האוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

פרויקט מתקדם במדעי המחשב

**מימוש selfish miner ברשת ביטקוין**

פרויקט מתקדם זה הוגש כחלק מהדרישות לקבלת תואר

"מוסמך למדעים" M.Sc. במדעי המחשב

באוניברסיטה הפתוחה

החטיבה למדעי המחשב

מגיש: רועי דימינטשטיין, ת.ז. 302792361

מנחה: פרופ' אהוד גודס

תוכן

[מבוא 6](#_Toc529473909)

[התקפות ביטקוין ומיקוד הפרויקט 6](#_Toc529473910)

[מטרת הפרויקט 8](#_Toc529473911)

[סוגיות מימוש והיקף העבודה 8](#_Toc529473912)

[עיצוב המערכת 9](#_Toc529473913)

[מפרט דרישות תוכנה (SRS) 9](#_Toc529473914)

[1. מבוא 9](#_Toc529473915)

[2. תיאור כללי 9](#_Toc529473916)

[3. דרישות פונקציונאליות 11](#_Toc529473917)

[Use Cases 13](#_Toc529473918)

[דיאגרמת ארכיטקטורה 16](#_Toc529473919)

[Class Diagram ותיאור הרכיבים 18](#_Toc529473920)

[הרכיב Blockchain 19](#_Toc529473921)

[הרכיב Wallet 24](#_Toc529473922)

[הרכיב Messages 24](#_Toc529473923)

[הרכיב Miners 25](#_Toc529473924)

[הרכיב Singletons 30](#_Toc529473925)

[פרטים טכניים בנוגע להרצת הסימולציה 31](#_Toc529473926)

[ממשק המשתמש 31](#_Toc529473927)

[מיקום קבצי העבודה והוראות הפעלה 34](#_Toc529473928)

[ביצוע מדידות 35](#_Toc529473929)

[ערכים הנמדדים 35](#_Toc529473930)

[selfishLeadRevenueRatio 35](#_Toc529473931)

[selfishConnectivityRatio 35](#_Toc529473932)

[פורמט קבצי התוצאות 36](#_Toc529473933)

[מוטיבציה לבחירת הערכים והמדידות 37](#_Toc529473934)

[תוצאות 38](#_Toc529473935)

[מספרים כלליים 38](#_Toc529473936)

[מקרי קיצון 39](#_Toc529473937)

[תוצאות סיכומיות 42](#_Toc529473938)

[נספחים 44](#_Toc529473939)

[קוד ראשי של MinerNode 44](#_Toc529473940)

[קוד ראשי של GreedyMinerNode 45](#_Toc529473941)

[דוגמא לקבצי פלט עבור הרצה אחת 45](#_Toc529473942)

[סיכום 46](#_Toc529473943)

[רשימת מקורות 47](#_Toc529473944)

רשימת איורים

[**איור 1 - פסאודו קוד של selfish miner** 7](#_Toc529473813)

[**איור 2 - context diagram של המערכת** 10](#_Toc529473814)

[**איור 3 - Use Cases Diagram** 13](#_Toc529473815)

[**איור 4 - state transition diagram של מיינר** 15](#_Toc529473816)

[**איור 5 - דיאגרמת ארכיטקטורה של המערכת** 16](#_Toc529473817)

[**איור 6 - Class Diagram כללי** 18](#_Toc529473818)

[**איור 7 - Blockchain Class Diagram** 19](#_Toc529473819)

[**איור 8 - TransactionInput class** 20](#_Toc529473820)

[**איור 9 - TransactionOutput Class** 20](#_Toc529473821)

[**איור 10 - Transaction Class** 21](#_Toc529473822)

[**איור 11 - Block Class** 22](#_Toc529473823)

[**איור 12 - Blockchain Class** 23](#_Toc529473824)

[**איור 13 - Wallet Class** 24](#_Toc529473825)

[**איור 14 - Node Class** 25](#_Toc529473826)

[**איור 15 - MinerNode Class** 25](#_Toc529473827)

[**איור 16 - Miner Flowchart** 27](file:///C%3A/Users/Roei/Desktop/%D7%AA%D7%95%D7%90%D7%A8/%D7%A4%D7%A8%D7%95%D7%99%D7%A7%D7%98/Advance%20Project%20-%20Selfish%20Mining%20Attack%20on%20Bitcoin.docx#_Toc529473828)

[**איור 17 - FlowChart של handleBlockMessage עבור מיינר רגיל** 28](#_Toc529473829)

[**איור 18 - GreedyMinerNode Class** 28](#_Toc529473830)

[**איור 19 - Flowchart של handleBlockMessage לפי selfish mining** 29](file:///C%3A/Users/Roei/Desktop/%D7%AA%D7%95%D7%90%D7%A8/%D7%A4%D7%A8%D7%95%D7%99%D7%A7%D7%98/Advance%20Project%20-%20Selfish%20Mining%20Attack%20on%20Bitcoin.docx#_Toc529473831)

[**איור 20 - מסך כניסה למערכת** 31](#_Toc529473832)

[**איור 21 - מסך הכנסת מספר צמתים ברשת** 32](#_Toc529473833)

[**איור 22 - רשת לדוגמא** 32](#_Toc529473834)

[**איור 23 - דוגמא להצגת מצבו של צומת** 33](#_Toc529473835)

[**איור 24 - רווחי selfish miners כתלות במספר סיבובים** 39](#_Toc529473836)

[**איור 25 - selfishLeadRevenueRation כתלות במספר הסיבובים** 40](#_Toc529473837)

[**איור 26 - רווחי ה selfish miners בריצה** 40](#_Toc529473838)

[**איור 27 - רווחי המיינרים התמימים בריצה** 41](#_Toc529473839)

[**איור 28 - רווחים בפועל אל מול רווחים תאורטיים של מיינר selfish** 41](#_Toc529473840)

[**איור 29 - רווחי מיינרים selfish אל מול תמימים** 42](#_Toc529473841)

[**איור 30 - רווחי הselfish miners כתלות בחלקם היחסי** 42](#_Toc529473842)

[**איור 31 - selfishLeadRevenueRatio כתלות ב selfishRatio** 43](#_Toc529473843)

[**איור 32 - selfishLeadRevenueRatio כתלות ב selfishConnectivityRatio** 43](#_Toc529473844)

[**איור 33 - דוגמא לקובץ פלט 1** 45](#_Toc529473845)

[**איור 34 - דוגמא לקובץ פלט 2** 45](#_Toc529473846)

[**איור 35 - דוגמא לקובץ פלט 3** 46](#_Toc529473847)

# מבוא

## התקפות ביטקוין ומיקוד הפרויקט

רשת ביטקוין היא מערכת מבוזרת שמטרתה לממש מטבע אלקטרוני שהסחר בו יעשה באופן שהוא לגמרי decentralized, זאת אומרת, ללא trusted third party כמו בנק.

בעבודה המסכמת תואר לפרטים הפרוטוקול, האלגוריתמים וכלים נוספים כמו הארנק המרכיבים את ביטקוין. בנוסף, תוארו מספר משפחות של התקפות על הרשת:

1. **התקפות דה-אנונימיזציה** - התקפות במסגרתן התוקף מצליח לגלות מזהים כלשהם, מחוץ לדומיין של ביטקוין, לגבי משתמש ביטקוין או פעולות שמשתמש ביטקוין ביצע. זהותם של משתמשי ביטקוין אמורה להיות "מוסתרת" מאחורי pseudonym בצורת מפתח פומבי שבו הם השתמשו על מנת לבצע פעולה במערכת
2. **התקפות eclipse** - התקפות במסגרתן התוקף מצליח להשיג שליטה על החיבורים הרשתיים הנכנסים והיוצאים של צומת קורבן ברשת ביטקוין. תוקף במצב זה יכול לגרום לקורבן "להאמין" במה שהוא רוצה - הוא בעצם מנתק אותו מהרשת ו"מספר" לו דברים באופן סלקטיבי
3. **התקפות selfish mining** - התקפות במסגרתן, באמצעות סטייה מהפרוטוקול, מיינר ברשת עשוי להרוויח יותר מכוח החישוב היחסי אותו הוא משקיע. רווח זה ינתן למיינר הסורר על חשבון המיינרים האחרים מאחר וכריית ביטקוין היא משחק סכום אפס. פרוטוקול ביטקוין אמור להיות incentive-compatible בתאוריה (מכניזם שבמסגרתו הכי רווחי למשתתפים לעקוב אחרי הפרוטוקול כדי למקסם רווחים) אבל התקפות אלה מציגות תוצאות (תאורטיות) שמצביעות אחרת

בפרויקט זה התמקדתי בהתקפה מהסוג השלישי - selfish mining.

selfish mining היא התקפה שהוצגה על ידי איתי אייל, היום פרופסור בפקולטה להנדסת חשמל בטכניון, בנובמבר 2013. התקפה גנרית ברמת הפרוטוקול לביצוע selfish mining ועד היום משמשת בקהילת ביטקוין כרפרנס כמעט היחיד לביצוע התקפה מהסוג הזה. מאמרים נוספים פורסמו המפתחים את השיטה בצורות יותר מתוחכמות (אחת מהן המשלבת eclipse attack יחד עם selfish mining הוצגה בעבודה המסכמת).

לא ניכנס כאן לפרטים מאחר וההתקפה כבר תוארה לעומק בעבודה המסכמת. המוטיבציה הכללית היא - הפרוטוקול המקורי מורה למיינר המוצא בלוק חדש לפרסם אותו מיד על מנת לזכות בפרס לפני שמיינר אחר ימצא בלוק שיחליף אותו. ההתקפה "משחקת" עם הרעיון של לא לספר לכולם מיד. במקום זה, מספר מיינרים תוקפים, ינסו לתחזק שרשרת פרטית, הם לא יספרו על בלוקים שהם מוצאים לשאר המיינרים. בהנחה ושאר המיינרים ברשת מצאו n בלוקים אבל באותו זמן המיינרים התוקפים יצליחו למצוא k > n בלוקים אז המיינרים התוקפים יוכלו לפרסם את k הבלוקים האלה ומאחר ו k > n, כל המיינרים (גם התמימים) יסכימו לקבל את השרשרת שלהם. מצב זה גרם לכל המיינרים התמימים לעבוד על n בלוקים שפשוט נזרקו לפח.

אינטואיטיבית המצב הזה לא אמור לקרות כי כוח החישוב של התמימים גדול בהרבה משל התוקפים אז למה שהתוקפים יצליחו להשיג אותם ולכרות k בלוקים בזמן שכל השאר הצליחו לכרות n?

עפ"י אלגוריתם selfish mining, באמת לא מחכים "יותר מידי זמן" ברגע שבמקרה המיינרים התוקפים הצליחו להשיג יתרון גדול מספיק, הם מפרסמים את הבלוקים שמצאו.

הפסאודו קוד המוצג באיור 1 מתאר את ההתקפה (קוד זה רץ על כל צומת תוקף) -



**איור 1 - פסאודו קוד של selfish miner**

ניתן לראות את הגישה ה"דאגנית" של האלגוריתם - ברגע שה"אחרים" מצאו בלוק והיתרון הצטמצם ל2 בלוקים בלבד, מפרסמים את השרשרת הפרטית. כמו כן, כל פעם שמאבדים את היתרון לגמרי מקבלים את השרשרת הפומבית של השאר כשרשרת עליה כורים.

## מטרת הפרויקט

במסגרת המאמרים שנסקרו בעבודה המסכמת הוצגו הוכחות תאורטיות לנכונות האלגוריתמים והוצגו סימולציות המתבססות על המתמטיקה אליה הגיעו. במילים פשוטות יותר, מרבית האלגוריתמים ובאופן ספציפי, selfish mining, לא נבחן אמפירית. במקום זאת, הוצג מודל מתמטי המתאר את המערכת וניתוח הסתברותי של הצלחה.

מטרת פרויקט זה היא לממש את אלגוריתם selfish mining ולבחון את נכונותו

## סוגיות מימוש והיקף העבודה

ברמת המימוש, ביטקוין הוא פרויקט open source שהוקם ע"י satoshi (ממציא ביטקוין) ומתוחזק ע"י הקהילה מאז. לפרויקט קוראים bitcoin core והוא מממש תהליך שנקרא bitcoind המתפקד כמיינר כאשר הוא מותקן על מחשב המחובר לרשת. פרויקט זה הוא מימוש המיינר המרכזי, 99% מהמיינרים מריצים אותו. מספר עובדות לגבי bitcoin core:

1. מאחר והפרויקט צמח bottom-up באופן מאוד sketchy, אין שום מקור המציג design של המערכת, תיאור הרכיבים באופן מסודר וכו. באופן כללי בקהילה כאשר מפתח מתחיל מבקש כיוון, נהוג להפנות אותו אל ה source code
2. הקוד עצמו קשה מאוד לקריאה. לדוגמא, קובץ main מכיל כ 6000 שורות קוד השייכות לאלמנטים שונים ומגוונים במערכת
3. התהליך דורש המון משאבים. על מנת להריץ תהליך אחד של bitcoind יש צורך ב2GB זכרון RAM וכ 150GB זכרון דיסק פנוי שגם הוא חייב להיות SSD. וזה עוד לפני שמדברים על תעבורה. לכן, לא כל כך ריאלי להריץ מספר תהליכים כאלה על מחשב אחד

לכן, לאחר שחקרתי את bitcoin core, הגעתי למסקנה שלמרות ששימוש בו יחסוך חלקי מימוש שונים, הוא אינו מתאים לצרכים אקדמאיים ולפרויקט הזה.

המסקנה הייתה, מימוש מיינר ביטקוין מלא מאפס המבצע את כל הפעולות הבסיסיות שמיינר מבצע. מאחר ומדובר על צומת מלא, מומש גם ארנק וכך ניתן לבצע יצירת טרנזקציות, חתימה על טרנזקציות, שליחתן ברשת, תחזוקה של קונצנזוס מבוזר באמצעות בלוקצ'יין, ולידציה של טרנזקציות מול בלוקצ'יין, כריית בלוקים וכו.

כמובן שבהתאם פותח selfish miner המתבסס על המיינר הרגיל ודומה לו ברוב פעולותיו מלבד תהליך הכרייה.

לצורך הרצת סימולציות נוחה, את המימוש כתבתי מעל sinalgo שהיא framework להרצת סימולציות מבוזרות הכתובה ב java. אין זה משנה את המימוש או את תוכן הפרויקט. הרעיון הוא שלאחר מימוש המיינר, הקוד נכתב בתור מחלקה היורשת ממחלקת Node של sinalgo על מנת לאפשר לנו ליצור גרפים מצמתים אלה ולאפשר להם לשלוח הודעות אחד לשני. בהמשך יתואר עיצוב המערכת והמושגים האלה יובהרו.

# עיצוב המערכת

## מפרט דרישות תוכנה (SRS)

### מבוא

בחלק זה נתאר את המפרט של המערכת. המפרט מספק את הדברים הבאים:

* הגדרת סביבת העבודה של המערכת
* הגדרת יכולות המערכת
* ספסיפיקציה של דרישות המערכת הפונקציונאליות והלא פונקציונאליות
* דרישות ממשק משתמש

המערכת תתמוך בפעולות הבאות:

* יצירת גרף אקראי המורכב ממיינרים המממשים את פרוטוקול ביטקוין. יחד הם יהוו סימולציה לרשת
* הגדרת מספר המיינרים ה selfish מביניהם
* הרצת הסימולציה למשך מספר סיבובים מבוקש
* אפשרות לעצירת הסימולציה ו/או המשך למספר סיבובים
* הצגת סטאטוס רווחי ביטקוין נוכחיים בכל שלב
* קבלת מידע סיכומי לגבי הסטאטוס הנוכחי ברשת המסומלצת
* מחיקת גרף ויצירת גרף חדש

המערכת מפותחת על מנת לבדוק את ביצועי התקפת selfish mining על רשת ביטקוין מסומלצת.

למפרט זה מצורף תרשים ERD של המערכת כנספח.

המבנה הכללי של המפרט הוא:

1. תיאור כללי - תיאור של הסביבה הנדרש לצורך הפעלת האפליקציה
2. דרישות מפורטות - תיאור מפורט של דרישות פונקציונאליות

### תיאור כללי

מבט על

המערכת הינה מערכת עצמאית שאינה תלויה ואינה מחוברת למערכת חיצונית. למרות שבאמצעות תוספות ושינויים קטנים במחלקת ה Miner ניתן יהיה לממשק אותה אל רשת ביטקוין, כרגע מטרת המערכת היא לאפשר סימולציה של מספר מיינרים על אותו המחשב.

החומרה הנדרשת:

* בהנחה כי רוצים להריץ מספר רב של סיבובים (הבלוקצ'יין גדל עם הכרייה), 4GB עבור גרפים עד גודל של 7 צמתים ו 1GB נוסף עבור כל צומת נוסף
* כ 10MB לשמירת תוצאות מהרצות שונות

תוכנה נדרשת:

* JRE מעודכן (או JDK)

**Context Diagram**



**איור 2 - context diagram של המערכת**

המשתמש הוא היישות האקטיבית החיצונית היחידה, הדיסק הוא יישות חיצונית נוספת שלוקחת חלק כאשר המשתמש מבקש לכתוב תוצאות.

 פונקציות מערכת

1. יצירת גרף רשת
	1. מספר הצמתים מתקבל כקלט ומספר הצמתים מתוכם שהם selfish גם מתקבל כקלט
	2. ניתן לבקש ליצור גרף חדש וזה ימחוק את הקודם
2. סימלוץ הרשת למספר סיבובים
	1. מספר הסיבובים מתקבל כקלט
	2. ניתן לעצור הרצה באמצע גם במידה ולא הסתיימו מספר סיבובים שהתבקש
3. קבלת state של הרשת
	1. ניתן באמצעות כפתור לקבל את מאזן הביטקוין הכולל בין "הטובים" ל"רעים"
	2. ניתן באמצעות hover מעל אחד הצמתים לראות את מאזן הביטקוין שלו
		1. אם זה צומת selfish, ניתן לראות גם את מצב השרשרת הפרטית שלו ואת הרווח הפוטנציאלי שלו אם היה מפרסם אותה
	3. ניתן באמצעות כפתור לפלוט את הנתונים האלה ונתונים נוספים שיתוארו בהמשך ומטרתם לאפשר מדידת התוצאות

אילוצים

1. למרות שזאת סימולציה של צמתים מלאים, התוכנה עדיין memory consuming מאחר וקיים trade off. מאחר ואנחנו רוצים שיכרו די הרבה בלוקים כדי לבחון את ההתקפה, הבלוקצ'יין הולך ונעשה גדול כתלות במספר הצמתים, מספר הקשתות ומספר הסיבובים. בנוסף במקרים מסויימים צמתים יצטרכו למשוך אחד מהשני עותק מלא של הבלוקצ'יין וכאשר מגיעים לגדלים גדולים, זה עשוי להוביל למספר גדול מאוד של אובייקטי הודעה העוברים ברשת. לכן באופן תאורטי לא ניתן להריץ אינסוף סיבובים, לא משנה מה גודל הרשת המסומלצת (ניתן לחשוב על פתרון בעתיד בנוסח paging לדיסק ולהידמות שלב נוסף למיינר מלא..). בפועל, הצמדות לדרישות תאפשר לנו להריץ מספר מספק של סיבובים (מאות עד אלפים)

תלויות

1. לצורך ביצוע פעולות קריפטוגרפיות כמו חתימה דיגיטלית עם אלגוריתם Elliptic Curves או אפילו תיאור כתובת משתמש באמצעות Private-Public keypair, נעשה שימוש ב jar חיצוני פומבי בשם BouncyCastle המממש פונקציונאליות זאת

### 3. דרישות פונקציונאליות

1. יצירת גרף חדש

מטרה: יצירת רשת חדשה המכילה צמתים תמימים וצמתים selfish על גביה תוכל לרוץ הסימולציה. לכל צומת יש את האלגוריתם שהוא מריץ, מזהה ומצב המתאפיין בין היתר ברווחי הביטקוין הנוכחיים שלו והבלוקצ'יין אותו הוא מתחזק

קלטים:

* מספר הצמתים בגרף
* מספר הצמתים מתוכם שיהיו selfish

 תוצאת ההרצה:

נוצר גרף חדש, הצמתים ה selfish מסומנים באופן ברור למשתמש. אם היה גרף קודם, הוא נמחק

1. סימולציה
	1. סימלוץ סיבובים

מטרה: נותנים לכל המיינרים לנסות לכרות בלוק ולשלוח טרנזקציות המכילות העברות אקראיות אחד לשני למשך מספר סיבובים מסויים

קלטים:

* מספר הסיבובים

 תוצאות ההרצה:

* כל המיינרים ינסו לכרות בלוק, כל אחד לפי האלגוריתם שלו, כל מספר מסויים (מוגדר בקונפיגורציה של מיינר) של סיבובים הם מסמלצים טרנזקציה אקראי
* הודעות הנשלחות בין המיינרים יראו באופן גרפי באמצעות צביעת הקשתות דרכן עברו הודעות באדום
	1. עצירת סימולציה באמצע

מטרה: לפעמים סימולציה עלולה לקחת יותר מידי זמן או זכרון ורוצים לעצור אותה באמצע. יש תועלת באפשרות של עצירה באמצע

קלטים: אין

תוצאות הרצה: בעת לחיצה על כפתור "pause" תעצר ההרצה באמצע. ניתן יהיה להסתכל על ה state ויתאפשר להמשיך להריץ מאותה נקודה לאחר מכן

1. קבלת State של הרשת
	1. בקשת מאזן בין המיינרים התמימים ל selfish

מטרה: בסופו של דבר מטרת ההתקפה היא שה selfish miners ירוויחו יותר מחלקם היחסי ברשת. לכן, יש תועלת בפונקציה המאפשרת בכל רגע לראות את המאזן

קלטים: אין

תוצאות הרצה: בעת לחיצה על הכפתור המתאים, יוצג למשתמש המאזן בין המיינרים התמימים ל selfish

* 1. בקשת פרמטר connectivity

מטרה: פרמטר שיוסבר בהמשך ומטרתו לתאר במידה מסויימת את טיב החיבור של ה selfish miners לשאר המיינרים. בדומה למאזן, ערך מעניין עבור המשתמש

קלטים: אין

תוצאות הרצה: יוצג למשתמש בעת לחיצה

* 1. פליטת State לקובץ

מטרה: לצורך ניתוח של תוצאות רבות נרצה פונקציה אשר פולטת את התוצאות של ההרצה בפורמט מסויים לקובץ

קלטים: path לכתיבת הקובץ

תוצאות הרצה: כתיבת state אל הקובץ

## Use Cases

איור 3 מתאר את ה use cases הקיימים עבור המשתמש



**איור 3 - Use Cases Diagram**

נתאר את ה use cases.

Create Graph

**תיאור כללי**

 יצירת גרף חדש של צמתים תמימים ו selfish

**תנאים מקדימים**

אין

**שלבים בביצוע**

1. המשתמש לוחץ על כפתור יצירת גרף
2. המשתמש מתבקש להכניס את מספר הצמתים בגרף
3. המשתמש מתבקש להכניס את מספר הצמתים ה selfish
4. מערכת יוצרת ומציגה את הגרף

Output State

**תיאור כללי**

כותב את ה state הנוכחי של רשת הביטקוין המסומלצת לקובץ

**תנאים מקדימים**

המשתמש כבר יצר גרף

**שלבים בביצוע**

1. המשתמש לוחץ על כפתור שמירת state
2. המשתמש מתבקש להכניס path אליו ישמר
3. המערכת שומר את ה state ל path

Calculate Parameter

**תיאור כללי**

מתאר בקשה של המשתמש לחישוב והצגת ערך בעל עניין כלשהו

**תנאים מקדימים**

המשתמש כבר יצר גרף

**שלבים בביצוע**

1. המשתמש לוחץ על כפתור המתאר מה הוא הפרמטר לחישוב
2. המערכת מחשבת את הפרמטר כפונקציה של המידע השמור עבור כל מיינר
3. המערכת מציגה למשתמש את התוצאה

 Run Num of Rounds

 **תיאור כללי**

המשתמש יכול לבקש להריץ את כל המיינרים ברשת למשך מספר מסויים של סיבובים. כאן יש מקום לתת תיאור קצת של תהליך הכרייה כדי להבין את משמעות הסיבוב. באופן הכי high-level ניתן לתאר את מחזור החיים של מיינר (מנקודת מבט של כרייה) עם ה state-transition diagram המוצג באיור 4



**איור 4 - state transition diagram של מיינר**

בכל סיבוב או שנוצר בלוק חדש או שהבלוק הנוכחי נמצא בכרייה. כל מיינר עבוד על בלוק משלו. הרצת ה use case הזה מאפשר לכל המיינרים מספר מוגדר של סיבובים לעבוד על הבלוק שלהם ולנסות למצוא hash מתאים

 **תנאים מקדימים**

המשתמש כבר יצר גרף

 **שלבים בביצוע**

1. המשתמש מזין מספר סיבובים ולוחץ על כפתור ההרצה
2. המערכת מבצעת קריאות לפונקציות המתאימות של המיינרים, כמספר הסיבובים שנבחרו
3. המערכת מציגה למשתמש את מעבר ההודעות על גבי הקשתות תוך כדי ההרצה

## דיאגרמת ארכיטקטורה

נציג כעת דיאגרמת ארכיטקטורת על. עיקר חשיבות הדיאגרמה במקרה זה היא לראות את המערכת שפותחה במסגרת הפרויקט יחד עם הקונטקסט של sinalgo framework. איור 5 מציג אך ורק את הנחוץ על מנת לקבל הבנה כללית:



**איור 5 - דיאגרמת ארכיטקטורה של המערכת**

מה שמוכל בתוך Selfish mining project הן המחלקות שנכתבו במסגרת הפרויקט ובעלות ממשק עם ה framework. בהמשך נעשה zoom in לכלל מחלקות הפרויקט.

מה שנמצא בתוך Sinalgo Framework הן המחלקות העיקריות שהסביבה מספקת.

מחלקת Node אבסטרקטית המאפשרת להרחיב אותה על מנת להסביר לסביבה איך ה Node הספציפי פועל בכל סיבוב, מחלקת Edge ומחלקת Graph שבאמצעותן יוצרים גרף של Node-ים. קונפיגורציה שמאפשרת הגדרות שונות כמו יצירת גרפים מוגדרים בראש - באופן כללי לא נעשה שימוש גדול מידי בקונפיגורציה, הרוב נעשה באופן "ידני" בשביל גמישות. הערכים שכן רלוונטיים לקונפיגורציה יתוארו בהמשך. מחלקת Run היא המחלקה שמאחדת את הכל, בעצם המחלקה המכילה את main, פותחת thread-ים אם צריך, דואגת לקרוא לפונקציות המתאימות של Node בכל סיבוב, מקבלת ערכים מהקונפיגורציה ומנהלת את הסימולציה.

מחלקת Message היא מחלקת האב של כל ההודעות העוברות בין צמתי הגרף. כל סוגי ההודעות שמומשו במסגרת הפרויקט יורשות ממחלקת אב (שגם היא הוגדרה במסגרת הפרויקט) שנקראת TTLMessage.

## Class Diagram ותיאור הרכיבים

כעת נתחיל ב zoom in לתוך הרכיבים ומחלקות שמומשו במסגרת הפרויקט ואופן פעילותם.

איור 6 מציג class diagram של המערכת



**איור 6 - Class Diagram כללי**

המחלקות בדיאגרמה מקובצות לרכיבים Blockchain, Miner, Messages, Singletons ו Wallet.

נעבור על כל רכיב ונתאר את תפקיד המחלקות בו תוך פירוט המתודות החשובות (לא נעבור על כל המתודות בגלל הכמות וגם כי חלקן לא תורם מאוד להבנת המערכת)

### הרכיב Blockchain



**איור 7 - Blockchain Class Diagram**

באיור 7 מוצג רכיב blockchain המכיל את הקוד המממש את מבנה הנתונים המבוזר שכל מיינר מחזיק. רשימה מקושרת של בלוקים המכילים טרנזקציות. כל טרנזקציה מכילה את הטרנזקציות היוצאות והנכנסות שלה (input-ים ו output--ים, מאיפה הגיעו הביטקוינים ולאן הם נשלחים). קיימת טרנזקציה מיוחדת שהיא ה coinbase - הטרנזקציה חסרת ה input-ים הקיימת בכל בלוק ומשמשת עבור הפרס לכריית הבלוק. קיים בלוק מיוחד שהוא הבלוק הראשון בבלוקצ'יין, היחיד שאין לו בלוק מקדים בבלוקצ'יין, והוא בלוק ה Genesi

רכיב זה יכול לעמוד בפני עצמו ולשמש אפליקציות אחרות שרוצות לתחזק קונצנזוס באמצעות blockchain. כל צומת באפליקציה מבוזרת כזאת צריך ליצור אובייקט מסוג blockchain ואז הוא יכול לנסות לכרות בלוקים ולהוסיף אותם למבנה, לספר לצמתים אחרים על בלוקים שהוא כרה, לקבל מהם פידבק בצורת בלוקים שהם כרו וכו.

**TransactionInput**



**איור 8 - TransactionInput class**

**תיאור**

מתאר input כלשהו בטרנזקציה השולחת ביטקוין בין 2 משתמשים

**משתנים**

 **transactionOutputId** - מזהה שמצביע על output של טרנזקציה קודמת ממנה "נלקחים" הביטקוין ל input הזה

**פעולות**

 אין

**TransactionOutput**



**איור 9 - TransactionOutput Class**

**תיאור**

מתאר output כלשהו בטרנזקציה השולחת ביטקוין בין 2 משתמשים

**משתנים**

 id - מזהה שהוא תוצר hash על שאר המשתנים

 recipient - מפתח פומבי המתאר כתובת ביטקוין אליה שולחים

 value - כמה שולחים אל recipient

 parentTransactionId - מזהה הטרנזקציה המכילה את ה output הזה

**פעולות**

isMine - מקבלת מפתח פומבי ומחזירה true אם הoutput הזה שייך לכתובת המיוצגת ע"י המפתח הפומבי הזה

**Transaction**



**איור 10 - Transaction Class**

**תיאור**

מתאר טרנזקציה בסך value הנשלחת מ sender אל recipient. חתומה על ידי עם החתימה signature. מסתמכת על רשימת TransactionInputs הנקראת inputs כקלטים לטרנזקציה ומחלקת את הinput-ים האלה בין רשימת ה TransactionOutputs שנקראת outputs

**משתנים**

 **id** - מזהה שהוא תוצר hash על השולח, המקבל והערך

 **sender** - מפתח פומבי המתאר את השולח של הטרנזקציה. יש לשים לב כי שולח זה אמור להיות הבעלים של ה inputs, אחרת הטרנזקציה לא ואלידית

 **recipient** - מפתח פומבי המתאר את מקבל הטרנזקציה

 **value** - כמות ביטקוין שנשלח

 **signature** - חתימה דיגיטלית עם אלגוריתם EC החותמת על פרטי הטרנזקציה עם המפתח הפרטי של sender

 **inputs -** רשימת TransactionInput-ים המהוות קלטים לטרנזקציה

 **outputs -** רשימת TransactionOutput-ים המתארים לאן ישלחו הביטקוין

**פעולות**

 **isTransactionValid -** מחזיר true אם הטרנזקציה חתומה בהתאם וגם סכום הקלטים עולה על סכום הפלטים וגם הקלטים לא בוזבזו כבר בעבר

 **processTransaction -** מבצע בפועל את הטרנזקציה בצומת הנוכחי. ז"א, מעדכן את הבלוקצ'יין על כך שהקלטים בוזבזו והפלטים הם עכשיו ביטקוינים בבעלות ה reciepients. יש לדאוג לעדכון הצמתים האחרים ברשת לאחר פעולה זאת

**Block**



**איור 11 - Block Class**

**תיאור**

 בלוק בבלוקצ'יין. מכיל טרנזקציות וmetadata הקשור לחוקיות ברשימה המקושרת

**משתנים**

 **index -** מספר סידורי של הבלוק בבלוקצ'יין

 **previousHash -** ה SHA256 המזהה את הבלוק הקודם ברשימה

 **merkleRoot -** ה hash המתאר את שורש עץ המרקל של הטרנזקציות

 **coinbaseTransaction -** טרנזקציית הפרס עבור כריית הבלוק

 **transactions -** רשימה של Transasction המוכלות בבלוק

 **nonce -** הערך המשתנה עליו מונים על מנת להצליח לכרות את הבלוק

**פעולות**

 **mineBlockNSteps -** ביצוע n צעדים של שינוי nonce בניסיון להגיע לכך שה hash של הבלוק יכיל מספיק leading zeros (שהבלוק יכרה). מחזיר true אם הצליח ובכל מקרה משנה את ערך nonce

 **addTransaction -** הוספת טרנזקציה לבלוק. טרנזקציה חייבת להיות מבוצעת (קריאה ל processTransaction) בהצלחה על מנת להתווסף לבלוק מאחר ועתידו של הבלוק הוא להיכרות ולהישלח לרשת. במקרה שתתווסף טרנזקציה לא ואלידית, מיינרים אחרים לא יאשרו את הבלוק והפרס לא יתקבל

**Blockchain**



**איור 12 - Blockchain Class**

**תיאור**

 מבנה נתונים המחזיק בלוקים של טרנזקציות. המבנה בנוי בצורה של רשימה מקושרת בהתאם למתואר עבור בלוקצ'יין בפרוטוקול ביטקוין

**משתנים**

 **blockchain -** רשימה של בלוקים

 **UTXOs -** ה unspent transaction db. מבנה נתונים המחזיק את תמונת המצב של כל ה TransactionOutput שעדיין לא בוזבזו מבחינת הצומת הנוכחי. מבנה זה ממומש באמצעות HashMap בין ה id של כל TransactionOutput לבין ה TransactionOutput עצמו

**פעולות**

 **isChainValid -** פעולה הכבדה המוודאת את תקינותו של כל הבלוקצ'יין. עוברת על כל הבלוקים, מוודאת שאכן כל בלוק הוא בעל hash עם מספיק leading zeros וכי ה previous hash שלו הוא באמת ה hash של הבלוק הקודם, עבור כל בלוק מוודאת את תקינות הטרנזקציות ואז "היכולת" לבצע את אותן הטרנזקציות לפי תמונת המצב של אותו הרגע. פעולה זאת נחוצה למשל כאשר מיינר אחד מקבל בלוקצ'יין ממיינר אחר הוא צריך לוודא בעצמו שהשרשרת כולה אכן תקינה וכמו כן לייצר לעצמו את ה UTXOs בהתאם

 **tryMineAndAddBlock -** מקבל בלוק ומנסה לכרות אותו למספר סיבובים מסויים. אם הכרייה הצליחה, מנסה להוסיף אותו אל הבלוקצ'יין

 **tryAddMinedBlock -** מנסה להוסיף אל הבלוקצ'יין בלוק שכבר נכרה. פעולה זאת נחוצה כאשר מיינר שומע על בלוק חדש שנוצר ע"י מיינר אחר

### הרכיב Wallet

רכיב זה מממש ארנק ביטקוין ומטרתו בפרויקט לאפשר למיינר מסויים להחזיק בצורה נוחה רפרנס לביטקוין השייכים לו מתוך הבלוקצ'יין, לייצר טרנזקציות ולחתום ובעצם ליצור זהות של כתובת ביטקוין עבור אותו המיינר שאליה גם ישלחו הפרסים על בלוקים אותו הוא יצליח לכרות.

**Wallet**



**איור 13 - Wallet Class**

**תיאור**

ארנק ביטקוין

**משתנים**

 **blockchain -** רפרנס לבלוקצ'יין כדי שיוכל ממנו להסיק את הטרנזקציות השייכות למשתמש

 **privateKey -** המפתח הפרטי של המשתמש לצורך חתימה על טרנזקציות

 **publicKey -** המפתח הפומבי של המשתמש לצורך שיוך טרנזקציות

 **UTXOs -** הטרנזקציות השייכות רק למשתמש מתוך ה UTXO הגלובלי

**פעולות**

 **generateKeypair -** יצירת מפתחות ה Elliptic Curves

 **getBalance -** מחזיר את מאזן הביטקוין הנוכחי של המשתמש

 **sendFunds -** יוצר טרנזקציה בהתאם לפרמטרים המתקבלים, חותם עליה ומחזיר אותה. מעדכן את המאזן בארנק בהתאם

### הרכיב Messages

את רכיב זה לא נתאר לעומק משום ששמות ההודעות אינדיקטיביים מספיק. נסביר רק שכל ההודעות יורשות מ TTLMessage שהיא הודעה בעלת TTL כלשהו. ז"א, בכל פעם שמיינר יקבל הודעה כזאת, לפני שיעביר אותה לשכניו הוא יוריד לה hop אחד מה TTL, יבדוק האם עדיין מתקיים TTL > 0 ורק אם כן, יעביר אותה הלאה. שיטה זאת באה למנוע הודעות שלנצח יסתובבו ברשת ויתפסו זכרון שלא לצורך. בהינתן גדלי הגרפים שניתן לעבוד עליהם, TTL של מספר hop-ים בודד אמור להספיק

### הרכיב Miners

במסגרת הפרויקט מומשו 2 סוגים של מיינרים, האחד רגיל ושמו MinerNode והשני selfish ושמו GreedyMinerNode. באופן כללי ניתן ליצור מיינרים נוספים ומורכבים אף יותר כפי שראינו בעבודה המסכמת במאמר המציג התקפת eclipse משולבת עם selfish mining מורכב. כל מיינר הוא מימוש של צומת ברשת. צומת בתשתית sinalgo מיוצג על ידי המחלקה האבסטרקטית Node

**Node**



**איור 14 - Node Class**

**תיאור**

צומת ברשת שיכול לקבל ולשלוח הודעות לצמתים השכנים

**משתנים**

 **ID** - מזהה ייחודי לכל צומת

 **outgoingConnections -** רשימת חיבורים לצמתים השכנים של צומת זה

**פעולות**

 **preStep -** פונקציה הנקראת בתחילת כל סיבוב

 **handleMessages -** פונקציה שנקראת בסוף הסיבוב ומאפשרת טיפול בכל ההודעות שנשלחו לצומת זה

**MinerNode**



**איור 15 - MinerNode Class**

**תיאור**

מיינר תמים ברשת ביטקוין

**משתנים**

 **blockchain -** בלוקצ'יין המתאר את תמונת המצב ברשת מבחינת המיינר הזה

 **wallet -** הארנק של המיינר הזה

 **transactionDB -** מבנה נתונים המכיל את הטרנזקציות שהמיינר שמע עליהן משכניו. טרנזקציות אלה מחכות להיכנס לאחד הבלוקים שהוא ינסה לכרות

 **currentWorkInProgress -** הבלוק הנוכחי אותו המיינר מנסה לכרות

 **hashesPerRound -** מספר ניסיונות ה hash אותו כל מיינר מנסה בכל סיבוב. התהליך הוא הסתברותי אבל המספרים ורמת הקושי כרגע מכוונים כך שבממוצע כל 4 סיבובים מיינר מוצא בלוק

**פעולות**

 **tryToMine -** המיינר מנסה לכרות את הבלוק הנוכחי במשך hashesPerRound האשים אם קיים בלוק נוכחי או יוצר חדש ומוסיף לו טרנזקציות מה DB, אם הוא מצליח לכרות הוא מוסיף אותו לבלוקצ'יין ועושה broadcast עם הבלוק לשכנים שלו באמצעות BlockMessage

 **tryToSpend -** כל 3 בלוקים חדשים שנוצרים (אין באמת סיבה לבחירה הזאת..) מיינר מנסה ליצור טרנזקציה בה הוא מגריל שכן ושולח אליו מספר מסויים של ביטקוין. זאת על מנת לסמלץ עבודה באישור הטרנזקציות מול הבלוקצ'יין לצמתי הרשת. המיינר עושה broadcast לשכניו עם TransactionMessage

 **handleBlockMessage -** נקרא כאשר הגיעה BlockMessage. המיינר ינסה להוסיף את הבלוק לבלוקצ'יין. במקרים מסויימים המיינר ישים לב שהבלוקצ'יין שלו בפיגור ולכן יבקש משכניו את הבלוקצ'יין שלהם ויחליף אליו. בקשה זאת תעשה עם BlockchainRequestMessage

 **handleTransactionMessage -** הגיעה TransactionMessage. המיינר יוסיף את הטרנזקציה ל DB

 **handleBlockchainRequest -** הגיעה BlockchainRequestMessage. המיינר ישלח חזרה אל השכן השולח את הבלוקצ'יין שלו עם BlockchainResponseMessage

 **handleBlockchainResponse -** הגיעה BlockchainResponseMessage. המיינר יבדוק תקינות הבלוקצ'יין שנשלח ויחליף את הבלוקצ'יין שלו לבלוקצ'יין שנשלח

איור 16 מציג flowchart המתאר את פעולת המיינר בכל סיבוב



**איור 16 - Miner Flowchart**

גם את ה subroutines ניתן לתאר באמצעות Flowchart, נעשה זאת עבור handleBlockMessage מאחר והיא שונה באופן מהותי עבור MinerNode ו GreedyMinerNode והבדלים אלה מציגים הבדלים עיקריים בין שני המיינרים.



**איור 17 - FlowChart של handleBlockMessage עבור מיינר רגיל**

**GreedyMinerNode**

מיינר זה יורש מ MinerNode ולוקח ממנו את רוב ההתנהגות המתארת מיינר ברשת ביטקוין. בנוסף להתנהגות, הוא פועל לפי האלגוריתם של selfish mining ולכן מתחזק בלוקצ'יין פרטי ומפרסם בלוקים באופן סלקטיבי. יש לציין כי קיימת הנחה לפיה GreedyNodes הם חלק מקליקה. הנחה זאת היא סבירה מאחר ולפי האלגוריתם, צמתים selfish הם חלק מבריכה המנוהלת ע"י מפעיל הבריכה, צמתים אמנם לא מחזיקים באמת חיבור ב hop יחיד אחד לשני אבל דה-פקטו זה יכול לקרות ב2 hop-ים העוברים דרך המפעיל. GreedyMiner דורס התנהגויות נוספות אבל נתרכז כאן ה handleBlockMessage כדי לתת מוטיבציה מספקת תוך כדי המנעות מפרטים רבים מידי.

נציג את המחלקה ונראה את ההבדל ב Flowchart של handleBlockMessage.



**איור 18 - GreedyMinerNode Class**

**תיאור**

מימוש של מיינר ברשת ביטקוין הפועל לפי פרוטוקול selfish mining

**משתנים**

 **privateBlockchain -** בלוקצ'יין נוסף פרטי לבריכה של selfish miners. קונצנזוס עליו הם מסכימים ביניהם

 **greedyClique -** סט המכיל את צמתי הבריכה

**פעולות**

 **handleBlockMessage -** טיפול בבלוק חדש שהרשת דיווחה עליו לפי פרוטוקול selfish mining



**איור 19 - Flowchart של handleBlockMessage לפי selfish mining**

כפי שמתואר ב selfish mining, המיינר יוסיף לשרשרת הפרטית בלוקים שנמצאו ע"י חברי הבריכה ויאריך את הפרטית. כאשר מתגלים בלוקים לפומבית הוא ינסה "לבטל" אותם עם בלוקים פרטיים במידת הניתן. אם לא ניתן, שאר הרשת ניצחה והמיינר לוקח את הפומבית להיות השרשרת הפרטית וממשיך משם.

כפי שניתן לראות באיור 19, selfish miner משתמש הרבה ברוטינות שהוגדרו במחלקת האבא מאחר ויש דימיון רב. לדוגמא תחזוקת הבלוקצ'יין הפרטי זהה במקרים רבים לתחזוקת הבלוקצ'יין אותה עושה MinerNode בין צמתי הרשת, רק שבמקרה זה התחזוקה נעשית רק בין הצמתים ה selfish.

### הרכיב Singletons

רכיב זה מכיל מספר מחלקות הנותנות שירות לכלל המחלקות ולסימולציה

**Config**

הוא ייצוג בצורת מחלקה של קובץ Config.xml המכיל מספר ערכים שניתן לעשות להם התאמה כמו גודל ויזואלי של הצמתים, מרחק גאוגרפי מקסימלי המאפשר חיבור של צמתים עם קשת, מימדי הסימולציה ועוד

**StringUtil**

מחלקה המכילה בעיקר פונקציות הדורשות סריאליזציה כמו חתימה בEC על מידע באמצעות מפתח פרטי, ביצוע SHA256 על מחרוזת, החזרת ייצוג Gson של אובייקט לצורך debug, חישוב ייצוג string של merkle root על רשימה של טרנזקציות ועוד

**CustomGlobal**

מחלקה המוגדרת (אופן ריק) ב sinalgo. מכילה handler-ים ללחיצות, קבל קלטים והאופן כללי event-ים המתקבלים דרך ממשק המשתמש. לדוגמא, על מנת ליצור דרף חדש יש ללחוץ על כפתור build graph ב gui ולספק פרמטרים שמתבקשים. ביצוע פעולה זאת יקרא לפונקציה CustomGlobal.buildUnitDiskGraph אותה אני הגדרתי עם והיא תיצור את הגרף.

נתוני הגרף, רשימת צמתים וערכים נוספים שהייתי זקוק להם על מנת לתת שירותים נוספים דרף ה GUI (כמו לספק מאזן נוכחי בלחיצה בין המיינרים) נשמרו תחת מחלקה זאת בעת היצירה והם נגישים בכל event handler המתקבל מה GUI

# פרטים טכניים בנוגע להרצת הסימולציה

## ממשק המשתמש



**איור 20 - מסך כניסה למערכת**

כלל הממשקים של המשתמש עם המערכת נמצאים תחת Simulation Control או Project Control. באמצעות Simulation Control ניתן להזין מספר ב text box המסומן Rounds to do וללחוץ על כפתור ה Play כדי להריץ את הרשת למספר סיבובים. במצב הראשוני, אין עדיין רשת. על מנת ליצור רשת יש ללחות על Build Graph, המשתמש יתבקש להכניס מספר צמתים בגרף ומספר צמתים שהם Selfish מתוכם



**איור 21 - מסך הכנסת מספר צמתים ברשת**

לדוגמא אם נכניס 10 עבור מספר צמתים כולל ו3 עבור מספר צמתים selfish נקבל גרף הצורה הבאה:



**איור 22 - רשת לדוגמא**

בכל שלב ניתן ללחוץ על GetGlobalBalance על מנת לקבל את כמות הביטקוין הכוללת שמחזיקים הצמתים ה selfish והצמתים התמימים.

בכל שלב ניתן ללחוץ על GetSelfishConnectivity שיחזיר את ערך ה selfish connectivity הממוצע בגרף - מדד זה הוא ממד שבחרתי לבחון ואסביר אותו בפרק ביצוע המדידות.

בכל שלב ניתן ללחוץ ResultsToFile כדי לכתוב את התוצאות הנוכחיות של הסימולציה לקבצים. המשתמש יתבקש לספק תיקייה (רצוי לספק path שנמצא תחת c:\users\[username] על מנת להימנע מבעיות של הרשאות כתיבה לדיסק). כל לחיצה כזאת תוביל לכתיבה של המידע ל3 קבצים שונים שהמבנה שלהם יוסבר גם הוא בפרק ביצוע מדידות.

כמובן ניתן להריץ מספר סיבובים מסויים, לעצור, לבחון את הערכים. ניתן גם "לעמוד" מעל צומת (hover) ולראות את מצב הביטקוין הנוכחי שלו, מצב הבלוקצ'יין שהוא מתחזק.



**איור 23 - דוגמא להצגת מצבו של צומת**

באיור 23 ניתן לראות כי הצומת שעמדנו מעליו עם העכבר הוא צומת עם מזהה 3, יש לו 200 ביטקוין (הרצנו 30 סיבובים כפי שגם ניתן לראות), אורך הבלוקצ'יין שלו הוא 10 בלוקים וניתן לראות את ההאשים שלהם

## מיקום קבצי העבודה והוראות הפעלה

הפרויקט מוגש בצורת פרויקט ג'אווה שנכתב בסביבת אקליפס. לצורך הרצת הסימולציה קיימים גם jar-ים שנמצאים תחת התיקייה binaries. את קבצי הפרויקט עצמם שנכתבו לצורך הפרויקט ניתן למצוא תחת src\projects\BitcoinMiner\. שמות הקבצים והמחלקות הם אותם השמות כפי שהוצגו במסמך זה.

כל הקבצים מוגשים בתוך תיקייה שנקראת BitcoinSelfishSimulation. לצורך הרצת הסימולציה יש לפתוח cmd ולעשות dir אל התיקייה BitcoinSelfishSimulation ושם להריץ את הפקודה

java -cp binaries\bin;binaries\jdom.jar;binaries\bcprov-jdk15on-159.jar;binaries\gson-2.6.2.jar sinalgo.Run -project BitcoinMiner

לצורך נוחיות שורה זאת נמצאת בקובץ טקסט שנקרא "שורת הרצה.txt" בתיקייה BitcoinSelfishSimulation

# ביצוע מדידות

## ערכים הנמדדים

במסגרת המדידות נכתבים לקבצים מספר ערכים. כמעט כל הערכים שהגדרתי הם טריוויאליים ויתוארו בחלק הבא שמסביר את פורמט הקבצים אך ראשית נסביר בקצרה על 2 מהערכים

### selfishLeadRevenueRatio

מדד זה נותן תיאור כלשהו ליחס בו התגברו ה selfish miners על המיינרים התמימים.

נגדיר את



expectedGreedyRevenueRatio הוא היחס המצופה בין רווחי הצמתים ה greedy לבין רווחי הצמתים התמימים מאחר ועפ"י פרוטוקול ביטקוין הרווחים אמורים להתקבל ביחס לכוח החישוב.

נגדיר את



actualGreedyRevenueRatio הוא היחס בפועל בין רווחי שני הצדדים.

לבסוף נגדיר את היחס



כאשר selfishLeadRevenueRatio גדול מ-1, ה selfish miners מנצחים. בנוסף, יחס זה נותן לנו תחושה פי כמה גדל הפער בין הצמתים הרעים לצמתים הטובים. לדוגמא, אם selfishLeadRevenueRatio=2, ניתן להגיד כי עבור פרמטרי הסימולציה הנוכחית, כוח החישוב של selfish miner שווה לכוח החישוב של שני honest miners

### selfishConnectivityRatio

בניתוח התקפת selfish mining עשו שימוש כותבי המאמר בפרמטר שהם המציאו וקראו לו  . נזכיר כי פרמטר זה מוגדר כך:

בהינתן מצב של race (פורסמו שרשרת פרטית ושרשרת פומבית באותו אורך, במקביל)  יהיה החלק מתוך הצמתים התמימים שיבחרו דווקא בבלוק של השרשרת הפרטית (השייכים למיינרים ה selfish)

המשמעות הקונספטואלית של ברורה, אולם היא תלויה מאוד בגורמים משתנים והסתברותיים כמו ה latency בין צמתים ברשת וכו.

ניסיתי לחשוב ולהגדיר פרמטר המשפיע על  אך הוא קבוע עבור גרף מסויים כך שניתן יהיה לנסות למצוא קשר בינו לבין הצלחת ההתקפה.

הפרמטר selfishConnectivityRatio יוגדר עבור צומת בודד באופן הבא:



החלק היחסי של צמתים תוקפים מבין השכנים של צומת מסויים. התלות בין  לבין selfishConnectivityRatio של כל אחד מהצמתים היא די ברורה - ככל שלצומת יש יותר שכנים שהם selfish, כך עולה הסיכוי שבמצב של race הוא ישמע קודם את הבלוק אותו פרסמה הבריכה ה selfish.

בנוסף ננסה למצוא קשר לפרמטר חזק יותר שהוא הממוצע של פרמטר זה על גבי הצמתים התמימים ברשת:



בתקווה למצוא קשר בין ערך ממוצע זה לבין סבירות הצלחת ההתקפה

## פורמט קבצי התוצאות

לכל גרף שנוצר קיים מזהה GraphId. כל לחיצה על ResultsToFile תכתוב דגימה של ה State הנוכחי של הרשת לתוך 3 קבצים:

1. **SelfishSingleRun\_[GraphId].csv** - קובץ המכיל שורה אחת עבור כל צומת selfish בגרף. לכל צומת יכתבו הפרטים הבאים:
	1. roundNum - מספר הסיבוב הנוכחי (מספר הסיבובים שהסימולציה עברה, משמש כציר ה"זמן")
	2. nodeId - מזהה הצומת
	3. BTC - כמות הביטקוין שברשות הצומת
	4. privateBTC - כמות הביטקוין שברשותו עפ"י השרשרת הפרטית (תאורטית, אם היה מפרסם את השרשרת והיא הייתה מתקבלת)
	5. expectedBTC - כמות ש"אמור" להיות לו בהתחשב בחלקו היחסי ברשת וברווחים שנכרו עד כה
	6. isOverExpectation - בוליאני שמתאר האם expectedBTC > BTC
2. **HonestSingleRun\_[GraphId].csv** - קובץ המכיל שורה אחת עבור כל צומת תמים בגרף. באופן דומה לכל צומת יכתבו הפרטים הבאים: roundNum, nodeId, BTC, selfishConnectivityRatio, expectedBTC, isOverExpectation. כאשר selfishConnectivity הוא הפרמטר שהוסבר בחלק הקודם.
3. **GlobalInfo\_[GraphId].csv** - קובץ המכיל מידע סיכומי לגבי מצב הרשת. תכתב שורה אחת עם הפרטים הבאים:
	1. roundNum
	2. nodesNum - מספר הצמתים בגרף
	3. selfishRatio - החלק היחסי של צמתים selfish מתוך nodesNum
	4. averageSelfishConnectivityRatio - ממוצע על פני selfishConnectivityRatio של כל הצמתים התמימים
	5. selfishGain - כמות הביטקוין שהרוויחו הצמתים ה selfish
	6. expectedSelfishGain - כמות הביטקוין אותו היו "אמורים" הצמתים ה selfish להרוויח
	7. honestGain - בדומה ל selfishGain
	8. expectedGreedyRevenueRatio - היחס המצופה בין רווחי ה selfish miners לבין רווחי התמימים
	9. actualGreedyRevenueRatio - היחס שהתקבל בין הרווחים
	10. selfishLeadRevenueRatio - היחס בין actualGreedyRevenueRatio ל expectedGreedyRevenueRatio
	11. hasSelfishWon - האם ההתקפה הצליחה? במילים אחרות - האם הselfish miners הצליחו להרוויח יותר ממה שהם אמורים היו להרוויח

כל הקבצים נכתבים בפורמט CSV על מנת שיהיה נוח לפתוח אותם ב excel ולייצר מהם גרפים מתאימים.

תוצאות שנכתבות על ידי ResultToFile ישורשרו לקבצים המתאימים לגרף הנוכחי, במידה והם כבר קיימים מכתיבות קודמות לקבצים. אחרת, יווצר קובץ חדש.

## מוטיבציה לבחירת הערכים והמדידות

באמצעות הסימולציות נרצה ליצור דגימות של ערכים שונים של תכונות הרשת בזמנים שונים ולנסות למצוא קורלציה כלשהי בין תכונות אלה לבין מידת ההצלחה של ההתקפה.

באופן ספציפי, ננסה לענות על השאלות

1. באיזו סבירות ההתקפה מצליחה?
2. כתלות באילו פרמטרים ההתקפה מצליחה (הצלחה במובן בינארי)?
3. כמה הפרמטרים האלה משפיעים על מידת ההצלחה (הצלחה במובן לא בינארי אלא השוואה של גובה הרווחים)

# תוצאות

סומלצו הרצות עבור 20 גרפים אקראיים שונים בגדלים 7-10 עם 2-4 צמתים selfish המהווים תמיד פחות מ50% מכוח החישוב. החלוקה נעשתה באופן הבא:

* 3 גרפים בגודל 7 עם 2 צמתים selfish, 3 selfish ו3 selfish
* 3 גרפים בגודל 8 עם 2 צמתים selfish, 3 selfish ו3 selfish
* 4 גרפים בגודל 9 עם 2-4 צמתים selfish, 4 selfish
* 10 גרפים בגודל 10, 2 מהם עם 2 צמתים selfish, 4 מהם עם 3 צמתים selfish ו4 מהם עם 4 צמתים selfish

נלקחו בחשבון רק גרפים קשירים שנוצרו מאחר וגרף שאינו קשיר לא מבטא נכון את רשת ביטקוין וגם לא "שכונה" ברשת ביטקוין מאחר ועדכונים בקונצזוס חייבים להגיע לכל חלקי הרשת לצורך תפקוד תקין.

עבור כל גרף בוצע:

1. הרצת 200 סיבובי סימולציה
2. לקיחת דגימה כל 10 סיבובים

סה"כ נלקחו 400 דגימות (מצורפות כקבצי אקסל יחד עם הגשת פרויקט זה). לאחר יצירת הדגימות נותחו התוצאות. הגרפים ועיקרי התוצאות יוצגו כעת.

## מספרים כלליים

מתוך 20 הרשתות השונות שסומלצו, הדגימה האחרונה מראה כי המיינרים ה selfish הרוויחו יותר מערכם היחסי ב13 מהרשתות. תוצאה זאת נראית חלשה יחסית למספרים המצופים. אולם, אם בוחנים את המקרים בהם averageSelfishConnectivityRatio > 0.5 (בממוצע לפחות חצי מהחיבורים של צומת תמים הם לצומת selfish) אז ברשתות אלה, המיינרים ה selfish ניצחו ב100% מההרצות. דבר המצביע על תלות חזקה מאוד בפרמטר זה. פה שווה להזכיר את התקפת ה eclipse שתוארה בעבודה המסכמת שהראתה שהקושי ליצור מספר חיבורים גדול של צומת תוקף לצמתים הוא לא גדול מאחר וניתן לחבר את אותו הצומת התוקף באמצעות הרבה חיבורים לאותו צומת תמים אחר וכן לצמתים אחרים ועלות חישובית ורשתית נמוכה.

ברוב המוחלט של ההרצות ברגע שה selfish miners הצליחו לצבור פער כלשהו, הם גם הצליחו להחזיק בו ולהגדיל אותו, כצפוי מהתקפה זאת.

## מקרי קיצון

קיים מספר בודד של הרצות שראוי להציג באופן פרטני מאחר והתקבלו בהן תוצאות קיצוניות יותר מבאחרות.

1. אחת ההרצות של רשת בעלת 10 צמתים ומתוכם 4 selfish היא היחידה בה לאורך כל ההרצה, החל מהסיבוב הראשון ועד האחרון, בכל דגימה המיינרים ה selfish היו בעלי הון הגדול מחלקם היחסי. מזהה ההרצה הוא e7ed312b-b9d3-4825-b4fa-d7373305a50a לצורך חיפוש כל הדגימות בקבצי התוצאות. באופן לא מפתיע, ערכו של averageSelfishConnectivityRatio בהרצה זאת היה 0.75. נזכיר שהיו הרצות נוספות בהן הפרמטרים האחרים כמו היחס של מיינרים selfish היה זהה אך לא התקבלו תוצאות קיצוניות כאלה.

**איור 24 - רווחי selfish miners כתלות במספר סיבובים**

באיור 24 ניתן לראות את הירידה שהחלה בסיבוב 100, ירידה זאת צפויה לקרות באופן כללי בהתקפה זאת מאחר וכוח החישוב של המיינרים התמימים גדול יותר והם מדביקים את הפער. עקב הדבקת הפער, המיינרים ה selfish משחררים כמה בלוקים פרטיים על מנת לחזור לשליטה ולרווחים.

איור 25 מציג את selfishLeadRevenueRatio כתלות במספר הסיבובים. ניתן לראות כי הוא תמיד מעל 1 מאחר והמיינרים ה selfish תמיד הרוויחו מעל המצופה מהם בהרצה זאת. כמו כן רואים peak-ים מאוד קיצוניים שנובעים מכך שההתקפה "אילצה" את המיינרים ה selfish להילחם בהדבקת הפער של המיינרים הרגילים באמצעות פרסומים של בלוקים פרטיים. בעקבות הפרסומים האלה המיינרים ה selfish זכו ברווחים זמניים (מנקודת המבט שלהם) עד שהרשת עומתה עם הפרסומים של המיינרים התמימים. לבסוף ניתן לראות התייצבות על רווחים של בערך פי 3 למיינר selfish אל מול מיינר תמים.

**איור 25 - selfishLeadRevenueRation כתלות במספר הסיבובים**

כאשר מסתכלים על התוצאות הפרטניות של המיינרים ניתן להבחין בכך שלאורך הריצה קיימים מיינרים תמימים רבים שלא הרוויחו ביטקוין בכלל(!) ומצד שני, כמעט בכל דגימה כל המיינרים ה selfish מרוויחים. אפילו לאחר כל הסיבובים בדגימה האחרונה ניתן לראות את האנומליה הזאת



**איור 26 - רווחי ה selfish miners בריצה**

ואלו עבור המיינרים התמימים:



**איור 27 - רווחי המיינרים התמימים בריצה**

מתוצאות אלו ניתן לראות דבר אחד מאוד בולט - 3 מיינרים כמעט ולא הרוויח (בלוק יחיד מזכה ב100 ביטקוין), מיינרים אלה הם בעלי selfishConnectivityRatio=1. במקרה זה, כל שכניהם הם מיינרים selfish שמנעו מהם רווחים.

דבר נוסף שניתן לראות הוא שלמיינרים ה selfish היו עוד רווחים תאורטיים לא מבוטלים - privateBTC שהם הביטקוין שלהם על פי השרשרת הפרטית כולה. איור 28 מציג את רווחיו התאורטיים והמעשיים של מיינר מספר 7 בריצה זו.

**איור 28 - רווחים בפועל אל מול רווחים תאורטיים של מיינר selfish**

הרווח בין הגרפים מתאר את "מרווח הבטחון" ש selfish miner שומר לעצמו, במקום תמיד לפרסם את הבלוקים שהוא מוצא, לזכות ברווחים ובכך בעצם שני הגרפים היו מתלכדים, הוא משתדל לפרסם רק כשהוא חייב

1. דוגמא הפוכה היא ההרצה בעלת המזהה 71316674-2fe2-45da-b8cd-2961320f6faa שבה המיינרים התמימים ניצחו לאורך כל ההרצה.

בהרצה זאת התקבל הערך averageSelfishConnectivityRatio=0.2958. ערכים בסדר גודל זה כבר התקבלו בהרצות אחרות שבהן אפילו ניצחו ה selfish miners נצחון מוחץ. אולם הרצה זאת גם כללה רק 2 מיינרים selfish מתוך ה10 אז נראה שפה זה הפרמטר שגרם לתוצאה הקיצונית.

**איור 29 - רווחי מיינרים selfish אל מול תמימים**

 באיור 29 ניתן לראות שהמיינרים התמימים משגשגים ומנגד ה selfish miners מדשדשים.

המסקנה שניתן להסיק היא שכאשר החלק היחסי של כוח החישוב שלך קטן **וגם** הקשירות שלך לשאר צמתי הרשת אינה גדולה, כריית ביטקוין ב selfish mining תוביל אותך להפסדים.

## תוצאות סיכומיות

לצורך חלק זה איחדתי את המידע בכל קבצי ה Global של ההרצות השונות כדי לנסות לקבל תחושה לגבי תוצאות כלליות לרשתות מהסוג הזה ולהתקפה המתוארת ולהציגן.

נתייחס בניתוח לדגימה האחרונה שנלקחה עבור כל גרף כדגימה המייצגת את "תוצאת ההרצה".

**איור 30 - רווחי הselfish miners כתלות בחלקם היחסי**

מאחר ולא ביצענו ניסויים על המון גדלים שונים אז יש לנו יחסית הרבה דגימות השייכות למעט יחסים כאלה (יש גם הגבלה שחלקם היחסי של selfish miners יהיה תמיד פחות מ 0.5) אבל בגדול ניתן לראות את קו המגמה הממוצע שמראה את הקורלציה הצפויה בין חלקם היחסי של המיינרים ה selfish מתוך כלל האוכלוסיה לבין רווחיהם.

**איור 31 - selfishLeadRevenueRatio כתלות ב selfishRatio**

באיור 31 רואים פיזור מעט גדול יותר אבל תוצאה דומה. ניתן לראות גם שכאשר חלקם היחסי של ה selfish miners מגיע לאיזור ה 0.3 אז ההתקפה נעשית יעילה (כבר בטווח היחסית קצר שסימולציה זו בוחנת) מאחר וערכו של selfishLeadRevenueRatio גדול מ1.

ננסה לבחון את הערכים האלה ביחס לקשירות של ה selfish miner באמצעות פרמטר selfishConnectivityRatio.

**איור 32 - selfishLeadRevenueRatio כתלות ב selfishConnectivityRatio**

נראה שלא ניתן להסיק קשר כל כך ישיר כפי שקיוויתי בין השניים. בכל זאת כן ניתן להסיק שמעל selfishConnectivity של 0.5 זה game over והמיינרים ה selfish מנצחים ב100% . תוצאה זאת כשלעצמה היא לא טריוויאלית מאחר וזה תנאי הרבה יותר חלש משליטה ממשית בחלק גדול מהרשת (כפי שהוסבר בעבודה תחת נושא התקפת eclipse) ומתוצאות הניסויים נראה שזה מבטיח בהסתברות מאוד גבוהה רווחים גבוהים בכל שלב.

# נספחים

## קוד ראשי של MinerNode

בכל סיבוב נקראת ראשית הפונקציה preStep ולאחר מכן נשלחות הודעות בין הצמתים ונקראת handleMessages

@Override

**public** **void** preStep() {

tryToMine(blockchain);

 **if** (isMiningFastEnoughComparedToSpending() &&

(blockchain.getSize() % 3 == 0)) { // make sure we dont flood the network dbs with txs

tryToSpend();

 }

}

@Override

**public** **void** handleMessages(Inbox inbox) {

 **while** (inbox.hasNext()) {

 TTLMessage m = (TTLMessage) inbox.next();

 **if** (m.originalSenderID == **this**.ID) { // msg bounced back

 **continue**;

 }

 handleMessage(m);

 }

}

**protected** **void** handleMessage(TTLMessage m) {

 **if** (m **instanceof** TransactionMessage) {

 handleTransactionMessage((TransactionMessage) m);

 } **else** **if** (m **instanceof** BlockMessage) {

 handleBlockMessage((BlockMessage) m, inbox.getSender());

 } **else** **if** (m **instanceof** BlockchainRequest) {

handleBlockchainRequest((BlockchainRequest) m, inbox.getSender());

 } **else** **if** (m **instanceof** BlockchainResponse) {

 handleBlockchainResponse((BlockchainResponse) m);

 }

}

## קוד ראשי של GreedyMinerNode

@Override

**public** **void** preStep() {

 deltaPrev = privateBlockchain.getSize() - blockchain.getSize();

 tryToMine(privateBlockchain);

 **if**(deltaPrev != privateBlockchain.getSize() - blockchain.getSize()){

 //new block was mined

 isMyPoolFound = **true**;

 applyGreedySteps(blockchain.getSize());

 }

 **if**(isMiningFastEnoughComparedToSpending() && (blockchain.getSize() % 3 == 0)) { //make sure we dont flood the network dbs with txs

 tryToSpend();

 }

}

הפונקציה handleMessages זהה ל MinerNode וממומשת רק שם.

## דוגמא לקבצי פלט עבור הרצה אחת

קבצי פלט של הרצה אחת, בגרף 8 צמתים, 3 מהם selfish, סומלצו 100 סיבובים ואז בוצעה בקשה לפלט. לאחר מכן סומלצו עוד 100 סיבובים ובוצעה בקשה נוספת לפלט.

כתוצאה מכך נוצרו הקבצים הבאים:

1. SelfishSingleRun\_1b32461c-d3ba-4924-ac09-8e7511810a17.csv



**איור 33 - דוגמא לקובץ פלט 1**

1. HonestSingleRun\_1b32461c-d3ba-4924-ac09-8e7511810a17.csv



**איור 34 - דוגמא לקובץ פלט 2**

1. GlobalInfo\_1b32461c-d3ba-4924-ac09-8e7511810a17.csv



**איור 35 - דוגמא לקובץ פלט 3**

# סיכום

פרויקט זה הראה מימוש של התקפה על רשת ביטקוין הנקראת selfish mining. מטרתה של העבודה הייתה להמחיש כי פרוטוקול ביטקוין אינו לגמרי incentive compatible וכי באמצעות סטייה מהאלגוריתם ניתן למקסם את הרווחים. התקפה זאת היא אחת מבין כמה משפחות של התקפות שניתן לנצל כנגד הרשת. בפועל כדאי לכל משתמש או משקיע לצורך העניין בביטקוין להיות מודע לפירצות האלהו לנזקים שהן עלולות לגרום. ואולי אפילו למפתחים נמרצים המחפשים דרכים להתפתח ולפתח, להצטרף לקהילה ה open-source-ית של ביטקוין ולעזור לעולם בדרכו אל מטבעות אלקטרוניים מאובטחים יותר.

# רשימת מקורות

1. JA Garay, A Kiayias, N Leonardos (June 23, 2017). The Bitcoin Backbone Protocol: Analysis and Applications. National and Kapodistrian University of Athens.

2. F Tschorsch, B Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. Humboldt University of Berlin. (Published in: IEEE Communications Surveys & Tutorials ( Volume: 18, Issue: 3, thirdquarter 2016 )

3. S Nakamoto (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin contributers

4. Nicolas T. Courtois, Lear Bahack. On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency. University College London, UK, Open University of Israel. (2 December 2014)

5. Ayelet Sapirshtein, Yonatan Sompolinsky, Aviv Zohar. Optimal Selfish Mining Strategies in Bitcoin. School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel. FC 2016: Financial Cryptography and Data Security pp 515-532 (23 July 2015)

6. Ittay Eyal, Emin G¨un Sirer. Majority is not Enough: Bitcoin Mining is Vulnerable. FC 2014: Financial Cryptography and Data Security pp 436-454 (09 November 2014)

7. Ethan Heilman and Alison Kendler, Boston University; Aviv Zohar, The Hebrew University of Jerusalem and MSR Israel; Sharon Goldberg, Boston University. Eclipse Attacks on Bitcoin’s Peer-to-Peer Network. included in the Proceedings of the 24th USENIX Security Symposium August 12–14, 2015 • Washington, D.C.

8. Kartik Nayak, Srijan Kumar, Andrew Miller, Elaine Shi. University of Maryland, College Park, Cornell University, Ithaca. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. Security and Privacy (EuroS&P), 2016 IEEE European Symposium (24 March 2016)

9. Pedro Franco, Understanding Bitcoin: Cryptography, Engineering, and Economics (24 October 2014)