האוניברסיטה הפתוחה
החטיבה למדעי המחשב
תואר ראשון

סדנה בתקשורת מחשבים

20588

תיק פרויקט

סנברון מוזיקה

שירה עשהאל, 314676503
איתי פאינשטיין, 205356587
מנחה: דני כלפון

דצמבר 2019

שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

תוכן עניינים

שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

1. מבוא

Syncalong הינה תוכנה המאפשרת לנגן מוזיקה בצורה מסונכרנת בין מספר מכשירים. התוכנה מאפשרת לשתף קבצי מוזיקה בין מספר תחנות, ולהפעיל קבצי מוזיקה בו-זמנית בכל התחנות. באופן זה, מאפשרת syncalong להפוך מספר מכשירים המחוברים רשתית זה לזה למערכת סאונד שלמה.

המערכת מורכבת משרת יחיד, אליו ניתן לחבר מספר תחנות לקוח. ממשק המערכת בשרת מאפשר לנהל את החיבורים לשרת ואת רשימת השירים להשמעה, וכן לשלוט בנגינת המוזיקה בתחנות המרוחקות. ממשק המערכת בצד הלקוח מאפשר להתחבר לשירות או להתנתק ממנו, ומראה את המוזיקה המנוגנת כעת.

הפעולות הנתמכות על ידי המערכת הינן:

- שליחת קבצי מוזיקה לתחנות במחוברות לשרת, במידה והם אינם קיימים בתחנת הקצה
- ניגון, השהייה ועצירה של קובץ מוזיקה
- בניית רשימת השמעה והפעלתה באופן רציף בכל התחנות

2. דרישות קדם

המערכת מותאמת לעבודה על כל מערכות ההפעלה, ונבדקה על מחשבי Windows 10 ו- Ubuntu 12.04. יש לוודא כי על תחנות הקצה מותקן כרטיס קול וישנם רמקולים מחוברים.

על מנת להריץ את התוכנה יש להתקין python בגרסה 3.7.4 ומעלה.

3. הוראות התקנה

על מנת להתקין את המערכת, יש להיכנס לתיקייה הראשית (syncalong), ומשם להריץ את הפקודה: python setup.py install. הפקודה תתקין על המחשב הנוכחי הן את תוכנת השרת והן את תוכנת הלקוח, יחד עם כל התלויות הנלוות למערכת.

הגדרות המערכת ניתנות לשינוי ישירות בממשק התוכנה.

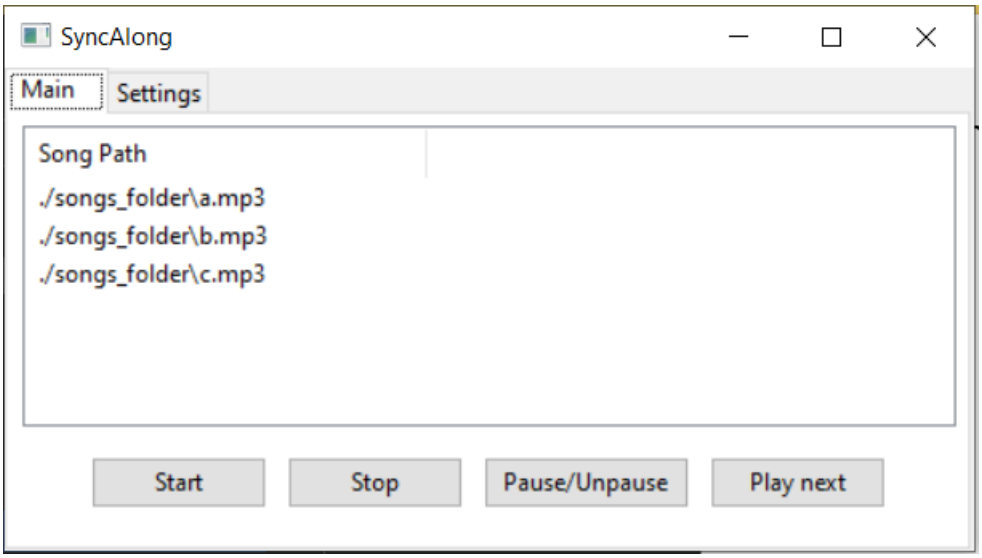
שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

4. הוראות שימוש

בחלק זה נדבר על הממשק הגרפי של התוכנה שנכתב בעזרת wx-python.

4.1. צד שרת

זהו החלון שנפתח כשמריצים את server.pyw (אפשר להריץ גם את server_main.py ואז לראות גם בחלון cmd את הdebug prints של התוכנה)



Main:

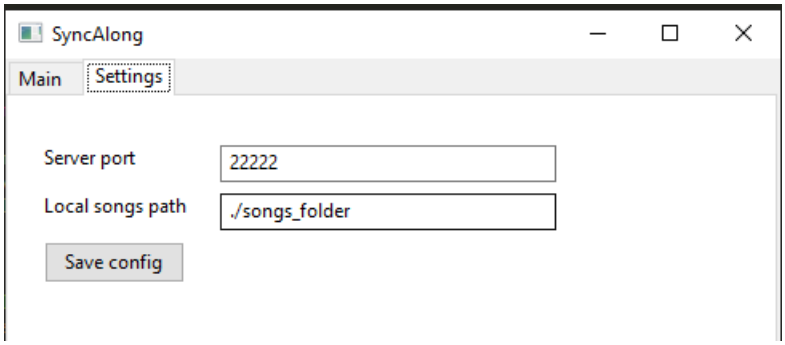
חלון עם כל השירים שעתידים להישלח למשתמשים, נתאר בהמשך פעולות שאפשר לבצע על רשימה זו.

Start – מריץ את השרת ואת שרת הntp. עד שלא לוחצים על start השרת לא מחכה לחיבורים.

Stop – שולח הודעת עצירה לכל המשתמשים המחוברים ומכבה את השרתים.

Pause/Unpause – כיוצא מהשם, שולח הודעת הפסקה/המשך ניגון לכל המשתמשים.

Play next – ממשיך לשיר הבא.



:Settings

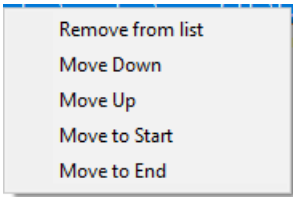
Server port – אפשר להגדיר על איזה פורט יאזין השרת, דיפולטית הוא 22222.

את הפורט של שרת הntp לא איפשרנו להגדיר והוא קבוע ל123.

Local songs path – השרת תומך בקריאת כל השירים מתיקייה קבוע מראש ככה שלא צריך בכל תחילת ריצה להכניס אליו את כל השירים, אפשר רק לדאוג שהם יהיו בתיקייה הרלוונטית, וגם היא קונפיגורבילית. דיפולטית היא מוגדרת לתיקייה songs_folder שנמצאת בתוך syncalong.

Save Config – שומר את ההגדרות הנתונות לקובץ הקונפיגורציה כדי שהם ישמרו בין שימושים שונים בתוכנה.

פעולות על רשימת השירים:



Remove from list – מסיר את השיר מהרשימה.

Move down – מוריד את השיר מקום אחד ברשימה.

Move up – מעלה את השיר מקום אחד ברשימה.

Move to start – מעביר את השיר להיות ראשון ברשימה.

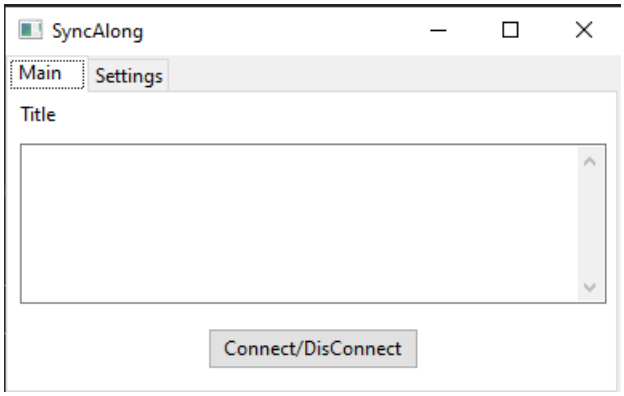
Move to end – מעביר את השיר להיות האחרון ברשימה.

בנוסף לכל החלונות הקיימים בשרת הוא גם תומך בגרירה של שירים לתוך הרשימה, פעולה שמוכרת כ-drag&drop.

4.2. צד לקוח

זהו החלון שנפתח כשמריצים את ב-client.pyw (אפשר להריץ גם את client_main.py ואז לראות גם בחלון cmd את הdebug prints של התוכנה)

שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

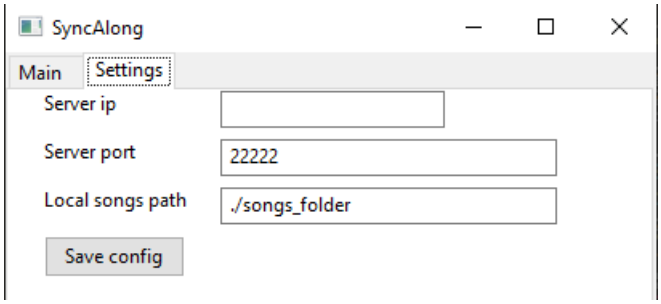


:Main

תצוגה של השיר אשר מתנגן כעט.

Connect/DisConnnect – מתחבר/מתנתק מהשרת, תלוי מצב – מחובר/מנותק.

:Settings



Server ip – ה-ip של השרת, אין הגדרה דיפולטית.

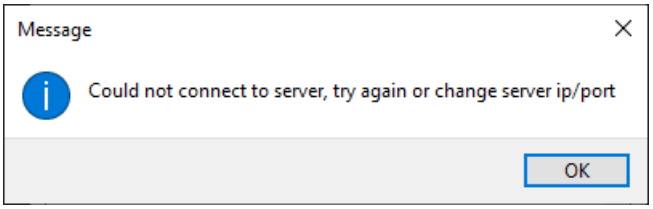
Server port – הפורט של השרת, דיפולטית 22222.

Local songs path – התיקייה בה ישמרו השירים לניגון, דיפולטית ב-songs_folder בתיקייה syncalong.

Save Config – שומר את ההגדרות הנתונות לקובץ הקונפיגורציה כדי שהם ישמרו בין שימושים שונים בתוכנה.

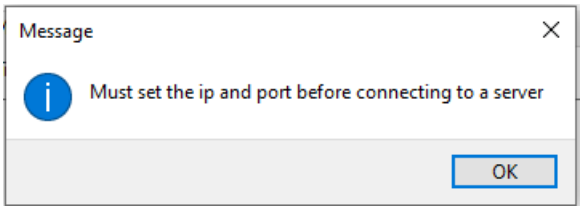
חלונות נוספים:

במצב בו לא הצלחנו להתחבר לשרת, יקפוץ החלון הבא:



שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

במצב בו לא הוגדרו כל הנתונים כדי להתחבר לשרת יקפוץ החלון הבא:



5. ארכיטקטורת תוכנה

5.1. סקירה

התוכנה בנויה מצד שרת, וצד לקוח. באחריות השרת לקבל חיבורים חדשים, לנהל את רשימת השירים להשמעה ולסנכרן את הפעולות השונות מול הלקוחות. בצד הלקוח ניתן לקבל קבצים מהשרת, וכן להשמיע מוזיקה על פי הוראות השרת. לצורך התקשורת בין השרת ללקוח, המערכת עושה שימוש במספר טכנולוגיות:

1. התקשורת בין השרת ללקוחות עבור שליחת פקודות וסנכרון קבצים נעשית מעל פרוטוקול TCP. לצורך העברת המסרים יצרנו פרוטוקול משלנו, עליו יפורט בהמשך.
2. לבניית ופרסור פקטות השתמשנו בספרייה scapy, המאפשרת להגדיר פרוטוקול ייחודי להגדרת המתכנת. הספרייה מאפשרת לייצר אובייקטים לפי מבנה ההודעה הרצוי, לבצע dispatch למידע הגולמי המתקבל מה-socket ולהכניסו לתוך אותם אובייקטים וכן לבצע סריאליזציה של האובייטקים הללו לבתים, כך שיוכלו להישלח מעל ל-socket.
3. הסנכרון בין השרת ללקוחות, שבאמצעותו כלל התחנות מנגנות את המוזיקה באותו רגע בדיוק, נעשה באמצעות פרוטוקול NTP. השרת המרכזי עצמו משמש גם כשרת NTP, כך שכלל התחנות מסונכרנות לפיו. בהמשך הפרק נציג הסבר נוסף על הפרוטוקול ואופן השימוש בו בתוכנה.
4. קבלת חיבורים חדשים בצד השרת נעשית באופן א-סינכרוני, כך שניתן לבצע פעולות למול לקוחות קיימים גם בזמן טיפול בלקוחות חדשים. דבר זה נעשה באמצעות שימוש ב-thread ייעודי לקבלת לקוחות בצד השרת.

צד השרת מכיל שני תהליכים מקבילים: שרת מוזיקה, ושרת NTP. הודעות הנשלחות משרת המוזיקה נשלחות ב-TCP, ואילו ההודעת שנשלחות לשרת ה-NTP נשלחות ב-UDP על פי התקן.

השימוש ב-TCP לפרוטוקול הייעודי של המערכת עבור העברת האותות וסנכרון הקבצים, מאפשר לשמור על אמינות גבוהה הנדרשת בתהליכים אלו. יחד עם זאת, משמעות הבחירה בפרוטוקול זה היא שהתקשורת בין השרת ללקוחות אורכת זמן רב, יותר מאשר לו היינו בוחרים להשתמש ב-UDP. בחרנו להעדיף אמינות על פני מהירות מכמה סיבות:

5. על העברת הקבצים מהשרת ללקוחות להיות אמינה. התוכנה בנויה לתמיכה בנגינת כל הפורמטים המוכרים לקבצי מוזיקה (כרגע תומכת רק בmp3), והרבה מפורמטים אלו מסתמכים על שלמות המידע שבקובץ. אילו חלק מהקובץ יועבר באופן פגום או לא שלם, לא יהיה ניתן לנגן את הקובץ בצורה תקינה אצל הלקוח.
6. קובץ שהועבר באופן חלקי עלול ליצור פערים בסנכרון שבין הלקוחות. אילו הקובץ נקטע לפני הסוף, למשל, הלקוח יחווה שקט ולא יועבר ישירות לשיר הבא.
7. עלינו לוודא כי כל הלקוחות מקבלים את הפקודות מהשרת. מכיוון שהלקוחות מסתנכרנים מול שרת ה-NTP על מנת לבצע את הפקודה ברגע הנכון, עלינו לוודא רק שכלל הלקוחות אכן קיבלו

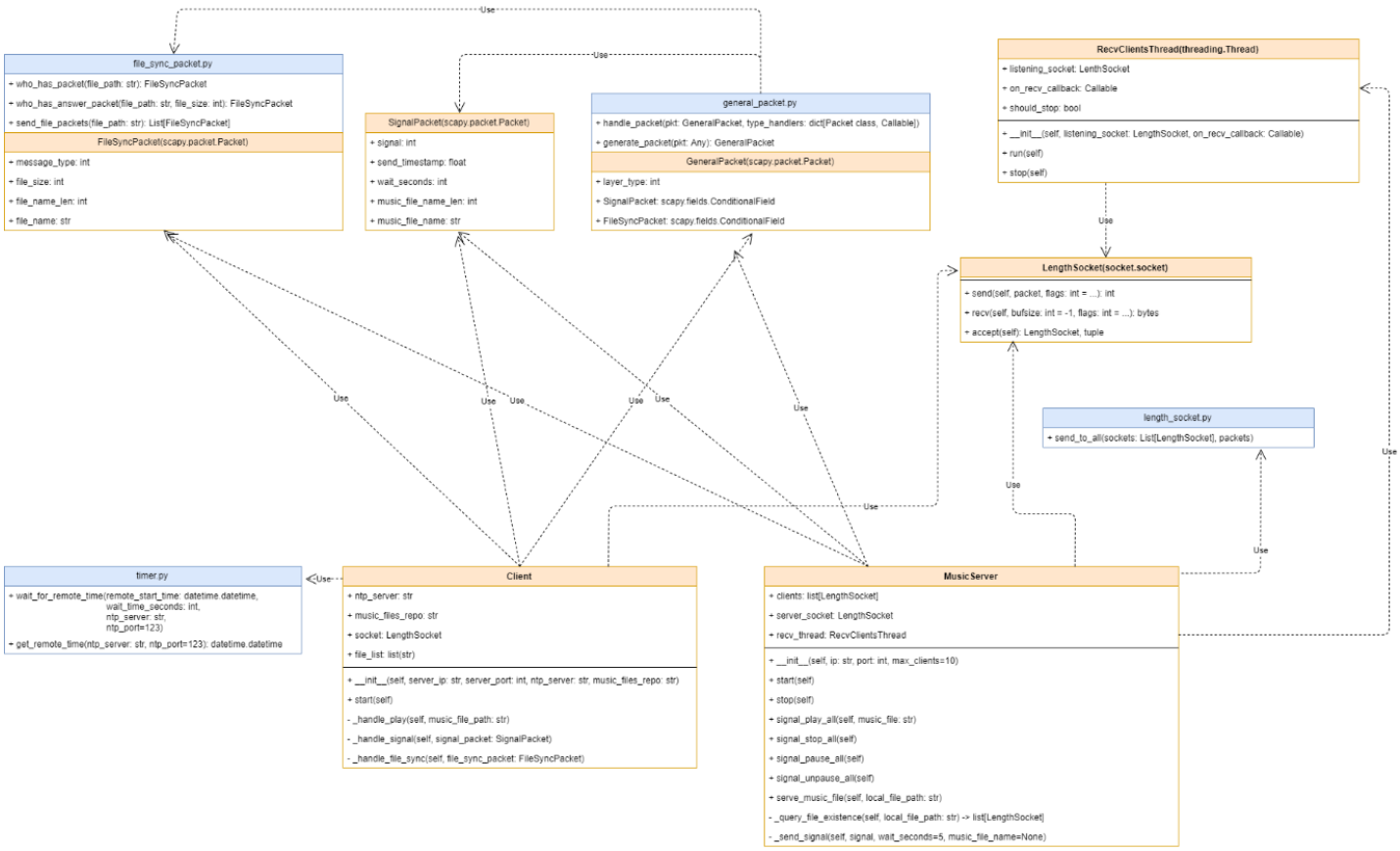
שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

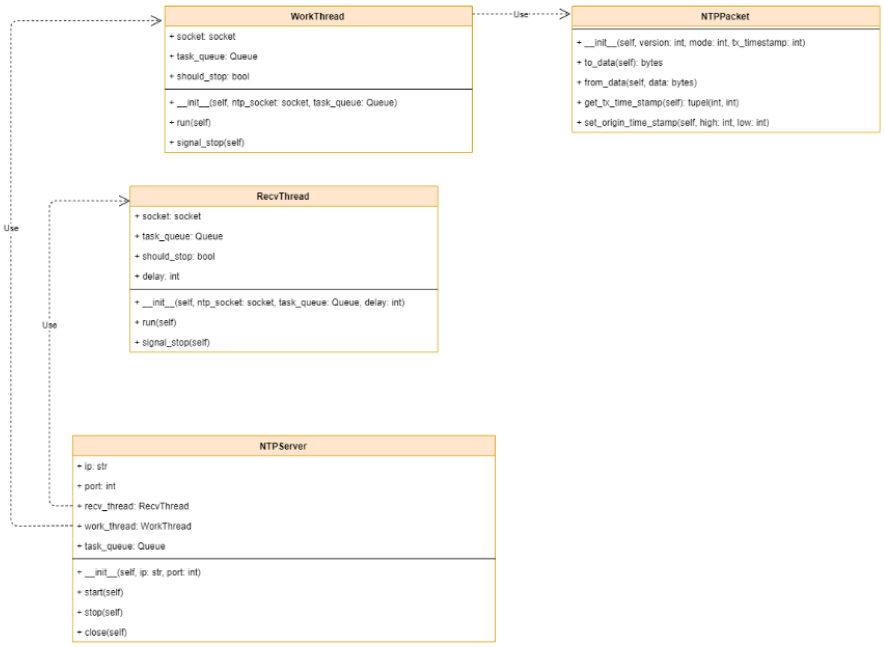
את הפקודה. זמן ההגעה של האות מהשרת נלקח בחשבון בצד הלקוח בעת חישוב הזמן המתאים לביצוע הפקודה (ראו פירוט על אופן הסנכרון בהמשך).

שירה עשהאל, 314676503
איתי פאינשטיין, 205356587

5.2. מבנה מחלקות

להלן תרשים של מבנה המחלקות בתוכנית והיחסים ביניהן (לצורך פשטות, הושמטו בשרטוט מחלקות הממשות את ממשק המשתמש):





5.3. תיאור המחלקות

5.3.1. החבילה Common

5.3.1.1. המחלקה LengthSocket(socket.socket) והמודול length_socket.py

המחלקה LengthSocket הינה האובייקט הבסיסי העוטף socket במערכת, לכתיבה ולקריאה הן בצד השרת והן בצד הלקוח. המחלקה יורשת מהאובייקט socket הפייתוני, ומשמשת לשליחה נוחה של הפרוטוקול שבשימוש המערכת. כל הודעה שנשלחת באמצעות LengthSocket נעטפת ב-header המתאים (ראו GeneralPacket), ולפניו נשלח גודל ההודעה; בכל קבלה המתבצעת על ידי LengthSocket, נקרא מה-socket גודל ההודעה ולפיו, ההודעה עצמה.

המחלקה חושפת שלוש פונקציות: send, recv ו-accept. פונקציות אלו דורסות את המימוש של socket הפייתוני, כך שיפעלו כמתואר לעיל (כמובן שפנימית, נעשה שימוש בפונקציות המקוריות על מנת לבצע פעולות למול ה-socket עצמו).

בנוסף, במודול בו מתוארת המחלקה (length_socket.py) קיימת פונקציית עזר חופשית בשם "send_to_all", המקבלת רשימה של אובייקטים מסוג LengthSocket ורשימה של הודעות לשליחה. הפונקציה שולחת את כל ההודעות לכל ה-socket-ים, ומדווחת על שגיאות במידה וקרו.

5.3.1.2. המחלקה FileSyncPacket(scapy.packet.Packet) והמודול file_sync_packet.py

המחלקה FileSyncPacket יורשת מהמחלקה Packet הכללית של scapy, ומשמשת להעברת הודעות לסנכרון קבצי מוזיקה מהשרת ללקוחות. המחלקה מגדירה את מבנה ההודעה הבסיסי לסוג הודעות זה (ראו בהמשך).

במודול file_sync_packet.py מיוצאות מספר פונקציות המשמשות לבנייה נוחה של הודעות מסוג FileSyncPacket:

1. הפונקציות who_has_packet ו-who_has_answer_packet – פונקציות לבניית הודעות שאלה / תשובה מהשרת / מהלקוח (בהתאמה), לבדיקה אצל מי מהלקוחות חסר קובץ מסוים ויש צורך בשליחתו מהשרת.
2. הפונקציה send_file_packets, הבונה הודעת FileSyncPacket המכילה תוכל של קובץ לסנכרון, תוך קריאת הקובץ מהדיסק.

5.3.1.3. המחלקה SignalPacket(scapy.packet.Packet)

מחלקה זו, הנמצאת במודול signal_packet.py, יורשת אף היא ממחקלת Packet ב-scapy, ותפקידה להגדיר את מבנה ההודעות לשליחת איתותים בין השרת ללקוחות. ארבעת סוגי האיתותים האפשריים הם: play, stop, pause, unpause, ומתוארים בפירוט בהמשך העבודה, עם תיאור הפרוטוקול. לכמה מן האיתותים נדרשים פרמטרים נוספים. פרמטרים אלו מוגדרים גם הם במחלקה, וכן ערכיהם הדיפולטיים.

5.3.1.4. המחלקה GeneralPacket(scapy.packet.Packet) והמודול general_packet.py

המחלקה GeneralPacket הינה מחלקת מעטפת עבור שתי מחלקות ההודעות המפורטות בסעיפים הקודמים. כל הודעה הנשלחת מה-socket נעטפת על ידי אובייקט זה. באמצעות הפרמטרים המוגדרים ב-GeneralPacket ניתן לחלץ את המידע הרצוי מתוך ההודעה, על פי סוגה (במקרה שלנו קיימים שני סוגי הודעות: FileSyncPacket, SignalPacket, אולם המחלקה כתובה כך שניתן להרחיבה לסוגי הודעות נוספים). חילוץ זה נעשה באופן אוטומטי על ידי scapy.

הפונקציה generate_packet במודול משמשת לבנייה נוחה של הודעה מסוג GeneralPacket העוטפת הודעה מסוג אחר.

הפונקציה handle_packet משמשת לעבודה נוחה מול הודעות המתקבלות ב-socket מסוים. הפונקציה מקבלת אובייקט מסוג GeneralPacket, ומילון הממפה בין סוג הודעה לבין פונקציה המטפלת בהודעה זו. הפונקציה דואגת להרצת לוגיקת הטיפול המתאימה לסוג ההודעה העטופה ב-GeneralPacket. דוגמת שימוש בפונקציה קיימת במחלה Client, בפונקציה start.

5.3.2. החבילה Server

5.3.2.1. המחלקה RecvClientsThread(threading.Thread)

המחלקה RecvClientsThread יורשת ממחלקת Thread הפייתונית. במחלקה זו ממומש thread ההאזנה ללקוחות בו משתמש שרת המוזיקה (MusicServer). המחלקה מקבלת שני פרמטרים: socket במצב האזנה, ופונקציית callback. כאשר ה-thread מאותחל, הוא קורא ל-accept על ה-socket שקיבל, ועם כל חיבור שנקלט קורא לפונקציית ה-callback. ה-thread ניתן לעצירה באמצעות שליחת Event פנימי.

5.3.2.2. המחלקה MusicServer

המחלקה הראשית של שרת המוזיקה. במחלקה זו מתוחזק מערך הלקוחות המחוברים למערכת, ובאמצעותה ניתן לשלוח ללקוחות השונים קבצים והוראות. המחלקה מחזיקה מופע יחיד של RecvClientsThread, וכל לקוח שמתקבל על ידי thread זה מוכנס לרשימת החיבורים הקיימים של השרת.

המחלקה תומכת בפעולות הבאות:

1. start – להתחלת האזנה ללקוחות.
2. close – סגירת ה-socket של השרת והפסקת האזנה ללקוחות.
3. signal_play_all, signal_stop_all, signal_pause_all, signal_unpause_all – פונקציות לשליחת פקודות play, stop, pause ו-unpause לכל הלקוחות המחוברים, בהתאמה.
4. server_music_file – שליחת קובץ מוזיקה בודד לכלל הלקוחות שקובץ זה אינו ברשותם.

5.3.2.3. המודול ntp_server.py

במודול זה ממומשת המחלקה NTPServer – מחלקה המשמשת שרת NTP, על פי התקן (ראו פירוט בהמשך). המחלקה תומכת בקבלת בקשות מלקוחות בצורה א-סינכרונית, באמצעות שני thread-ים: RecvThread – לקבלת שאילתות; ו-WorkThread – לחישוב התשובה ושליחתה.

כמו-כן, במודול מוגדרת המחלקה NTPPacket הממשת את מסגרת הפרוטוקול.

מחלקה זו נכתבה בהשראת קוד פתוח הנמצא בקישור הבא:

https://github.com/limifly/ntpserver/blob/master/ntpserver.py

5.3.3. החבילה Client

5.3.3.1. המודול timer.py

מודול זה מכיל שתי פונקציות עזר למימוש סנכרון הזמנים בין הלקוח לשרת:

1. get_remote_time – פונקציה פנימית העוטפת את התקשורת מול שרת ה-NTP, ומחזירה את התשובה כאובייט datetime פייתוני.
2. wait_for_remote_time – פונקציה להמתנת זמן באופן מכוייל למול שרת NTP. הפונקציה מקבלת זמן התחלה (נסמנו כאן – t), זמן המתנה (נסמן – x) ופרמטרים לחיבור לשרת NTP, מתשאלת את השרת לזמן הנוכחי ($t + y$), וממתינה בדיוק $x - y$ שניות.

5.3.3.2. המחלקה Client

מחלקת ה-Client ממשת את לוגיקת צד הלקוח. המחלקה חושפת פונקציה מרכזית אחת: start. פונקציה זו יוצרת חיבור אל השרת, ומאזינה להודעות ממנו. המחלקה ממשת לוגיקות פנימיות על מנת להתמודד עם כל סוג הודעה שנשלחת מהשרת.

5.3.4. החבילה gui

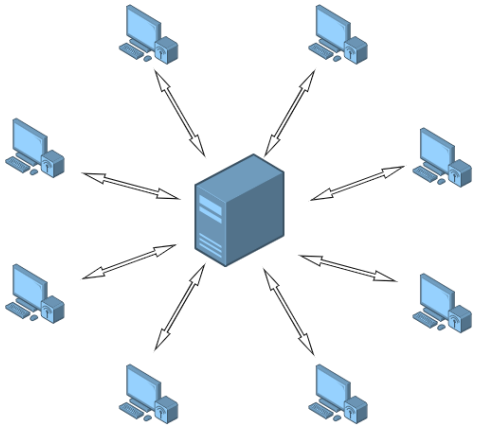
בחבילה זו מודול יחיד בשם gui_general.py, החושף קבועים ופונקציות עזר לשימוש קוד ממשק המשתמש.

5.3.5. החבילה main

בחבילה זו שני מודולים: server_main.py ו-client_main.py. במודולים אלו ממומשת הלוגיקה הראשית של צד השרת וצד הלקוח, יחד עם ממשק המשתמש הייעודי לכל אחד מהם.

5.4. פרוטוקול התקשרות

מבנה הרשת המוגדר על ידי המערכת הוא מבנה כוכב, בו כל הלקוחות מתחברים אל שרת מרכזי אחד, מסתנכרנים מולו ומקבלים ממנו פקודות לביצוע. כאמור בסקירה, כל התקשורת בין השרת ללקוחות וחזרה נעשית מעל פרוטוקול TCP בשליחת קבצים והוראות, וב-UDP לצורך סנכרון מול שרת ה-NTP.

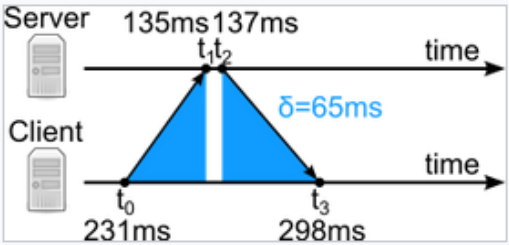


איור 1 - מבנה הרשת המוגדר על ידי התוכנית: שרת מרכזי אחד, המסנכרן בין מספר קליינטים.

5.4.1. פרוטוקול NTP

Network Time Protocol הוא פרוטוקול המאפשר להתקני רשת לקבל את השעה המדויקת מהתקני רשת אחרים המשמשים כ Time Server. NTP הוא אחד הפרוטוקולים הישנים הקיימים היום באינטרנט, והחלו להשתמש בו עוד לפני 1985.

התקשורת עם השרת נעשית בפורט UDP 123. מועבר בה מידע על השעון על-גבי מבנה נתונים המכיל 32 ביט המתארים את השניות, ועוד 32 ביט לתיאור חלקי השניות – מה שמאפשר בתיאוריה דיוק של עשיריות-ננו שניות. הדיוק הממשי הוא של כ-10 מיליוניות השנייה.



איור 2 - אילוסטרציה של סנכרון זמנים למול שרת NTP.

אחת הבעיות שהפרוטוקול מתמודד איתן היא הזמן שלוקח לשדר להגיע מרכיב לרכיב ברשת. כאשר מחשב מבקש להסתנכרן מול שרת זמן הוא שולח לשרת הזמן את הבקשה ואת השעה המקומית אצלו. כאשר חוזר משרת הזמן שדר המכיל את השעה הנכונה יחד עם השעה שנשלחה על ידי השרת המבקש, מושווה השעה המקומית הנוכחית לשעה שנשלחה בשדר המקורי הקיימת גם בשדר שהוחזר ואז מבוצע חישוב של הזמן הממוצע שלוקח לשדר לעשות את המרחק משרת אחד למשנהו. הממוצע מתקבל לאחר שליחת בקשה לסנכרון מספר פעמים (לרוב 4).

5.4.2. פרוטוקול פנימי – העברת מסרים וקבצים

עם עלייתו, השרת פותח thread המאזין לחיבורים חדשים. לקוח חדש המעוניין להתחבר לשרת שולח לו הודעת פתיחה. השרת מקבל את הלקוח ומוסיף אותו לרשימת הלקוחות שלו.

המבנה הבסיסי של כל הודעה הנשלחת במערכת בין השרת ללקוחות הוא כדלהלן:

0-3	4-7
length	Type
payload …	

כאשר length מציין את גודל ההודעה כולה (כולל type), ו-type את סוגה: FileSyncPacket או SignalPacket. ה-payload מכיל את תוכן ההודעה מהסוג המתאים.

5.4.2.1. פרוטוקול סנכרון קבצים

מבנה הודעת FileSyncPacket הוא:

0-3	4-7	8-11
message type	file size	file name length
file name …		

השדה file size מציין את גודל הקובץ לסנכרון. השדה file name מציין את שם הקובץ, ו- file name size מציין את אורכו של שדה השם. השדה message type מציין את סוג המסר המועבר מבין הארבעה:

8. who has – 2

הודעה מסוג זה נשלחת מהשרת לכלל הלקוחות, ומטרתה לתשאל כל לקוח האם הקובץ הרצוי שמור אצלו לוקאלית.

9. have – 1

הודעת תשובה להודעת who has, הנשלחת על ידי לקוח שברשותו הקובץ הרצוי במלואו.

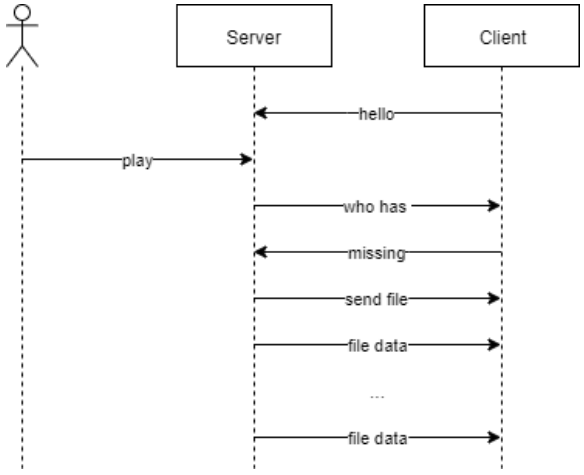
10. missing – 0

הודעת תשובה להודעת who has, הנשלחת על ידי לקוח שהקובץ הרצוי אינו קיים אצלו, או קיים אצלו באופן לא תקין (בגודל קובץ שונה).

11. send file – 4

הודעה מקדימה לשליחה של הקובץ עצמו. הודעה זו נשלחת מהשרת ללקוחות שיש לסנכרן עימם את קבצי השמע. לאחריה, ישלח השרת את המידע של הקובץ עצמו.

להלן תרשים זרימה המתאר את פעולות הפרוטוקול עבור סנכרון קובץ. האיור מתאר תקשורת למול לקוח אחד, אולם אותו התהליך משוכפל על ידי המערכת עבור יתר הלקוחות אשר התחברו לשרת.



איור 3 - סנכרון קובץ לנגינה בין שרת ללקוח. (1) הלקוח מתחבר אל השרת בהודעת פתיחה; (2) המשתמש מורה לשרת להתחיל לנגן; (3) השרת מתשאל את הלקוח: האם הקובץ קיים אצלו; (4) הלקוח משיב שהקובץ לא ברשותו; (5) השרת שולח הודעה לציון תחילת העברת הקובץ; (6) השרת שולח את הקובץ.

פרוטוקול סנכרון הקבצים של המערכת הינו חסכוני בתעבורה, שכן קבצים נשלחים ברשת אך ורק ללקוחות הזקוקים לקובץ. אמנם שליחה אוטומטית של הקבצים לכלל הלקוחות, ללא בדיקה מקדימה של הימצאות הקובץ, הייתה דורשת שליחת פחות הודעות (שכן הייתה חוסכת את הודעות ה-who has והמענה עליהן), אולם הגודל היחסי של הודעות סנכרון אלו לעומת גודלם של קבצי מוזיקה הוא קטן מאוד, ועל כן החיסכון בתעבורה הינו משמעותי.

5.4.2.2. פרוטוקול שליחת אותות וסנכרון זמנים

אותות נשלחים תמיד מהשרת אל הלקוחות, באמצעות הודעות מסוג SignalPacket. מבנה הודעת SignalPacket הינו:

0-3	4-7	8-11	12-15
signal	send timestamp		wait seconds
file name length	file name …		

שדה ה-signal מציין את הפקודה שיש לבצע: play (1), stop (2), pause (3), unpause (4). השדות file name ו-file name length תואמים זה לזה, ונצרכים רק בעת שימוש בפעולת ה-play (ביתר המקרים, שדות אלו אינם בשימוש).

השדה send timestamp מכיל double עם חתימת הזמן בו *נשלח* האות אל הלקוח מהשרת. שדה זה ישמש לסנכרון זמן ביצוע הפקודה בין הלקוחות השונים, כך שכולם יבצעו את הפקודה באותו הרגע בדיוק.

שדה ה-wait seconds מציין כמה זמן על הלקוח להמתין, לפני ביצוע הפקודה. שדה זה נחוץ על מנת להתגבר על מקרי קצה הנובעים מן העברת מסרים ברשת, אשר עשויים להשפיע על הדיוק בביצוע הפעולה בתיאום מדויק בין הלקוחות – כגון עיכובים, איבודים וכיו"ב.

נפרט על אופן בצוע הפקודות והסנכרון בין השרת ללקוחות:

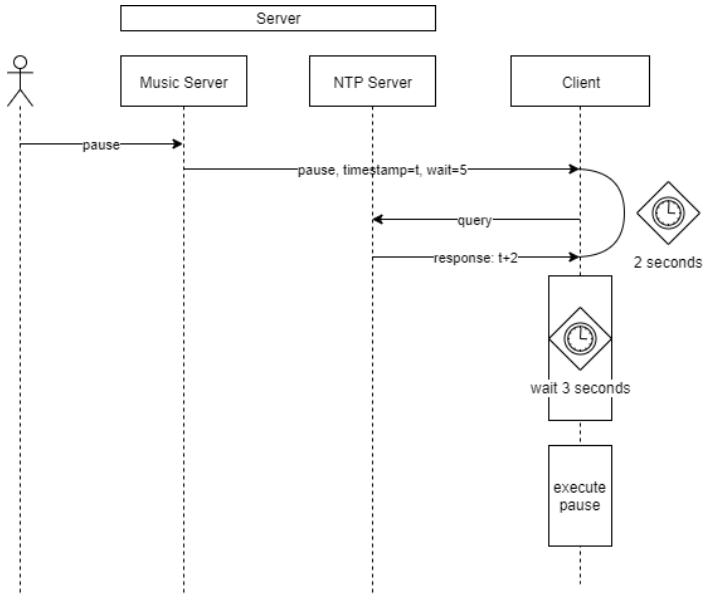
כאשר לקוח מקבל פקודה כלשהי לביצוע, עליו לבצע פקודה זו בדיוק x שניות לאחר *שליחת* ההוראה מהשרת, כאשר x הוא המספר המצויין ב-SignalPacket בשדה wait seconds. מכיוון שעם שליחת האות נשלח גם הזמן שבו יצאה ההודעה מן השרת (בשדה send timestamp), יכול הלקוח לחשב בדיוק כמה זמן עליו להמתין על מנת לבצע את הפקודה בזמן.

נמחיש באמצעות דוגמה: נניח שהשרת שלח הוראה כלשהי בזמן t_0, ולקח לה 3 שניות להגיע ללקוח A – כלומר A קיבל את האות בזמן $t_0 + 3$. נניח עוד כי השרת דרש לבצע את הפקודה המתאימה בתוך 5 שניות - $x = 5$. אזי על הלקוח A להמתין בדיוק שתי שניות בטרם יבצע את הפקודה.

באופן זה, אם נקבע קבוע x גדול מספיק, נבטיח שכלל הלקוחות יבצעו את הפעולה הנדרשת באותו הזמן – שכן גם אם קיים לקוח המקבל הודעות מהר יותר או לאט יותר מהאחרים, הפער יגושר באמצעות זמן ההמתנה המכוייל לזמני השרת.

כפי שצוין בסקירה, השרת משמש למעשה לשני תהליכים מרכזיים – שרת מוזיקה (המחלקה MusicServer) – לסנכרון קבצים והעברת פקודות ללקוחות, ושרת NTP (המחלקה NTPServer) – לסנכרון זמנים בין הלקוחות. כאשר לקוח מבקש לחשב את זמן ההמתנה הנכון בעבורו, הוא בודק מהו הזמן אצל השרת – ולפיו מסיק את זמן קלת המסר אצלו. באופן זה, אנו מבטיחים אי-תלות בשעונים המקומיים בעמדות הלקוחות.

נסכם פרק זה בתרשים זרימה המדגים את פרוטוקול שליחת המסרים, תוך סנכון מול שרת ה-NTP:



איור 4 - שליחת אות pause מהשרת ללקוח. השרת מחולק לשני תהליכים: שרת מוסיקה, ושרת NTP. לאחר שהמשתמש מבקש להשהות את הנגינה, שולח השרת ללקוח הוראת pause, עם זמן שליחה t וזמן המתנה של 5 שניות. הלקוח מקבל את הפקודה ושולח NTP query לשרת. שרת ה-NTP מחזיר ללקוח את הזמן הנוכחי של העמדה בה רץ השרת המרכזי: t+2. הלקוח מסיק שעליו להמתין 3 שניות, ולאחר מכן מבצע את פקודת ההשהיה.