בפרויקט הזה בחרתי לבנות מערכת זדונית לשליטה מרוחקת במכשירים.

**פרוטוקול VNC**

העברת המידע מתבססת על אלמנטים של הפרוטוקול VNC שהוא פרוטוקול שמאפשר שליטה על מחשבים מרוחקים בצורת peer to peer ושיתוף המסך של המחשב הנשלט.

שיתוף המסך של הפרוטוקול עובד ע"י שליחת הפריימים של מסך המחשב הנשלט ישירות למחשב השולט, בניגוד לפרוטוקולים אחרים כמו RDP שבהם נשלחים הוראות למחשב השולט על איך צריך לבנות מחדש כל פריים שנשלח מהמחשב הנשלט.

אז נשאלת השאלה איך הפרוטוקול הזה יכול לתפקד בצורה חלקה, הרי להעביר פריימים שלמים בצורה חלקה זה דבר שדורש הרבה משאבים.

התשובה היא שהפרוטוקול משתמש בדרכים שונות להעברת הפריימים שיעילות הרבה יותר מהעברת כל הפריימים כמו שהם, כל דרך להעברת הפריימים נקרא Encoding (או קידוד).

הקידוד הפשוט ביותר שכל שרת VNC חייב שיהיה לו נקרא Raw Encoding – זוהי דרך להעברת פריימים שבה הפריים הכי עדכני מושווה עם הפריים הקודם (הפריים שמוצג לקליינט) ונמצאים רק האזורים בפריים העדכני שהשתנו מהפריים הקודם וכך ניתן לשלוח רק את האזורים שהשתנו במקום את כל הפריים.

האזורים שהשתנו הם בעצם אזורים מלבניים וכל פריים פשוט נשלח לקליינט רשימה של מלבנים כך שלכל מלבן מכיל metadata, כלומר המידע על התמונה (הקורדינאטות של המלבן, הגובה שלו, האורך שלו) ואת התמונה שלקוחה מהפריים בקורדינאטות האלו כך שהקליינט ידע איפה באיזה אזור בתמונה צריך להוסיף את החלקים שהשתנו מהפריים הקודם.

כדי למצוא את האזורים שמשתנים מהפריים הקודם נשתמש בOpen CV- כך:

קודם כל נהפוך את הפריים העדכני והפריים הקודם ל-GrayScale (שחור-לבן) ונבצע את הפעולה XOR על שתי התמונות (XOR היא פעולה שמחזירה 0 אם שני המספרים הם אותו דבר ומספר שהוא לא 0 אם הם לא שווים), כך שנקבל תמונה חדשה שבה כל החלקים שלא השתנו הם שחורים (מכיוון ש-0 מתורגם לצבע שחור) וכל החלקים שנשארו דומים הם לא שחורים.

אחר כך נשנה את הצבע של כל האזורים הלא שחורים ללבנים ולבסוף נמצא את הקורדינאטות, האורך והגובה של כל האזורים הלבנים בתמונה ואז ניקח את כל המלבנים בפריים העדכני בעזרת הקורדינאטות שמצאנו.

**VidGear**

בפרויקט הזה השתמשתי ב-Framework שנקרא VidGear שאחראי על כל מה שקשור בהעברת וידאו (שיתוף מסך, סטרימים, הפעלת מצלמה, כתיבה של סטרים לקובץ ועוד...).

עד עכשיו השתמשתי בשני כלים של VidGear (שנקראים Gears): ScreenGear ו-NetGear.

VidGear נמצא בקבצי הפרוייקט מכיוון שהתאמתי אותו אישית לפי הצרכים שלי ולכן אי אפשר להוריד את החבילה מהאינטרנט.

ScreenGear הוא כלי מעטפת ל-pyscreenshot רק שהוא מוסיף multithreading.

השתמשתי בגרסה מותאמת אישית של ScreenGear שמוסיפה מחסום.

pyscreenshot הוא בעצמו מעטפת להרבה פתרונות לצילום מסך (PyQt5, mss, PIL…), אני אישית בחרתי להשתמש ב-mss בגלל שהוא נתמך בכל פלטפורמה והוא מהיר מאוד.

NetGear הוא כלי שנועד לשליחה של וידאו ברשת שמשתמש בחבילה pyzmq (חבילה להעברת מידע ברשת) אבל הוא מוסיף גם multithreading.

באמצעות NetGear אפשר להתחבר עם מחשב אחר ולהתחיל להסטרים אליו מידע.

ב-NetGear יש גם אפשרות ל-lossy compression, כלומר דחיסה שבה נאבד מידע מהתמונה, כלומר הורדת האיכות של התמונה, באמצעות הספרייה simplejpeg כך שגודל התמונה יורד וכך העברת התמונה תהיה מהירה יותר על חשבון הזמן שלוקח לכווץ את התמונה.

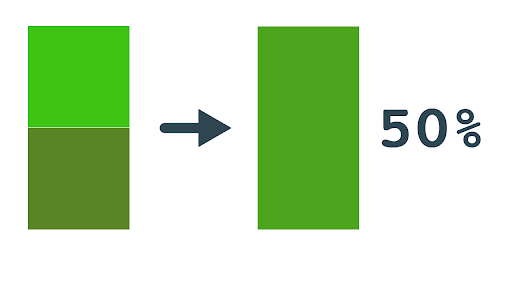
הגרסה של NetGear שבה השתמשתי היא גרסה מותאמת אישית: הוספתי גם את הקטנת ממדיי התמונה לפני שליחתה כך שתהיה קטנה יותר בזיכרון ויהיה קל יותר להעביר אותה ברשת ובנוסף השתמשתי בloseless compression-, כלומר דחיסה שבה לא נאבד מידע מהתמונה אבל היא תהיה אפילו קטנה יותר בזיכרון, ע"י הספרייה zlib.

כל אלו תורמים לחיבור להיות חלק מאוד וכמעט בלי delay.

בהמשך גם אשלב את ה-Raw Encoding ב-NetGear כדי שהעברת הפריימים תהיה אפילו מהירה יותר.

**loseless/lossy compression**

שתי שיטות הכיווץ האלו שונות לגמרי אבל אפשר לשלב את שתיהם כדי לכווץ תמונה.

lossy compression הוא כיווץ שבו מאבדים מידע מהתמונה, על ידי שימוש באלגוריתם שמוריד את איכות התמונה בכך שהוא מחבר כמה פיקסלים עם צבעים שונים לצבע אחד שהוא הממוצע של הצבעים של כל הפיקסלים שהתחברו לדוגמא: 

אם האלגוריתם החליט לחבר את שני הפיקסלים האלה הוא יחליף אותם בצבע שדומה מספיק לשניהם.

loseless compression הוא כיווץ שבו לא מאבדים שום מידע מהתמונה, על ידי שימוש האלגוריתם שבעצם משמש לכיווץ כל מידע ולא רק תמונות.

האלגוריתם עובר על הבתים שמרכיבים את התמונה ואם הוא מוצא רצף שחוזר על עצמו הוא מכווץ אותם כך: אם האלגוריתם מצא רצף של בתים 0xFF 0xFF 0xFF 0xFF 0xFF הוא מכווץ את הרצף לבתים 0x05 0xFF.

בעצם האלגוריתם החליף את הרצף של חמש הבתים הדומים בבית אחד שאומר כמה בתים חוזרים על עצמם יש ברצף (0x05) ועוד בית שאומר מה חוזר על עצמו (0xFF) וכך לא איבדנו שום מידע.

אפשר להשתמש ב-lossy compression קודם ואז ב-loseless compression כדי לכווץ את התמונה מאוד.

**Framerate**

הגבלת ה-fps חשובה מכיוון שגורם להעברת הפריימים להיות בקצב קבוע, בנוסף ניתן להשתמש בשיטה זו כדי להוריד את השימוש במעבד ואת השימוש ב-network בשביל מחשבים שלא יכולים לעמוד בשיתוף המסך הכבד.

על מימוש מחסום ה-fps נדבר אחר כך.

**UI/UX**

בפרויקט הזה השתמשתי בספרייה PyQt5 כדי ליצור את הממשק למשתמש.

הממשק יאפשר למשתמש ליצור session כך שמשתמש אחר יוכל להתחבר או יאפשר להתחבר ל-session שנפתח ע"י מחשב אחר.

כאשר שני משתמשים יתחברו יהיה ניתן להפעיל את שיתוף המסך (הסטרים יופיע באזור המסך השחור).

הממשק משתמש בקובץ json פשוט שנקרא "config.json"כדי לשמור את הגדרות המשתמש.

**utils.py**

קובץ זה מכיל פונקציות עזר שבהם השתמשתי, משתנים וקבועים.

קובץ זה כולל פעולות עזר למימוש הגבלת הפריימים לשנייה:

כדי להגביל את כמות הפריימים לשנייה צריך איכשהו להבין כמה זמן עובר בין כל פעם שframe- מתחיל עד לסופו.

כדי להשיג את הזמן המדויק בכל רגע, נשתמש בפקודה perf\_counter() מהספרייה המובנת time.

הסיבה שלא נשתמש הפקודה time.time היא שהפקודה הזאת לא מדוייקת, perf\_counter() היא הדרך המדויקת ביותר לקבל את הזמן הנוכחי.

נשתמש בפקודה פעמיים, פעם אחת בתחילת הפריים ופעם אחת בסופו (לאחר הפעולה הנדרשת), נחסר את הסוף וההתחלה ונקבל את הזמן שלקח לבצע את הפרים הנוכחי.

השלב הבא יהיה לראות כמה זמן עבר (הפרש הזמנים) ולחכות את הזמן שנשאר לדוגמה: אם נרצה שה-fps יהיה 24 נצטרך שכל פריים ימשך שניות ולכן נחסיר את הפרש הזמנים מהזמן שהפריים אמור להימשך כך: *.*

מה שקיבלנו הוא הזמן שצריך לחכות עד לפריים הבא ולכן נכניס את הערך הזה לפונקציית העזר halt, העוצרת את התוכנית למספר מסוים של שניות.

הסיבה שלא השתמשתי בפקודה time.sleep היא שהפקודה הזאת לא מדויקת לרמה של מילישניות וגם לא עקבית – פעם אחת הפקודה יכולה לחכות קצת פחות המזמן המבוקש ופעם קצת יותר ולכן בניתי פעולה משלי שתהיה מדויקת ועקבית.

הפעולה SpinLock היא גם פעולה שמחכה מספר מסוים של שניות, היא הפעולה הכי מדויקת.

הפעולה בודקת את הזמן הנוכחי ללא הפסקה בלולאת while, הזמן הנוכחי מוחסר מהזמן שבו הפונקציה החלה לפעול ונעשית בדיקה האם הזמן שעבר מאז תחילת הפונקציה קטן המזמן שהמשתמש ביקש לחכות, אם כן הפעולה מסתיימת.

הבעיה בשיטה הזו היא שלולאת ה-while מבזבזת הרבה cpu ולכן יש את הפונקציה halt.

הפונקציה halt היא קצת פחות מדויקת מ-SpinLock אבל היא מדויקת מספיק ולא משתמשת בהרבה משאבים בעזרת שימוש משולב בין הפונקציה SpinLoop לפונקציה time.sleep.