**שרת ולקוח לשיתוף קבצים חברתי**

**מבוא:**

עד שנת 1999 והמצאת נפסטר, כל לקוח שרצה לשתף קובץ ברשת למספר רב של אנשים היה צריך להעלות אותו לשרת מרכזי. אנשים שרצו להוריד את הקובץ היו ניגשים לשרת והוא היה שולח אליהם אותו. כתוצאה מכך על השרת נוצר עומס רב של בקשות ותגובות ולכן השרת היה נהיה יקר או איטי ולכן קצב ההורדה לא היה מהיר. בנוסף, היו נשמרים בו הרבה קבצים שהצטברו למידע רב וגרמו לשרת להיות יקר.

לכן, בשביל לנצל את היכולת של השיתוף החברתי נוצרו תוכנות כגון נפסטר וביטורנט. הלקוחות שמשתמשים בביטורנט לדוגמא, נעזרים בשרת מרכזי שיקשר ביניהם. המעלים מודיעים לשרת שהם מעלים והשרת מודיע למורידים ממי להוריד. כך המידע של הקבצים היה נשמר אצל הלקוחות והשרת לא היה צריך לשלוח מידע רב להרבה אנשים. בכך יצרנו מערכת זולה יותר והרבה יותר יעילה.

אני הולך לבנות שרת ולקוח המתבססים על עקרונות הביטורנט בשפת Python ובסביבת העבודה של PyCharm. אני אבנה את הפרוטוקולים בדרך שלי ואבנה את כל המרכיבים המרכזיים שהשרת והלקוח צרכים על מנת שהמערכת תעבוד.

**הסבר כללי:**

התוכנה עובדת עם קבצים בעלי הסיומת yftf. קבצים אלו הם למעשה מצביעים שמטרתם להפנות את התוכנה למידע הדרוש על מנת להוריד את הקובץ המבוקש. קובץ ה-yftf מפנה את התוכנה ל-Tracker, שהוא למעשה שרת המכוון את כל התנועה בין המשתמשים המורידים את אותו קובץ.

התוכנה שולחת ומקבלת מספר חלקים מקובץ ההורדה בו-זמנית, על פי הוראות ה-Tracker. היכולת לשלוח ולקבל חלקים שונים של הקובץ בו זמנית, ממשתמשים שונים, מאפשרת לתוכנה לשמור על קצב העברה גבוה מאוד.

**דוגמא להורדת קובץ שכבר נמצא ברשת:**

* אדם מסוים רוצה להוריד את הקובץ A באמצעות התוכנה.
* הוא תר את האינטרנט אחר קובץ yftf המצביע על הקובץ A, ומפנה את התוכנה לשרת Tracker.
* ברגע שתוכנה מתחברת לשרת, היא מקבלת ממנו הוראות באשר למי מהמשתמשים השונים עליה להתחבר.
* התוכנה מתחברת למשתמשים רבים ומורידה מכולם את הקובץ בו-זמנית.
* כל חלק מהקובץ שהורד והוא כבר מוכן, הופך להיות זמין למשתמשים אחרים דרך התוכנה. כלומר, ברגע שהתוכנה מסיימת להוריד חלק מסוים, היא חולקת אותו עם שאר המשתמשים על ידי ידוע שרת ה-Tracker.
* בו בזמן, השרת מפנה משתמשים אחרים אל המחשב של אותו אדם, על מנת שיוכלו להוריד ממנו חלקים מהקובץ שהורדתם הסתיימה.

**דוגמא להעלאת קובץ חדש לרשת:**

* אדם מסוים רוצה להעלות את הקובץ A באמצעות התוכנה.
* התוכנה יוצרת בשבילו קובץ yftf שמכיל את הפרטים עליו וכתובת שרת ה-Tracker.
* בשרת ה-Tracker הקובץ A נרשם והשרת מתחיל לנהל את האנשים שמורידים ומעלים אותו.
* אותו אדם מפרסם את קובץ ה-yftf ברשת.
* לקוחות חדשים שמורידים את הקובץ A מצטרפים לרשימה הלקוחות בשרת ה-Tracker.
* אותו אדם ששיתף את הקובץ A, מעלה אותו למורידים החדשים עד שהוא מעלה את כל הקובץ למספר מורידים חדשים שיוכלו להחליף אותו בעבודתו להעלות את הקובץ למורידים חדשים.
* כאשר הם מסיימים להוריד את הקובץ A (או חלקים ממנו) הם מתחילים להעלות אותו לאנשים חדשים שמורידים אותו.

**מרכיבים:**

* **מעלה ראשוני-** האיש הראשוני שמעלה את הקובץ החדש לרשת. הוא מעלה את הקובץ למורידים הראשונים.
* **קובץ yftf-** מכיל מידע על הקובץ אותו משתפים ועל כתובות שרתי ה-Trackers שעוקבים אחרי שיתוף הקובץ ברשת.
* **שרתים ברשת-** הם אלו שמכילים את קבצי ה-yftf ולקוחות שרוצים להוריד קובץ מסוים, מורידים מהם את קובץ ה-yftf שמפנה אותם לשרתי ה-Tracker שעוזרים להם להוריד את הקובץ במהירות בזכות השיתוף בין האנשים.
* **שרתי Tracker-** שרתים שמקשרים בין המעלים והמורידים של קבצים מסוימים שהם מטפלים בהם. הם גורמים לשיתוף של הקובץ בין אנשים רבים ובכך הם גורמים להורדה מהירה יותר של הקבצים. הם גורמים לאנשים שכבר הורידו קובץ מסוים (חלקו או כולו), להעלות חלקים ממנו לאנשים אחרים שמורידים את אותם חלקים.
* **לקוחות שמורידים קבצים-** נמצאים ברשימות הלקוחות שבשרתי ה-Tracker כמורידים של הקובץ אותו הם מורידים. שרתי ה-Tracker מקשרים אותם לאנשים שמעלים את אותו הקובץ (חלקים ממנו שלבסוף יהיו כל הקובץ). יכולים גם בו זמנית להעלות חלקים מהקובץ שהם כבר הורידו לרשת לאנשים אחרים.
* **לקוחות שמעלים קבצים-** נמצאים ברשימות הלקוחות שבשרתי ה-Tracker כמעלים של הקובץ אותו הם מעלים והם מעלים חלקים ממנו לאנשים שמורידים את הקובץ ושרתי ה-Tracker מפנים אותם אליהם. יכולים גם בו זמנית להוריד את אותו הקובץ מהרשת ומאנשים אחרים.

**פירוט תוכנה:**

**לקוח:**

* **משתף קובץ חדש-** תהיה אפשרות לשתף לרשת קובץ חדש. הלקוח יבחר באפשרות זאת ויבחר את הקובץ שהוא רוצה להעלות שנמצא במחשב שלו. התוכנה תייצר לו קובץ yftf שיכיל את המידע על הקובץ החדש והוא יכיל את הכתובת לשרת ה-Tracker של התוכנה בלבד (משום שלתוכנה יהיה פרוטוקול משלה שלא יוכל לתקשר עם שרתי Tracker אחרים ברשת). התוכנה תקשר עם שרת ה-Tracker ושרת ה-Tracker יוסיף רשימה חדשה של המעלים והמורידים של הקובץ החדש. הלקוח ישתף את קובץ ה-yftf באינטרנט ויהיה עליו להעלות את הקובץ החדש למורידים החדשים עד שמספר אנשים יורידו את כולו (אם הוא לא יעלה אותו לאנשים לא יהיה שיתוף של הקובץ בין כולם ולכן שיתוף הקובץ ברשת לא יעבוד).
* **מוריד-** לקוח מוריד מהאינטרנט קובץ yftf ותהיה בתוכנה אפשרות להורדת קובץ. כאשר הוא יבחר באפשרות זאת ויבחר בקובץ ה-yftf, התוכנה תצרף אותו לשרת ה-Tracker לרשימה של הלקוחות של אותו קובץ כמוריד. ואז התוכנה תמשיך לתקשר עם שרת ה-Tracker והוא יפנה אותה לחיבור עם אנשים שמעלים חלקים מאותו קובץ והלקוח יוריד מהם את החלקים שהם משתפים עד שלבסוף יבנה הקובץ כולו. התוכנה תבדוק את האמינות החלק שהיא הורידה על ידי הנתונים שנמצאים בקובץ ה-yftf, אם הוא לא תקין היא תפיל אותו ותתחיל להוריד אותו מחדש מלקוח אחר שמעלה את אותו החלק של הקובץ. בו בזמן הלקוח יוכל להעלות חלקים מהקובץ שהוא כבר הוריד.
* **מעלה-** ברגע שלקוח הוריד את החלק הראשון מהקובץ, התוכנה מתקשרת עם שרת ה-Tracker ובו הלקוח יתווסף לרשימת הלקוחות כמעלה הקובץ. כל עוד הלקוח מוריד את הקובץ ולא עוצר את העלאתו כאשר הסתיימה ההורדה, הלקוח יעלה חלקים מהקובץ שהוא כבר הוריד לאנשים שמורידים את אותם חלקים. שרת ה-Tracker יגיד ללקוח איזה חלק מהקובץ להעלות ולאיזה כתובת. ברגע שהוא מסיים להעלות לאותה כתובת הוא חוזר על אותו התהליך כל עוד הלקוח נותן לתוכנה להמשיך להעלות.

**שרת ה-Tracker:**

הוא אחרי על קישור המורידים והמעלים של הקבצים ברשת בהם הוא מטפל. הוא מתקשר עם התוכנות של הלקוחות ומודיע להם לאיזה כתובות להעלות איזה חלקים של קובץ מסוים ואיזה חלקים של אותו הקובץ להוריד מאיזו כתובת. בשיתוף קובץ חדש השרת יבנה רשימה חדשה של מורידים ומעלים של הקובץ. כאשר יתווספו מורידים של הקובץ, תוכנת הלקוח תודיע לשרת ה-Tracker והוא יוסיף אותם לרשימת הלקוחות כמורידים שבה ירשם איזה חלקים הלקוח כבר הוריד. כאשר יתווספו מעלים של הקובץ, תוכנת הלקוח תודיע לשרת ה-Tracker והוא יוסיף אותם לרשימת הלקוחות כמעלים. בנוסף, במשך כל תהליך ההורדה, השרת עוקב על איזה חלקים הלקוח כבר הוריד והוא שומר אותם ברשימה. מכאן השרת יצור ביעילות המרבית את שיתוף הקובץ (חלקיו) ברשת בין המורידים למעלים.

**קבצי yftf:**

מורכב מהנתונים הבאים (המחרוזות מקודדות ב-UTF-8):

* **Announce-** כתובת ה-URL של שרת ה-Tracker.
* **Info-** ספרייה שמכילה את כל הנתונים הבאים על הקובץ או הקבצים המשותפים:
  + **Name-** שם הקובץ או התיקייה (שמכילה את הקבצים) המשותפים.
  + **Piece Length-** אורך כל חלק של כל קובץ או קבצים ב-Bytes (חוץ מהחלק האחרון שיכול להכיל פחות Bytes). במקרה ויהיה מספר קבצים שונים, החלקים יוכלו להכיל קבצים שונים. אורך כל חלק יהיה 256 KiB.
  + **Num Pieces-** כמות החלקים שאליו התחלק הקובץ המשותף.
  + **Pieces Hash-** (במקרה של שיתוף קובץ אחד בלבד) רשימה של Hash-ים של כל חלק בקובץ. ה-Hash יהיה מסוג SHA-1. כל Hash יהיה באורך 160 Bit.
  + **Length-** (במקרה של שיתוף קובץ אחד בלבד) אורך הקובץ המשותף ב-Bytes.
  + **Hash-** (במקרה של שיתוף קובץ אחד בלבד) Hash מסוג SHA-1 באורך 160 Bit שיתבצע על הקובץ.
  + **Files-** (במקרה של שיתוף יותר מקובץ אחד) רשימה שמכילה ספריות לכל קובץ שמכילות את הנתונים הבאים על כל קובץ:
    - **Path-** מחרוזת של מיקום הקובץ בתוך התיקייה הראשית.
    - **Length-** אורך הקובץ המשותף ב-Bytes.
    - **Hash-** Hash מסוג SHA-1 באורך 160 Bit שיתבצע על כל קובץ.
    - **Pieces Hash-** רשימה של Hash-ים של כל חלק בקובץ. ה-Hash יהיה מסוג SHA-1. כל Hash יהיה באורך 160 Bit.

**פרוטוקולי YFT:**

הלקוח ושרת ה-Tracker יוכלו לנהל מספר שיחות במקביל בעזרת שימוש ב-thread (ספרייה ב-Python העוזרת לנהל מספר שיחות במקביל) וב-Tornado (ספרייה ב-Python לשרת ולקוח שפועלים על פרוטוקול HTTP שעל TCP בצורה אסינכרונית).

**בין הלקוח לשרת ה-Tracker:**

חלק ראשון בשרת ה-Tracker. החלק שמנהל את השיחה בין הלקוחות שמעלים ומורידים לבין השרת.

בבקשה הלקוח שולח פרטים על עצמו (מזהה וכתובת ip), על הקובץ שהוא רוצה להוריד ועל מצב ההורדה/העלאה של הלקוח בנוגע לקובץ המסוים. אם הוא מוריד הוא מודיע על האינדקס של החלק שהוא רוצה להוריד. על הדרך הוא גם מודיע על אינדקס של חלק שהוא סיים להוריד (אם יש). בנוסף, אם הלקוח הוא מעלה (הודיע על כך שהוא יכול להעלות), הוא מודיע על הפורט שעליו הוא מאזין ומחכה לבקשה ממוריד.

בתגובה או שהשרת יחזיר הודעת שגיאה אם הייתה, או שיחזיר את המידע שהלקוח ביקש. המידע יכיל תמיד את מזהה הקובץ וסוג התגובה (אם למעלה או מוריד). אם סוג התגובה היא למוריד אז כתובת ה-ip תפנה אותו ללקוח שיעלה אליו את המידע המבוקש על הפורט שישלח אליו גם בתגובה. בנוסף, ישלח אליו מספר החלק שאותו הוא צריך לבקש מהמעלה. אם סוג התגובה היא למעלה אז התגובה תודיע לו על אישור בקשתו אם אין שגיאה. מציאת המעלה המתאים תתבצע מהמידע שנמצא באובייקטים השמורים בשרת.

במקרה והשרת לא מוצא התאמה בין מעלה למוריד, השרת ישלח שגיאה מתאימה. עד אשר תמצא התאמה, הלקוח יצטרך לשלוח בקשה חדשה כל כמה שניות.

פרוטוקול התקשורת מבוסס על פרוטוקול HTTP שעל TCP.

**כותרות הבקשה:**

YFT-Upload-Piece ו-YFT-Request-Piece-Index לא יכולים להיות באותה בקשה יחד. אם יהיו, השרת לא יתייחס ל-YFT-Upload-Piece.

* **YFT-Info-Hash-** Hash מסוג SHA-1 באורך 160 Bit שיתבצע על חלק ה-Info בקובץ ה-yftf. זה הוא המזהה של הקובץ שהלקוח רוצה להוריד או להעלות.
* **YFT-Peer-id-** מחרוזת רנדומלית באורך 20 תווים המכילה את מזהה הלקוח.
* **YFT-Peer-ip-** כתובת ה-ip של הלקוח.
* **YFT-Peer-Status-** מצב ההורדה/העלאה של הלקוח בנוגע לקובץ המסוים. אם 0, זאת אומרת שהלקוח מתחיל להוריד את הקובץ ולכן השרת יצרף אותו לרשימת המורידים/מעלים (השרת יבדוק קודם אם הוא כבר לא נמצא שם). אם 1 זאת אומרת שהלקוח כבר מוריד או מעלה והוא נמצא ברשימה. אם 2, השרת ימחק את הלקוח מרשימת המורידים/מעלים והוא יתעלם משאר הכותרות.
* **YFT-Port-** (במקרה והלקוח מעלה) מספר הפורט שעליו הלקוח מאזין ומחכה לבקשה מלקוח אחר שמוריד.
* **YFT-Request-Piece-Index-** (במקרה והלקוח מוריד) האינדקס של החלק הבא שהלקוח רוצה להוריד.
* **YFT-Finished-Piece-Index-** (במקרה והלקוח סיים להוריד חלק) האינדקס של החלק שהלקוח סיים להוריד ויכול להעלות עכשיו.
* **YFT-Upload-Piece-** (במקרה והלקוח סיים להעלות חלק/ים או יכול להתחיל להעלות חלק לראשונה) מכיל 1 אם הלקוח יכול להעלות חלק/ים.

**כותרות התגובה:**

* **YFT-Info-Hash-** Hash מסוג SHA-1 באורך 160 Bit שיתבצע על חלק ה-Info בקובץ ה-yftf. זה הוא המזהה של הקובץ שהלקוח רוצה להוריד או להעלות.
* **YFT-Error-** (במקרה ויש שגיאה) מידע על השגיאה (לאחריו לא יופיעו הכותרות הבאות).
* **YFT-Type-** יכיל 1 אם הלקוח (שאליו נשלחת התגובה) הוא מעלה ו-0 אם הוא מוריד.
* **YFT-ip-** (במקרה והלקוח מוריד) כתובת ה-ip של הלקוח שמעלה.
* **YFT-Piece-Index-** (במקרה והלקוח מוריד) האינדקס של החלק שאותו הלקוח המוריד יבקש מהלקוח המעלה.
* **YFT-Port-** (במקרה והלקוח מוריד) מספר הפורט שעליו מאזין הלקוח המעלה ומחכה לבקשה מהמוריד.
* **YFT-Port-** (במקרה והלקוח מעלה) מספר הפורט הלקוח המעלה להאזין עד לסיום שיחתו עם המוריד.

**בין הלקוח המשתף קובץ חדש לשרת ה-Tracker:**

החלק השני בשרת ה-Tracker. החלק שאחראי על יצירת האובייקטים שמנהלים את המעלים והמורידים של הקובץ החדש.

הלקוח שולח בקשה לשרת שיחל לנהל את הקובץ שלו. הבקשה מכילה (באזור ה-body) את קובץ ה-yftf (שנשמר בשרת) ומידע על הלקוח (בכותרות). בתגובה השרת יבנה את האובייקט (הוא גם יחשב את מספר החלקים בקובץ) ויוסיף את הלקוח לרשימת הלקוחות באובייקט כלקוח שמעלה (שם יהיה רשום שיש בידיו את כל החלקים) וישלח לו בתגובה הודעת אישור או שגיאה אם קרתה. האובייקטים יישמרו בשרת כקבציי טקסט בעזרת שימוש בפונקציית pickle.

פרוטוקול התקשורת מבוסס על פרוטוקול HTTP שעל TCP.

**הכותרות וה-body בבקשה:**

* **YFT-Peer-id-** מחרוזת באורך 20 תווים רנדומליים המכילה את מזהה הלקוח.
* **YFT-yftf-Hash-** Hash מסוג SHA-1 באורך 160 Bit שיתבצע על קובץ ה-yftf. זה בדיקה שהקובץ שנשלח בגוף הבקשה הגיעה ללא שגיאות.
* **YFT-Peer-ip-** כתובת ה-ip של הלקוח.
* **YFT-Peer-Status-** מצב ההורדה/העלאה של הלקוח בנוגע לקובץ המסויים. יכול להכיל 0 או 1 או 2 כמו שהוסבר כותרת הבקשה של הפרוטוקול בין הלקוח לשרת ה- Tracker (במקרה זה יהיה חייב להיות 0).
* **YFT-Port-** מספר הפורט שעליו הלקוח מאזין ומחכה לבקשה מלקוח אחר שמוריד.
* **YFT-Upload-Piece-** מכיל 1 אם הלקוח יכול להעלות חלק/ים (במקרה הזה יהיה חייב להיות 1).
* **Body-** קובץ ה-yftf שמכיל את המידע על הקובץ המשותף.

**כותרות התגובה:**

* **YFT-Info-Hash-** Hash מסוג SHA-1 באורך 160 Bit שיתבצע על חלק ה-Info בקובץ ה-yftf. זה הוא המזהה של הקובץ המשותף.
* **YFT-Error-** (במקרה ויש שגיאה) מידע על השגיאה (לאחריו לא יופיעו הכותרות הבאות).
* **YFT-Type-** יכיל 1 אם הלקוח (שאליו נשלחת התגובה) הוא מעלה ו-0 אם הוא מוריד (במקרה הזה יהיה 1).
* **YFT-Port-** (במקרה והלקוח מעלה) מספר הפורט הלקוח המעלה להאזין עד לסיום שיחתו עם המוריד.

**המבנה של אובייקט:**

כל הניהול של הקובץ המשותף יתבצע בעזרת שימוש באובייקט שלו שבשרת. באובייקט ישמרו מזהה הקובץ המשותף, מספר החלקים שבו והלקוחות (המזהה שלהם) שמורידים ומעלים אותו והמידע עליהם. כל עוד הלקוח לא שלח בכותרת YFT-Peer-Status 2 אז הוא יופיע ברשימה. כאשר לקוח מבקש להוריד חלק מסוים, השרת יחפש את המעלה המתאים ביותר על ידי כך שהוא יעבור על רשימת הלקוחות, יבדוק אם הם יכולים להעלות ואם הם יכולים להעלות את החלק המתאים. לקוח שיכול להעלות הוא לקוח ששלח לשרת בקשה עם YFT-Upload-Piece שבו יש 1 ולכן הסטאטוס של הלקוח יהיה 1. כאשר השרת ימצא התאמה הוא יהפוך את הסטאטוס של הלקוח ל-0, ישלח למוריד את הנתונים הנחוצים על המעלה מהטבלה וימחק את הפורט שעליו מאזין המעלה. בנוסף השרת ישמור רשימה של כל החלקים שהלקוח הוריד וישמור את כתובת ה-ip שלו.

**דוגמא לשני אובייקטים של קובץ yftf שרשימת הלקוחות שלהם מכילה אובייקטים של לקוח:**

|  |  |
| --- | --- |
| Info-Hash | Info-Hash |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  | | --- | | Peer-id, Peer-ip, State, [Pieces], Port | | Peer-id, 10.1.1.4, 0, [0, 1, 3], None | | Peer-id, 10.1.1.5, 1, [0, 1, 2], 1024 | | Num-Pieces | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  | | --- | | Peer-id, Peer-ip, State, [Pieces], Port | | Peer-id, 10.1.1.1, 0, [0, 1], None | | Peer-id, 10.1.1.2, 1, [0, 1, 2], 8080 | | Num-Pieces | |

**בין הלקוח המוריד ללקוח המעלה:**

תקשורת בין שני לקוחות ששרת ה-Tracker קישר ביניהם, אחד המוריד והאחר המעלה. המוריד קיבל מהשרת את כתובת ה-ip של המעלה, את הפורט שעליו המעלה מאזין ואת החלק שעליו לבקש מהמעלה. המעלה הוא בעצם השרת והמוריד הוא הלקוח. המוריד מתחיל שיחה עם המעלה ומבקש ממנו את הקובץ שהשרת אמר לו לבקש. במקרה של שגיאה, השיחה תתנתק. בתגובה המעלה שולח למוריד את ה-Info-Hash ואת המידע של החלק. כאשר הכול נשלח בהצלחה השיחה מסתיימת והמוריד בודק את אמיתות המידע עם ה-hash המתאים שבקובץ ה-yftf. במקרה ולא זהה, המוריד ישלח בקשה חדשה לשרת לאותו חלק. בכל מהלך השיחה קיימת הגבלת זמן כך שבמקרה של תקלה בצד אחד, הצד האחר לא יתקע. במקרה ונגמרת הגבלת הזמן, המוריד/מעלה ישלח בקשה חדשה לשרת.

השיחה היא על פרוטוקול P2P שעל TCP.

**בקשת הלקוח:**

* **Info-Hash-** Hash מסוג SHA-1 באורך 160 Bit (כ-40 תווים כי בבסיס 16) שיתבצע על חלק ה-Info בקובץ ה-yftf. זה המזהה של הקובץ המשותף שהמעלה ישלח למוריד.
* **Piece-Index-** יכיל את האינדקס של החלק שאותו המוריד רוצה לקבל מהמעלה (יהיה באורך שמונה תווים. במקרה ויש עודף בתווים, התווים הנותרים משמאל יהיו 0).

**מבנה המידע:**

* **Info-Hash-** Hash מסוג SHA-1 באורך 160 Bit (כ-40 תווים כי בבסיס 16) שיתבצע על חלק ה-Info בקובץ ה-yftf. זה המזהה של הקובץ המשותף שהמעלה שולח למוריד.
* **Data-** המידע שמכיל החלק שאותו צריך המעלה לשלוח למוריד (המוריד יודע את אורך המידע משום שזה מופיע בקובץ ה-yftf).

**דומה ושונה מ-BitTorrent:**

**דומה:**

* הלקוחות שולחים את המידע של הקובץ המשותף אחד לשני ולא דרך שרת מרכזי.
* יש את שרת ה-Tracker שמנהל את הקישורים בין המורידים למעלים.
* יש בדיקה אם הקובץ שנשלח תקין והמידע שלו לא שונה.

**שונה:**

* הלקוח יכול להחליט את דרך ההורדה שלו. הוא יכול להחליט אם הוא מוריד כל הזמן חלקים רנדומליים או שהוא מוריד את החלקים בסדר שלהם (Streaming, שזה מה שיהיה בלקוח שלנו).
* קובץ ה-yftf מחלק כל קובץ בנפרד. אם יש כמה קבצים משותפים בו זמנית בשיתוף אחד אז הוא לא מחלק את כל המידע ביחד לחלקים אלה כל מידע של כל קובץ בנפרד. זאת על מנת שהלקוח לא יצטרך להוריד את כל הקבצים המשותפים.
* בעת הורדה או העלאה של קובץ מסוים, הלקוח יכול לתקשר רק עם שרת Tracker אחד של אותו הקובץ ולא עם כמה במקביל.
* חסרים כמה חלקים של השיחה בין הלקוחות שקיימים ב-BitTorrent, כגון Choke ו-Interested.
* חסרים כמה אלגוריתמים נוספים לשיתוף קבצים מהיר יותר שקיימים ב-BitTorrent, כגון DHT ו-Fast Peers Extensions.

**צעדים להמשך:**

ישנם הרבה שיפורים נוספים שאפשר להוסיף לפרויקט שלי ויכול להיות שאוסיף אותם בהמשך:

* ליישם אלגוריתמים נוספים לשיתוף יעיל של הקבצים המשותפים.
* ליישם דרכים נוספות לשיתוף קבצים מהיר יותר, כמו DHT.
* ליישם פרוטוקול P2P שעובד גם מאחורי שרתי NAT.
* ליישם שיחה יעילה יותר בפרוטוקול P2P כך שלא ייוצרו תקלות ותקיעות מיותרות.
* לאבטח יותר את השיחה בין הלקוחות כך שלא יהיו בעיות אבטחה.
* ליישם לקוח שיוכל לתקשר עם כמה שרתי Tracker במקביל בנוגע לקובץ מסוים שהוא רוצה להוריד או להעלות.
* ליישם לקוח שיוכל להוריד או להעלות כמה קבצים במקביל.
* ליישם לקוח שאם הוא יוצא מהתוכנה וחוזר אליה אז הוא יוכל להמשיך להעלות קבצים ישנים שהוריד ועדיין נמצאים על המחשב שלו.

**סיכום:**

במהלך הפרויקט חקרתי ולמדתי הרבה דברים חדשים:

* חקרתי את BitTorrent לעומק, למדתי על הפרוטוקולים שלו ועל איך שהוא עובד.
* חקרתי את פרוטוקול P2P ולמדתי על הבעיות שבו.
* למדתי הרבה דברים חדשים על שיתוף קבצים חברתי ברשת.
* למדתי איך לחקור נושאים חדשים שאני לא מכיר.
* למדתי איך לתכנן בניית פרויקטים כך שהבנייה תתבצע בצורה יעילה.

**קוד:**

**לקוח:**

על מנת להריץ את הלקוח יש להפעיל את client\_gui.py.

**client\_action.py:**

*"""  
Manging the client actions file.  
"""***import** hashlib  
**import** json  
**from** tornado.httpclient **import** HTTPRequest  
**from** tornado.httputil **import** HTTPHeaders  
*# from random import randint***import** client\_server\_protocol  
  
  
**class** ClientAction(object):  
 *"""  
 The class the manage the client actions.  
 """* **def** \_\_init\_\_(self, command, yftf\_files, info\_hash, peer\_id, peer\_ip, port\_range, num\_workers, queue\_size):  
 *"""  
 Starting a new action.  
 """* self.peer\_id = peer\_id  
 self.peer\_ip = peer\_ip  
 self.port\_range\_in\_use = dict(zip(port\_range, [False] \* len(port\_range)))  
  
 self.yftf\_files = yftf\_files  
 self.info\_hash = info\_hash  
 self.pieces\_requested\_index = dict()  
 self.finished\_pieces\_index = list()  
  
 self.num\_workers = num\_workers  
 self.queue\_size = queue\_size  
  
 self.num\_requests = 0  
 self.first\_uploader = 0  
 self.command = command  
  
 **def** basic\_request(self, headers):  
 *"""  
 Basic request to server.  
 """* **return** HTTPRequest(url=self.yftf\_files[self.info\_hash][0][**"Announce"**], method=**"GET"**,  
 headers=HTTPHeaders(headers), allow\_nonstandard\_methods=True)  
  
 **def** find\_unused\_port(self):  
 *"""  
 Finding unused port.  
 """* port = int()  
  
 **for** port, in\_use **in** self.port\_range\_in\_use.iteritems():  
 **if not** in\_use:  
 **return** port  
  
 **return** port  
  
 **def** upload\_request(self):  
 *"""  
 Upload request to server.  
 """* port = self.find\_unused\_port()  
  
 **if not** port:  
 **return  
  
 if** len(self.finished\_pieces\_index) > 0:  
 **if** len(self.finished\_pieces\_index) == 1:  
 finished\_piece = self.finished\_pieces\_index[0]  
 **else**:  
 finished\_piece = self.finished\_pieces\_index  
  
 req = self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.upload\_request(self.info\_hash, self.peer\_id, self.peer\_ip,  
 port, finished\_piece))  
 self.finished\_pieces\_index = list()  
 **else**:  
 req = self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.upload\_request(self.info\_hash, self.peer\_id, self.peer\_ip,  
 port))  
  
 **return** req  
  
 **def** request(self):  
 *"""  
 Getting the right request of the action.  
 """* **if** self.num\_requests **is** 0 **and** self.command **is** 0:  
 req = self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.start\_new\_download\_request(self.info\_hash, self.peer\_id,  
 self.peer\_ip, 0))  
  
 **if** self.info\_hash **not in** self.pieces\_requested\_index:  
 self.pieces\_requested\_index.update({self.info\_hash: []})  
  
 self.pieces\_requested\_index[self.info\_hash] += [self.num\_requests]  
 self.num\_requests += 1  
  
 **return** req  
  
 **elif** self.first\_uploader **is** 0 **and** self.command **is** 1:  
 port = self.find\_unused\_port()  
  
 **if not** port:  
 **return** self.first\_uploader += 1  
  
 req = self.basic\_request(client\_server\_protocol.ClientServerProtocol.new\_share\_request(  
 hashlib.sha1(json.dumps(self.yftf\_files[self.info\_hash][0])).hexdigest(), self.peer\_id, self.peer\_ip,  
 port))  
 req.body = json.dumps(self.yftf\_files[self.info\_hash][0])  
  
 **return** req  
  
 **elif** self.command **is** 2:  
 **return** self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.finish\_sharing\_request(self.info\_hash, self.peer\_id,  
 self.peer\_ip))  
  
 **elif** self.command **is** 1:  
 req = self.upload\_request()  
  
 **if not** req:  
 **return  
  
 return** req  
  
 **elif** self.command **is** 0 **and** int(self.yftf\_files[self.info\_hash][0][**"Info"**][**"Num Pieces"**]) <= self.num\_requests:  
 req = self.upload\_request()  
  
 **if not** req:  
 **return  
  
 return** req  
  
 **else**:  
 *# if randint(0, 1):  
 # req = self.upload\_request()  
 #  
 # if not req:  
 # return  
 #  
 # return req  
 #  
 # else:* **if** len(self.finished\_pieces\_index) > 0:  
 **if** len(self.finished\_pieces\_index) == 1:  
 finished\_piece = self.finished\_pieces\_index[0]  
 **else**:  
 finished\_piece = self.finished\_pieces\_index  
  
 req = self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.download\_request(self.info\_hash, self.peer\_id,  
 self.peer\_ip, self.num\_requests,  
 finished\_piece))  
 self.finished\_pieces\_index = list()  
  
 self.pieces\_requested\_index[self.info\_hash] += [self.num\_requests]  
 self.num\_requests += 1  
  
 **return** req  
  
 **else**:  
 req = self.basic\_request(  
 client\_server\_protocol.ClientServerProtocol.download\_request(self.info\_hash, self.peer\_id,  
 self.peer\_ip, self.num\_requests))  
  
 self.pieces\_requested\_index[self.info\_hash] += [self.num\_requests]  
 self.num\_requests += 1  
  
 **return** req  
  
 **def** handle\_response(self, yftf\_files, response\_headers):  
 *"""  
 Handling the response from the server.  
 """* self.yftf\_files = yftf\_files  
  
 **return** client\_server\_protocol.ClientServerProtocol.handle\_response(self.yftf\_files, self.pieces\_requested\_index,  
 response\_headers)

**client\_gui.py:**

*"""  
The GUI file.  
Run this to start the client.  
"""***from** Tkinter **import** \*  
**import** tkFileDialog  
**import** yft\_client  
  
  
**class** ClientGUI(Tk):  
 *"""  
 The main GUI class.  
 """* **def** \_\_init\_\_(self):  
 *"""  
 Starting the GUI.  
 """* Tk.\_\_init\_\_(self)  
  
 yft\_client\_obj = yft\_client.YFTClient()  
 container = Frame(self)  
  
 container.pack(side=**"top"**, fill=**"both"**, expand=True)  
  
 container.grid\_rowconfigure(0, weight=1)  
 container.grid\_columnconfigure(0, weight=1)  
  
 self.frames = {}  
  
 **for** page **in** (StartPage, NewDownloadPage, NewSharePage, StopUploadPage):  
 frame = page(container, self, yft\_client\_obj)  
 self.frames[page] = frame  
  
 frame.grid(row=0, column=0, sticky=**"nsew"**)  
  
 self.show\_frame(StartPage)  
  
 **def** show\_frame(self, page):  
 *"""  
 Show different frame.  
 """* frame = self.frames[page]  
 frame.tkraise()  
  
  
**class** StartPage(Frame):  
 *"""  
 The start page class.  
 """* **def** \_\_init\_\_(self, parent, controller, yft\_client\_obj):  
 *"""  
 Organize its frame.  
 """* Frame.\_\_init\_\_(self, parent)  
  
 self.yft\_client\_obj = yft\_client\_obj  
  
 change\_downloads = Button(self)  
 change\_downloads[**"text"**] = **"Change Downloads Dir"** change\_downloads[**"command"**] = self.choose\_downloads\_path  
  
 self.download\_path = Entry(self, textvariable=self.yft\_client\_obj.downloads\_dir\_path, width=80)  
 self.download\_path.config(state=DISABLED)  
  
 change\_yftf = Button(self)  
 change\_yftf[**"text"**] = **"Change yftf Dir"** change\_yftf[**"command"**] = self.choose\_yftf\_path  
  
 self.yftf\_path = Entry(self, textvariable=self.yft\_client\_obj.downloads\_dir\_path, width=80)  
 self.yftf\_path.config(state=DISABLED)  
  
 change\_downloads.grid(row=0, column=0, padx=10, pady=10)  
  
 self.download\_path.grid(row=0, column=1, padx=10, pady=10)  
  
 change\_yftf.grid(row=1, column=0, padx=10, pady=10)  
  
 self.yftf\_path.grid(row=1, column=1, padx=10, pady=10)  
  
 new\_download\_button = Button(self)  
 new\_download\_button[**"text"**] = **"New Download"** new\_download\_button[**"command"**] = **lambda**: controller.show\_frame(NewDownloadPage)  
  
 new\_share\_button = Button(self)  
 new\_share\_button[**"text"**] = **"New Share"** new\_share\_button[**"command"**] = **lambda**: controller.show\_frame(NewSharePage)  
  
 stop\_update\_button = Button(self)  
 stop\_update\_button[**"text"**] = **"Stop Update/Download"** stop\_update\_button[**"command"**] = **lambda**: controller.show\_frame(StopUploadPage)  
  
 new\_download\_button.grid(row=2, column=0, padx=10, pady=10)  
 new\_share\_button.grid(row=2, column=1, padx=10, pady=10)  
 stop\_update\_button.grid(row=3, column=0, padx=10, pady=10)  
  
 **def** choose\_downloads\_path(self):  
 *"""  
 Choose downloads path.  
 """* self.yft\_client\_obj.downloads\_dir\_path = tkFileDialog.askdirectory(  
 parent=self, initialdir=**"/"**,  
 title=**'Please select the directory you want your downloads to be saved'**)  
  
 self.download\_path.config(state=NORMAL)  
 self.download\_path.delete(0, END)  
 self.download\_path.insert(0, self.yft\_client\_obj.downloads\_dir\_path)  
 self.download\_path.config(state=DISABLED)  
  
 **def** choose\_yftf\_path(self):  
 *"""  
 Choose yftf path.  
 """* self.yft\_client\_obj.yftf\_dir\_path = tkFileDialog.askdirectory(  
 parent=self, initialdir=**"/"**,  
 title=**'Please select the directory you want your yftf files to be saved'**)  
  
 self.yftf\_path.config(state=NORMAL)  
 self.yftf\_path.delete(0, END)  
 self.yftf\_path.insert(0, self.yft\_client\_obj.yftf\_dir\_path)  
 self.yftf\_path.config(state=DISABLED)  
  
  
**class** NewDownloadPage(Frame):  
 *"""  
 New download frame class.  
 """* **def** \_\_init\_\_(self, parent, controller, yft\_client\_obj):  
 *"""  
 Organize the frame.  
 """* Frame.\_\_init\_\_(self, parent)  
  
 self.yft\_client\_obj = yft\_client\_obj  
  
 home\_button = Button(self, text=**"Back to Home"**, command=**lambda**: controller.show\_frame(StartPage))  
  
 change\_yftf = Button(self)  
 change\_yftf[**"text"**] = **"Change yftf Path"** change\_yftf[**"command"**] = self.choose\_yftf\_path  
  
 self.yftf\_path = Entry(self, textvariable=self.yft\_client\_obj.yftf\_path, width=80)  
 self.yftf\_path.config(state=DISABLED)  
  
 download\_button = Button(self)  
 download\_button[**"text"**] = **"Download"** download\_button[**"command"**] = self.download  
  
 home\_button.grid(row=0, column=0, padx=10, pady=10)  
 change\_yftf.grid(row=1, column=0, padx=10, pady=10)  
 self.yftf\_path.grid(row=1, column=1, padx=10, pady=10)  
 download\_button.grid(row=2, column=0, padx=10, pady=10)  
  
 **def** choose\_yftf\_path(self):  
 *"""  
 Choose yftf path.  
 """* self.yft\_client\_obj.yftf\_path = tkFileDialog.askopenfilename(  
 parent=self, initialdir=**"/"**,  
 title=**'Please select the yftf file you want to download from'**)  
  
 self.yftf\_path.config(state=NORMAL)  
 self.yftf\_path.delete(0, END)  
 self.yftf\_path.insert(0, self.yft\_client\_obj.yftf\_path)  
 self.yftf\_path.config(state=DISABLED)  
  
 **def** download(self):  
 *"""  
 Start download.  
 """* self.yft\_client\_obj.command = 0  
 self.yft\_client\_obj.new\_action()  
  
  
**class** NewSharePage(Frame):  
 *"""  
 New share frame class.  
 """* **def** \_\_init\_\_(self, parent, controller, yft\_client\_obj):  
 *"""  
 Organize the frame.  
 """* Frame.\_\_init\_\_(self, parent)  
  
 self.yft\_client\_obj = yft\_client\_obj  
  
 home\_button = Button(self, text=**"Back to Home"**, command=**lambda**: controller.show\_frame(StartPage))  
  
 change\_shared\_files\_dir\_path = Button(self)  
 change\_shared\_files\_dir\_path[**"text"**] = **"Change Shared Files Dir Path"** change\_shared\_files\_dir\_path[**"command"**] = self.choose\_yftf\_path  
  
 self.shared\_files\_dir\_path = Entry(self, textvariable=self.yft\_client\_obj.shared\_files\_dir\_path, width=80)  
 self.shared\_files\_dir\_path.config(state=DISABLED)  
  
 tracker\_url\_label = Label(self)  
 tracker\_url\_label[**"text"**] = **"Tracker Url:"** self.tracker\_url = Entry(self, textvariable=self.yft\_client\_obj.tracker\_url, width=80)  
  
 share\_button = Button(self)  
 share\_button[**"text"**] = **"Share"** share\_button[**"command"**] = self.share  
  
 home\_button.grid(row=0, column=0, padx=10, pady=10)  
 change\_shared\_files\_dir\_path.grid(row=1, column=0, padx=10, pady=10)  
 self.shared\_files\_dir\_path.grid(row=1, column=1, padx=10, pady=10)  
 tracker\_url\_label.grid(row=2, column=0, padx=10, pady=10)  
 self.tracker\_url.grid(row=2, column=1, padx=10, pady=10)  
 share\_button.grid(row=3, column=0, padx=10, pady=10)  
  
 **def** choose\_yftf\_path(self):  
 *"""  
 Choose yftf path.  
 """* self.yft\_client\_obj.shared\_files\_dir\_path = tkFileDialog.askdirectory(  
 parent=self, initialdir=**"/"**,  
 title=**'Please select the directory which there are the files you want to share'**)  
  
 self.shared\_files\_dir\_path.config(state=NORMAL)  
 self.shared\_files\_dir\_path.delete(0, END)  
 self.shared\_files\_dir\_path.insert(0, self.yft\_client\_obj.shared\_files\_dir\_path)  
 self.shared\_files\_dir\_path.config(state=DISABLED)  
  
 **def** share(self):  
 *"""  
 Start sharing the new file.  
 """* self.yft\_client\_obj.command = 1  
 self.yft\_client\_obj.tracker\_url = self.tracker\_url.get()  
 self.yft\_client\_obj.new\_action()  
  
  
**class** StopUploadPage(Frame):  
 *"""  
 Stop upload/download frame.  
 """* **def** \_\_init\_\_(self, parent, controller, yft\_client\_obj):  
 *"""  
 Organize the frame.  
 """* Frame.\_\_init\_\_(self, parent)  
  
 self.yft\_client\_obj = yft\_client\_obj  
  
 home\_button = Button(self, text=**"Back to Home"**, command=**lambda**: controller.show\_frame(StartPage))  
  
 change\_yftf = Button(self)  
 change\_yftf[**"text"**] = **"Change yftf Path"** change\_yftf[**"command"**] = self.choose\_yftf\_path  
  
 self.yftf\_path = Entry(self, textvariable=self.yft\_client\_obj.yftf\_path, width=80)  
 self.yftf\_path.config(state=DISABLED)  
  
 stop\_button = Button(self)  
 stop\_button[**"text"**] = **"Stop"** stop\_button[**"command"**] = self.stop  
  
 home\_button.grid(row=0, column=0, padx=10, pady=10)  
 change\_yftf.grid(row=1, column=0, padx=10, pady=10)  
 self.yftf\_path.grid(row=1, column=1, padx=10, pady=10)  
 stop\_button.grid(row=2, column=0, padx=10, pady=10)  
  
 **def** choose\_yftf\_path(self):  
 *"""  
 Choose yftf path.  
 """* self.yft\_client\_obj.yftf\_path = tkFileDialog.askopenfilename(  
 parent=self, initialdir=**"/"**,  
 title=**'Please select the yftf file you want to stop download/upload'**)  
  
 self.yftf\_path.config(state=NORMAL)  
 self.yftf\_path.delete(0, END)  
 self.yftf\_path.insert(0, self.yft\_client\_obj.yftf\_path)  
 self.yftf\_path.config(state=DISABLED)  
  
 **def** stop(self):  
 *"""  
 Stop upload/download.  
 """* self.yft\_client\_obj.command = 2  
 self.yft\_client\_obj.new\_action()  
  
  
**def** main():  
 *"""  
 Starting the client.  
 """* app = ClientGUI()  
 app.mainloop()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

**client\_server\_protocol.py:**

*"""  
The client server protocol file (client side).  
"""***class** ClientServerProtocol(object):  
 *"""  
 The client server protocol class.  
 """* @staticmethod  
 **def** new\_share\_request(info\_hash, peer\_id, peer\_ip, port):  
 *"""  
 Making new share request to the server.  
 """* **return** {**"YFT-Peer-id"**: peer\_id, **"YFT-Peer-ip"**: peer\_ip, **"YFT-yftf-Hash"**: info\_hash, **"YFT-Peer-Status"**: str(0),  
 **"YFT-Upload-Piece"**: str(1), **"YFT-Port"**: str(port)}  
  
 @staticmethod  
 **def** download\_request(info\_hash, peer\_id, peer\_ip, request\_piece\_index, finished\_piece\_index=None):  
 *"""  
 Making download request to the server.  
 """* headers = ClientServerProtocol.basic\_request(info\_hash, peer\_id, peer\_ip)  
 headers.update({**"YFT-Peer-Status"**: str(1), **"YFT-Request-Piece-Index"**: str(request\_piece\_index)})  
  
 **if** finished\_piece\_index:  
 **if** isinstance(finished\_piece\_index, list):  
 finished\_piece\_index = **', '**.join(finished\_piece\_index)  
 **else**:  
 finished\_piece\_index = str(finished\_piece\_index)  
  
 headers.update({**"YFT-Finished-Piece-Index"**: finished\_piece\_index})  
  
 **return** headers  
  
 @staticmethod  
 **def** upload\_request(info\_hash, peer\_id, peer\_ip, port, finished\_piece\_index=None):  
 *"""  
 Making upload request to the server.  
 """* headers = ClientServerProtocol.basic\_request(info\_hash, peer\_id, peer\_ip)  
 headers.update({**"YFT-Peer-Status"**: str(1), **"YFT-Upload-Piece"**: str(1), **"YFT-Port"**: str(port)})  
  
 **if** finished\_piece\_index:  
 **if** isinstance(finished\_piece\_index, list):  
 finished\_piece\_index = **', '**.join(finished\_piece\_index)  
 **else**:  
 finished\_piece\_index = str(finished\_piece\_index)  
  
 headers.update({**"YFT-Finished-Piece-Index"**: finished\_piece\_index})  
  
 **return** headers  
  
 @staticmethod  
 **def** start\_new\_download\_request(info\_hash, peer\_id, peer\_ip, request\_piece\_index):  
 *"""  
 Starting new download request to the server.  
 """* headers = ClientServerProtocol.basic\_request(info\_hash, peer\_id, peer\_ip)  
 headers.update({**"YFT-Peer-Status"**: str(0), **"YFT-Request-Piece-Index"**: str(request\_piece\_index)})  
  
 **return** headers  
  
 @staticmethod  
 **def** finish\_sharing\_request(info\_hash, peer\_id, peer\_ip):  
 *"""  
 Finished sharing request to the server.  
 """* headers = ClientServerProtocol.basic\_request(info\_hash, peer\_id, peer\_ip)  
 headers.update({**"YFT-Peer-Status"**: str(2)})  
  
 **return** headers  
  
 @staticmethod  
 **def** basic\_request(info\_hash, peer\_id, peer\_ip):  
 *"""  
 Basic request to the server.  
 """* **return** {**"YFT-Info-Hash"**: info\_hash, **"YFT-Peer-id"**: peer\_id, **"YFT-Peer-ip"**: peer\_ip}  
  
 @staticmethod  
 **def** handle\_response(yftf\_files, pieces\_requested\_index, response\_headers):  
 *"""  
 Handle response from the server.  
 """* **if "Yft-Info-Hash" not in** response\_headers.keys() **or** response\_headers[**"Yft-Info-Hash"**] **not in** yftf\_files.keys():  
 **return** 0, **"ERROR: Response not valid"** yftf\_json = yftf\_files[response\_headers[**"Yft-Info-Hash"**]][0]  
  
 **if "Yft-Error" in** response\_headers.keys():  
 **return** 0, yftf\_json[**"Info"**][**"Name"**] + **" - ERROR:"** + response\_headers[**"Yft-Error"**]  
  
 **if "Yft-Type" not in** response\_headers.keys():  
 **return** 0, yftf\_json[**"Info"**][**"Name"**] + **" - ERROR: Header is missing"  
  
 if** response\_headers[**"Yft-Type"**] **is** str(1):  
 **return** 1, response\_headers[**"Yft-Port"**]  
  
 **if** response\_headers[**"Yft-Info-Hash"**] **not in** pieces\_requested\_index.keys():  
 **return** 0, yftf\_json[**"Info"**][**"Name"**] + **" - ERROR: You didn't requested from this file"  
  
 if** int(response\_headers[**"Yft-Piece-Index"**]) **not in** pieces\_requested\_index[response\_headers[**"Yft-Info-Hash"**]]:  
 **return** 0, yftf\_json[**"Info"**][**"Name"**] + **" - ERROR: You didn't requested this piece"  
  
 return** 2, int(response\_headers[**"Yft-Piece-Index"**]), response\_headers[**"Yft-Ip"**], int(  
 response\_headers[**"Yft-Port"**])

**client\_worker.py:**

*"""  
The client worker file.  
"""***from** tornado.ioloop **import** IOLoop  
**from** tornado **import** gen, queues  
**from** tornado.httpclient **import** AsyncHTTPClient  
**import** socket  
**import** clients\_protocol  
**import** client\_action  
*# import thread***class** ClientWorker(object):  
 *"""  
 The class that manages the client worker.  
 """* **def** \_\_init\_\_(self, yftf\_files, num\_workers, queue\_size):  
 *"""  
 Organize the client worker.  
 """* self.yftf\_files = yftf\_files  
  
 self.num\_workers = num\_workers  
 self.queue\_size = queue\_size  
 self.queue = queues.Queue(self.queue\_size)  
  
 self.actions = {}  
  
 AsyncHTTPClient.configure(None, max\_clients=self.num\_workers)  
 self.http\_client = AsyncHTTPClient()  
  
 **def** handle\_response(self, response):  
 *"""  
 Handling the response from the server.  
 """* **if** response.error:  
 **print "Error:"**, response.error  
 **else**:  
 **if "Yft-Info-Hash" in** response.headers.keys():  
 data = list(  
 self.actions[response.headers[**"Yft-Info-Hash"**]].handle\_response(self.yftf\_files, response.headers))  
 **elif "Yft-Error" in** response.headers.keys():  
 **print** response.headers[**"Yft-Error"**]  
 **return  
 else**:  
 **print "Error: Response not valid"  
 return  
  
 if** data[0] **is** 0:  
 **print** data[1]  
  
 **elif** data[0] **is** 1:  
 port = int(data[1])  
 *# thread.start\_new\_thread(self.upload, (response.headers["Yft-Info-Hash"], port))* self.upload(response.headers[**"Yft-Info-Hash"**], port)  
  
 **else**:  
 piece\_index = data[1]  
 uploader\_ip = data[2]  
 uploader\_port = data[3]  
 *# thread.start\_new\_thread(self.download,  
 # (response.headers["Yft-Info-Hash"], piece\_index, uploader\_ip, uploader\_port))* self.download(response.headers[**"Yft-Info-Hash"**], piece\_index, uploader\_ip, uploader\_port)  
  
 @staticmethod  
 **def** send(sock, data):  
 *"""  
 Sending data to a client.  
 """* length = len(data)  
 length = str(length).zfill(8)  
  
 sock.send(length + data)  
  
 @staticmethod  
 **def** receive(sock):  
 *"""  
 Receiving data from a client.  
 """* length = int(sock.recv(8))  
 data = **""  
 while** length != 0:  
 data += sock.recv(1)  
 length -= 1  
  
 **return** data  
  
 @gen.coroutine  
 **def** download(self, info\_hash, piece\_index, uploader\_ip, uploader\_port):  
 *"""  
 Download the data from a client.  
 """* sock = socket.socket()  
 sock.settimeout(30)  
 sock.connect((uploader\_ip, uploader\_port))  
  
 ClientWorker.send(sock, clients\_protocol.ClientProtocol.request(info\_hash, piece\_index))  
  
 clients\_protocol.ClientProtocol.handle\_response(ClientWorker.receive(sock),  
 self.actions[info\_hash].pieces\_requested\_index, self.yftf\_files)  
  
 self.actions[info\_hash].pieces\_requested\_index[info\_hash].remove(piece\_index)  
 self.actions[info\_hash].finished\_pieces\_index.append(piece\_index)  
  
 sock.close()  
  
 @gen.coroutine  
 **def** upload(self, info\_hash, port):  
 *"""  
 Uploading data to a client.  
 """* self.actions[info\_hash].port\_range\_in\_use[port] = True  
  
 server\_socket = socket.socket()  
  
 *# temp\_sock = socket.socket()  
 #  
 # if temp\_sock.connect\_ex(("127.0.0.1", port)) == 0:  
 # print "Error: Port is in use."  
 # temp\_sock.close()  
 # return  
 #  
 # temp\_sock.close()* server\_socket.bind((**'0.0.0.0'**, port))  
  
 server\_socket.listen(1)  
  
 (client\_socket, client\_address) = server\_socket.accept()  
  
 data = clients\_protocol.ClientProtocol.handle\_request(ClientWorker.receive(client\_socket), self.yftf\_files)  
  
 ClientWorker.send(client\_socket, data)  
  
 self.actions[info\_hash].port\_range\_in\_use[port] = False  
  
 client\_socket.close()  
 server\_socket.close()  
  
 @gen.coroutine  
 **def** worker(self):  
 *"""  
 Sending request to the server.  
 """* **while** True:  
 req = **yield** self.queue.get()  
  
 **try**:  
 **yield** self.http\_client.fetch(req, self.handle\_response)  
  
 **finally**:  
 self.queue.task\_done()  
  
 @gen.coroutine  
 **def** do\_work(self):  
 *"""  
 Starting the workers.  
 """* **for** worker **in** range(self.num\_workers):  
 IOLoop.current().spawn\_callback(self.worker)  
  
 **while** True:  
 **if not** self.actions:  
 **continue  
  
 for** info\_hash, action **in** self.actions.iteritems():  
 req = action.request()  
  
 **if not** req:  
 **continue  
  
 yield** self.queue.put(req)  
  
 **yield** self.queue.join()  
  
 **def** start\_client(self):  
 *"""  
 Starting the client worker.  
 """* IOLoop.current().run\_sync(self.do\_work)  
  
 **def** add\_action(self, command, yftf\_files, info\_hash, peer\_id, peer\_ip, port\_range, num\_workers, queue\_size):  
 *"""  
 Adding an action to the client worker.  
 """* self.yftf\_files = yftf\_files  
  
 self.actions.update({info\_hash: client\_action.ClientAction(command, yftf\_files, info\_hash, peer\_id, peer\_ip,  
 port\_range, num\_workers, queue\_size)})  
  
 **def** stop\_action(self, info\_hash):  
 *"""  
 Stopping an action.  
 """* **del** self.actions[info\_hash]

**clients\_protocol.py:**

*"""  
The file of the protocol between the clients.  
"""***import** os  
**import** hashlib  
  
  
**class** ClientProtocol(object):  
 *"""  
 The class of the protocol between the clients.  
 """* @staticmethod  
 **def** handle\_request(request, yftf\_files):  
 *"""  
 Handling a request from a downloader.  
 """* info\_hash = request[0:40]  
 piece\_index = int(request[40:48])  
  
 **if** info\_hash **not in** yftf\_files.keys():  
 **return** None  
  
 yftf\_data = yftf\_files[info\_hash][0]  
 shared\_files\_dir\_path = yftf\_files[info\_hash][1]  
  
 file\_path = str()  
 file\_piece\_index = int()  
  
 pieces\_counter = 0  
 **for** shared\_file\_info **in** yftf\_data[**"Info"**][**"Files"**]:  
 pieces\_counter += len(shared\_file\_info[**"Pieces Hash"**])  
  
 **if** pieces\_counter > piece\_index:  
 file\_piece\_index = piece\_index - (pieces\_counter - len(shared\_file\_info[**"Pieces Hash"**]))  
 file\_path = shared\_file\_info[**"Path"**]  
 **break  
  
 if not** file\_path **or not** os.path.exists(shared\_files\_dir\_path + file\_path):  
 **return** None  
  
 shared\_file = open(shared\_files\_dir\_path + file\_path, **'rb'**)  
 shared\_file.read(file\_piece\_index \* yftf\_data[**"Info"**][**"Piece Length"**])  
 data = shared\_file.read(yftf\_data[**"Info"**][**"Piece Length"**])  
 shared\_file.close()  
  
 **return** info\_hash + data  
  
 @staticmethod  
 **def** request(info\_hash, piece\_index):  
 *"""  
 Request from a downloader to uploader.  
 """* **return** info\_hash + str(piece\_index).zfill(8)  
  
 @staticmethod  
 **def** handle\_response(response, requests, yftf\_files):  
 *"""  
 Handling a response from uploader.  
 """* info\_hash = response[0:40]  
  
 **if** info\_hash **not in** yftf\_files.keys() **and** info\_hash **not in** requests.keys():  
 **return** None  
  
 yftf\_data = yftf\_files[info\_hash][0]  
 shared\_files\_dir\_path = yftf\_files[info\_hash][1]  
 pieces\_index\_requested = requests[info\_hash]  
  
 data = response[40:40 + yftf\_data[**"Info"**][**"Piece Length"**]]  
 data\_hash = hashlib.sha1(data).hexdigest()  
  
 file\_path = str()  
 file\_piece\_index = int()  
 piece\_index = int()  
 **for** piece\_index **in** pieces\_index\_requested:  
 pieces\_counter = 0  
  
 **for** shared\_file\_info **in** yftf\_data[**"Info"**][**"Files"**]:  
 pieces\_counter += len(shared\_file\_info[**"Pieces Hash"**])  
  
 **if** pieces\_counter > piece\_index:  
 file\_piece\_index = piece\_index - (pieces\_counter - len(shared\_file\_info[**"Pieces Hash"**]))  
  
 **if** file\_piece\_index < 0 **or** file\_piece\_index > len(shared\_file\_info[**"Pieces Hash"**]) - 1:  
 **continue  
  
 if** str(shared\_file\_info[**"Pieces Hash"**][file\_piece\_index]) == data\_hash **or** str(  
 shared\_file\_info[**"Hash"**]) == data\_hash:  
 file\_path = str(shared\_file\_info[**"Path"**])  
 **break  
  
 if** file\_path:  
 **break  
  
 if not** file\_path:  
 **return** None  
  
 directories = file\_path.split(**'\\'**)[0:-1]  
  
 **if not** os.path.isdir(shared\_files\_dir\_path + **'\\'** + yftf\_data[**"Info"**][**"Name"**]):  
 os.makedirs(shared\_files\_dir\_path + **'\\'** + yftf\_data[**"Info"**][**"Name"**])  
  
 shared\_files\_dir\_path += **'\\'** + yftf\_data[**"Info"**][**"Name"**]  
  
 **for** directory **in** directories:  
 **if not** os.path.isdir(shared\_files\_dir\_path + **'\\'** + directory):  
 os.makedirs(shared\_files\_dir\_path + **'\\'** + directory)  
  
 shared\_file = open(shared\_files\_dir\_path + file\_path, **'wb'**)  
 shared\_file.seek(file\_piece\_index \* yftf\_data[**"Info"**][**"Piece Length"**])  
 shared\_file.write(data)  
 shared\_file.close()  
  
 **return** piece\_index

**yft\_client.py:**

*"""  
The file that manages the client.  
"""***import** socket  
**import** random  
**import** string  
**import** client\_worker  
**import** os  
**import** json  
**import** hashlib  
**import** yftf\_creator  
**import** thread  
  
  
**class** YFTClient(object):  
 *"""  
 The class the manages the client.  
 """* **def** \_\_init\_\_(self):  
 *"""  
 Starting new client.  
 """* self.peer\_id = **''**.join(random.SystemRandom().choice(string.ascii\_letters + string.digits) **for** \_ **in** range(20))  
 self.peer\_ip = YFTClient.get\_host\_ip()  
  
 self.yftf\_files = dict()  
  
 self.thread\_counter = 0  
 self.start\_port\_from = 6000  
 self.num\_port\_per\_thread = 10  
  
 self.downloads\_dir\_path = **""** self.yftf\_dir\_path = **""** self.yftf\_path = **""** self.shared\_files\_dir\_path = **""** self.tracker\_url = **""** self.command = -1  
  
 self.worker\_object = client\_worker.ClientWorker(self.yftf\_files, 1, 1)  
  
 thread.start\_new\_thread(self.worker\_object.start\_client, ())  
  
 **def** new\_action(self):  
 *"""  
 Client new action.  
 """* **if** self.command **is** -1:  
 **return  
  
 if** self.command **is** 0:  
 self.new\_download()  
 **return  
  
 if** self.command **is** 1:  
 self.new\_share()  
 **return  
  
 if** self.command **is** 2:  
 self.stop\_upload()  
 **return  
  
 def** correct\_path(self):  
 *"""  
 Correcting the path string.  
 """* self.downloads\_dir\_path = self.downloads\_dir\_path.replace(**'/'**, **'\\'**)  
 self.yftf\_dir\_path = self.yftf\_dir\_path.replace(**'/'**, **'\\'**)  
 self.yftf\_path = self.yftf\_path.replace(**'/'**, **'\\'**)  
 self.shared\_files\_dir\_path = self.shared\_files\_dir\_path.replace(**'/'**, **'\\'**)  
  
 **def** new\_download(self):  
 *"""  
 Action of new download.  
 """* self.correct\_path()  
  
 **if not** os.path.isfile(self.yftf\_path):  
 **print "Error: Your yftf file doesn't exists"  
 return** yftf\_file = open(self.yftf\_path, **'r'**)  
 yftf\_data = yftf\_file.read()  
 yftf\_json = json.loads(yftf\_data)  
 yftf\_file.close()  
  
 info\_hash = hashlib.sha1(json.dumps(yftf\_json[**"Info"**])).hexdigest()  
  
 self.yftf\_files.update({info\_hash: [yftf\_json, self.downloads\_dir\_path]})  
  
 self.worker\_object.add\_action(0, self.yftf\_files, info\_hash, self.peer\_id, self.peer\_ip,  
 range(self.start\_port\_from + (self.num\_port\_per\_thread \* self.thread\_counter),  
 self.start\_port\_from + (  
 self.num\_port\_per\_thread \* (self.thread\_counter + 1))),  
 self.num\_port\_per\_thread, self.num\_port\_per\_thread \* 10)  
  
 self.thread\_counter += 1  
  
 **def** new\_share(self):  
 *"""  
 Action of new share.  
 """* self.correct\_path()  
  
 yftf\_creator.YftfCreator(self.shared\_files\_dir\_path, self.yftf\_dir\_path, self.tracker\_url)  
  
 yftf\_file = open(  
 os.path.join(self.yftf\_dir\_path, self.shared\_files\_dir\_path.split(**'\\'**)[-1].split(**'.'**)[0] + **".yftf"**), **'r'**)  
 yftf\_data = yftf\_file.read()  
 yftf\_json = json.loads(yftf\_data)  
 yftf\_file.close()  
  
 info\_hash = hashlib.sha1(json.dumps(yftf\_json[**"Info"**])).hexdigest()  
  
 self.yftf\_files.update({info\_hash: [yftf\_json, self.shared\_files\_dir\_path]})  
  
 self.worker\_object.add\_action(1, self.yftf\_files, info\_hash, self.peer\_id, self.peer\_ip,  
 range(self.start\_port\_from + (self.num\_port\_per\_thread \* self.thread\_counter),  
 self.start\_port\_from + (  
 self.num\_port\_per\_thread \* (self.thread\_counter + 1))),  
 self.num\_port\_per\_thread, self.num\_port\_per\_thread \* 10)  
  
 self.thread\_counter += 1  
  
 **def** stop\_upload(self):  
 *"""  
 Action of stop upload/download.  
 """* self.correct\_path()  
  
 shared\_file\_name = self.yftf\_path.split(**'\\'**)[-1].split(**'.'**)[0]  
 info\_hash = **""  
  
 for** info\_hash, data **in** self.yftf\_files.iteritems():  
 **if** data[0][**"Info"**][**"Name"**] **is** shared\_file\_name:  
 **del** self.yftf\_files[info\_hash]  
 **break  
  
 if** len(info\_hash) < 0:  
 **print "ERROR: You don't upload this files"  
 return** self.worker\_object.add\_action(2, self.yftf\_files, info\_hash, self.peer\_id, self.peer\_ip,  
 range(self.start\_port\_from + (self.num\_port\_per\_thread \* self.thread\_counter),  
 self.start\_port\_from + (  
 self.num\_port\_per\_thread \* (self.thread\_counter + 1))), 1,  
 self.num\_port\_per\_thread \* 10)  
  
 self.worker\_object.stop\_action(info\_hash)  
  
 self.thread\_counter += 1  
  
 @staticmethod  
 **def** get\_host\_ip():  
 *"""  
 Gets the client ip.  
 """* sock = socket.socket()  
 sock.connect((**"google.com"**, 80))  
 ip = sock.getsockname()[0]  
 sock.close()  
  
 **return** ip

**yftf\_creator.py:**

*"""  
Containing the yftf creator class.  
"""***import** os  
**import** json  
**import** hashlib  
  
  
**class** YftfCreator(object):  
 *"""  
 Class that makes the yftf file for the file/s you want to share.  
 """* **def** \_\_init\_\_(self, path, yftf\_path, tracker\_url):  
 *"""  
 Creating the yftf file.  
 """* self.piece\_length = 262144  
 self.is\_file = False  
 self.num\_pieces = 0  
 self.tracker\_url = tracker\_url  
  
 exists = self.is\_exist(path, yftf\_path)  
  
 **if** exists[0]:  
 self.path = path  
 self.create\_yftf(yftf\_path)  
 **else**:  
 **print "ERROR: "** + exists[1]  
  
 **def** is\_exist(self, path, yftf\_path):  
 *"""  
 Checking if the file or directory exists.  
 """* **if** os.path.isfile(path):  
 self.is\_file = True  
  
 **if not** os.path.isdir(yftf\_path):  
 **return** False, **"Saving path location not exist!"  
  
 if not** os.path.isdir(path) **and not** self.is\_file:  
 **return** False, **"File or directory not exist!"  
  
 return** True, **''  
  
 def** get\_data(self):  
 *"""  
 Organizing all the data that needs for the yftf file.  
 """* yftf\_data = dict()  
 yftf\_data[**'Announce'**] = self.tracker\_url  
 yftf\_data[**'Info'**] = dict()  
  
 yftf\_data[**'Info'**][**'Name'**] = self.path.split(**'\\'**)[-1]  
  
 **if** self.is\_file:  
 yftf\_data[**'Info'**][**'Name'**] = self.path.split(**'\\'**)[-1]  
  
 yftf\_data[**'Info'**][**'Piece Length'**] = self.piece\_length  
  
 files\_data = self.get\_files\_info(self.path)  
  
 **if** len(files\_data) **is** 1:  
 yftf\_data[**'Info'**].update(files\_data[0])  
 **return** yftf\_data  
  
 yftf\_data[**'Info'**][**'Files'**] = files\_data  
  
 yftf\_data[**'Info'**][**'Num Pieces'**] = self.num\_pieces  
  
 **return** yftf\_data  
  
 **def** create\_yftf(self, path):  
 *"""  
 Creating the yftf file.  
 """* yftf\_data = self.get\_data()  
  
 **if** self.is\_file:  
 yftf = open(os.path.join(path, **''**.join(yftf\_data[**'Info'**][**'Name'**].split(**'.'**)[:-1]) + **'.yftf'**), **'w'**)  
 **else**:  
 yftf = open(os.path.join(path, yftf\_data[**'Info'**][**'Name'**] + **'.yftf'**), **'w'**)  
  
 yftf.write(json.dumps(yftf\_data))  
  
 **def** get\_files\_info(self, path):  
 *"""  
 Organizing all the files info.  
 """* files = []  
  
 **if** self.is\_file:  
 files.append(self.get\_file\_info(path))  
 **return** files  
  
 **for** new\_path **in** os.listdir(path):  
 new\_path = path + **'\\'** + new\_path  
  
 **if** os.path.isdir(new\_path):  
 dir\_files = self.get\_files\_info(new\_path)  
 files += dir\_files  
 **continue** files.append(self.get\_file\_info(new\_path))  
  
 **return** files  
  
 **def** get\_file\_info(self, path):  
 *"""  
 Organizing all the file info.  
 """* file\_info = dict()  
 file\_data = open(path, **'rb'**).read()  
  
 **if not** self.is\_file:  
 file\_info[**'Path'**] = path.replace(self.path, **''**).decode(**'iso-8859-8'**).encode(**'utf-8'**)  
  
 file\_info[**'Length'**] = os.path.getsize(path)  
 file\_info[**'Hash'**] = YftfCreator.get\_data\_hash(file\_data)  
 file\_info[**'Pieces Hash'**] = self.get\_file\_pieces\_hashes(file\_data)  
  
 **return** file\_info  
  
 **def** get\_file\_pieces\_hashes(self, data):  
 *"""  
 Dividing the file to pieces and make on each piece a hash.  
 """* pieces\_hashes = []  
  
 **for** index **in** range(0, (len(data) / self.piece\_length) + 1):  
 **if** index == (len(data) / self.piece\_length) + 1:  
 pieces\_hashes.append(YftfCreator.get\_data\_hash(data[index \* self.piece\_length:]))  
 **else**:  
 pieces\_hashes.append(  
 YftfCreator.get\_data\_hash(data[index \* self.piece\_length:(index + 1) \* self.piece\_length]))  
  
 self.num\_pieces += 1  
  
 **return** pieces\_hashes  
  
 @staticmethod  
 **def** get\_data\_hash(data):  
 *"""  
 Make a SHA-1 hash on the data.  
 """* **return** hashlib.sha1(data).hexdigest()

**שרת:**

על מנת להריץ את השרת יש להפעיל את yft\_server.py.

**client\_server\_protocol.py:**

*"""  
The file of the protocol between the client and the server.  
"""***import** hashlib  
**import** json  
**import** shared\_file\_table  
**import** os  
  
  
**class** ClientServerProtocol(object):  
 *"""  
 The class of the protocol between the client and server (server side).  
 """* **def** \_\_init\_\_(self, saved\_tables\_path):  
 *"""  
 Sets the protocol needed information.  
 """* **if** os.path.isfile(**"data\_save.obj"**):  
 data\_save\_file = open(**"data\_save.obj"**, **'rb'**)  
 saved\_data = json.loads(data\_save\_file.read())  
 data\_save\_file.close()  
  
 self.saved\_tables\_path = saved\_data[**"saved\_tables\_path"**]  
 yftf\_files\_info\_hash = saved\_data[**"yftf\_files\_info\_hash"**]  
  
 self.yftf\_files = dict()  
 **for** info\_hash **in** yftf\_files\_info\_hash:  
 self.yftf\_files.update(  
 {info\_hash: shared\_file\_table.SharedFileTable(info\_hash, self.saved\_tables\_path)})  
  
 **return** self.saved\_tables\_path = saved\_tables\_path  
 self.yftf\_files = dict()  
  
 **def** handle\_request(self, request\_headers, request\_body=**""**):  
 *"""  
 Handle request.  
 """* **if not** request\_headers:  
 **return** {**"YFT-Error"**: **"You probably miss some headers"**}  
  
 **if** {**"Yft-Peer-Id"**, **"Yft-Peer-Status"**, **"Yft-Upload-Piece"**, **"Yft-Yftf-Hash"**, **"Yft-Port"**, **"Yft-Peer-Ip"**}.issubset(  
 set(request\_headers.keys())):  
 **return** self.handle\_new\_share(request\_headers, request\_body)  
  
 **if** {**"Yft-Info-Hash"**, **"Yft-Peer-Id"**, **"Yft-Peer-Ip"**, **"Yft-Peer-Status"**}.issubset(set(request\_headers.keys())):  
 **if** request\_headers[**"Yft-Info-Hash"**] **not in** self.yftf\_files:  
 **return** {**"YFT-Error"**: **"This file is not shared"**}  
  
 **if** request\_headers[**"Yft-Peer-Status"**] **is** str(2):  
 self.remove\_peer(request\_headers)  
 **return  
  
 if** request\_headers[**"Yft-Peer-Status"**] **is** str(0):  
 self.add\_peer(request\_headers)  
  
 **if "Yft-Finished-Piece-Index" in** request\_headers.keys():  
 self.handle\_finished\_piece(request\_headers)  
  
 **if "Yft-Request-Piece-Index" in** request\_headers.keys():  
 **return** self.handle\_downloader\_request(request\_headers)  
  
 **if** {**"Yft-Port"**, **"Yft-Upload-Piece"**}.issubset(set(request\_headers.keys())) **and** request\_headers[  
 **"Yft-Upload-Piece"**] **is** str(1):  
 **return** self.handle\_uploader\_request(request\_headers)  
  
 **return** {**"YFT-Error"**: **"You probably miss some headers"**}  
  
 **def** remove\_peer(self, request\_headers):  
 *"""  
 Removes peer from the right table.  
 """* self.yftf\_files[request\_headers[**"Yft-Info-Hash"**]].remove\_peer(request\_headers[**"Yft-Peer-Id"**])  
  
 **def** add\_peer(self, request\_headers):  
 *"""  
 Adding peer to the right table.  
 """* self.yftf\_files[request\_headers[**"Yft-Info-Hash"**]].add\_peer(request\_headers[**"Yft-Peer-Id"**],  
 request\_headers[**"Yft-Peer-Ip"**])  
  
 **def** handle\_finished\_piece(self, request\_headers):  
 *"""  
 Handle finished piece from peer.  
 """* self.yftf\_files[request\_headers[**"Yft-Info-Hash"**]].add\_piece(request\_headers[**"Yft-Peer-Id"**], map(int, str(  
 request\_headers[**"Yft-Finished-Piece-Index"**]).replace(**"["**, **""**).replace(**']'**, **''**).split(**', '**)))  
  
 **def** handle\_downloader\_request(self, request\_headers):  
 *"""  
 Handling a download request from peer.  
 """* table = self.yftf\_files[request\_headers[**"Yft-Info-Hash"**]]  
  
 uploader\_data = table.find\_uploader(int(request\_headers[**"Yft-Request-Piece-Index"**]))  
  
 **if not** uploader\_data:  
 **return** {**"YFT-Info-Hash"**: table.get\_info\_hash(), **"YFT-Error"**: **"Could not find an uploader"**}  
  
 **return** {**"YFT-Info-Hash"**: table.get\_info\_hash(), **"YFT-Type"**: str(0), **"YFT-ip"**: uploader\_data[0],  
 **"YFT-Piece-Index"**: request\_headers[**"Yft-Request-Piece-Index"**], **"YFT-Port"**: str(uploader\_data[1])}  
  
 **def** handle\_uploader\_request(self, request\_headers):  
 *"""  
 Handling upload request from peer.  
 """* table = self.yftf\_files[request\_headers[**"Yft-Info-Hash"**]]  
  
 table.set\_peer\_waiting(request\_headers[**"Yft-Peer-Id"**], int(request\_headers[**"Yft-Port"**]))  
  
 **return** {**"YFT-Info-Hash"**: table.get\_info\_hash(), **"YFT-Type"**: str(1),  
 **"YFT-Port"**: str(request\_headers[**"Yft-Port"**])}  
  
 **def** handle\_new\_share(self, request\_headers, request\_body):  
 *"""  
 Handle new share from peer.  
 """* **if not** request\_body:  
 **return** {**"YFT-Error"**: **"There is no yftf file in body"**}  
  
 **if** hashlib.sha1(request\_body).hexdigest() != request\_headers[**"Yft-Yftf-Hash"**]:  
 **return** {**"Yft-Error"**: **"yftf file corrupted"**}  
  
 yftf\_json = json.loads(request\_body)  
 info\_hash = hashlib.sha1(json.dumps(yftf\_json[**"Info"**])).hexdigest()  
  
 **if** info\_hash **not in** self.yftf\_files.keys():  
 table = shared\_file\_table.SharedFileTable(info\_hash, self.saved\_tables\_path, request\_body)  
 **else**:  
 **return** {**"YFT-Info-Hash"**: info\_hash, **"YFT-Error"**: **"File is already shared"**}  
  
 peer\_id = request\_headers[**"Yft-Peer-Id"**]  
  
 **if** request\_headers[**"Yft-Peer-Status"**] **is** str(0):  
 table.add\_peer(peer\_id, request\_headers[**"Yft-Peer-Ip"**])  
 **else**:  
 **return** {**"YFT-Info-Hash"**: info\_hash, **"YFT-Error"**: **"Your status must be 0"**}  
  
 **if** request\_headers[**"Yft-Upload-Piece"**] **is** str(1):  
 table.set\_peer\_waiting(peer\_id, str(request\_headers[**"Yft-Port"**]))  
 **else**:  
 **return** {**"YFT-Info-Hash"**: info\_hash, **"YFT-Error"**: **"Your must share"**}  
  
 table.add\_piece(peer\_id, range(0, table.get\_num\_pieces()))  
  
 self.yftf\_files.update({info\_hash: table})  
  
 **return** {**"YFT-Info-Hash"**: info\_hash, **"YFT-Type"**: str(1), **"YFT-Port"**: str(request\_headers[**"Yft-Port"**])}  
  
 **def** \_\_del\_\_(self):  
 *"""  
 When delete the protocol, it saves its information.  
 """* data\_save\_file = open(**"data\_save.obj"**, **'wb'**)  
  
 data\_save\_file.write(  
 json.dumps({**"saved\_tables\_path"**: self.saved\_tables\_path, **"yftf\_files\_info\_hash"**: self.yftf\_files.keys()}))  
 data\_save\_file.close()

**peer.py:**

*"""  
The file of the peer class.  
"""***class** Peer(object):  
 *"""  
 The class of the peer that contains all his data.  
 """* **def** \_\_init\_\_(self, peer\_ip):  
 *"""  
 New peer constructor.  
 """* self.peer\_ip = peer\_ip  
 self.state = 0  
 self.pieces = list()  
 self.port = None  
  
 **def** add\_piece(self, index):  
 *"""  
 Adding a finished piece.  
 """* **if** isinstance(index, list):  
 self.pieces += index  
 **return** self.pieces.append(index)  
  
 **def** set\_waiting\_state(self, port):  
 *"""  
 Sets his state to waiting.  
 """* self.state = 1  
 self.port = port  
  
 **def** set\_working\_state(self):  
 *"""  
 Sets his state to working.  
 """* self.state = 0  
 self.port = None  
  
 **def** get\_peer\_ip(self):  
 *"""  
 Get peer ip.  
 """* **return** self.peer\_ip  
  
 **def** get\_state(self):  
 *"""  
 Get peer state.  
 """* **return** self.state  
  
 **def** get\_pieces(self):  
 *"""  
 Get peer pieces.  
 """* **return** self.pieces  
  
 **def** get\_port(self):  
 *"""  
 Get peer listening port.  
 """* **return** self.port  
  
 **def** \_\_str\_\_(self):  
 *"""  
 String of peer data.  
 """* **return ', '**.join([self.peer\_ip, str(self.state), str(self.pieces), str(self.port)])

**shared\_file\_table.py:**

*"""  
The file of the table class.  
"""***import** json  
**import** os  
**import** peer  
  
  
**class** SharedFileTable(object):  
 *"""  
 The table class which contains the data of the shared file (peers info and file info).  
 """* **def** \_\_init\_\_(self, info\_hash, saved\_tables\_path, yftf\_data=**""**):  
 *"""  
 Starting a new table.  
 """* **if** os.path.isfile(os.path.join(saved\_tables\_path, info\_hash + **".obj"**)):  
 table\_data = open(os.path.join(saved\_tables\_path, info\_hash + **".obj"**), **'rb'**)  
 self.\_\_dict\_\_ = json.loads(table\_data.read())  
 table\_data.close()  
  
 **for** peer\_id, peer\_data **in** self.peers.iteritems():  
 peer\_obj = peer.Peer(**""**)  
 peer\_obj.\_\_dict\_\_ = peer\_data  
 self.peers[peer\_id] = peer\_obj  
  
 **return** self.saved\_tables\_path = saved\_tables\_path  
 self.info\_hash = info\_hash  
 self.yftf\_data = yftf\_data  
  
 self.yftf\_json = json.loads(self.yftf\_data)  
  
 self.num\_pieces = self.yftf\_json[**"Info"**][**"Num Pieces"**]  
 self.peers = dict()  
  
 **def** get\_info\_hash(self):  
 *"""  
 Get the info hash of the shared file of this table.  
 """* **return** self.info\_hash  
  
 **def** get\_num\_pieces(self):  
 *"""  
 Get number of pieces of the shared file.  
 """* **return** self.num\_pieces  
  
 **def** add\_peer(self, peer\_id, peer\_ip):  
 *"""  
 Adding a peer to the table.  
 """* self.peers.update({peer\_id: peer.Peer(peer\_ip)})  
  
 **def** set\_peer\_waiting(self, peer\_id, port):  
 *"""  
 Sets the peer state to waiting.  
 """* self.peers[peer\_id].set\_waiting\_state(port)  
  
 **def** set\_peer\_working(self, peer\_id):  
 *"""  
 Sets the peer state to working.  
 """* self.peers[peer\_id].set\_working\_state()  
  
 **def** add\_piece(self, peer\_id, piece):  
 *"""  
 Adding a piece to the peer.  
 """* self.peers[peer\_id].add\_piece(piece)  
  
 **def** find\_uploader(self, piece\_index):  
 *"""  
 Finding uploader.  
 """* **for** peer\_id, peer\_obj **in** self.peers.iteritems():  
 **if** peer\_obj.get\_state():  
 **if** piece\_index **in** peer\_obj.get\_pieces():  
 port = peer\_obj.get\_port()  
 peer\_obj.set\_working\_state()  
  
 **return** peer\_obj.get\_peer\_ip(), port  
  
 **return** None  
  
 **def** remove\_peer(self, peer\_id):  
 *"""  
 Remove a peer from the table.  
 """* **del** self.peers[peer\_id]  
  
 **def** \_\_str\_\_(self):  
 *"""  
 String of data about the table.  
 """* **return ', '**.join([self.yftf\_data, self.info\_hash, str(self.num\_pieces), str(self.peers)])  
  
 **def** \_\_del\_\_(self):  
 *"""  
 Saves the table when deleted.  
 """* table\_data = open(os.path.join(self.saved\_tables\_path, self.info\_hash + **".obj"**), **'wb'**)  
 **for** peer\_id **in** self.peers.keys():  
 self.peers[peer\_id] = self.peers[peer\_id].\_\_dict\_\_  
  
 table\_data.write(json.dumps(self.\_\_dict\_\_))  
 table\_data.close()

**yft\_server.py:**

*"""  
The file that runs the server.  
"""***from** tornado.httpserver **import** HTTPServer  
**from** tornado.ioloop **import** IOLoop  
**import** client\_server\_protocol  
  
  
**class** YFTServer(object):  
 *"""  
 The server class.  
 """* **def** \_\_init\_\_(self, listen\_port, http\_version, saved\_tables\_path):  
 *"""  
 Preparing the server.  
 """* self.protocol = client\_server\_protocol.ClientServerProtocol(saved\_tables\_path)  
  
 self.http\_version = http\_version  
 self.http\_server = HTTPServer(self.handle\_request)  
  
 self.http\_server.listen(listen\_port)  
  
 **def** handle\_request(self, request):  
 *"""  
 Handling the request.  
 """* headers = self.protocol.handle\_request(request.headers, request.body)  
  
 response = self.write\_response(headers)  
  
 request.write(response)  
 request.finish()  
  
 **def** write\_response(self, headers, body=**""**):  
 *"""  
 Writing the response.  
 """* response = self.http\_version + **" 200 OK\r\n"  
  
 for** header **in** headers:  
 response += header + **": "** + str(headers[header]) + **"\r\n"  
  
 if "Content-Length" not in** headers.keys() **and** body **is not ""**:  
 response += **"Content-Length: "** + str(len(body)) + **"\r\n"** response += **"\r\n"** + body  
  
 **return** response  
  
 @staticmethod  
 **def** start\_server():  
 *"""  
 Starting the server.  
 """* IOLoop.instance().start()  
  
  
**def** main():  
 *"""  
 The main function that start the server.  
 """* listen\_port = raw\_input(**"What is the port you want to listen on?"**)  
 http\_version = **"HTTP/1.1"** saved\_tables\_path = raw\_input(**"What is the path you want to save the tables to?"**)  
  
 server = YFTServer(listen\_port, http\_version, saved\_tables\_path)  
  
 server.start\_server()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

**ביבליוגרפיה:**

<http://www.kristenwidman.com/blog/33/how-to-write-a-bittorrent-client-part-1/>

<http://bittorrent.org/beps/bep_0003.html>

<https://wiki.theory.org/BitTorrentSpecification>

<https://en.wikipedia.org/wiki/BitTorrent>

<https://en.wikipedia.org/wiki/BitTorrent_index>

<http://www.howtogeek.com/howto/31846/bittorrent-for-beginners-how-get-started-downloading-torrents/>

<http://www.ccs.neu.edu/home/amislove/teaching/cs4700/fall09/lectures/lecture20.pdf>

<https://he.wikipedia.org/wiki/%D7%91%D7%99%D7%98%D7%95%D7%A8%D7%A0%D7%98>

<http://www.kaspersky.com/au/images/camilo_andr%D1%83s_gonzalez_toro.pdf>

<https://en.wikipedia.org/wiki/Torrent_file>