פרויקט סייבר במסגרת תכנית גבהים

Onion Routing

לירון ברגר

209142090

תיכון ליאו באק

הגנת סייבר

מנחים

שרית לולב

אלון בר־לב

תוכן עניינים

מבוא

פרויקט זה נועד לממש טכניקה לתקשורת אנונימית ברשת מחשבים, הקרויה ניתוב בצל - **Onion Routing**.

הפרויקט מממש מערכת המאפשרת למשתמשיה **תקשורת** דרך דפדפן האינטרנט אשר **אינה מאפשרת מעקב** אחר מקורן של ההודעות הנשלחות מהמשתמש. המערכת מבוססת על פרוטוקול socks5 עליו יפורט בהמשך.

המערכת בנויה מרשת של נתבי בצל או צמתים - nodes. המידע הנשלח מהדפדפן עובר דרך צומת הכניסה מועבר לשלושה צמתים אקראיים נוספים, עד שלבסוף מנותב אל היעד הרצוי. המידע מוצפן בשכבות לפי המפתחות הסודיים של הצמתים, ומכאן האנלוגיה לבצל.

המערכת מורכבת ממספר משתתפים:

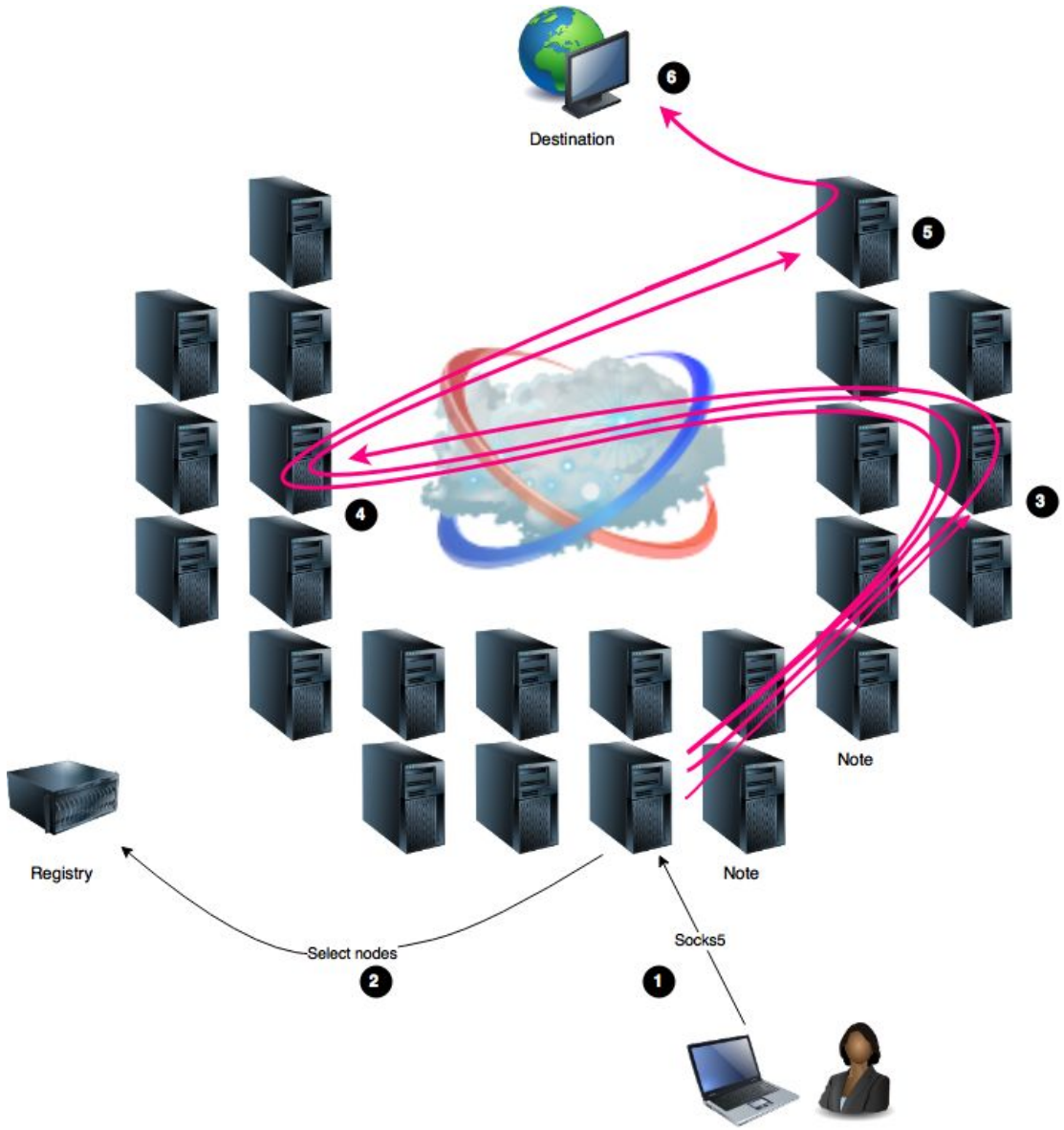
- **Entry Node** - לקוח socks5 אליו מתחבר הדפדפן. הצומת אחראי על יצירת החיבור בין הדפדפן ליעדו תוך כדי מעבר דרך צמתים נוספים.
- **Onion Node** - שרת socks5 אשר משמש לניתוב הבצל.
- **Registry** -שרת HTTP המשמש לצורך אחסון מידע על נתבי הבצל (צמתים) המחוברים כעת.

כלל המשתתפים פועלים באופן א-סינכרוני המאפשר עבודה מקבילית על מספר רב של חיבורים.

למערכת ממשק משתמש נוח וקל לשימוש המאפשר קבלת מידע על צמתים מחוברים לרשת הבצל וסטטיסטיקה על התעבורה ברשת.

ארכיטקטורה

ההודעות הנשלחות מהדפדפן נשלחות אל צומת הכניסה ולאחר מכן עוברות דרך מספר צמתים נוספים. כל צומת מנתב את ההודעה אל הצומת הבא. לאחר התחברות אל הצומת הרביעי והאחרון מנתבת ההודעה נשלחת אליו ההודעה המקורית של הדפדפן, והצומת מחבר אותו אל היעד הרצוי.



	הסבר
1	הדפדפן פותח חיבור socks5 לצומת הכניסה - אחד מנתבי הבצל.
2	הצומת בוחר באופן אקראי שלושה צמתים נוספים מה-Registry.
3	צומת הכניסה פותח חיבור socks5 לצומת הראשון.
4	צומת הכניסה פותח חיבור socks5 לצומת השני באמצעות החיבור לצומת הראשון.
5	צומת הכניסה פותח חיבור socks5 לצומת השלישי באמצעות החיבור לצומת השני.
6	צומת הכניסה מנתב את הודעות socks5 המתקבלות מהדפדפן ל-socks5 של הצומת השלישי.

התקשורת בפרויקט מבוססת על **פרוטוקול socks5**. פרוטוקול זה נבחר מכיוון שמטרתו היא **להעביר חבילות מיד**ע בין שרת ללקוח **דרך שרת פרוקסי**. הלקוח מתחבר לשרת הפרוקסי ומיידע אותו ברצונו ליצור חיבור ליעד מסוים. שרת הפרוקסי מבצע את החיבור. כך, באמצעות העברת המידע אודות היעד הבא לחיבור, צומת הכניסה יכול לפתוח חיבור בפרוטוקול זה לצומת הראשון של המערכת וכך להתחבר לשאר הצמתים, והדפדפן יכול להתחבר ליעדו דרך רשת הבצל.
כמו כן, התקשרת בין כל רכיבי המערכת מוצפנת במפתחות הצפנה שונים לפי הצמתים אליהם נוצר החיבור.

בפועל, הפרויקט יוצר מצב שבו הדפדפן שהתחבר לצומת הכניסה בחיבור socks5, לכאורה מתחבר ישירות לצומת השלישי, ומשם הודעותיו מנותבות אל היעד שלו ליעדו.

דיאגרמות רצף

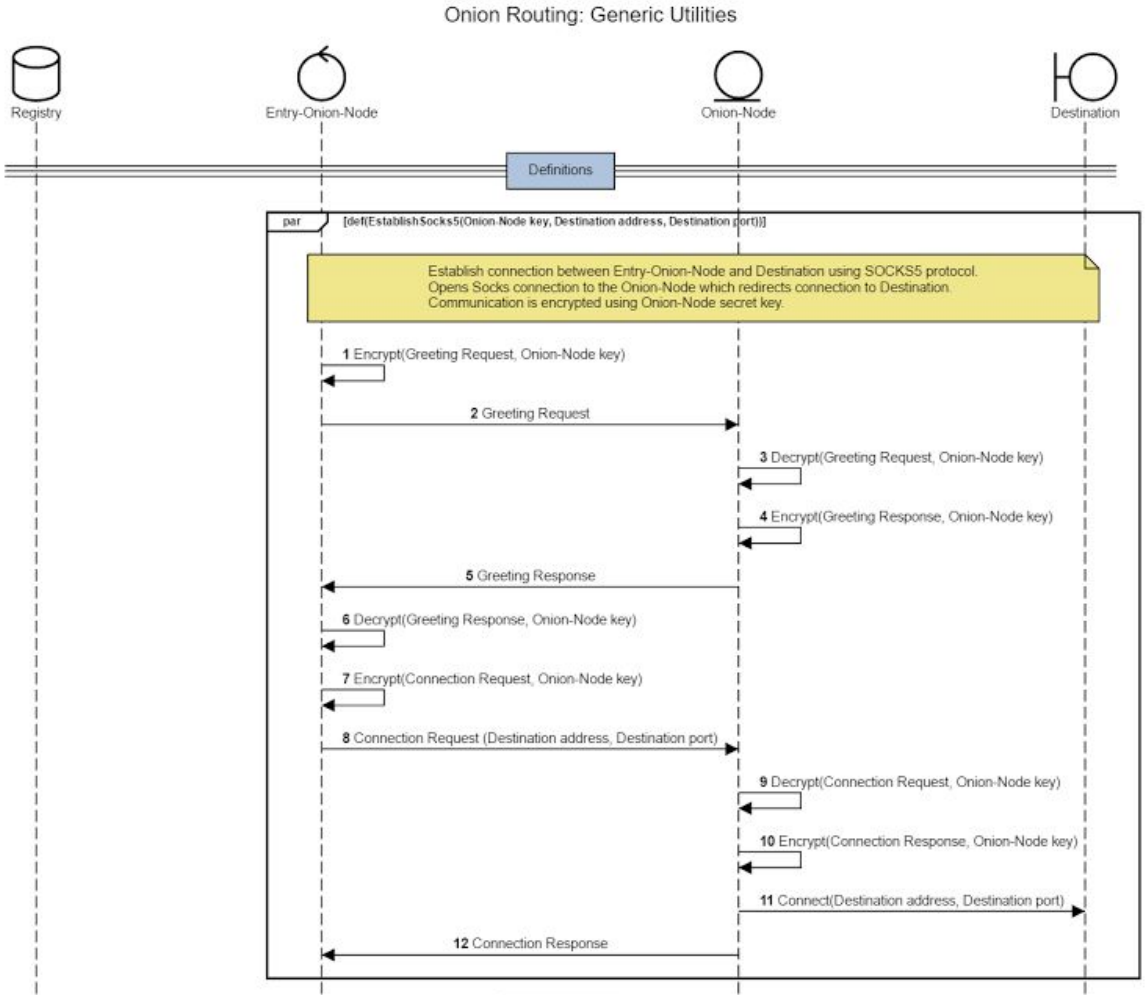
על מנת להמחיש כיצד כל רכיבי הארכיטקטורה בפרויקט, להלן דיאגרמת רצף אשר מתארת בצורה ברורה את הפעולות השונות המתרחשות בין הרכיבים.

דיאגרמות כלליות

דיאגרמות אלו הן גנריות. ישנו צורך בדיאגרמות מסוג זה מכיוון שבארכיטקטורת הפרויקט ישנם שלושה עצמים זהים - Onion Nodes.

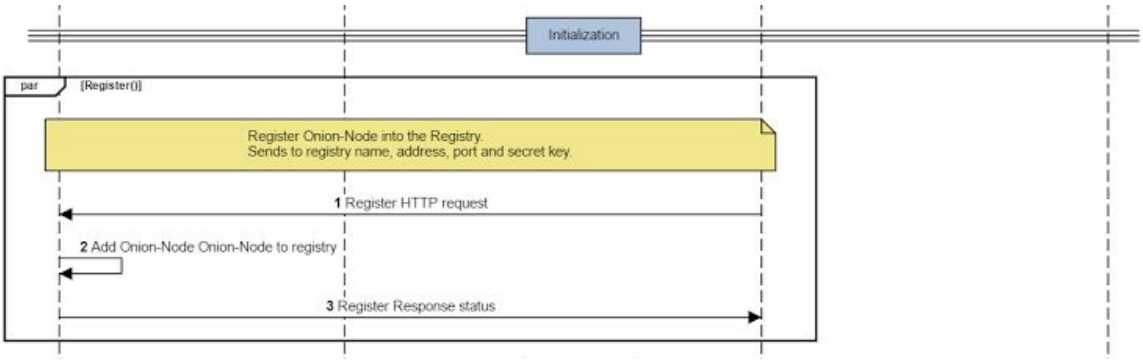
- **יצירת חיבור socks5**

  הרצף מתאר את תהליך יצירת חיבור socks5. כל ההודעות בין השרתים מוצפנות במפתח הצפנה של Onion Node. הסבר מפורט על כל אחד מהשלבים המתוארים ניתן לראות בפרוטוקול socks5 בפרק המימוש.



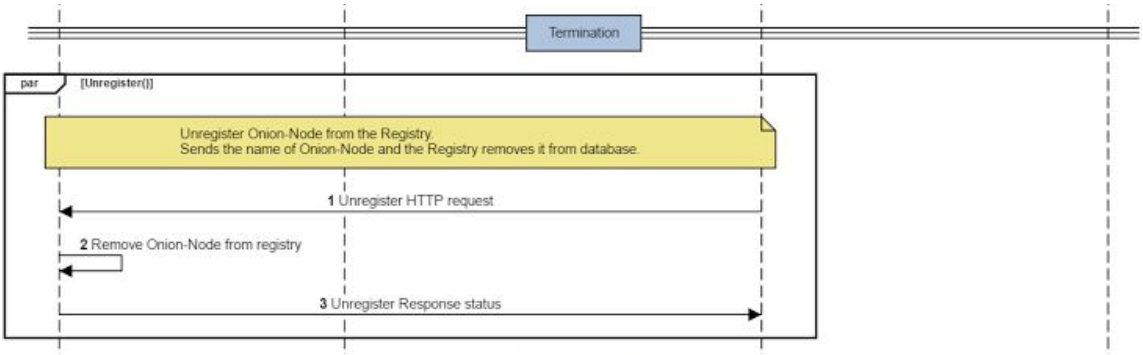
- **התחברות לשרת ה-Registry**

תהליך זה מאפשר צירוף של Onion Node חדש אל ה-Registry. ההתחברות מתבצעת על ידי שליחת הודעת חיבור מהצומת אל שרת ה-Registry עם פרטי הצומת החדש.



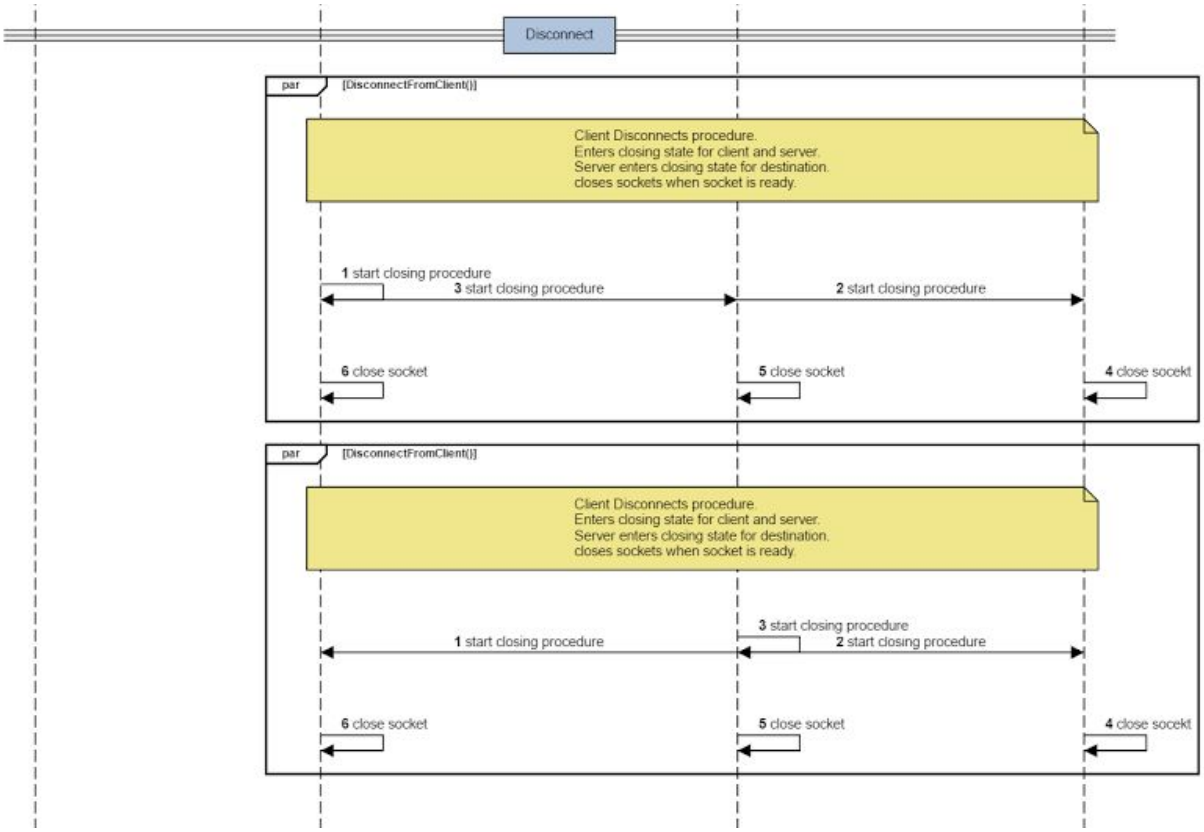
- **התנתקות משרת ה-Registry**

תהליך זה מאפשר ל-Onion Node שרוצה להתנתק למחוק את עצמו מה-Registry. הפעולה מתבצעת באמצעות שליחת בקשת התנתקות לשרת ה-Registry עם שם הצומת, וה-Registry מבצע את הבקשה.



- **התנתקות של אחד הצמתים**

תהליך זה מציג סגירה מסודרת של חיבורים שונים בעת התנתקותם של אחד או יותר מגורמי. ישנו צורך בסגירת החיבורים הרלוונטיים על מנת לא לבצע שליחה/קריאה מחיבור סגור.

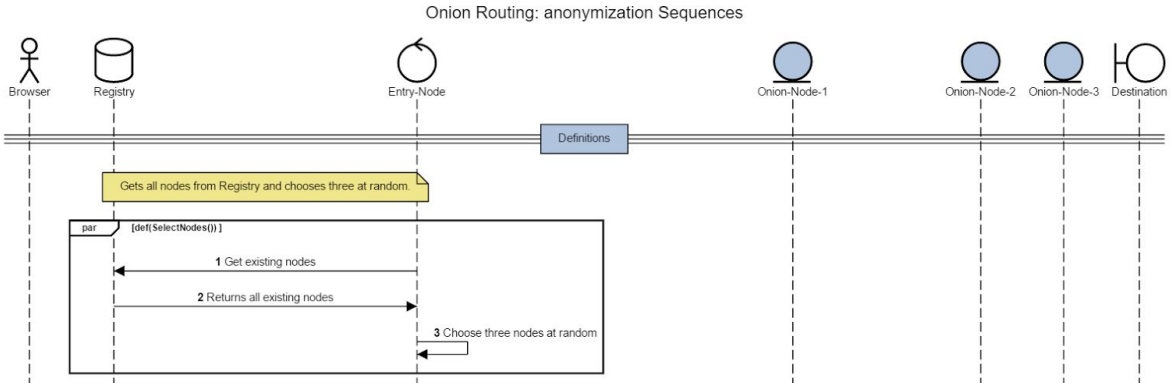


דיאגרמות רצף לאנונימיזציה - ניתוב בצל

דיאגרמה זו מראה את מהלך הפעולות שעושה המערכת על מנת לגרום להודעה מהדפדפן להגיע אל יעדה בעילום שם באמצעות שיטת Onion Routing.

- **בחירת צמתים עבור ניתוב בצל**

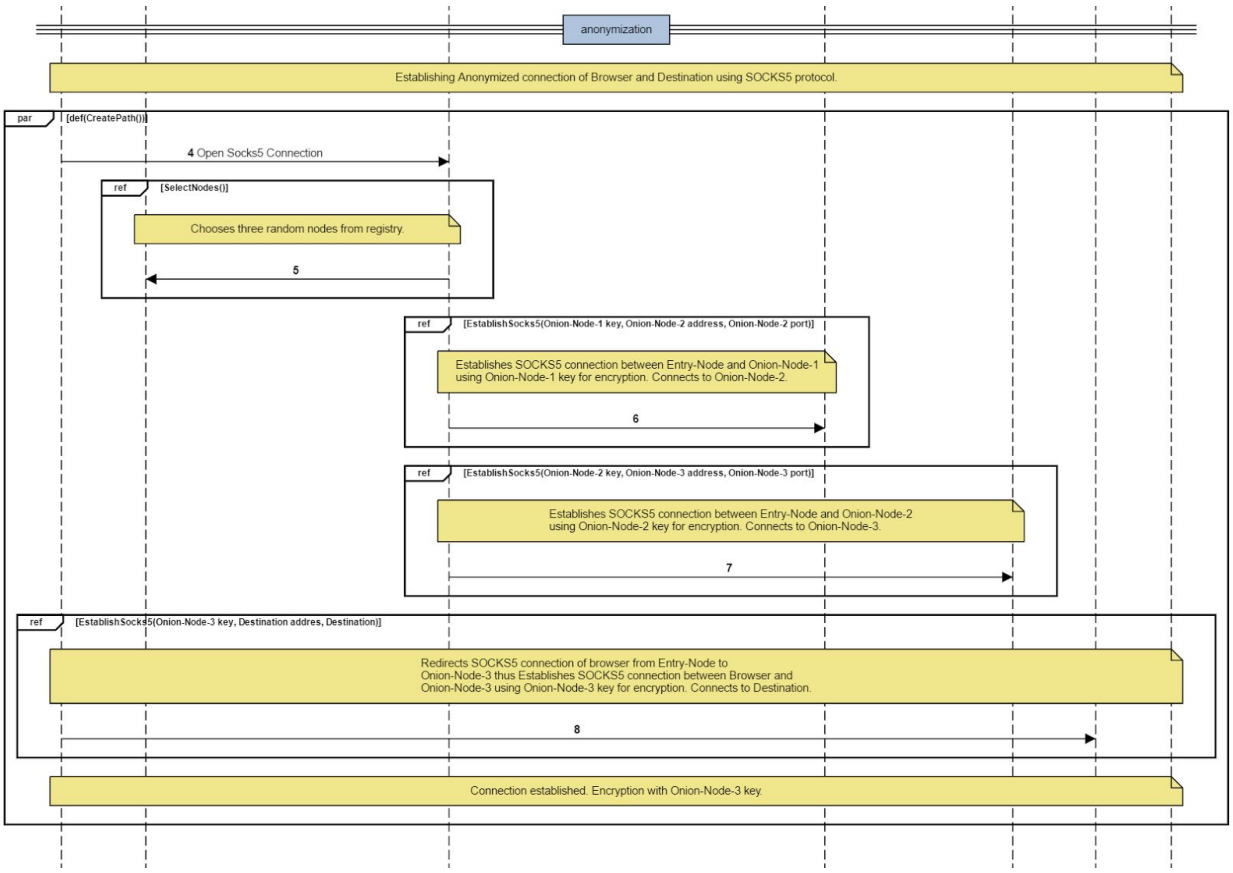
תהליך זה מציג כיצד נשלחת בקשה מצומת הכניסה לשרת ה-Registry, עבור קבלת כל הצמתים הקיימים במערכת. לאחר קבלת התשובה בוחר הצומת שלושה צמתים אקראיים עבור



ניתוב הבצל.

- **שליחת הודעות בדפדפן אל יעד בעילום שם**

שליחת הודעה מדפדפן ללקוח ע"י שימוש הצמתים שנבחרו על ידי צומת הכניסה. השליחה מתבצעת באמצעות יצירת חיבורי socks5 לצמתי הבצל השונים, ולאחר מכן ניתוב ההודעה מהדפדפן דרך הצמתים אל היעד הרצוי.



רקע תאורטי

ניתוב בצל - Onion Routing

בימינו, רשת האינטרנט אינה מגינה על פרטיותם של לקוחות שונים. כל שרת יכול בקלות לזהות את מקור ומיקום השולח כמו גם מידע רב אחר העובר דרך החיבור לשרת. בנוסף לכך, לצערנו, אין שום אמצעי למניעת מעקב זה, ושרתים מסוימים אף אוספים במכוון מידע אודות כל לקוח ומשתמשים בו למטרות רווח או מעקב.

אחת השיטות הידועות למניעת אפשרויות מעקב אחרי לקוחות, וגלישה בעילום שם באינטרנט היא Onion Routing, או בעברית - ניתוב בצל.

הרעיון הראשוני של ניתוב בצל הוצג לראשונה בשנת 1981 במאמר של דויד צ'אום על דואר אלקטרוני אנונימי. הרעיון היה שימוש במספר צמתים שנקראו MIX שקיבלו אימייל מוצפן, השתמשו במפתח הסודי שלהם לצורך פענוח ושלחו אותו לצומת הבא. בנוסף, כל אחד מהצמתים ביצעו שינויי זמנים על מנת להקשות לצופה מבחוץ לזהות את מסלול ההודעה. בגלל ששיטה זו התבססה על שינויי זמנים היא אינה מותאמת לתקשורת בזמן אמת.

הרעיון הזה פותח בשנות ב90 על ידי הצי האמריקאי למה שקרוי כיום Onion Routing.

מקור השם " בצל" מגיע משכבות ההצפנה השונות בהן נעטפת כל הודעת לקוח, עליהן יורחב בהמשך.

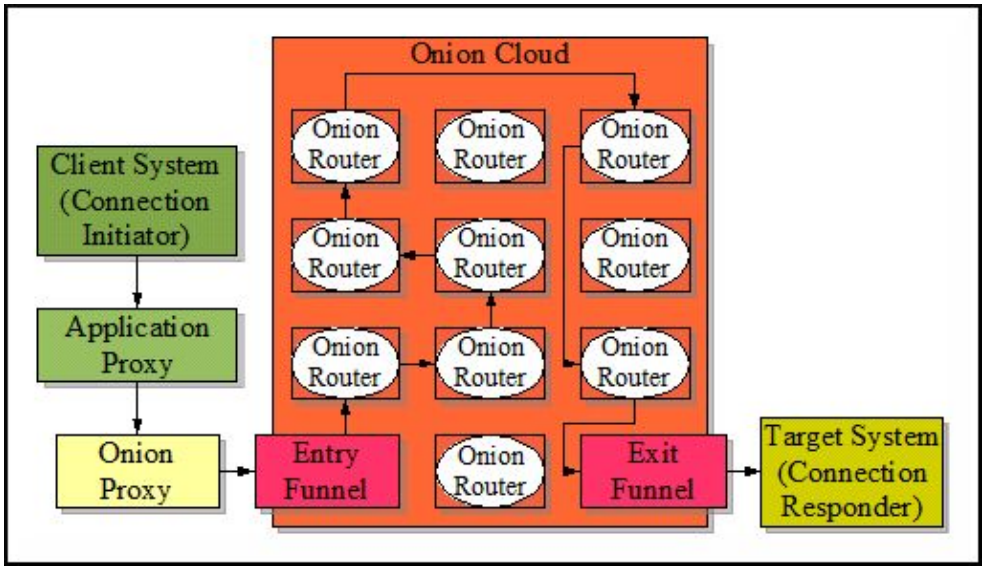
אופן פעולה

המערכת מורכבת ממספר מרכיבים הקרויים צמתי בצל. כל בצמתים מתקשרים ביניהם באמצעות פרוטוקול TCP. חלק מהצמתים יכולים גם לתפקד כצמתי כניסה - הם יכולים לקבל לקוחות מהרשת

(דפדפן). חלק מהצמתים יכולים לתפקד כצמתי יציאה, הם מממשים את החיבור הראשוני שהתבקש על ידי הלקוח ועבר דרך רשת ניתוב הבצל.

בעת קבלת חיבור חדש באחד מצמתי הכניסה, צומת הכניסה פותח חיבור למספר צמתים נוספים, עד שהצומת האחרון, צומת יציאה, מעביר את ההודעה אל היעד הרצוי.

ניתן לראות הסבר לאופן מעבר ההודעות במערכת בתרשים הבא:

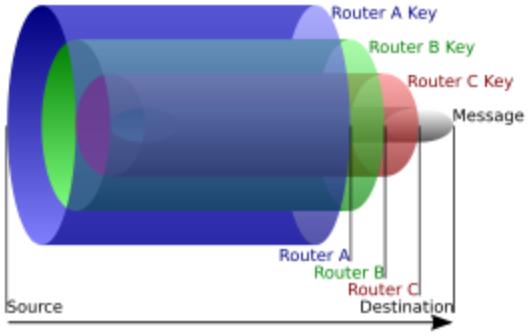


לפי דוגמה זאת, ניתן לראות את תהליך מעבר ההודעה מלקוח מסוים לרשת הפרוקסי (בתרשים - Onion Cloud).

ניתוב בצל מבוסס על הצפנה באמצעות מפתח ציבורי לכל צומת, אשר מאפשר ליצור שכבות הצפנה שונות, וכל שכבה יכולה להיות מפוענחת באמצעות המפתח הפרטי של כל צומת.

בצורה זו, כל צומת במערכת מקבל מידע אודות הלקוח המתחבר אליו, והיעד אליו הוא שולח את ההודעה בהמשך. כך, כל אחד שמאזין לרשת בניסיון למצוא את מקור ההודעה יכול לדעת אך ורק את המקור והיעד - ולכן, כבר לאחר מעבר של שלושה צמתים בודדים יעד ומקור ההודעה מוסתרים מהמאזין.

כפי שנאמר, ההצפנה מתבצעת באמצעות הצפנה בשכבות. ניתן לראות המחשה לכך בתרשים הבא:



כל צומת שמקבל את ההודעה בתורו יכול לפענח את השכבה האחרונה בלבד, וכך לבסוף, הצומת האחרון מפענח את המסר המקורי ומנתב אותו אל היעד.

פרויקט תור - Tor Project והבדל מרכזי

רשת ניתוב הבצל המתקדמת ביותר כיום קרויה פרויקט תור. ברשת כיום פעילים מעל ל-7000 צמתים שונים המאפשרים את התהליך שהוסבר קודם לכן.

בפרויקט תור, השלב הראשון בעת ניסיון של לקוח לגשת ליעד מסוים הוא העברת כלל המידע דרך שרת הפועל בתמיכה עם Tor. שרת זה לוקח את ההודעה ומרכיבה עליה מספר שכבות של הצפנה - קודם היא מצפינה אותי במפתח אחד, ההודעה המוצפנת, לאחר מכן, מוצפנת באמצעות מפתח שני וכך הלאה. לאחר מכן מעבירה השרת את ההודעה באמצעות פרוטוקול TCP את ההודעה לצמתים שונים בתור רשת הבצל של תור עד להגעתה ליעד הרצוי.
מימוש זה שונה מהמימוש בפרויקט שלי מכיוון שבתכנית שמימשתי ישנו שימוש בפרוטוקול socks5. הסבר על הפרוטוקול יפורט לעומק בהמשך בפרק המימוש, אך על מנת לחבר שני צמתים בתקשורת socks5 ישנו צורך בהעברת מספר הודעות בין לקוח ושרת, ולכן ההצפנה עם מפתח מתבצעת בין ההודעות העוברות בין הצמתים השונים, ואין מספר הצפנות עבור הודעה יחידה.

הסכנות בשימוש ב-Onion Routing

ניתוב בצל כפי שהוא משומש כיום בפרויקט Tor נותן מענה טוב לאנונימיות ברשת, אם כי לא מושלם. רשת ניתוב בצל בטוחה רק כל עוד כל הצמתים במסלול התקשורת מקיימים את הפרוטוקול כנדרש. לכן, על ידי החדרת מספר מסוים של צמתים אשר יתחזו לצמתים אשר פועלים באין מפריע ולפי הצורה

המתאימה לרשת, אך בפועל יפעלו באופן שונה ויאספו מידע אודות התקשורת המועברת, ניתן לנטר את מרבית התעבורה ברשת באין מפריע.
פרויקט Tor מנסה למנוע התחזויות מסוג זה, אם כי התקבלו בעבר שמועות דיווחים על כך שארגוני מודיעין ממשלתיים ניסו להחדיר מחשבים לרשת Tor ולעקוב אחר משתמשים ברשת.

בנוסף לכך, שיטה נוספת באמצעותה ניתן לאסוף מידע אודות חיבורים שונים ברשת ניתוב בצל, היא על ידי מידע מקדים אודות חיבור מסוים שעתיד לקרות. אימות החיבור בין שני הגורמים מתבצע על ידי בדיקת זמני שליחת המידע מצד אחד וקבלתו בצד השני. ממידע זה ניתן להסיק בהסתברות גבוהה מאוד ששני הגורמים מבצעים ביניהם תקשורת. בשיטה זאת אין אפשרות לתוקף המערכת לפענח את התוכן שהועבר בין הגורמים.

מושגי תקשורת

- **פרוטוקול SOCKS 5**

פרוטוקול תקשורת בשכבת התעבורה המאפשר יצירת ערוץ תקשורת בין לקוח אל יעדו תוך מעבר דרך שרת socks אליו מתחבר תחילה הלקוח. תקשורת דרך פרוטוקול זה מתבצעת ע"י שליחה של מספר הודעות בין הלקוח לשרת socks:

- **Greeting request** - בקשת תחילת חיבור ואפשרויות אופן ההזדהות.
- **Greeting response** - תגובת השרת ובחירת אופן ההזדהות
- **Connection request** - בקשת הלקוח לביצוע משימה כלשהי (command) אל יעד מסוים.
- **Connection response** - תגובת הלקוח וסטטוס ביצוע המשימה.

בתום רצף הודעות זה מתפקד שרת ה-socks5 כשרת פרוקסי בין הלקוח ויעדו.

- **פרוטוקול HTTP**

פרוטוקול תקשורת בשכבת היישום של מודל OSI ובשכבת היישום של מודל TCP/IP הנועד להעברת דפי HTML ברשתות אינטרנט. תקשורת בפרוטוקול HTTP מבוססת על בקשות הלקוח ותגובת השרת הקרויות **HTTP request** ו-**HTTP response.** כל בקשה/תגובה

מורכבת מחלקים קבועים והם - שיטת הבקשה (status), שדות כותרת (headers) ותוכן הבקשה (content).

הפרויקט תומך בסוג אחד של בקשות HTTP הקרויה GET. סוג זה של בקשה מיועד לקבל אובייקט אשר נמצא על השרת בכתובת שנמצאת בבקשה.

- **פרוטוקול TCP**

פרוטוקול תקשורת בשכבת התעבורה המשמש להעברת נתונים בין שתי תחנות ברשת מחשבים באופן אמין וללא איבוד מידע.

- **שרת פרוקסי Proxy**

זהו שרת שתפקידו לחבר בין לקוח ליעד מסוים, כאשר הוא משמש כמתווך ביניהם. כלומר, הלקוח מתחבר לשרת הפרוקסי ושולח אליו את המידע והיעד מקבל את המידע הזה משרת הפרוקסי ולהפך.

- **Socket**

הוא נקודת קצה (endpoint) עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים.

- **IPV4 Address**

הגרסה הנפוצה ביותר של פרוטוקול IP, המהווה את הפרוטוקול עליו מבוססות רשתות מחשבים.

כתובת IPv4 מורכבת מ-32 סיביות ומיוצגת באמצעות 4 מספרים עשרוניים המופרדים בנקודה. כל מספר מהווה מקבץ של 8 סיביות וגודלו נע בין 0 ל-255.

- **Port**

הוא תהליך ספציפי שדרכו יכולות תוכנות להעביר נתונים באופן ישיר. השימוש הנפוץ ביותר בפורט הוא בתקשורת מחשבים במסגרת הפרוטוקולים הנפוצים בשכבת התעבורה:TCP ו-UDP. פורט מזוהה לכל כתובת או פרוטוקול מסוים על ידי מספר באורך 16 ביטים היוצר 65536 כתובות אפשריות לUDP ו-65535 כתובות אפשריות לTCP. כתובת זו נקראת "מספר הפורט".

מושגי מערכות הפעלה

- **Asynchronous IO**

סוג תקשורת המאפשר לשלוח מסרים בתוך ערוץ תקשורת מבלי לחכות שהצד המקבל מוכן. באמצעות השימוש במערכת אסינכרונית קיימת אפשרות לעבודה מקבילית הן של שליחה והן

של קבלת הודעות ממספר חיבורים, ובכך ליצור תקשורת רבת משתמשים תוך שימוש בתהליך יחיד.

- **POSIX**

אוסף תקנים בסיסיים שנאגד במטרה לשמור על תאימות בין מערכות הפעלה, בעיקר בין מערכות מבוססות UNIX.

- **daemon**

תכנית מחשב שרצה כתהליך ברקע ואינה נמצאת בשליטה של אינטרקציה עם משתמש. בפרויקט זה קיימת תמיכה של הרצת תכניות השרתים בתור תכניות daemon.

- **file descriptor**

מספר המייצג קובץ פתוח במערכת ההפעלה.

- **Signals**

אות (signal) הוא הודעה אסינכרונית שנשלחת ל-thread מסוים בתוך תהליך כלשהו, כדי להודיע על כך שאירוע מסוים התרחש. בפרויקט זה ישנו שימוש בשלושה אותות שונים:

  - SIGINT
  - SIGTERM
  - SIGALRM

הצפנה

תהליך בו על ידי אלגוריתם כלשהו מעבדים מידע על מנת להפוך אותו לגורמים מסוימים ולאבטח אותו מפני גורמים אחרים.

- **הצפנת בסיסים מבוססת xor**

הפרויקט משתמש בהצפנה בסיסית אשר לוקחת מידע ומשנה אותו באמצעות פעולת ה-xor לפי מפתח נתון כלשהו.

שפות תכנות

- **שפת Python**

שפת תכנות דינמית נפוצה. תוכננה לצורך שימוש בפשטות במבני נתונים מסובכים.

- **שפת HTML**

שפת תגיות לתצוגה ועיצוב של דפי אינטרנט בדפדפן.

- **CSS**

פורמט לעיצוב דפי אינטרנט.

- **שפת JavaScript**

שפת תכנות דינמית מונחית עצמים המותאמת להרצה בדפי אינטרנט.

- **XML**

תקן לייצוג מבני נתונים במחשב. מאפשר דרך נוחה לשמור נתונים על אובייקטים.

מושגים נוספים

- **קובץ log**

קובץ שמטרתו היא תיעוד ההתרחשויות השונות בתכנית בזמן הרצתה. פרויקט זה משתמש בספריית logging של Python לצורך כתיבה לקובץ ה-log, ולשם נכתבים פרטים אודות אירועים משמעותיים בזמן ההרצה. ניתן לפתוח את ה-log במספר רמות, ולפיהן מוחלט רמת הפירוט הנכתבת ל-log.

- **קובץ Configuration**

קובץ המכיל מידע שמשמש כהגדרות בסיס עבור יישום כלשהו. לכל מרכיב בפרויקט ישנו קובץ קונפיגורציה עם הגדרות אלו.

- **תכנות מונחה עצמים - Object Oriented Programming**

שיטת תכנות המבוססת על חלוקת התכנית לאובייקטים בעלי תכונות ופונקציות ייחודיות. מאפשרת לחלק את התכנית לתיאורים נפרדים של כל רכיבי המערכת המפותחת.

- **תכנות מונחה אירועים - Event Driven programming**

שיטת תכנות לפיה ישנם חלקים במחשב אשר מחכים לקבלת אות מסויים. האות נקרא אירוע (event) ועם קבלתו יודעת התכנית לבצע פעולות מתאימות.

- **תרשים רצף - Sequence diagrams**

תרשים רצף הנועד לתאר תהליך ולהסביר כיצד המערכת ורכיביה מבצעים תהליך זה.

- **גיט - Git**

מערכת ניהול גרסאות פרויקטים מבוזרת.

- **Doxygen**

כלי אוטומטי לתיעוד של פרוייקטים.

מימוש

פרויקט זה משתמש באובייקטים רבים על מנת לבצע את כל הפעולות הנדרשות להשגת המטרה הסופית. מימוש המערכת מתבצע על ידי מספר מרכיבים מרכזיים:

Poller

כל מרכיבי המערכת מתבססים על תקשורת אסינכרונית על מנת לשלוח מסרים בתוך ערוץ תקשורת מבלי לחכות שהצד המקבל מוכן וכך לעבוד עם מספר רב של חיבורים במקביל. לשם כך משתמש הפרויקט בשיטת polling. לפי שיטה זו מוחזרת רשימה של חיבורי socket הפתוחים, ופעם בפרק זמן קבוע עבור כל אחד מהם מחזירה האם אחד מהם פנוי לקריאה, כתיבה או שהתקבלה שגיאה.

- POLLIN - אירוע קריאה.
- POLLOUT - אירוע כתיבה.
- POLLERR - אירוע שגיאה.

הפרויקט תומך בשתי שיטות polling:

- **Poll** - עובד רק בסביבת UNIX.
- **Select** - מתאים לכל מערכות ההפעלה.

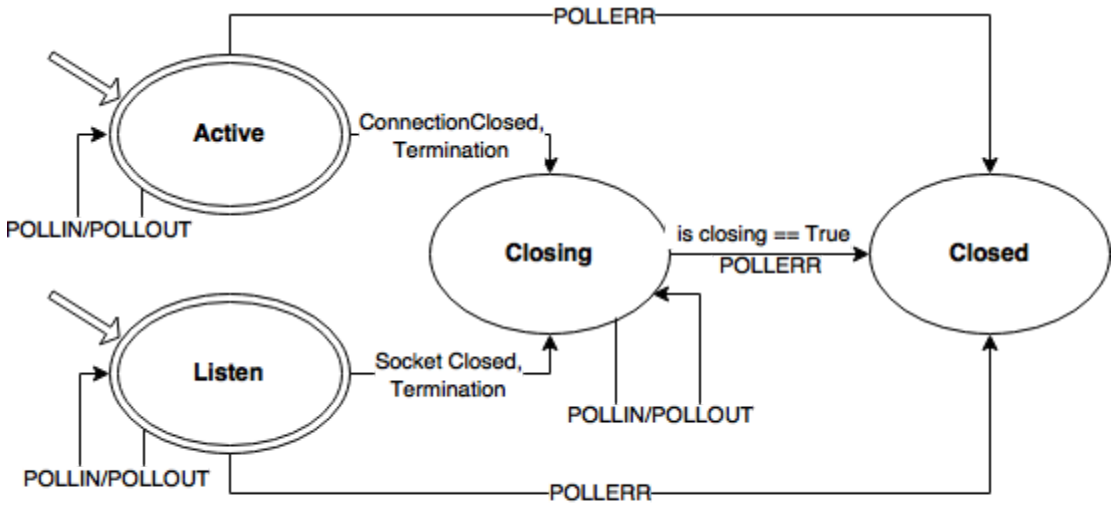
על מנת לאפשר התייחסות זהה לכל שיטה, הפרויקט מממש אובייקטים אשר מאפשרים פלט וקלט זהים לשני השיטות.

שרת אסינכרוני - AsyncServer

שרת אסינכרוני שעובד בשיטת polling באמצעות העצם Pollable. השרת יוצר עצם מסוג Poller, ועבור כל אחד מהחיבורים הפתוחים קורא לפונקציית קריאה/כתיבה/סגירה מתאימה.

השרת משתמש במעטפת עבור כל האובייקטים מסוג socket הקרויה Pollable.

להלן מכונת מצבים עבור כל עצם מסוג Pollable.



Pollable Object State Machine

מצב	הסבר
Active/Listen	אירוע קריאה - במקרה זה קורא AsyncServer לפונקציה on_read. אירוע כתיבה -במקרה זה קורא AsyncServer לפונקציה on_write. אירוע שגיאה - במקרה זה קורא AsyncServer לפונקציה on_error.
Closing	כל Pollable בוחר את האירועים להם הוא רוצה להקשיב לפני יציאה.
Closed	סגירה של Pollable.

שרת ה-Registry

שרת המבוסס על פרוטוקול HTTP. השרת משתמש במספר שירותים שונים על מנת לטפל בבקשות הלקוח:

- **שירות התחברות של צומת** - regsiter
  - השרת מקבל מהלקוח את שם, כתובת, פורט ומפתח סודי של צומת.
  - השרת מוסיף את הצומת את למילון הצמתים.
  - השרת מחזיר ללקוח הודעה על הצלחה או שגיאה במידה והצומת קיים במערכת.

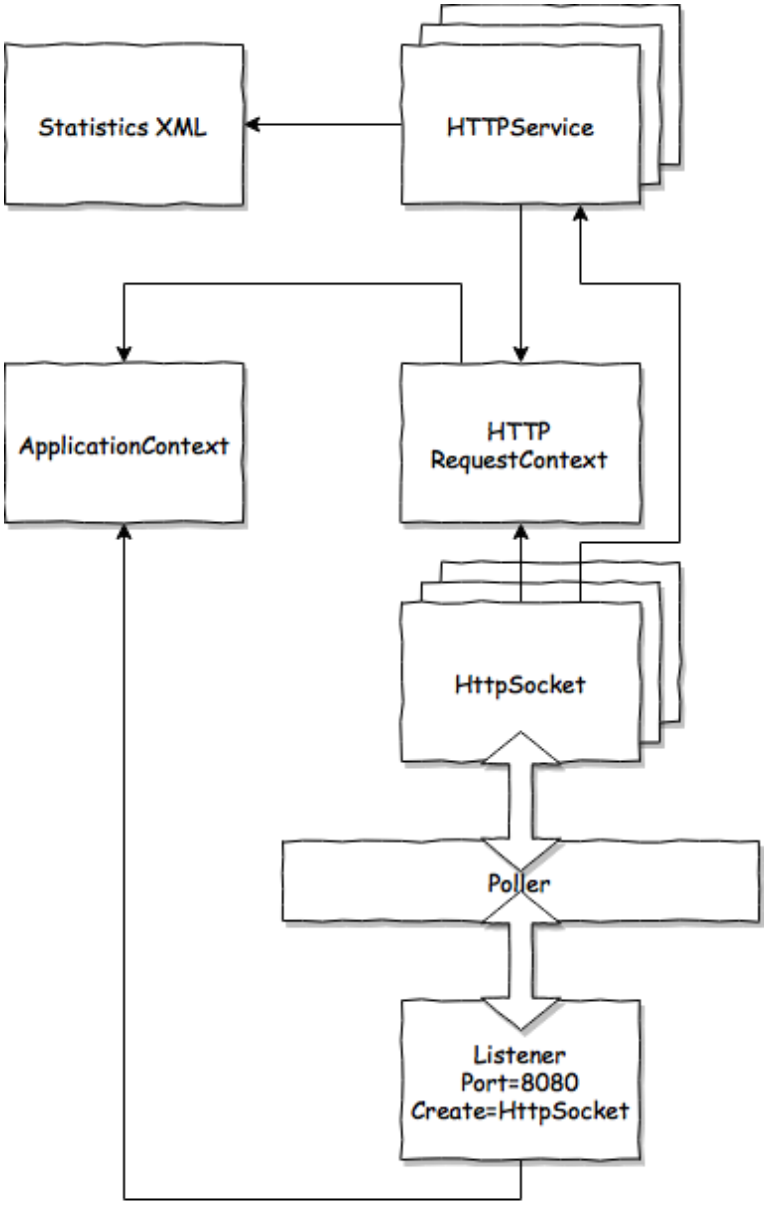
  שירות זה עונה על דרישת המערכת לאפשר כניסה של צמתים נוספים בכל שלב בהרצה. כל צומת שנכנס שולח את פרטיו ל-Registry וכך צומת הכניסה יוכל להשתמש בפרטים האלו על מנת לבחור שלושה צמתים אקראיים.
- **שירות התנתקות של צומת** - unregister
  - השרת מקבל מהלקוח שם של צומת.
  - השרת מוחק את הצומת ממילון הצמתים.
  - השרת מחזיר ללקוח הודעה על הצלחה או שגיאה במידה והצומת לא קיים במערכת.
- **שירות החזרת הצמתים המחוברים** - nodes
  - השרת מקבל מהלקוח בקשה לקבלת הצמתים.
  - השרת מחזיר ללקוח מילון של כל הצמתים המחוברים.
- **שירות קריאת קובץ** - (file name)
  - השרת מקבל מהלקוח שם של קובץ.
  - השרת עובר על התיקייה files ומחפש את הקובץ המבוקש.
  - השרת מחזיר ללקוח את תוכן הקובץ.

  שירות זה הוא השירות עליו מבוסס ממשק המשתמש של המערכת עליו יפורט בפרק התקנה ותפעול.

אופן פעולה

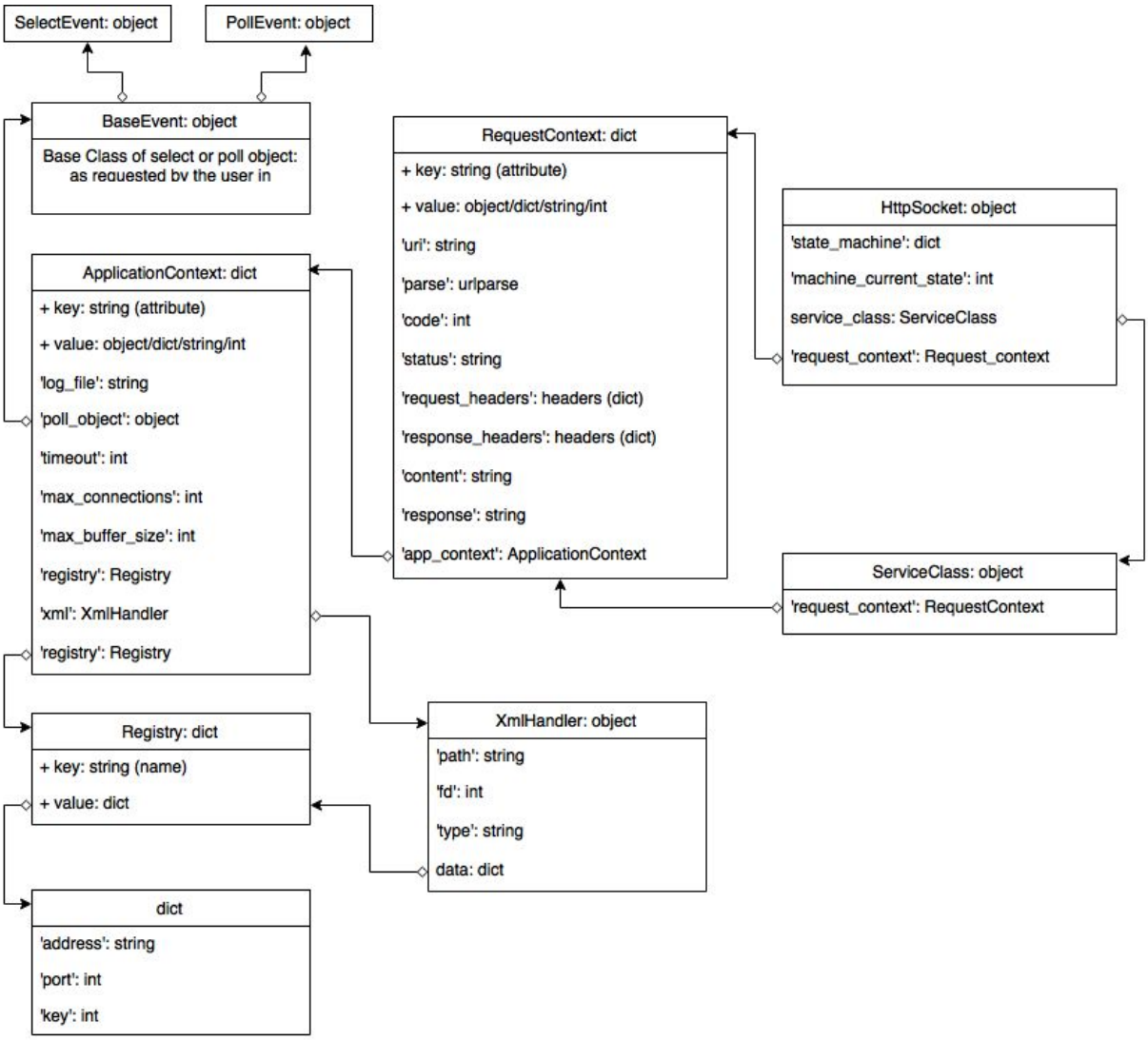
- המערכת פותחת אובייקט מסוג Listener ומוסיפה אותו ל-Poller.
- ה-Listener מאזין בפורט 8080.
- עם קבלת חיבור פותח אובייקט מסוג HttpSocket ומוסיף ואותו לPoller.
- HttpSocket מטפל בבקשות הלקוח (אופן הטיפול יוסבר תחת פרוטוקולים).

דיאגרמת בלוקים



Registry Block Diagram

דיאגרמת נתונים



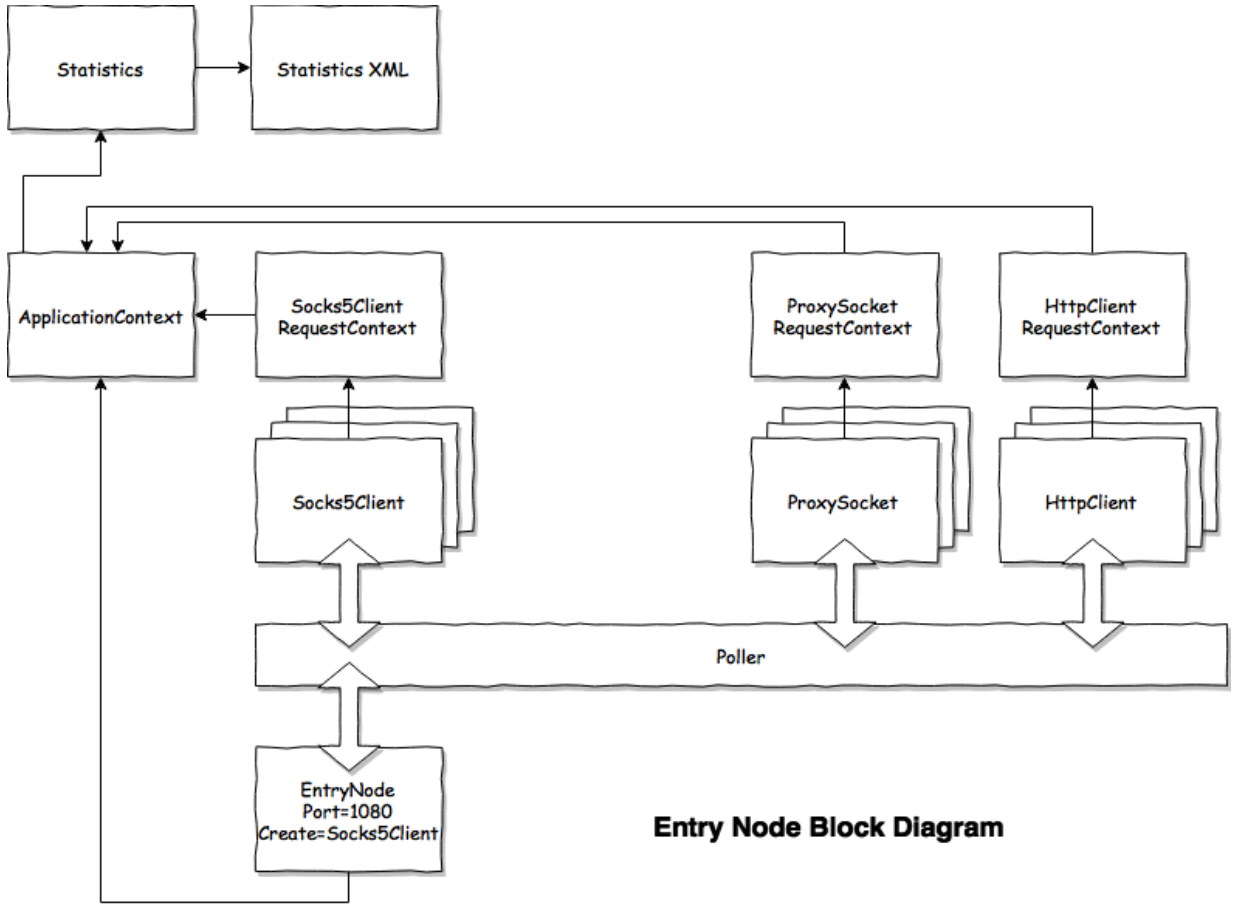
צומת הכניסה - Entry Node

צומת הכניסה הוא שרת מבוסס על פרוטוקול socks5 שמטרתו היא לקבל חיבור מהדפדפן (הלקוח) ולהעביר את המידע שנשלח דרך סדרה של צמתים אחרים על מנת שההודעה תשלח בעילום שם.
הצומת פועל כלקוח socks5 על מנת להתחבר ל-socks5 של שאר הצמתים ולאחר שהצומת התחבר לכל שאר הצמתים הוא פועל כשרת פרוקסי רגיל להעברת המידע שנשלח מהדפדפן.

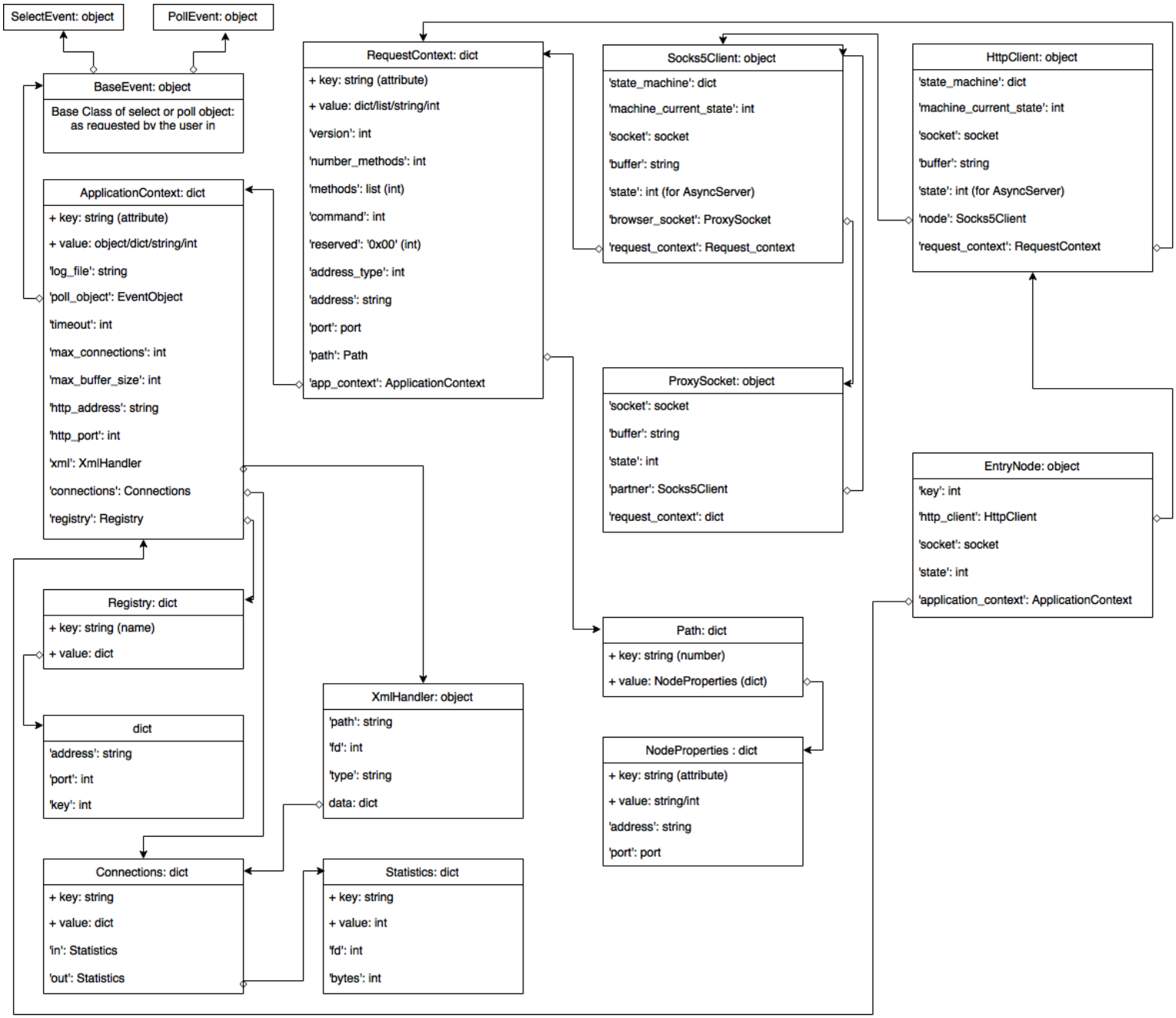
אופן פעולה

- המערכת פותחת אובייקט מסוג EntryNode ו-HttpClient ומוסיפה אותם ל-Poller.
- HttpClient שולח ל-Registry בקשת HTTP בשימוש בשירות ההתחברות של צומת חדש. במקרה של שגיאה המערכת נסגרת.
- צומת הכניסה מאזין בפורט 1080.
- עם קבלת חיבור שולח EntryNode שולח בקשה ל-Registry באמצעות HttpClient לקבלת כל הצמתים המחוברים כעת למערכת.
- צומת הכניסה בוחר באקראי שלושה צמתים מבין הצמתים שהתקבלו.
- צומת הכניסה פותח שני אובייקטים ומוסיף אותם ל-Poller:
  - אובייקט מסוג Socks5Client המשמש להתחברות לשאר הצמתים אשר מתחבר לשאר הצמתים ע"י חיבור socks5 (אופן ההתחברות תוסבר תחת פרוטוקולים), ולאחר מכן פועל כProxy בין ProxySocket והצמתים.
  - אובייקט מסוג ProxySocket שמקבל את החיבור שהתקבל מהלקוח ל-Socks5Client לאחר שזה סיים את תהליך ההתחברות לשאר הצמתים.
- עם סגירת התכנית HttpClient שולח ל-Registry בקשת התנתקות של הצומת ולאחר מכן מתבצעת סגירה של כל התכנית.

דיאגרמת בלוקים



דיאגרמת נתונים



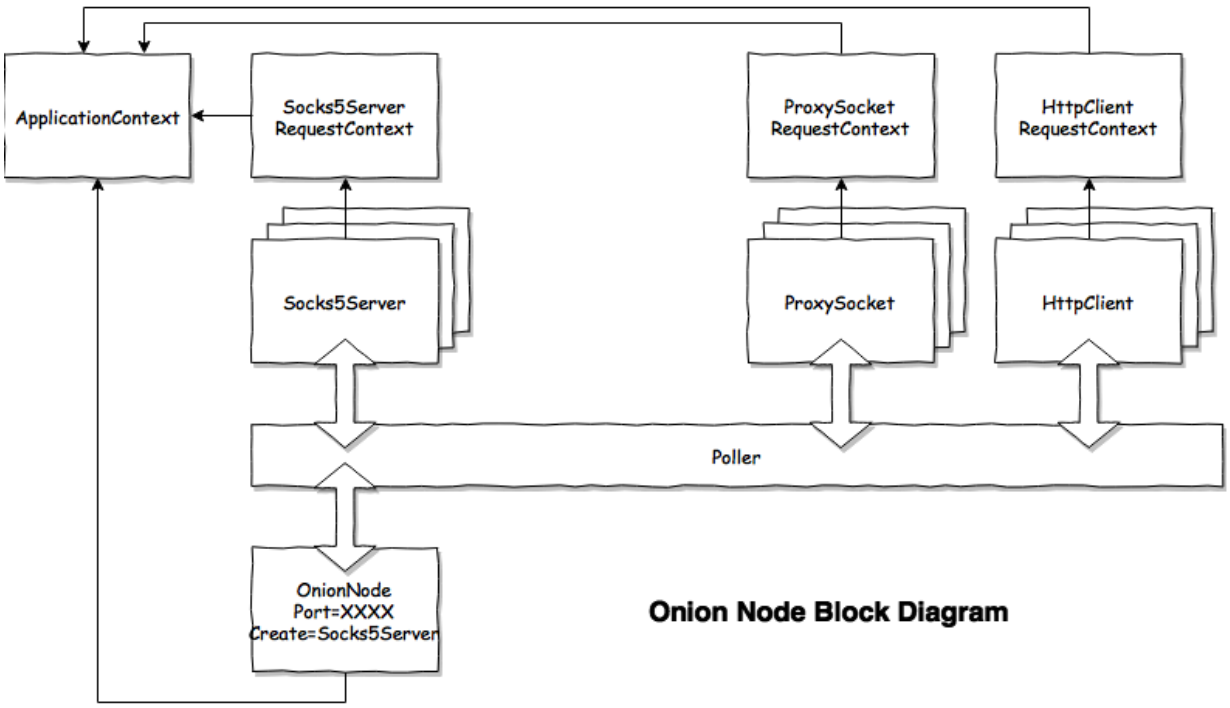
צומת בצל - Onion Node

צומת הכניסה הוא שרת מבוסס על פרוטוקול socks5 שמטרתו היא לקבל חיבור מהדפדפן (הלקוח) ולהעביר את המידע שנשלח דרך סדרה של צמתים אחרים על מנת שההודעה תשלח בעילום שם.
הצומת פועל כלקוח socks5 על מנת להתחבר ל-socks5 של שאר הצמתים ולאחר שהצומת התחבר לכל שאר הצמתים הוא פועל כשרת פרוקסי רגיל להעברת המידע שנשלח מהדפדפן.

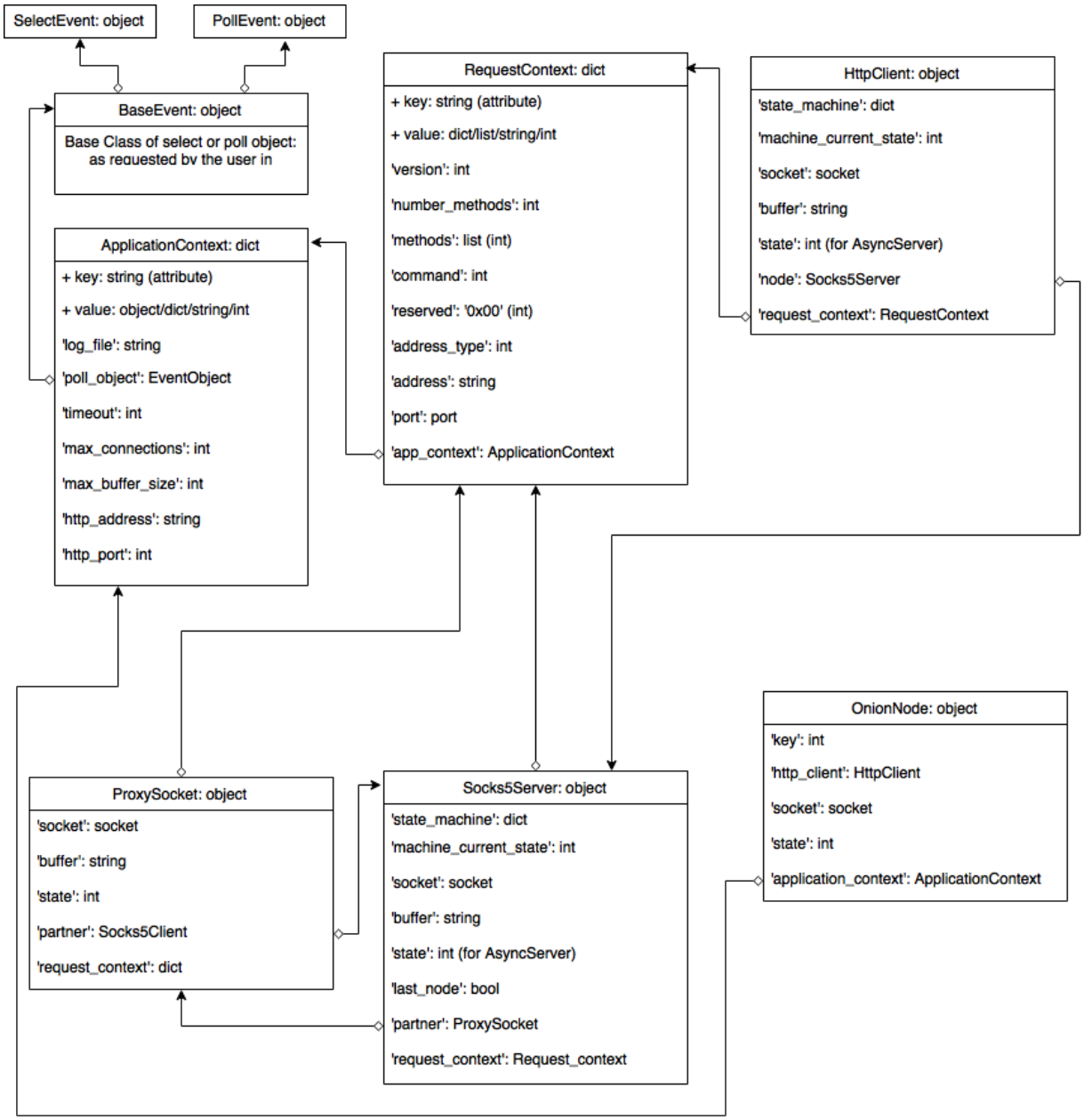
אופן פעולה

- המערכת פותחת אובייקט מסוג OnionNode ו-HttpClient ומוסיפה אותם ל-Poller.
- HttpClient שולח ל-Registry בקשת HTTP בשימוש בשירות ההתחברות של צומת חדש. במקרה של שגיאה המערכת נסגרת.
- הצומת מאזין בפורט בפורט מסוים שהוזן בפתיחתו (יוסבר תחת התקנה ותפעול).
- עם קבלת חיבור הצומת פותח אובייקט מסוג Socks5Server ומוסיף אותו ל-Poller. אובייקט זה משמש כשרת socks5 המקבל בקשה מלקוח להתחברות ליעד כלשהו, ולאחר השלמת החיבור מתפקד כ-Proxy בין הלקוח והיעד.
- עם סגירת התכנית HttpClient שולח ל-Registry בקשת התנתקות של הצומת ולאחר מכן מתבצעת סגירה של כל התכנית.

דיאגרמת בלוקים



דיאגרמת נתונים



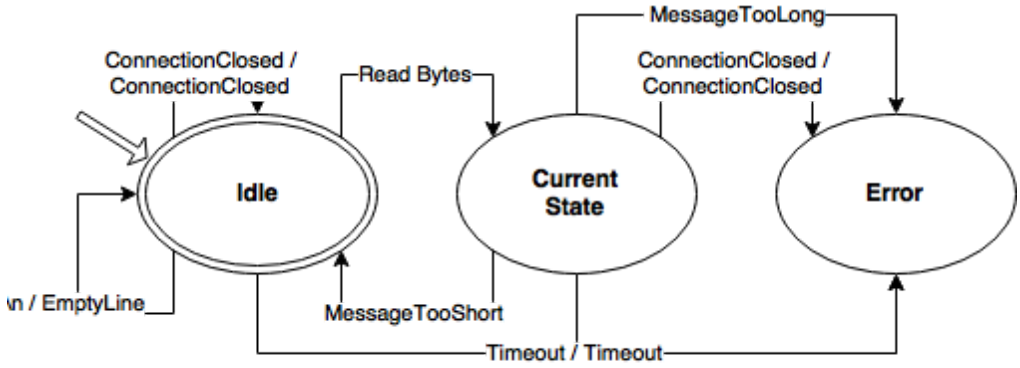
פרוטוקולי תקשורת

התקשורת בין המרכיבים השונים בתכנית מבוססת על פרוטוקולי תקשורת קיימים. הפרויקט מממש אפשרויות לשימוש בפרוטוקולים אלו בשיטות שיוסברו לעיל:

פרוטוקול Socks5

התקשורת בין הדפדפן והצמתים השונים מתבצעת באמצעות פרוטוקול Socks5.

כל בקשה בפרוטוקול (עליהן יורחב בהמשך) שמתקבלת ישנו צורך בהתייחסות שונה מצד הן השרת והן הלקוח של socks5. מכונת המצבים הבאה מתארת את התהליך לפיו מפוענחת כל הודעה המתקבלת בין הצדדים השונים:



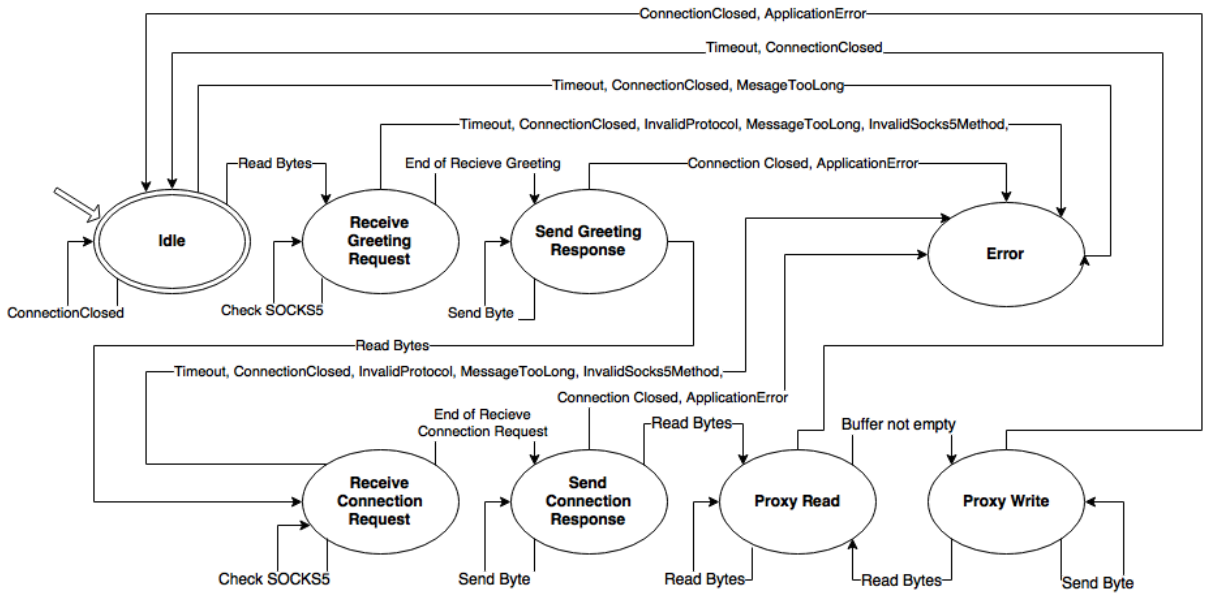
Check SOCKS5 State Machine

בפרויקט ישנו מימוש הן לשרת והן ללקוח של socks5, ולכן עלינו להפריד את שני המקרים.

שרת socks5 (מימוש ב-Socks5Server)

על מנת לבצע ניתוב בצל בין נתבי הבצל השונים כל אחד מהם פועל כשרת socks5. השרת מקבל בקשות socks5 ומתחבר ליעד של לקוח מסוים. לאחר החיבור השרת הופך לשרת פרוקסי ומנתב הודעות בין שני הקצוות המחוברים.

להלן מכונת מצבים עבור שרת socks5 הנותנת אפשרות לטיפול אסינכרוני בבקשת socks5 המתקבלת מלקוח.



SOCKS5 Server State Machine

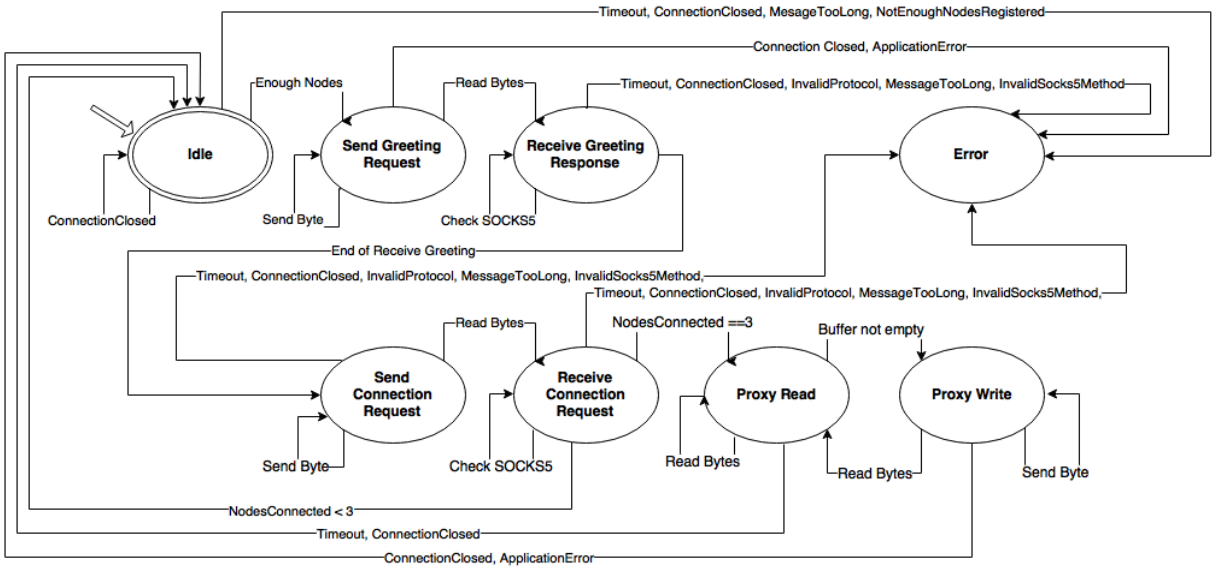
<table>
<tr><th>מצב</th><th>הסבר</th></tr>
<tr><td>Idle</td><td>שלב המתנה לבקשה.</td></tr>
<tr><td>Receive Greeting Request</td><td>שלב קבלת הודעה מסוג greeting request, הודעת החיבור הראשונית, וקבלת השיטה הרצויה לפתיחת החיבור (בפרויקט זה רק שיטת חיבור רגילה נתמכת).
מבנה:
<table><tr><th>version</th><th>Number of methods</th><th>methods</th></tr><tr><td>1</td><td>1</td><td>1 to 255</td></tr></table></td></tr>
<tr><td>Send Greeting Response</td><td>שלב שליחת תגובה מסוג greeting response, בחירת שיטת החיבור הרצויה (או התנתקות).
מבנה:</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>version</td><td>method</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>Receive Connection Request</td><td>שלב קבלת הודעה מסוג connection request, הודעה המכילה את פרטי היעד אליו הלקוח רוצה להתחבר. בשלב זה מנסה השרת לפתוח חיבור ליעד.
מבנה:
<table>
<tr><td>version</td><td>command</td><td>reserved</td><td>address type</td><td>dst address</td><td>dst port</td></tr>
<tr><td>1</td><td>1</td><td>"0x00"</td><td>1</td><td>Veriable size not) (fixed</td><td>2</td></tr>
</table>
</td></tr>
<tr><td>Send Connection Response</td><td>שלב שליחת תגובה מסוג connection reponse. ההודעה מכילה את פרטי היעד אליו השרת ניסה להתחבר וסטטוס ההצלחה במשימה.
מבנה:
<table>
<tr><td>version</td><td>reply</td><td>reserved</td><td>address type</td><td>dst address</td><td>dst port</td></tr>
<tr><td>1</td><td>1</td><td>"0x00"</td><td>1</td><td>Veriable</td><td>2</td></tr>
</table>
</td></tr>
<tr><td>Proxy Read</td><td>שלב הקריאה כשרת פרוקסי. השרת מעביר מידע מהלקוח לשרת.</td></tr>
<tr><td>Proxy Write</td><td>שלב הכתיבה כשרת פרוקסי. השרת מעביר מידע מהיעד ללקוח.</td></tr>
<tr><td>Error</td><td>במידה באחד מהשלבים שתוארו התקבלה שגיאה בשרת, השרת מנתק את החיבור.</td></tr>
</table>

לקוח socks5 (מימוש ב-Socks5Client)

על מנת לבצע חיבור בין הדפדפן לבין יעדו דרך רשת בצל עלינו להעביר את החיבור של הדפדפן דרך סדרה של צמתים לפני התייחסות להודעת ה-socks5 של הדפדפן עצמו. לשם כך נוצר לקוח socks5 אשר בו משתמש צומת הכניסה ויוצר שלושה חיבורי socks5 עם הצמתים האחרים עד שלבסוף מנתב כשרת פרוקסי את הודעות הדפדפן לשאר הצמתים.

להלן מכונת מצבים עבור לקוח socks5 הנותנת אפשרות התחברות אסינכרוני ב-socks5 לשרת.



SOCKS5 Client State Machine

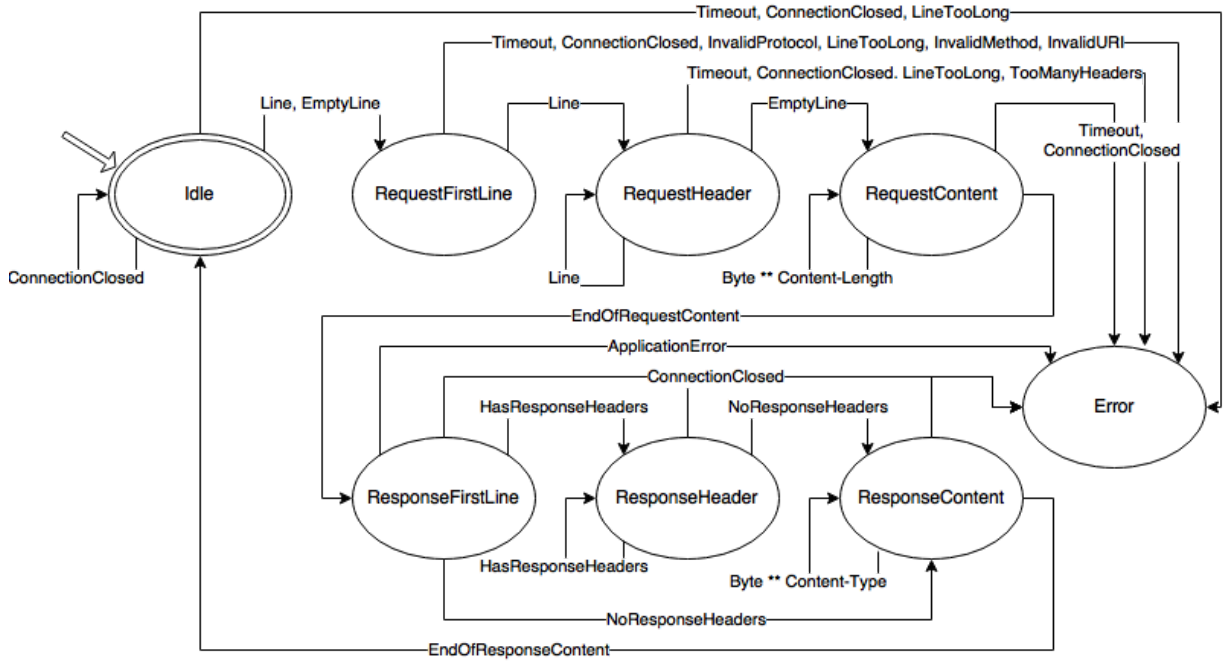
מצב	הסבר
Idle	שלב המתנה לבקשה מדפדפן. זהו גם השלב אליו חוזר הלקוח עד שהוא לא התחבר לכל שלושת הצמתים הרצויים.
Send Greeting Request	שלב שליחת בקשת greeting request לצומת הרצוי.
Receive Greeting Response	שלב קבלת תגובה מסוג greeting response לצומת הרצוי.
Send Connection	שלב שליחת בקשת connection request. ההודעה מכילה את פרטי היעד

Request	של הצומת הבא.
Receive Connection Response	שלב קבלת תגובה מסוג connection reponse, הודעה המכילה את פרטי היעד של הצומת הבא, וסטטוס הצלחה. בשלב זה הלקוח חוזר לתחילת התהליך אם הוא עדין לא התחבר לשלושת הצמתים. במידה וכל החיבורים נעשו, הלקוח עובר למצב פרוקסי.
Proxy Read	שלב הקריאה כשרת פרוקסי. הלקוח מעביר מידע מהדפדפן לצומת הראשון.
Proxy Write	שלב הכתיבה כשרת פרוקסי. הלקוח מעביר מידע מהצומת הראשון לדפדפן.
Error	במידה באחד מהשלבים שתוארו התקבלה שגיאה בשרת, השרת מנתק את החיבור עם השרת ואת החיבור עם הדפדפן.

פרוטוקול HTTP

התקשורת בין צומת הכניסה ל-Registry והשימוש בממשק המשתמש מתבצעת באמצעות פרוטוקול HTTP.

על מנת ששרת ה-Registry יעבוד בצורה מהירה ואסינכרונית בעת קבלת בקשות GET ישנם מספר שלבים אשר מבצע השרת המפורטים במכונת המצבים הנ"ל:

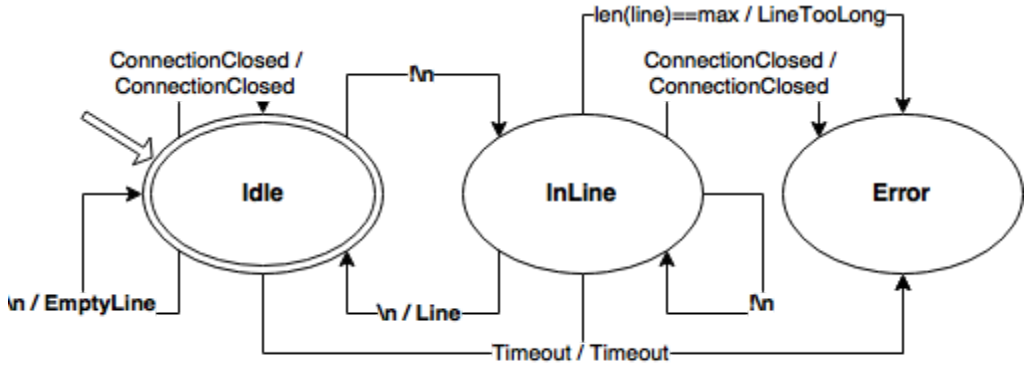


HTTP State Machine

מצב	הסבר
Idle	שלב המתנה לבקשה.
Request First Line	שלב קבלת שורת הסטטוס. השרת מקבל את שם השירות המבוקש.
Request Header	שלב קריאת ה-Headers. מכילים מידע אודות סוג ואורך ההודעה המתקבלת ועליהם מבוסס הפרוטוקול.
Request Content	שלב קריאת תוכן ההודעה.
Response First	שלב שליחת שורת הסטטוס של התגובה . השרת מודיע ללקוח על כישלון או

Line	הצלחה בביצוע השירות.
Response Headers	שלב שליחת ה-Headers. השרת שולח ללקוח מידע אודות התגובה.
Response Content	שלב שליחת תוכן התגובה (עבור ממשק המשתמש, תוכן התגובה יהיה דף HTML המכיל את הדף הרצוי).
Error	במידה באחד מהשלבים שתוארו התקבלה שגיאה בשרת, השרת מנתק את החיבור.

כל שורה בפרוטוקול HTTP נגמרת בשורה חדשה ('r/n/'). לכן קבלת התוכן בפרוטוקול התבצע על ידי קריאת שורות מלאות, על מנת לטפל בהן לאחר מכן. להלן מכונת מצבים אשר מתארת את תהליך קריאת השורה:



Get Line State Machine

בשלב קבלת שורת הסטטוס, השרת מקבל את שם השירות ויוצר אובייקט שירות מתאים מתוך רשימת השירותים הקיימים. בכל אחד מהמצבים המתוארים, השרת קורא לפעולות מתאימות מתוך השירות אשר מאפשרות טיפול אסינכרוני בבקשה.

בעיות ידועות

הבעיה העיקרית בפרויקט היא בכך שלאובייקט select **ישנה מגבלה לפתיחת sockets** ל-256.

כמו כן, סביבת ההרצה של הפרויקט, cygwin, מציבה רף גם היא ל-3172 לכמות ה-sockets שניתן לפתוח.

בשני המקרים מתקבלות השגיאות הבאות המקריסות את המערכת:

```
CRITICAL:root:Traceback (most recent call last):
  File "common/async/async_server.py", line 121, in run
    for fd, event in self._create_poller().poll(self._timeout):
  File "common/async/event_object.py", line 111, in poll
    r, w, x = select.select(rlist, wlist, xlist, timeout)
ValueError: filedescriptor out of range in select()
```

```
CRITICAL:root:Traceback (most recent call last):
  File "common/async/async_server.py", line 121, in run
    for fd, event in self._create_poller().poll(self._timeout):
  File "common/async/event_object.py", line 69, in poll
    return self._poller.poll(timeout)
error: (14, 'Bad address')
```

בשיטה עליה מבוסס פרויקט זה, שימוש בספריית select של פייטון, אין פתרון לבעיה כרגע. פתרון אפשרי הוא הרצת התכנית עם select. עם מעבר הכמות המקסימלית של החיבורים יצירת תהליך, thread, חדש אשר יוצר אובייקט select נוסף. כך ניתן יהיה לטפל בחיבורים נוספים האובייקט הנוסף מבלי לעלות על רף החיבורים באף אחד מהם. עם זאת, לא נעשה מחקר וניסיון לתקן את הבעיה הזו. (הבעיה הזו משפיעה לרוב במקרים בהם מנסים להתחבר מהדפדפן למספר אתרים כבדים בו זמנית, לדוגמה CNN ו-Ynet).

מגבלה נוספת היא **אי התמיכה של הפרויקט במערכת ההפעלה Windows**, אלא בסביבת Unix בלבד. הוספת תמיכה עבור Windows דורשת הגדרה חלופית לכלל הספריות שאינן נתמכות ע"י Windows.

התקנה ותפעול

התקנה

על מנת להריץ את הפרויקט יש לבצע את השלבים הבאים:

1. הורידו את גרסת הפרויקט מתוך releases ב-github. קישור בסוף התיק, תחת הכותרת - קוד פרויקט.
2. הורידו את דפדפן אינטרנט מודרני.

ישנו צורך בשינוי הגדרות הפרוקסי של הדפדפן ולכן מומלץ להוריד את דפדפן Firefox, בו ניתן לקבוע את ההגדרות בפשטות ללא שינוי הגדרות הפרוקסי של המחשב.

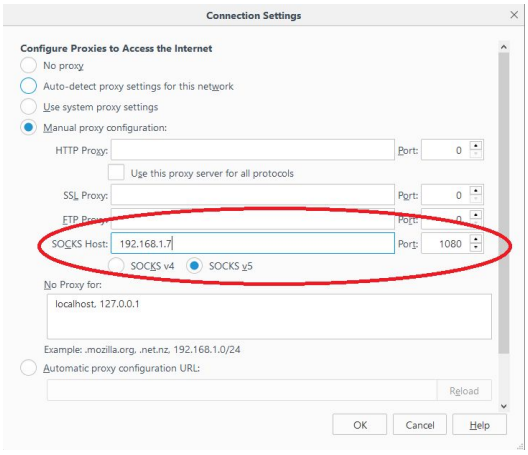
3. שנו את הגדרות הדפדפן כך שישתמש בפרוקסי בפרוטוקול socks5.

ההוראות הבאות מתאימות למשתמשים בדפדפן Firefox:

a. כנסו ל-Options.

b. לחצו על Advanced בסרגל הכלים בצד, וכנסו ללשונית Network.

c. תחת הכותרת Connections לחצו על Settings. הגדירו את שרת הפרוקסי בכתובת של המחשב שלכם ובפורט מסוים. ברירת המחדל עבור הפורט היא 1080. ניתן לראות את שינוי ההגדרות בדוגמה הנ"ל:



4. במידה ואתם משתמשים במערכת ההפעלה ווינדוס, הורידו והתקינו את סביבת העבודה Cygwin מהקישור הבא: http://cygwin.com/ (אין צורך לבצע שלב זה על מחשב הפועל על מערכת POSIX).
5. שינוי קבצי Config.ini

Config.ini - Registry

Section Name	Key Name	Value Type	Default Value	Explanation
Registry	bind.address	string	0.0.0.0	כתובת ה־IP שאליה מבצע השרת bind.
Registry	bind.port	int	8080	פורט ההאזנה של השרת.
xmlFile	path	string	files/statistics.xml	מיקום קובץ ה-xml המשמש לסטטיסטיקות.
nodesFile	path	string	files/nodes.xml	מיקום קובץ ה-xml לצפייה בצמתים המחוברים.

Config.ini - Entry Node

Section Name	Key Name	Value Type	Default Value	Explanation
Registry	bind.address	string	0.0.0.0	כתובת ה־IP שאליה מבצע השרת bind.
Registry	bind.port	int	8080	פורט ההאזנה של השרת.
EntryNode	bind.address	string	0.0.0.0	כתובת ה־IP שאליה מבצע צומת הכניסה bind.
EntryNode	bind.port	int	1080	פורט ההאזנה של צומת הכניסה.
xmlFile	path	string	files/statistics.xml	מיקום קובץ ה-xml המשמש לסטטיסטיקות.

Config.ini - Onion Node

Section Name	Key Name	Value Type	Default Value	Explanation
Registry	bind.address	string	0.0.0.0	כתובת ה־IP שאליה מבצע השרת bind.
Registry	bind.port	int	8888	פורט ההאזנה של השרת.
xmlFile	path	string	files/statistics.xml	מיקום קובץ ה-xml המשמש לסטטיסטיקות.

הרצת התכנית

על מנת להריץ את התכנית כנסו לתיקיית האב של הפרויקט דרך ה-command line (או Cygwin).

הרצת Registry:

```
/tmp/onion_routing
liron@DESKTOP-HQ47JFI /tmp/onion_routing
$ python -m registry
```

הרצת Entry Node - צומת הכניסה:

```
/tmp/onion_routing
liron@DESKTOP-HQ47JFI /tmp/onion_routing
$ python -m node_entry
```

הרצת Onion Node - צומת בצל:

```
/tmp/onion_routing
liron@DESKTOP-HQ47JFI /tmp/onion_routing
$ python -m node_server --bind-port 2080
```

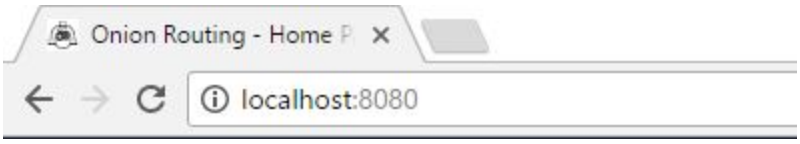
חשוב לציין! הרצת ה-Registry לפני הרצת הצמתים היא הכרחית, מכיוון שהצמתים פותחים חיבור ל-Registry.

גישה של לקוח למערכת

לאחר ביצוע ההוראות, ניתן להתחבר למערכת דרך הדפדפן עם הגדרות הפרוקסי בצורה רגילה, אם כי כעת הגלישה היא בעילום שם.

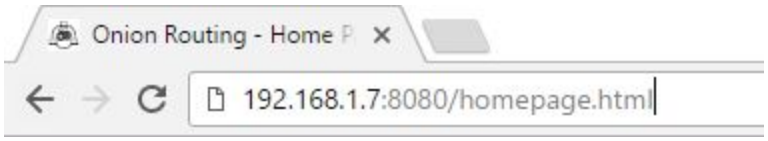
כמו כן, הלקוח יכול לגשת לממשק המשתמש ע"י ביצוע השלבים הבאים:

להיכנס לדפדפן כלשהו, ולכתוב:



אי הכנסה של שום דבר תעביר את הלקוח ישירות לדף הבית - homepage.html

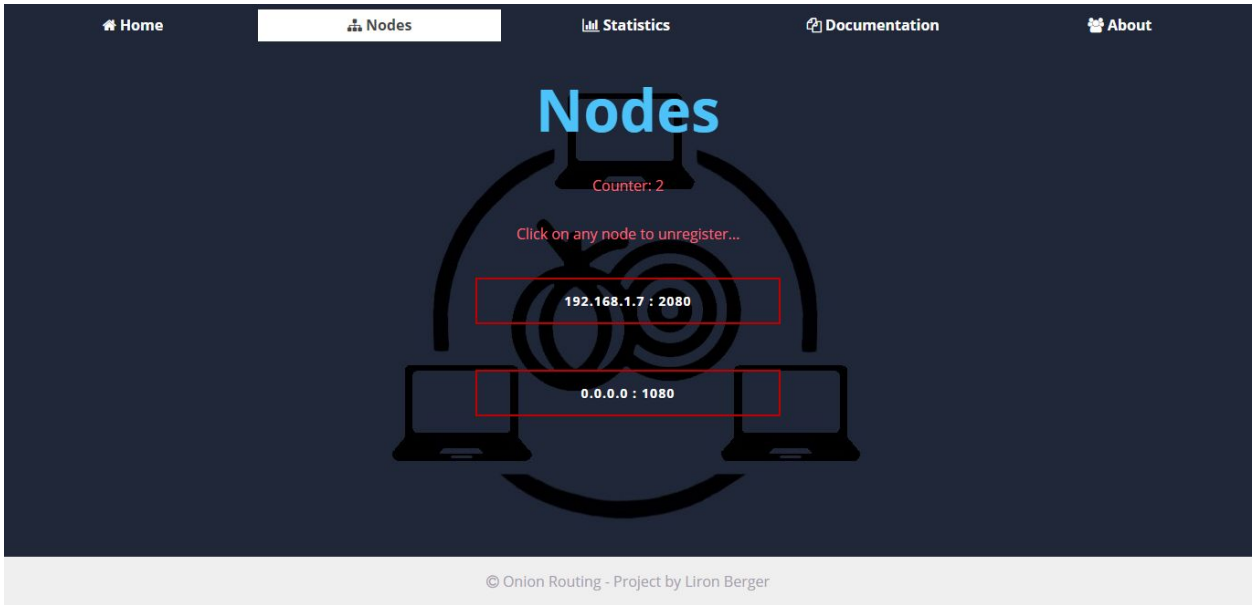
חשוב לציין! במידה ושיניתם את כתובת ה-bind עליכם להכניס אותה במקום הlocalhost כמו בדוגמה הבאה:



כעת הדפדפן יציג לכם את חלון הכניסה הבא:

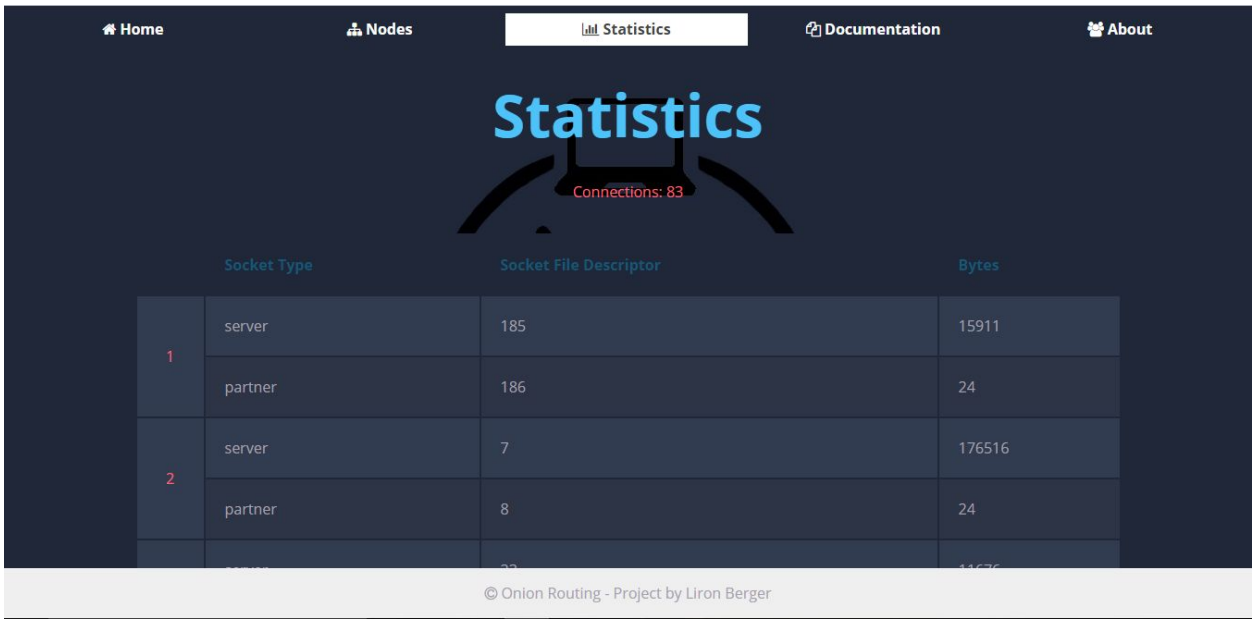
בסרגל הכלים בראשית הדף ישנן אפשרויות שונות שממשק המשתמש מספק עבור הלקוח.

אם תיכנסו ל-Nodes, יתקבל מידע אודות כלל הצמתים במערכת, כפי שנראה בדף הבא:



לחיצה על אחד הצמתים תפעיל את שירות ההתנתקות, ותוציא את הצומת מה-Registry.

אם תיכנסו ל-Statistics, תתקבל סטטיסטיקה אודות המידע העובר דרך צומת הכניסה, כפי שנראה בדף הבא:



תוכניות עתיד

כיום, פותחו דרכים לעקוב אחרי הודעות ב-Onion Routing, ולכן, על מנת להגדיל את הקושי בהשגת המידע ברשת הבצל היו לי מספר תכניות להוסיף לפרויקט, אשר לא הוספו מפאת קוצר הזמן:

שיפור ההצפנה

כעת, הפרויקט משתמש בהצפנה מבוססת על xor, בין המרכיבים השונים במערכת. הצפנה זו היא הצפנה בסיסית מאוד הקלה לפענוח, ומטרתה היה בעיקר לבניית התשתית לאלגוריתמים מסובכים יותר של הצפנה.

ישנם שני סוגים של הצפנה שניתן היה לממש:

- הצפנה סימטרית Symmetric Encryption - הצפנת בלוקים סימטרית נפוצה שנחשבת לבעלת יכולת אבטחה גבוהה. ההצפנה בשיטה זו מבוססת על מפתח מסוים ווקטור אתחול IV מוגדר מראש.

- הצפנה א-סימטרית Asymmetric Encryption - הצפנה המבוססת על שני מפתחות הידועים לבעליהם בלבד. הצפנה זאת מאפשרת אפשרות של authentication בנוסף על encryption וכך יוצרת רמת אבטחה גבוהה יותר.

תמיכה מורכבת במספר רב של צמתי כניסה

המערכת כפי שהיא מממשת כעת אינה תומכת במספר צמתי כניסה שיוכנסו ל-Registry. על מנת להוסיף מידע על צמתי הכניסה לפרויקט ניתן לבצע שימוש בload balancer.
באמצעות יצירת שרת שיפעל כ-load balancer, השרת יקבל את כלל הצמתים הקיימים מה-Registry, ויתחיל בניסיונות ליצירת חיבור עם כל אחד מהם. אם מתקבלת שגיאה הוא ממשיך בתהליך. כאשר שגיאה אינה מתקבלת משמעות הדבר היא שהשרת מצא EntryNode, אשר יכול לטפל בצורה נכונה בהודעה. בעת מציאת צומת הכניסה יהפוך השרת לשרת מסוג proxy, וינתב את כלל ההודעות אל צומת הכניסה שהנבחר. מכאן, תהליך ההתחברות יהיה זהה לתהליך המתואר לאורך הפרויקט.

פרק אישי

פרויקט זה נתן לי ראייה והתנסות ראשונית בעבודה על פרויקטים גדולים במחשבים. במהלך תהליך כתיבת הפרויקט נוכחתי שעלי לחקור וללמוד נושאים תאורטיים רבים, וכך העשרתי רבות את הידע שלי בעולם המחשבים. העבודה על הפרויקט פיתחה אותי במובנים רבים במסגרת לימודי פיתוח התכנה שלי, והקנתה לי אינספור מיומנויות בתחום.
אחד הדברים המרכזיים שלמדתי במהלך הפרויקט הוא שארגון, סדר ותכנון אפילו לפני תחילת הקידוד עצמו הם תהליכים חשובים ביותר. לא אחת נתקלתי במצבים בהם הקוד שלי בתחילת הפרויקט היה לא ברור ולא מסודר והקשה עלי במימוש דברים מסובכים יותר אשר התבסו עליו. מצבים אלו גרמו לי לעיקובים רבים, ואף למימוש חוזר של מחלקות מסוימות והן נבעו ישירות מאי הניסיון שלי בתכנון נכון

של עבודה על פרויקט במחשבים. הפרויקט לימד אותי כיצד לתכנן בצורה יעילה ונכונה את מימוש הפרויקט, וניתן לראות הבדל משמעותי וברור בין הפרויקט בתחילת דרכו ובסיומה מהבחינה הזו.
הקושי המרכזי שעלה לעתים קרובות במהלך הפרויקט הוא חיפוש אחר שגיאות בלתי צפויות בתכנית וגיבוג. בעיות מסוימות היווה מכשולים שנמשכו על גבי ימים ארוכים והיו מתסכלות ביותר. עם זאת, הבעיות נפתרו בסופו של דבר וכל אחת נתנה לי כלים חדשים ומסקנות כיצד להימנע מהן בעתיד.
אין ספק שללא התמיכה העזרה והתדרוך של שני מנחי העבודה שלי, שרית לולב ואלון בר לב, אשר אתם עברתי תהליך למידה ממושך שערך שלוש שנים, הפרויקט לא היה נמצא במצב שלו כיום.
העבודה על הפרויקט גרמה לי עניין רב בתחום מדעי המחשב, תקשורת, ופרטיות ברשת, והעלתה שאלות רבות אשר אני מקווה למצוא להן תשובה בהמשך דרכי בתחום.

קוד פרויקט

את קוד הפרויקט ניתן למצוא ב-releases במשתמש ה-github שלי, בקישור הבא:
https://github.com/Liron-Berger/Onion-Routing/releases
עליכם להוריד את ההפצה האחרונה של הפרויקט ולחלץ אותה מתוך קובץ ה-zip.

תיעוד פרויקט

את התיעוד לפרויקט ניתן למצוא גם כן ב-releases, בקובץ המצורף לפרויקט תחת השם - gen-doc.zip.
לצורך צפייה בתיעוד יש לחלץ את הקבצים מתוך קובץ ה-zip, ולהיכנס לקובץ index.html.

נספחים

נספח א' - Anonymization Sequences

לשימוש באמצעות: http://sequencediagram.org/

```
title Onion Routing: anonymization Sequences

actor Browser
database Registry
control Entry-Node
```

```
entity Onion-Node-1 #LightSteelBlue
entity Onion-Node-2 #LightSteelBlue
entity Onion-Node-3 #LightSteelBlue
boundary Destination

==Definitions== #LightSteelBlue

autonumber 1
note over Registry,Entry-Node #khaki:Gets all nodes from Registry and chooses three at random.
par def(SelectNodes())
    Registry<-Entry-Node:Get existing nodes
    Registry->Entry-Node:Returns all existing nodes
    Entry-Node->Entry-Node:Choose three nodes at random
end

==anonymization== #LightSteelBlue

note over Browser,Destination #khaki:Establishing Anonymized connection of Browser and
Destination using SOCKS5 protocol.
par def(CreatePath())
    Browser->Entry-Node:Open Socks5 Connection
    ref SelectNodes()
        note over Registry,Entry-Node #khaki:Chooses three random nodes from registry.
        Registry<-Entry-Node:
    end
    ref EstablishSocks5(Onion-Node-1 key, Onion-Node-2 address, Onion-Node-2 port)
        note over Entry-Node,Onion-Node-1 #khaki:Establishes SOCKS5 connection between
Entry-Node and Onion-Node-1\nusing Onion-Node-1 key for encryption. Connects to Onion-Node-2.
        Entry-Node->Onion-Node-1:
    end
    ref EstablishSocks5(Onion-Node-2 key, Onion-Node-3 address, Onion-Node-3 port)
        note over Entry-Node,Onion-Node-2 #khaki:Establishes SOCKS5 connection between
Entry-Node and Onion-Node-2\nusing Onion-Node-2 key for encryption. Connects to Onion-Node-3.
        Entry-Node->Onion-Node-2:
    end
    ref EstablishSocks5(Onion-Node-3 key, Destination addres, Destination)
        note over Browser,Destination #khaki:Redirects SOCKS5 connection of browser from
Entry-Node to\nOnion-Node-3 thus Establishes SOCKS5 connection between Browser
and\nOnion-Node-3 using Onion-Node-3 key for encryption. Connects to Destination.
        Browser->Onion-Node-3:
    end
note over Browser,Destination #khaki:Connection established. Encryption with Onion-Node-3 key.
end
```

נספח ב' - Generic Sequences

לשימוש באמצעות: http://sequencediagram.org/

```
title Onion Routing: Generic Utilities

database Registry
control Entry-Onion-Node
entity Onion-Node
boundary Destination

==Definitions== #LightSteelBlue
autonumber 1
par def(EstablishSocks5(Onion-Node key, Destination address, Destination port))
    note over Entry-Onion-Node,Destination #khaki:Establish connection between
Entry-Onion-Node and Destination using SOCKS5 protocol.\nOpens Socks connection to the
Onion-Node which redirects connection to Destination.\nCommunication is encrypted using
Onion-Node secret key.
    Entry-Onion-Node->Entry-Onion-Node:Encrypt(Greeting Request, Onion-Node key)
    Entry-Onion-Node->Onion-Node:Greeting Request
    Onion-Node->Onion-Node:Decrypt(Greeting Request, Onion-Node key)
    Onion-Node->Onion-Node:Encrypt(Greeting Response, Onion-Node key)
    Entry-Onion-Node<-Onion-Node:Greeting Response
    Entry-Onion-Node->Entry-Onion-Node:Decrypt(Greeting Response, Onion-Node key)
    Entry-Onion-Node->Entry-Onion-Node:Encrypt(Connection Request, Onion-Node key)
    Entry-Onion-Node->Onion-Node:Connection Request (Destination address, Destination port)
    Onion-Node->Onion-Node:Decrypt(Connection Request, Onion-Node key)
    Onion-Node->Onion-Node:Encrypt(Connection Response, Onion-Node key)
    Onion-Node->Destination:Connect(Destination address, Destination port)
    Entry-Onion-Node<-Onion-Node:Connection Response
end

==Initialization== #LightSteelBlue
autonumber 1
par Register()
    note over Registry,Onion-Node #khaki:Register Onion-Node into the Registry.\nSends to
registry name, address, port and secret key.
    Registry<-Onion-Node:Register HTTP request
    Registry->Registry:Add Onion-Node Onion-Node to registry
    Registry->Onion-Node:Register Response status
end

==Termination== #LightSteelBlue
```

```
autonumber 1
par Unregister()
    note over Registry,Onion-Node #khaki:Unregister Onion-Node from the Registry.\nSends the
name of Onion-Node and the Registry removes it from database.
    Registry<-Onion-Node:Unregister HTTP request
    Registry->Registry:Remove Onion-Node from registry
    Registry->Onion-Node:Unregister Response status
end

==Disconnect== #LightSteelBlue
autonumber 1

par DisconnectFromClient()
    parallel
        note over Entry-Onion-Node,Destination #khaki:Client Disconnects procedure.\nEnters
closing state for client and server.\nServer enters closing state for destination.\ncloses
sockets when socket is ready.
        space
        Entry-Onion-Node->Entry-Onion-Node:start closing procedure
        Onion-Node->Destination:start closing procedure
        Entry-Onion-Node->Onion-Node:start closing procedure
    parallel off
    space

    parallel
        Destination->Destination:close socekt
        Onion-Node->Onion-Node:close socket
        Entry-Onion-Node->Entry-Onion-Node:close socket
    parallel off
end

par DisconnectFromClient()
autonumber 1
    parallel
        note over Entry-Onion-Node,Destination #khaki:Client Disconnects procedure.\nEnters
closing state for client and server.\nServer enters closing state for destination.\ncloses
sockets when socket is ready.
        space
        Entry-Onion-Node<-Onion-Node:start closing procedure
        Onion-Node->Destination:start closing procedure
        Onion-Node->Onion-Node:start closing procedure
    parallel off
    space

    parallel
```

```
        Destination->Destination:close socekt
        Onion-Node->Onion-Node:close socket
        Entry-Onion-Node->Entry-Onion-Node:close socket
    parallel off
end
```