		
C:\Users\Home\Desktop\projects\v8 -- server\gui\test_files\Thumbs.db		



## סביבת עבודה

### שפת התכנות

Python 3.6

### פירוט סביבת העבודה והכלים הנדרשים לפיתוח

שם הכלי	מטרה
Python 3.6	הרצה
Cmd/Powershell	בדיקות והרצות תוכנית / הורדת ספריות
Notepad++	עריכת טקסט וכתיבה
Firefox/Google	גלישה וחקירת נושאים
wxPython	יצירת ממשק המשתמש





## הוראות התקנה של הספריות החיצוניות וכיצד להריץ את הפרויקט

### ספריות חיצוניות

• wxPython

### הוראות התקנה

התקנת ספריות נעשית עם pip באופן הבא:

```
python3 -m pip install wxpython
```

### הוראות הרצה ותפעול

כדי להפעיל את התוכנה יש קודם כל להפעיל את קובץ ה server.py, אשר באופן דיפולטיבי יעבוד בפורט 13372.

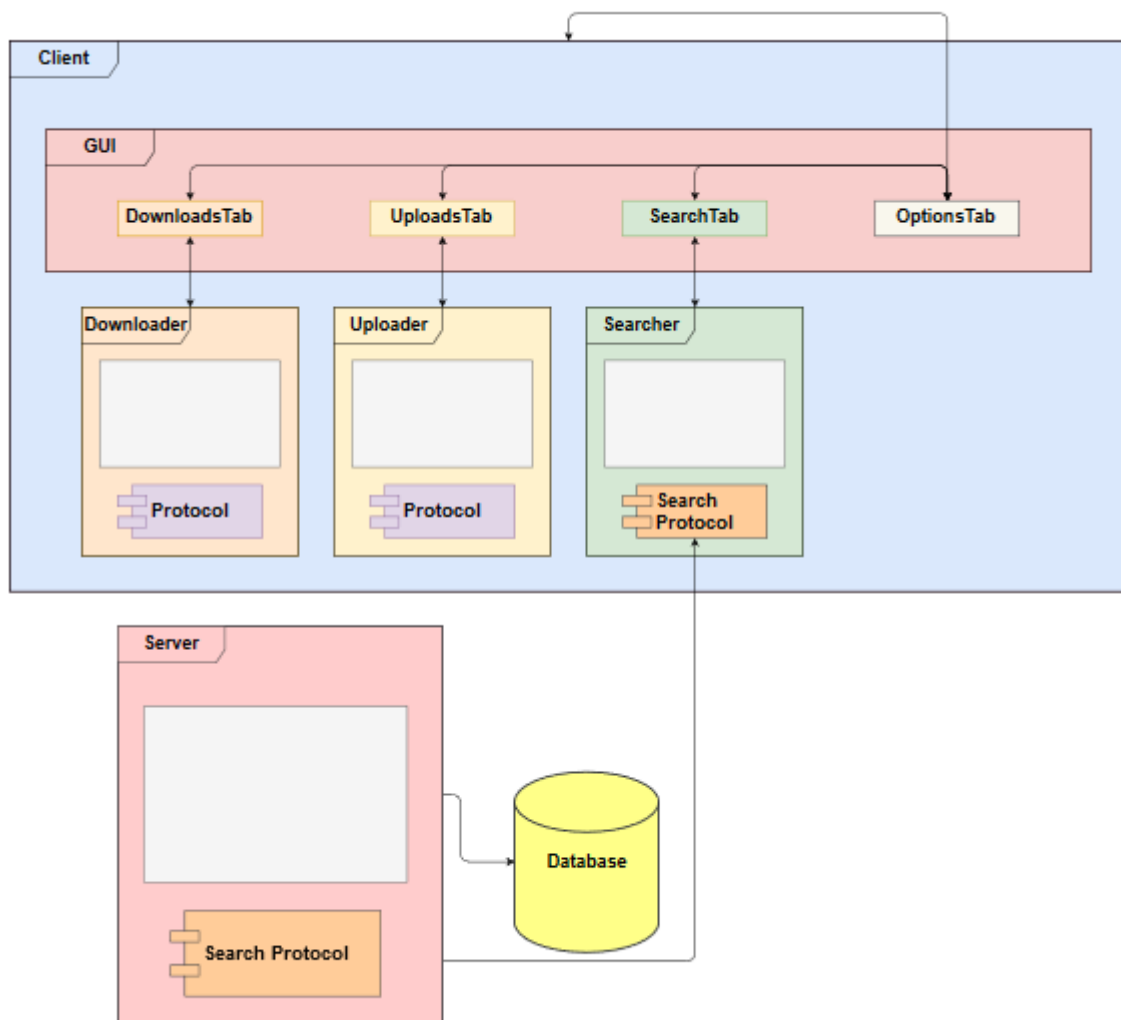
כדי להפעיל את הלקוח יש להריץ את קובץ ה gui\_wx.py בצורה הבאה:

```
python3 gui_wx.py [my_ip_address] [my_port] [server_ip] [server_port]
```

דרך ה cmd או טרמינל.

## תיאור ארכיטקטורה

### סקירת מודולים







מחלקה לטיפול בבקשות הורדה

שם הקובץ: request.py

שם המחלקה: RequestHandler

פעולות המחלקה: המחלקה מגדירה את אופן הטיפול בבקשת הורדה יחידה.

מחלקה לטיפול בפרוטוקול הקישור

שם הקובץ: connection.py

שם המחלקה: Connection

פעולות המחלקה: המחלקה מגדירה את אופי פרוטוקול התקשורת הבסיסי.

מחלקה להגדרת הודעה אשר נשלחת דרך פרוטוקול התקשורת.

שם הקובץ: message.py

שם המחלקה: Message

פעולות המחלקה: המחלקה מגדירה את מבנה ההודעה ותרגומה.



מחלקה ראשית

שם הקובץ: gui\_wx.py

שם המחלקה: Glomp

פעולות המחלקה: המחלקה מגדירה את מבנה ה GUI.

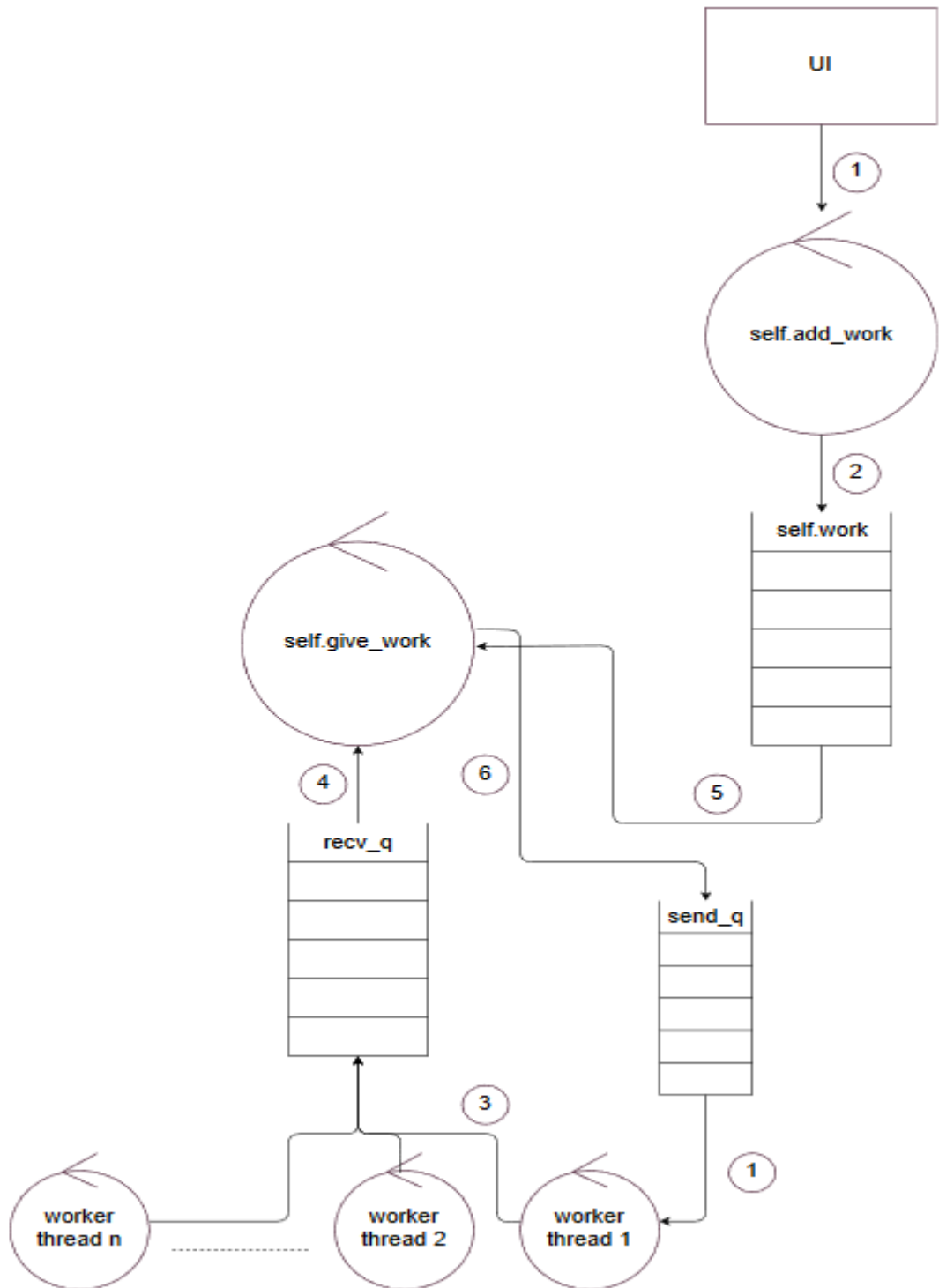
מחלקה ראשית

שם הקובץ: gui\_wx.py

שם המחלקה: Glomp

פעולות המחלקה: המחלקה מגדירה את מבנה ה GUI.

## דיאגרמה כרונולוגית





## הסבר

הדיאגרמה הבאה מציגה את תהליך הורדת קובץ יחיד מכמות מסוימת של מעלים. בהתחלה פרטי הבקשה מחולקים לתתי משימות (נקרא chunks) אשר מתווספים לתוך תור של משימות על ידי thread בשם `add_work`.

בזמן הזה, מספר עובדים גדול ככמות המעלים נוצרים, כל אחד עבור מעלה אחר, ואלו מבקשים מ-thread אחר בשם `give_work` לקבל עבודות. `give_work` נותן לכל אחד מהם חתיכה כשמבקשים ממנו ומשיב בחזרה. `give_work` מנהל את תהליך ההורדה בעוד שכל עובד (אשר נמצא ב-thread נפרד) אחראי להוריד חתיכה אחת ספציפית ממעלה אחד ספציפי.

חילוק העבודה הזו נעשה על ידי הרבה מאוד עבודה מאחרי הקלעים כדי לנהל את המקביליות של כל התהליך, טיפול בשגיאות ודינמיות: ישנו תור של בקשות עבודה אשר נקרא `recv_q`, אליו כל העובדים שולחים בקשות לעבודה, ופרטים מזהים, וכן ישנו תור ייחודי לכל עובד על מנת לשלוח לו את המטלות.

תהליך הזיהוי, טיפול בשגיאות וניהול העבודה הכללי נעשה לחלוטין על ידי `give_work`, אשר בו יש רצף עבודה המאפשר לפרק את העבודה בצורה יעילה וחסכונית בזיכרון כדי להוריד את הקובץ.

יעול הזיכרון נעשה באמצעות שימוש בתתי קבצים וגנרטורים במקום אחסון הקובץ במלואו על זיכרון התוכנית. תתי קבצים הכוונה לקבצים אשר כל עובד שומר בהן את כל בקשת החלק. הקבצים נשמרים בתיקה `tmp` אשר בתוך תיקיית הפרויקט.

אך על מנת לשמור את הקובץ, יש לאחסן אותו קודם כל בזיכרון(!). לא בדיוק. בפרויקט אני עושה שימוש נרחב בפיצ'ר שפה של פייתון הנקרא גנרטור (`generator`), גנרטורים מאפשרים לי להגדיר איטרטורים, אשר לא שומרים את כל המידע בזיכרון ביחד. איטרטור הוא כל אובייקט בפייתון אשר ניתן לעבור עליו בלולאה.

לולאת ה-`for` בפייתון מקבילה ללולאת `foreach` משפות אחרות, ולכן צריכה אובייקט אשר יש לו את היכולת להתנהג בצורה ראויה למבנה זה. בפייתון התנהגות זו ממומשת על ידי פרוטוקול האיטרטור, אשר מוגר על ידי פונקציית הקסם `__next__`, אשר מחזירה את "האובייקט הבא" ללולאה.

לפייתון יש מבנה פונקציה מיוחד אשר מאפשר בקלות ליצור גנרטורים, ואלו מאפשרים חסכון משמעותי בזיכרון(כיוון שלא צריך לשמור את הכל), וכמו כן חסכון משמעותי בזמן הריצה.

הגנרטור במקרה הזה לוקח חתיכות מידע מה-`socket`, מעביר אותן הלאה, לפעולה שעושה שימוש ב-`for`, וחוזר חזרה כדי להשיג עוד מידע.

כל זה ממומש בצורה אשר מחביאה את היישום הפרקטי של זה מאחרי הקלעים, ובך מאפשרת לי(או למישהו אחר שמשמש לי בקוד) להתייחס לזה כמו "קופסה שחורה שפשוט עובדת". המימוש המלא לא רלוונטי כמו ההשלכות שלו על הקוד, ולכן לא אכנס לפרטים יותר רלוונטיים.



## תיאור היבטי הסייבר בפרויקט

### Socket

בפרויקט נעשה שימוש נרחב בסוקטים. סוקטים הם קווי תקשורת בין אפליקציות שונות, אשר מאפשרים שליחת מידע ישיר באמצעות פרוטוקולי תקשורת כגון TCP או UDP. בפרויקט נעשה שימוש בסוקטים בתהליך שליחת הקובץ מן המעלה אל המוריד. דרך סוקטים ניתן לשלוח אך ורק מידע byte-י. מידע כזה מצריך ממני להגדיר פרוטוקול תקשורת המתרגם בין אובייקט נוח לשימוש בפרויקט לסוג הודעה הנוחה להעברה בסוקט. לשם כך יצרתי מספר קבצים ומחלקות המשמשות אותי לצורך התעבורה. העיקריים מביניהם אלו:

- Connection
- Message

המלקות האלו מאפשרות לי, בסופו של דבר להעביר את כל המידע הנחוץ לתקשורת בין המעלה למוריד של הקובץ. הגדרת הפרוטוקולים הזו חשובה במיוחד כיוון שהיציבות של הפרויקט תלויה בעיקר בתקשורת בין החלקים השונים.

דוגמאות קוד לא ממש הכרחיות כיוון שהחומר די בסיסי.

### Threads

בפרויקט נעשה כמו כן שימוש נרחב ביותר ב Threads. תהליך הורדה הקובץ הוא מקבילי ומתפרס על פני מספר מעלים שונים, פוטנציאלית. תכנון הפרויקט, בהתאם, חייב להשתמש ב Threads על מנת לאפשר את הפונקציונאליות להורדה במקביל. Threads הם למעשה ישויות המוקצות על ידי מערכת ההפעלה ומאפשרות לבצע מספר חלקי קוד במקביל. בפיתוח, שפת הפיתוח, לא ניתן עקרונית להריץ מספר חלקי קוד במקביל בעקבות מגבלות המוצבות על השפה (GIL), אך השימוש עקרונית זהה.

בפרויקט יש Threads האחראיים על הורדת הקובץ – workers, work\_giver, כמו כן Threads ראשיים כגון GUI, ספריית ההעלאה וספריית ההורדה. ישנן מספר Threads נוספים בכל אחד מן ה Threads הראשיים. באופן כללי הפרויקט עושה שימוש נרחב מאוד ב Threads בגלל האופי שלו אשר מצריך עבודה מסוג זה.

דוגמאות לשימוש ב Threads נמצא בתיאור הארכיטקטורה.





## רכיבי המערכת

### צד לקוח

שם פעולה/מחלקה	תיאור
RequestHandler	המחלקה הראשית, אחראית על פירוק העבודה לחלקים קטנים יותר וניהול העבודה.
request	הפעולה העיקרית בה משתמשים ספריות חיצוניות. הפעולה מפעילה את רצף הבקשה.
give_work	הפעולה הניהולית הראשית – מקבלת חלקי הורדה, יוצרת עובדים ומנחה את תהליך ההורדה.
add_work	הפעולה אשר מחלקת את העבודה לחלקים קטנים יותר.
worker	הפעולה אשר מתארת עובד יחיד – העובדה מקבל עבודה מהפעולה הניהולית ומוריד את החלק המתאים לכך מהמעלה.
process_part	פעולה אשר מעבדת חלק יחיד אשר הורד.
assemble_file	פעולה אשר מחברת את כל חלקי הקובץ אשר ירדו לקובץ אחד גדול.

### צד שרת

שם פעולה/מחלקה	תיאור
ServerHandler	מחלקה אשר מטפלת בבקשה יחידה אשר נשלחה אל המעלה.
handle	הפעולה הראשית אשר מטפלת בבקשה.
check_request	פעולה אשר בודקת את תקינות הבקשה.
handle_request	הפעולה אשר מנתחת את הבקשה ומגיבה לה.
Server	המחלקה המתארת את המעלה.



## פרוטוקול תקשורת

```
<title> <filename>\n<header>: <value>\n<header2>: <value2>\n...\n\n\n
```

זוהי הדרך המרכזית בה התקשורת עוברת בין השולח למוריד.

נשלחים ב headers פרטים נוספים הקשורים להעלאה כמו גודל וכדומה. השליחה מאוד מזכירה HTTP.

הקוד כדי לממש את זה נמצא ב-2 קבצים שונים – connection.py וכן message.py, אשר ביחד מגדירים את ספריית הפרוטוקול אשר אחראית לכל זה תהליך ההמרה וכיוצא בכך.

הפרוטוקול המלא הרבה יותר מורכב מן המצוין פה כיוון שהיישום שלו דורש עובדה עם חיבורים שונים במקביל.



## האלגוריתמים המרכזיים בפרויקט

מפורט בדיאגרמה הכרונולוגית(תיאור ארכיטקטורה ← דיאגרמה כרונולוגית)



## ביבליוגרפיה

- <https://docs.python.org/3/library/socketserver.html>
- <http://code.activestate.com/recipes/408859-socketrecv-three-ways-to-turn-it-into-recv>
- [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- <https://docs.python.org/3/library/socket.html>
- <https://docs.python.org/3.6/library/io.html>
- <https://docs.python.org/3/library/asyncio-task.html>
- <http://dabeaz.com/coroutines>
- <http://www.slsknet.org>
- <https://docs.python.org/3.6/library/socket.html#socket.socket.makefile>
- [https://developer.mozilla.org/en-US/docs/Web/HTTP/Range\\_requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests)
- <https://github.com/jefflovejapan/drench>
- <https://github.com/borzunov/bit-torrent>
- <https://stackoverflow.com/questions/1131430/are-generators-threadsafe>
- <http://www.dabeaz.com/generators/Generators.pdf>
- [https://en.wikipedia.org/wiki/Segmented\\_file\\_transfer](https://en.wikipedia.org/wiki/Segmented_file_transfer)
- <https://docs.python.org/3/library/queue.html>
- <https://docs.python.org/3/reference/datamodel.html#context-managers>
- <https://docs.python.org/3/library/stdtypes.html#typecontextmanager>
- <https://pdfs.semanticscholar.org/8f26/39212106215f20e32db49723a47c2813517a.pdf>
- <https://mm.aueb.gr/publications/2013-MMFTP-ICN.pdf>
- <http://ijcset.net/docs/Volumes/volume6issue12/ijcset2016061201.pdf>
- <http://www.cs.virginia.edu/~son/cs851/papers/dataMgmt.pdf>
- <https://www.cs.rutgers.edu/~rmartin/teaching/fall04/cs552/readings/by98.pdf>
- [http://www.ee.oulu.fi/~skidi/teaching/mobile\\_and\\_ubiquitous\\_multimedia\\_2002/Turner.pdf](http://www.ee.oulu.fi/~skidi/teaching/mobile_and_ubiquitous_multimedia_2002/Turner.pdf)
- <http://www.itu.int/rec/T-REC-I.371-200403-I/en>
- [https://en.wikipedia.org/wiki/Token\\_bucket](https://en.wikipedia.org/wiki/Token_bucket)



- [https://en.wikipedia.org/wiki/Leaky\\_bucket](https://en.wikipedia.org/wiki/Leaky_bucket)
- <http://www.cs.virginia.edu/~son/cs851/papers/dataMgmt.pdf> – חיבורי TCP מקביליים זה יעיל.
- <http://e-libdigital.com/download/protocols-for-high-speed-networks-iv.pdf> – חיבורי TCP במקביל יוצרים כמו כן בעיות(עמודים 349-360) – אולי בעתיד TCP ישתפר וזה ישחק יפה.
- <https://pdfs.semanticscholar.org/8f26/39212106215f20e32db49723a47c2813517a.pdf>