

תמיר אלביליה 325832681



פתיחה

רקע:

באינטרנט הנוכחי קיימים מספר רב של שירותים שונים ומגוונים, אך למרות זאת, במשותף לכל השירותים הללו הוא שיש לסמוך על שרת מרכזי על מנת לעשות בהם שימוש. אך מדוע לנו לסמוך על שרת מעבר למסך שאכן ינהג בהגינות כלפי כל משתמשיו? באמצעות הקריפטוגרפיה ניתן לעקוף את מתן הסמכות והצורך בשרת מרכזי למתן שירות ובכך למגר את עריצות השרת המרכזי.

הרעיון:

מטרת הפרויקט היא יצרת בלוקצ'יין – מסד נתונים המאוחסן ללא צורך בשרת מרכזי (Decentralized) כך שלא יהיה כוח רב לאף שרת יחיד שבו כל המחשבים המשתתפים במערכת מחוברים ברשת P2P מפוזרת. המערכת תוכל לתפקד כראוי גם כאשר חלק מן השרתים מנסים לתקוף את המערכת. בפרויקט זה מסד הנתונים יתמוך בהעברה של כסף דיגיטלי (Cryptocurrency) והפצה של הודעות ברבים.

שימוש:

המערכת שימושית עבור כל מי שמעוניין להעביר כסף דיגיטלי ללא הרצון לסמוך על שרת מרכזי כזה או אחר או לשדר מידע ברבים ללא צנזור.

הגדרת הבעיה ופתרונה

כעת באינטרנט שירותים כמו העברת כספים והפצת הודעות מחייבות מתן סמכות בשרת מרכזי. כדי שנוכל להסיר את מתן הסמכות מהמערכת ראשית יש לעבור לרשת עמית לעמית שבה כל מחשב במערכת מסוגל להתחבר לכל מחשב אחר במערכת ושאינה כוללת שרת מרכזי. כל המחשבים המחוברים במערכת יקליטו את רצף האירועים והעסקאות שקרו במערכת. אך מכך נובעת בעיה חדשה, כיצד ניתן להגיע לקונצנזוס ולהחליט מהו רצף האירועים האמיתי שקרה. על מנת לפתור בעיה זו על מסד הנתונים להיות חסין מפני שינוי ספונטני או זיוף וזהום של כל אחד מהעמיתים במערכת. נעשה זאת באמצעות הגדרת שרשרת בלוקים (בלוקצ'יין). שרשרת הבלוקים תהיה מורכבת מאוסף בלוקים שבו כל בלוק מקושר לבלוק הקודם לו באמצעות hash. כל בלוק כולל בתוכו את הhash שלו עצמו ואת הhash של הבלוק הקודם לו. מכיוון שהhash הוא בהגדרה ייחודי עבור כל בלוק, מתקבלת התופעה שעל מנת לשנות בלוק יחיד בשרשרת הבלוקים, יש לשנות את כל הבלוקים אחריו. בכדי להפוך תופעה זו לכלי הגנה מפני שינויים בשרשרת נשתמש בproof-of-work (הוכחת עבודה) ונגדיר שבשביל שhash של בלוק יהיה תקין, על הhash להיחשב כתקין, עליו לקיים ; יש לשים לב שמשום שהפלטים פונקציית הhash היא פסודו-אקראית, ההסתברות לקבל hash שעובר proof-of-work בקושי n היא . לכן כאשר מספר האפסים הנדרשים בתחילת כל hash גדול, שינוי של כל בלוק בשרשרת נעשה בלתי אפשרי עבור כל תוקף שאין לו לפחות כסך מחצי מכוח המחשוב במערכת, זאת משום שתוקף שכזה יצטרך לכרות מחדש את כל הבלוקים שאחרי הבלוק שברצונו לשנות ובו זמנית לעקוף את שאר חברי המערכת שעובדים תמידית להרחיב אותה. כאשר אין לתוקף שכזה מספיק כוח מחשוב, הוא לא יוכל לעקוף את עבודתם של שאר מחשבי הרשת. עקב כך ניתן לקבוע את השרשרת הנכונה כשרשרת המאסיבית ביותר אשר להכינה היה נדרש את כוח המחשוב הרב ביותר. מתקבל קונצנזוס משום שכל שאר המשתתפים במערכת ישתפו אחד עם השני על העדכונים ששמעו ממשתפים אחרים וישתמשו בשרשרת המאסיבית ביותר כשאר כל המשתתפים. כל אחד מהבלוקים בשרשרת מכילים העברות (transactions). כל העברה מסמלת פעולה שבה משתמש מעביר מהכסף האלקטרוני הרשום על חשבונו למספר חשבונות אחרים. על מנת לוודא שמחבר ההעברה הוא אכן בעל החשבון נשתמש בפתרון הכולל חתימות דיגיטליות ובכדי למנוע בזבוז כפול של כספים שכבר בוזבזו, נשתמש במבנה נתונים שמאפשר אכסון וארגון של כל ההעברות ההכשרות שתוצאותיהן עוד לא בוזבזו.

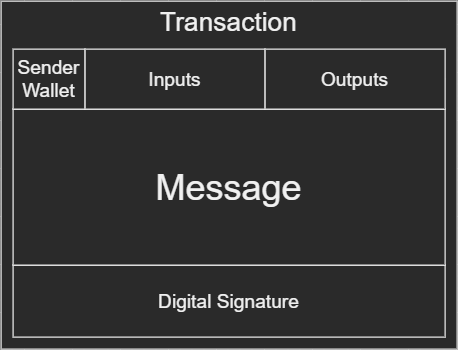
מבנה הבלוקצ'יין

העברה – Transaction

בבלוקצ'יין העברה היא המידע הבסיסי שמסד הנתונים מכיל, כל העברה היא הצהרה על מידע שיש לשמר בבלוקצ'יין. העברה מכילה הכרזה על העברת מטבעות דיגיטליים מארנק אחד לארנק אחר. ההעברות ישודרו בפומבי אל משתמשי הרשת על מנת שיועברו על להעברות מספר תנאים שיש לעמוד בהם על מנת לממש רכיב זה בבלוקצ'יין.

1. כל העברה חייבת להיות מיוצרת על ידי המקור המצוין בה בהכרח.
2. יש למנוע מכלל הלקוחות לשנות את התוכן של העברה לאחר שפורסמה ברשת
3. נדרשת היכולת להבדיל בין העברה אחת להעברה אחרת ללא תלות בתוכן ההעברה.

הינה תרשים של Transaction המתארת את מאפייני ההעברה:



* Sender wallet – הארנק של שולח ויוצר העברה והארנק של הגוף בהעברה הארנק יהווה מפתח ציבורי במערכת הצפנה א-סימטרית RSA.
* Inputs – מילון המכיל את כל מקורות ההעברה וערכם.
* Outputs – מילון המכיל את כל ארנקי מקבלי ההעברה וחלקם ממנה.
* Message – ההודעה שמכילה ההעברה, אורך מקסימלי של 222 תווים.
* Sender’s Digital Signature – החתימה הדיגיטלית של מחבר ההעברה, החתימה כוללת את כל העברה ומושפעת מערך הNonce. מטרתה להבטיח שימור של הערכים בהעברה ולהבטיח ששולח ההעברה הוא גם המחבר שלה.

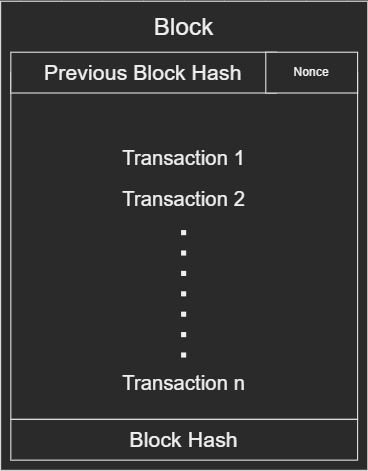
אתגרים במימוש ופתרונות

* כיצד ניתן למנוע מגוף זר לבצע העברה עבור לקוח אחר? על מנת לפתור בעיה זו נפנה למושג בקריפטוגרפיה הנקרא חתימה דיגיטלית. כל לקוח ייצר צמד מפתחות אחד פרטי ואחד ציבורי, כאשר לקוח מעוניין לחבר העברה, הוא יצמיד אליה את המפתח הציבורי שלו (הוא גלוי לכולם). בסוף ההעברה ייצר המחבר חתימה דיגיטלית המושפעת מגוף ההעברה ויצרף אותו אל סוף ההעברה. בסופו של דבר תתקבל מערכת שבה ניתן לוודא שהעברה מסוימת אכן חוברה על ידי השולח. כל לקוח כעת יוכל לאמת זאת באמצעות המפתח הציבורי המצורף בהודעה, והחתימה הדיגיטלית בסוף ההודעה.
* החתימה הדיגיטלית גם פותרת בעיה נוספת, היא מונעת מכול זר לשנות את התוכן הקיים בה, זאת משום שכל שינוי של התוכן אמור לשנות את ערך החתימה הדיגיטלית ולא ניתן לשנות את ערך החתימה משום שהמפתח הפרטי נדרש על מנת לייצר אותה, אך אין לאף זר מלבד השולח את אותו מפתח הפרטי. כל שינוי של התוכן יבטל את ההתאמה של המפתח הציבורי שצירף השולח לחתימה הדיגיטלית של ההעברה.
* כיצד נוכל לדעת אם העברה כלשהי היא חוקית ויש למחבר מספיק יתרה לבצע אותה? בכדי שההעברה תיחשב חוקית עליה לקיים את חוק שימור הכסף בinputs וoutputs שלה, כלומר sum(inputs) – sum(outputs) = 0.
* כיצד נדע איך כל אחד מהמקורות אכן על השרשרת ואינו בוזבז? שרשרת הבלוקים היא גדולה מאוד ולעבור על כולה עבור כל מקור זה אינו פתרון ממשי. על מנת לאכסן את מצב ההעברות שאינן בוזבזו נשתמש במבנה נתונים יעיל AVL שבו נאכסן את מצב כל הoutputs של כל העברה שהוכנסה לשרשרת באופן ממוין לפי החתימה הדיגיטלית של ההעברה אליהם הם משתייכים. בעץ AVL הסיבוכיות להוסיף או למצוא איבר במבנה הנתונים היא log(n) כאשר n זה מספר ההעברות בשרשרת. עקב כך נגדיר שמקור הוא חוקי רק כאשר מופיע בעץ ומחבר ההעברה הוא בעל המקור.

בלוק – (Block)

בלוק מכיל אוסף של העברות ומטרתו לשמר את סדר האירועים שבו העברות התקבלו. כל בלוק מכיל hash שלו עצמו וhash של הבלוק הקודם לו. על מנת שבלוק יהיה תקין אוסף ההעברות שלו חייב להיות תקין. על מנת שבלוק יתקבל על ידי משתתפי הרשת על hash הבלוק לעבור proof-of-work.

הינה תרשים של Block המתאר את מאפייני הבלוק:



* Previous Block Hash – הhash של הבלוק הקודם
* Nonce – ערך טקסט אקראי שמטרתו לשנות ולהשפיע על hash הבלוק, נקרא גם פתרון הבלוק
* transactions – רשימה של כל ההעברות שהבלוק מכיל.
* Block Hash – הhash של הבלוק הנוכחי.

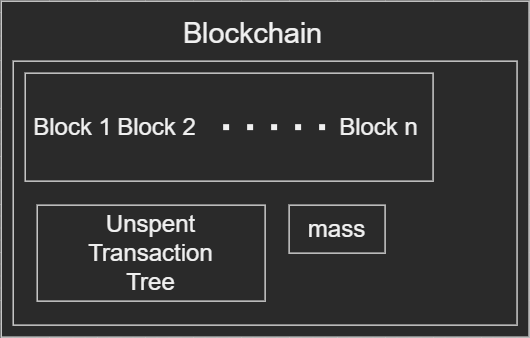
מאפיינים ועקרונות

* מכיוון שהhash של הבלוק מכיל את הhash של הבלוק הקודם לו כאשר משנים בלוק במקום הn, פעולה זו מבטלת את כל הבלוקים אחריו משום שהPrevious Block Hash שלהם אמור להשתנות בהתאמה ולכן גם הhash שלהם. כאשר על הhash.
* כאשר מייצרים בלוק אין זה סביר שהhash שיתקבל יתאים לproof-of-work הרצוי על ידי הרשת. לכן על מנת לשנות ולהשפיע על הhash, נשנה את ערך זה עד אשר יתקבל hash מתאים. תהליך זה נקרא כרייה והוא לוקח עוצמת מחשוב וזמן רבים ביחס לקושי הproof-of-work הרצוי.
* על מנת לעודד כורים להשקיע זמן וכוח מחשוב בכריית בלוקים, כאשר בלוק נכרה, לכורה הבלוק היכולת להוסיף העברה מיוחדת שמקורה אינה העברה קיימת ובכך מקבל הכורה פרס על עבודתו. גודל הפרס נקבע לפי מקום הבלוק בשרשרת. הגודל התחילי של הפרס שווה ל727 וכל 52560 בלוקים נחתך בחצי. מכיוון שמקור כל המטבעות בשרשרת מכריית בלוקים, ניתן לחשב את מספר המטבעות הכולל

שרשרת בלוקים (בלוקצ'יין) – Blockchain

שרשרת הבלוקים היא מבנה הנתונים העליון והיא מכילה אוסף של בלוקים המסודרים לפי ההhashes שלהם כך שעבור block n הPrevious Block Hash שלו שווה לHash של הבלוק הקודם לו בשרשרת והPrevious block hash של הבלוק העוקב לו שווה לBlock Hash שלו. לשרשרת הבלוקים גם עץ AVL שבו היא שומרת על איזה העברות כבר בוזבזו.

הינה תרשים של Blockchain המתארת את מאפייני שרשרת הבלוקים:

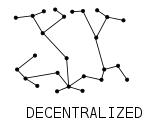


* chain – רשימה של הבלוקים שהשרשרת מכילה
* Unspent Transaction Tree – עץ AVL העוקב אחר אלו מקורות חוקיים עוד לא בוזבזו.
* mass – סכום הproof-of-work של כל הבלוקים בשרשרת.

מבנה התקשורת

התקשורת בפרויקט נעשית באמצעות רשת עמית לעמית (P2P) שבה כל עמית (peer) יוצר חיבורים ומקבל חיבורים במקביל. מטרת הרשת היא להיות מפוזרת ככל שאפשר ולכן אל לנו להשתמש בשרת מרכזי ורצוי שהרשת תהיה מפוזרת ככל שאפשר.

להלן ציור מופשט של מודל של רשת מפוזרת:



שרת - Server

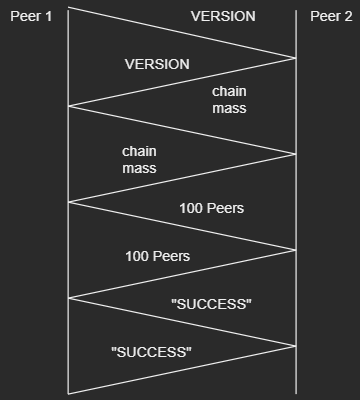
השרת מטרתו לנהל את כל החיבורים עם שאר העמיתים שאליהם עמית כלשהו מחובר. כל עמית פותח במחשבו שרת שביכולתו ליצור ולקבל חיבורים במקביל. השרת מתחיל להאזין לחיבורים לפי כתובת הIP והפורט שצוינו בקובץ הconfig של הפרויקט (פורט ברירת המחדל הוא 22222). לשרת כמות מוגבלת של חיבורים שלהם מוכן להסכים, פרמטר זה מוגדר גם כן על ידי קובץ הconfig וברירת המחדל שלו היא 150. כאשר מתבצע חיבור עם עמית אחר השרת מוסיף אותו למילון העמיתים שאליהם מחובר ביחד עם אובייקט הConnection שיפתח עבור אותו עמית. בעת פתיחת הConnection שני העמיתים עוברים תהליך של תסדיר.

חיבור – Connection

אובייקט החיבור מטרתו לנהל כל חיבור בין עמית יחיד לעמית יחיד בלבד ואת התקשורת ביניהם. בתחילת החיבור נפתח Connection run thread שבו מאזין החיבור להודעות מהעמית המחובר. לאחר מכן מתקיים תהליך של תסדיר setup שבו שני העמיתים המחוברים מחליפים ביניהם מידע לגבי עצמם. לאחר התסדיר החיבור יכול לשמש לצרכיהם של כל אחד מהעמיתים אשר יכולים לשלוח פקודות לעמית המחובר בהתאם לצרכיהם ומצב השרשרת שלהם.

**תסדיר - setup**

התסדיר מתקיים בעת פתיחת חיבור. והוא תהליך של העברת מידע על כל אחד מהעמיתים. יש לשים לב שמכיוון שהתוכנה היא סימטרית, על מנת לשמור על סדר דיבור הגיוני, נגדיר שהעמית הראשון לדבר הוא האחד בעל כתובות הIP הגדולה יותר. להלן תרשים של שיחת התסדיר:



ראשית מחליפים שני העמיתים את גרסת האפליקציה שלהם. העמיתים ממשיכים בשיחה רק אם הגרסה שלהם זהה, אחרת החיבור נסגר. הגרסה מוגדרת בקובץ הconfig של הפרויקט.

לאחר מכן הם מחליפים את מסת השרשראות שלהם, כל עמית שומר את מסת השרשרת של העמית שאליו הוא מחובר באובייקט הConnection.

לאחר מכן כל אחד מהם שולח עד 100 עמיתים בצורה של כתובות IP מברכת העמיתים המוכרים שלו אשר נבחרו אקראית. כל עמית יוסיף לברכת העמיתים המוכרים שלו את העמיתים ששותפו אתו כעת. שיתוף זה מרחיב את כמות המחשבים שעמית מסוים מכיר ברשת ובכך תורם לפיזור שלה.

לסיום כל עמית שולח "SUCCESS" להראות שהתהליך הסתיים בהצלחה.

**Connection – recv**

תפקיד הפעולה לקבל הודעה יחידה מן סוקט החיבור. פורמט הודעה הוא:

Size || Message כאשר Size הוא אורך ההודעה ואורכו שלו קבוע וגודלו 7 תווים. אורך הודעת פקודה הוא קבוע וגודלו 6 תווים. אורך הודעה רגילה הוא כאורך בלוק מקסימלי 2MB (2097152 בייטים). במקרה שאורך ההודעה אינו שבעה ספרות יש לעשות פאד לsize לגודל 7 תווים עם אפסים.

הפעולה קוראת כsize + message בייטים מההודעה ולפי האורך שצוין בsize מחלצת הפעולה את message ומחזירה אותו.

במידה והפעולה נכשלת או מתקבלת הודעה שאינה תקינה היא סוגרת את החיבור.

**Connection – send**

תפקיד הפעולה לשלוח הודעה לעמית המחובר. הפעולה מקבלת str message שהיא ההודעה שיש לשלוח. לפני שליחת ההודעה, הפעולה תוסיף את אורכה לתחילת ההודעה ותעשה לאורך פאד לפי הצורך. הפעולה אינה מחזירה דבר.

במידה והפעולה נכשלת היא סוגרת את החיבור.

**Connection – run**

כל עוד החיבור פתוח באובייקט החיבור רץ תרד השייך לחיבור על פעולת הRun של החיבור. הפעולה מאזינה להודעות מן העמית באמצעות פעולת הrecv, כאשר מתקבלת הודעה הפעולה תריץ את Connection.execute\_command אשר תבצע את הפקודה שהלקוח שלח במידה וההודעה אכן פקודה. הפעולה תשמור את ההודעה במאפיין באובייקט החיבור הנקרא last\_message כדי שפעולות אחרות באובייקט יוכלו לגשת אליו בקלות. במידה והתקבל "TERMINATE", הפעולה סוגרת את החיבור. הפעולה לא תריץ את execute\_command אילולא החיבור צלח את התסדיר.

**Connection – read\_last\_message**

פעולה זו מנסה לקרוא את הערך Connection.last\_message אשר תחילה מאותחל לNone. הפעולה תקרא את הערך רק כאשר ערכו אינו None (הפעולה run קיבלה הודעה ושינתה אותו). כאשר last\_message אינו None הפעולה קוראת את הערך, מאפסת אותו חזרה לNone ומחזירה את ערכו לפני האיפוס.

כל עוד last\_message שווה לNone הפעולה תחכה עד שערכו אינו None או שעברו 15 שניות. אם עברו 15 שניות ושום ערך לא נקרא הפעולה תעלה שגיאה (אך לא תסגור את החיבור)

**Connection – execute\_command**

הפעולה מקבלת הודעה str message ובמידה וההודעה היא פקודה ידועה, הפעולה תפתח בשיחת פקודה עם העמית כדבר הפקודה. במידה והפקודה צלחה הפעולה תחזיר True אחרת אם הפקודה נכשלה תחזיר False.

**פקודות Connection**

לכל פקודה אפשרית יש קוד משלה שהוא שמה של הפקודה מקוצר לשישה תווים. הפקודות החוקיות בחיבור הן SENDTX שליחת העברה, SENDBK שליחת בלוק ולבסוף CHNSYN סנכרון שרשראות בלוקים.

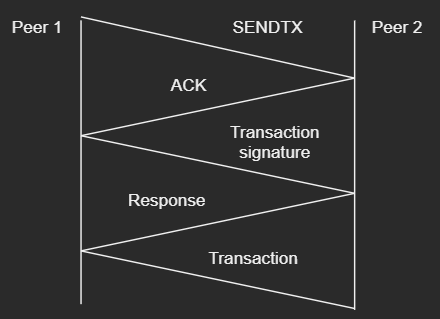
לכל פקודה אפשרית גם צמד שני פעולות המבצעות אותה בין העמיתים, פעולה של פוקד ונפקד. פעולות הנפקד מקומם בתוך execute\_command. ופעולות הפוקד הם פעולות של Connection.

כל פעולת פקודה פוקדת תמיד ראשית שולחת את קוד הפקודה אל העמית לפני תחילת ביצועה.

**SENDTX**

מטרת הפקודה היא לשתף בהעברה שהגיע אל הפוקד, הפוקד מעוניין לשלוח את ההעברה לנפקד

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח את קוד הפקודה לנפקד והנפקד מחזיר תשובה שקיבל את ההודעה. לאחר מכן הפוקד שולח את החתימה הדיגיטלית של ההעברה והנפקד מחזיר כתשובה אם הוא רוצה לקבל את ההעברה. אם כן הפוקד שולח את ההעברה אחרת לא נשלח דבר והפקודה הסתיימה.

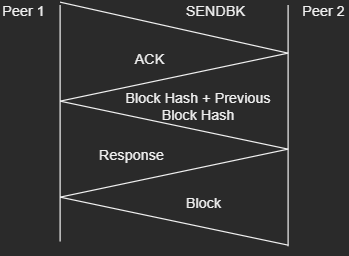
הנפקד מעוניין בהעברה כל עוד הוא אינו מכיר אותה בעצמו או היא איננה על שרשרת הבלוקים שלו.

לאחר שההעברה התקבלה על ידי הפוקד, הפוקד יאמת שהיא תקינה ושהחתימה הדיגיטלית שלה שווה לזו שציין הפוקד לפני שליחתה. אם ההעברה אומתה הנפקד יוסיף אותה אל בריכת ההעברות שלו וישתף אותה (יריץ SENDTX) עם כל שאר עמיתיו שהם אינם הפוקד. אחרת אם החתימה הדיגיטלית אינה זהה, הפעולה תסגור את החיבור.

**SENDBK**

מטרת הפעולה היא לשתף בלוק ששורשר בצד הפוקד אל הנפקד. הפוקד מעוניין לשלוח את הבלוק לנפקד. עמית מעוניין לשלוח בלוק לעמית אחר רק כאשר שרשר בלוק חדש.

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח את קוד הפקודה לנפקד והנפקד מחזיר תשובה שקיבל את ההודעה. לאחר מכן הפוקד שולח את הhash של הבלוק וגם את הhash של הבלוק הקודם לבלוק שאותו מעוניין לשלוח והנפקד מחזיר כתשובה אם הוא רוצה לקבל את הבלוק. אם כן הפוקד שולח את הבלוק אחרת לא נשלח דבר והפקודה הסתיימה.

הנפקד מעוניין בבלוק רק עם הprevious\_block\_hash שנשלח שווה לhash של הבלוק העליון בשרשרת הנוכחית של הנפקד (הבלוקים מסוגלים להשתרשר).

לאחר שהבלוק התקבל על ידי הפוקד, הנפקד ינסה לשרשר אותו לשרשרת שלו. (באמצעות הפונקציה Server.try\_chain\_block)

כאשר מתקבל הBlock hash מן הפוקד, הנפקד מעדכן את peer\_chain\_mass באובייקט החיבור שלו ומוסיף לו את גודל הProof of work של הhash שקיבל.

**CHNSYN**

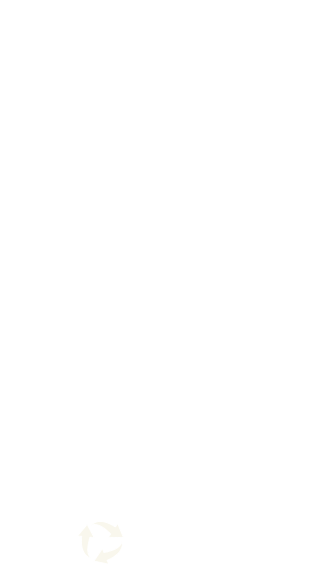
מטרת הפעולה היא לסנכרן את השרשרת של הפוקד עם השרשרת הכבדה יותר של הנפקד. עמית מעוניין להסתנכרן עם שרשרת של עמית אחר רק כאשר שם לב שהשרשרת של עמית שאליו מחובר כבדה משמעותית מהשרשרת שלו. שרשרת כבדה משמעותית מהשרשרת הנוכחית כאשר מתקיים:

peer\_chain\_mass >= my\_chain\_mass \* difficulty\_target \* TRUST\_HEIGHT

כאשר difficulty\_target זה קושי הProof of work המינימלי שנדרש על מנת לשרשר בלוק

ו TRUST\_HEIGHT זה קבוע שגודלו 6 שמטרתו למנוע משרשראות להסתנכרן מוקדם מדי כשעדיין לא ברור שהן השרשרת הפופולרית ביותר האמיתית (אני ארחיב על זה בהמשך)

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח לנפקד את קוד הפקודה. לאחר מכן הנפקד שולח את מסת השרשרת שלו. הפוקד שולח בחזרה את מסת השרשרת של עצמו ומספר הבלוקים בה והנפקד משיב עם “ACK”.

לאחר מכן הפוקד שולח את מאה הBlock hashes העליונים בשרשרת שלו (או פחות אם אין מספיק). במידה והנפקד מוצא הצטלבות בין אחד מהBlock hashes שקיבל לבין בלוק בשרשרת שלו ידוע כי באינדקס אחד אחרי, זוהי נקודת הפיצול בין השרשראות (Split Index). אם פיצול נמצא הוא מוחזר על ידי הנפקד, אחרת הנפקד עונה "continue" והפוקד שולח את מאה הבלוקים הבאים עד אשר הSplit Index נמצא או שנגמרו הבלוקים בשרשרת הפוקד או הנפקד.

אם נמצא Split Index זה אומר שהשרשראות של העמיתים נבעו מבסיס משותף ואין צורך לשלוח את החלק המשותף.

אם לא נמצא הנפקד מחזיר את הערך 0 שאומר שאין נקודת פיצול משום שהשרשראות שונות.

לאחר מכן הפוקד יוצר אובייקט שרשרת חדשה שעתידה להחליף את השרשרת הנוכחי והפוקד משרשר אליה את כל הבלוקים שכבר ידועים לו לפי Split Index.

לאחר מכן הפוקד שולח "start" המעיד על מוכנותו להתחיל לקבל הזרמה של בלוקים שישרשר אל השרשרת החדשה.

הנפקד מתחיל בלשלוח בלוקים מהשרשרת שלו. הוא שולח בלוק אחד עבור כל הודעה והוא מתחיל בשליחת הבלוק בSplit Index ועולה מעלה. בתשובה לכל בלוק הפוקד שולח "continue" בשביל להראות שהוא מוכן לקבל את הבלוק הבא. כאשר הנפקד מקבל בלוק הוא משרשר אותו לשרשרת החדשה. אם השרשור נכשל הפוקד סוגר את החיבור. התהליך חוזר חלילה עד אשר מסת השרשרת החדשה שווה למסה שהובטחה לו בתחילת הפקודה על ידי הנפקד. במידה והשרשרת החדשה איננה בגודל שהובטח (הנפקד פסק לספק בלוקים באמצע התהליך) השרשרת החדשה נמחקת והחיבור נסגר.

בסוף הפקודה כאשר השרשרת החדשה התקבלה במלואה, אנו מעבירים אליה את המטהדאטא של השרשרת הישנה ואז אנו מחליפים את השרשרת הישנה בחדשה.