# Analiza scenariuszy zaburzeń systemu

## Dominacja obliczeniowa jednego węzła

Występuje w sytuacji, gdy jeden węzeł jest w stanie wykopać blok z dużą większą prędkością niż pozostali kopacze.

Scenariusz jest realizowany poprzez dodanie dodatkowej trudności (parametr DIFFICULTY + 2) do wymagań kopania dla każdego węzła oprócz 5001, który wciela się w rolę dominatora. Węzły wykopane przez niego są wciąż poprawne, ponieważ trudność wymagana przy weryfikacji bloku nie ulega zmianie.

### Przypadek bez dominatora

Do porównania wyników wykorzystano wizualizacje oraz liczność bloków wykopanych przez różne węzły co można sprawdzić odpytując wybrany węzeł o listę bloków, grupując je według kopacza i przedstawiając liczebności (program gb.js do tego grupowania został wygenerowany przez Copilot i sprawdzony osobiście – było to możliwe ponieważ jest relatywnie prosty). Przy zwykłej pracy sieci:

Miner Counts: { '5001': 5, '5002': 5, '5003': 3, '5004': 5, undefined: 1 }

Undefined to blok GENESIS. W pozostałych przypadkach widać, że różni kopacze wykopali podobne ilości bloków. W skrypcie wysyłane jest 9 transakcji, a suma jest dużo większa – jest to skutek losowo tworzonych forków, które nie przeszkadzają w badaniu.

### Przypadek uczciwy

* Trudność dla dominatora: 2
* Trudność dla kopaczy: 2 + 2

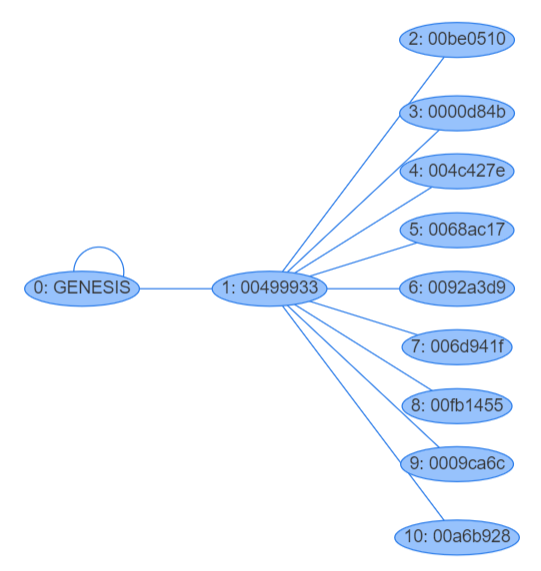
Przypadek w którym jeden węzeł dominuje, ale jest uczciwy sprawia, że wszystkie bloki są wykopane przez niego. Wynikiem przykładowego skryptu jest liczność bloków:

Miner Counts: { '5001': 9, undefined: 1 }

Łatwo zauważyć dominację 5001 oraz fakt, że nie dochodzi do żadnych forków. W przypadku założenia, że kopacz nie dostaje nagrody za dany blok taka sytuacja nie ma większego znaczenia, natomiast w prawdziwej sieci sprawiło by to, że 5001 bogacił by się znacznie szybciej niż inni. Mogło by to spowodować zarówno, że pozostali zrezygnowali by z Coina powodując jego upadek, ale mogło by to też zadziałać jako motywacja do zwiększenia możliwości obliczeniowych w celu rywalizacji.

### Złośliwy węzeł dominujący

Test zakładał włączenie złośliwości 5001 w postaci celowego rozwidlania łańcucha (kopie nowy blok nie dla poprzedniego bloku będącego końcem łańcucha tylko dla przedostatniego). Liczebności wykopanych bloków są takie same jak w poprzedniej podsekcji. To co jest ważne to wygląd łańcucha:



Powstały w tym scenariuszu łańcuch pokazuje pełną kontrolę węzła nad siecią. Żaden inny węzeł nie ma szansy wykopać bloku, który wydłuży łańcuch bo brakuje mu mocy, a węzeł dominujący celowo tego nie robi – oznacza to **całkowite zatrzymanie sieci**!

Ryzyko całkowitego zniszczenia sieci zależy od rozmiaru i parametrów trudności dla waluty. W bardzo rozległym systemie z odpowiednio krótkim czasem kopania istnieje szansa, że węzły-sąsiedzi tego, który inicjalizuje transakcje byli by w stanie wykopać ją szybciej niż rozpropagowała by się do daleko położonego dominatora. Nie zmienia to faktu, że nawet w optymistycznym przypadku sieć i jej działanie było by **zaburzone**.

### Propozycja rozwiązania

Najlepszą metodą wydaje się być znaczne **zwiększenie trudności hasha**. Dla bardzo wysokiej trudności element losowości przy poszukiwaniu będzie miał dużo większe znaczenie niż dla łatwych hashy, gdzie przeszukanie wszystkich kombinacji jest błyskawiczne. Spośród innych wymyślonych metod żadna nie wydawała się nawet podobnie skuteczna, co ta.

W celu przetestowania zmienione zostały parametry trudności:

* Trudność dla dominatora: 5
* Trudność dla kopaczy: 5 + 2

Miner Counts: { '5001': 7, undefined: 1 } (zatrzymane wcześniej z powodu czasochłonności)

Niestety nie pomogło to w badanym problemie. Hipotezą jest to, że **trudność jest wciąż zbyt mała**, a **dysproporcja między węzłami** (trudność +2) wręcz **nierealnie duża**, ale po zmniejszeniu jej do +1 dominacja nie była zbyt zauważalna nawet dla mniejszych trudności. Żeby upewnić się, że metoda rozwiązania ma sens należałoby sprawdzić jeszcze większe trudności, bądź zastosować inną metodę spowalniania innych węzłów np. poprzez krótki timeout na każdą próbę jednego nonce.

## Awaria węzła przegubowego

Scenariusz to sytuacja, w której awarii ulega połączenie typu most. Prowadzi to do utworzenia dwóch podsieci, które mogą kopać zupełnie inne transakcje. Test odbył się poprzez utworzenie sieci, gdzie 5002 jest węzłem przegubowym mającym dwa mosty. Pauzowanie go, czyli wyłączanie odpowiedzi na jakikolwiek broadcast symulowało awarię. W testach wykorzystywane są endpointy leave\_network i pause – więcej o nich w sekcji o nieautoryzowanej kontroli.

### Kontrolowane odłączenie

Test polegał na przetestowaniu działanie mechanizmu kontrolowanego odłączenia. Kiedy węzeł otrzyma sygnał wykonuje następujące czynności:

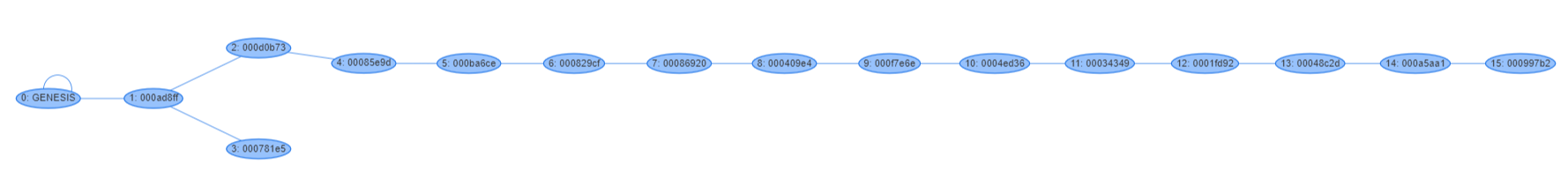
* Odpytuje swojego mastera (adres do którego dołączył na początku), czy może opuścić sieć – ma to na celu zapobiec sytuacji, w której sąsiedzi są przekierowywani do nowego węzła, który zaraz opuści sieć
* Po otrzymaniu zezwolenia wysyła wszystkim sąsiadom adres nowego mastera
* Węzeł się zatrzymuje

Metoda jest skuteczna i dobrze zapobiega powstawaniu nadmiarowych połączeń, ale niestety jest podatna na oszustwo. **Złośliwy węzeł master mógłby nie pozwalać na odłączenie się węzła**. Żeby temu zapobiec odłączający węzeł mógłby rozesłać wszystkim pełną listę sąsiadów, ale spowodowało by to tylko nadmiarowe połączenia a w przypadku, gdy jest to węzeł przegubowy, a jego master też się odłącza to **sieć została by rozcięta**. W aktualnej implementacji niestety żadna wersja nie jest w 100% pewna. Potencjalne rozwiązania w dalszych sekcjach.

Test w uczciwym scenariuszu polegał na:

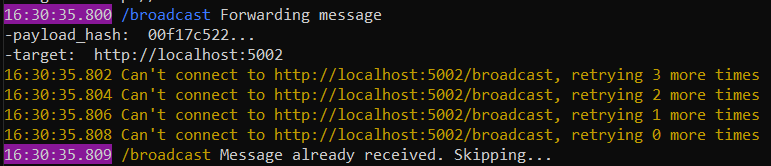
* Ustawienie sieci
* Wysłanie 2 transakcji startowych
* Wyłączenie węzła 5002 (bez systemu odłączania powodowałoby dziurę w sieci)
* 7 transakcji od 5001
* 5 transakcji od 5003

Po zakończeniu skryptu blockchain był identyczny dla obu 5001 i 5003, co potwierdza, że **mechanizm zadziałał w tym scenariuszu**.

  
Blockchain po badaniu (identyczny dla obu sieci)

### Trwałe wstrzymanie przekazywania danych

Podstawowa symulacja awarii pokazała reakcje okolicznych węzłów. Poniższy zrzut ekranu z węzła 5001 pokazuje, że **utrata połączenia została wykryta**. W aktualnej implementacji węzeł nie reaguje na to ostrzeżenie, więc **sieć przestaje działać**. Dodanie kodu, który spowodowałby kontrolowane wyłączenie się węzła z sieci i zwrócenie ostrzeżenia użytkownikowi po danej liczbie prób połączenia byłby jednak prosty do dodania.

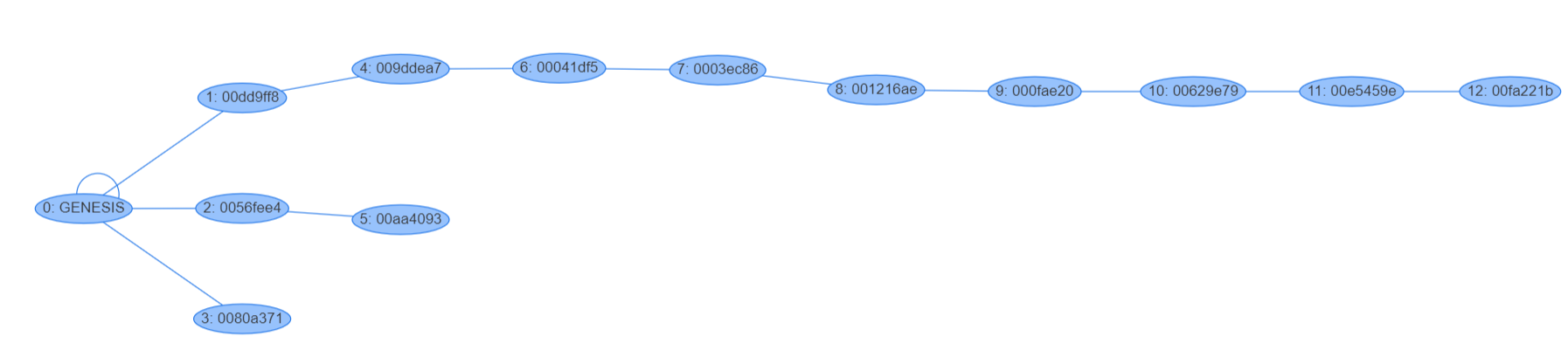
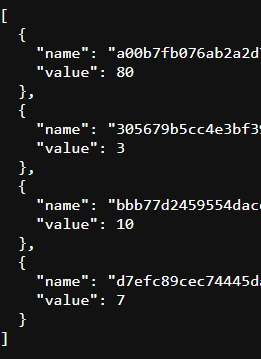
  
Zrzut ekranu z logów węzła 5001

### Tymczasowe wstrzymanie przekazywania danych

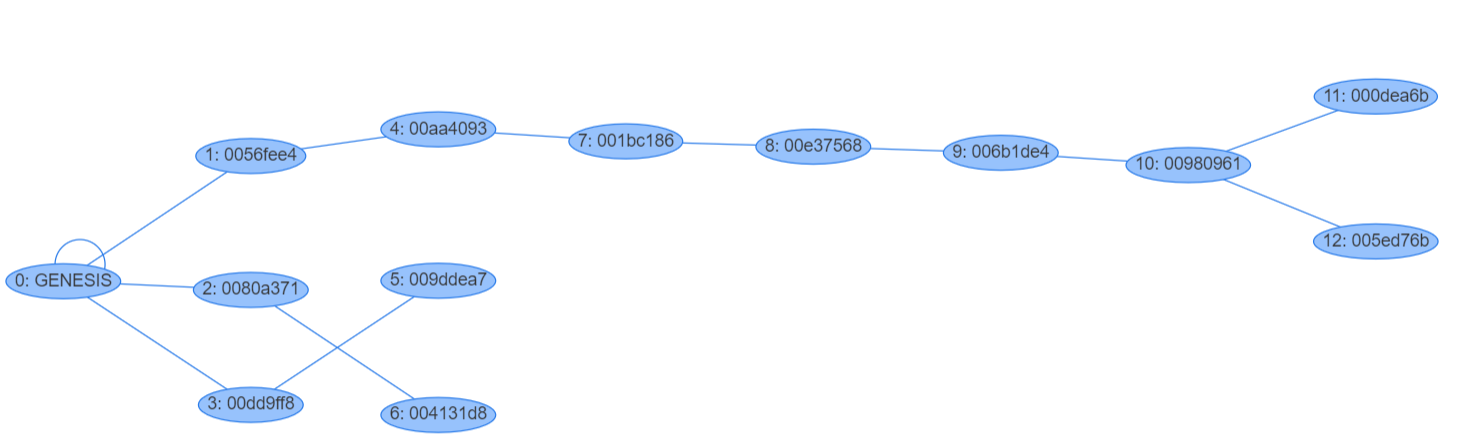
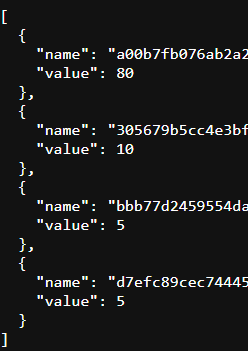
Ten scenariusz działa analogicznie do poprzedniego, z tą różnicą że węzeł przegubowy wraca po pewnym czasie. Zrealizowano to wykorzystując endpoint /pause węzła 5002. Scenariusz testu to:

* Utworzenie sieci
* Wysłanie 2 transakcji zasilających konta 5001 oraz 5003
* Wyłączenie mostu
* Wysłanie 7 transakcji dla podsieci 1
* Wysłanie 5 transakcji dla podsieci 2
* Włączenie mostu
* Wysłanie kilku (3) transakcji wywołujących mechanizm synchronizacji

Stan przed włączeniem mostu (do przedostatniego kroku włącznie).

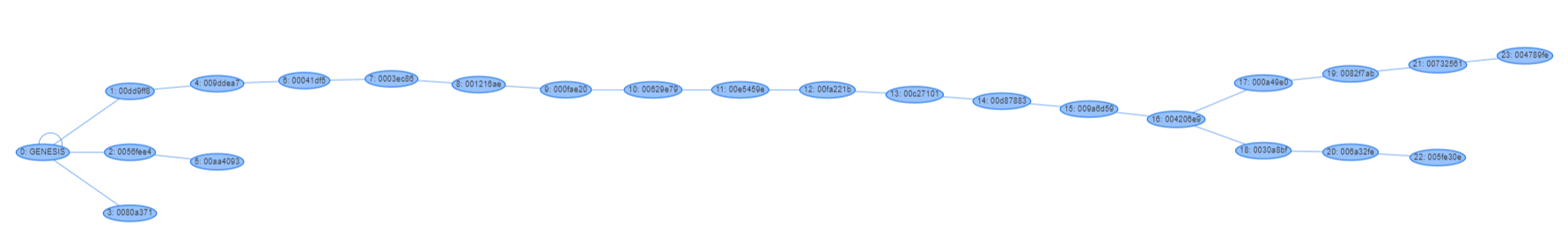
Podsieć 1 i stany kont (węzeł 5001)

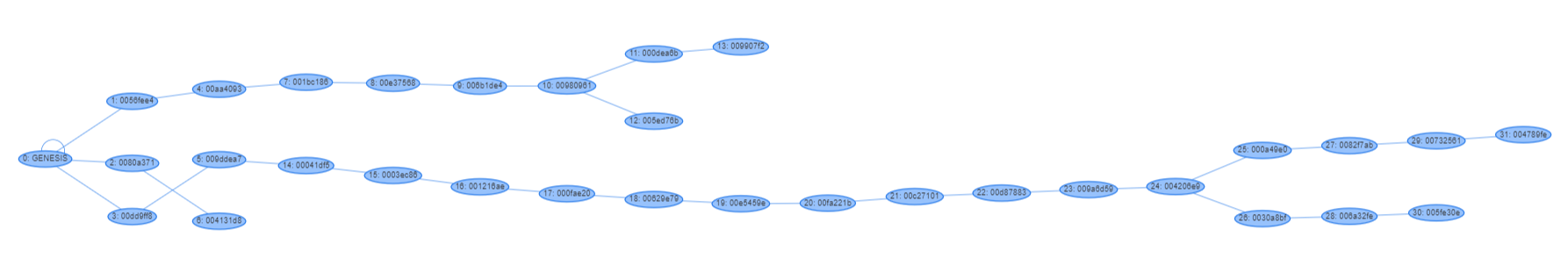
  


Podsieć 2 i stany kont (węzeł 5003)

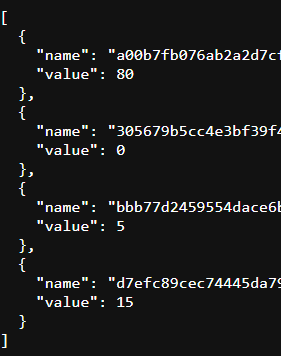
Można zaobserwować, że zaakceptowane (najdłuższe) łańcuchy są inne. Różnią się też długością – w podsieci 1 długość to 2 (wstępne zasilanie) + 7 (transakcje w podsieci 1), natomiast dla podsieci 2 jest to 2 + 5.

Po włączeniu mostu:

  
Podsieć 1 (5001) – blok końcowy 004789fe

  
Podsieć 2 (5003) – blok końcowy 004789fe

Dłuższy łańcuch podsieci wygrał i został zaakceptowany jako ten ważniejszy. Aby nie naruszać spójności sieci, wszystkie transakcje z podsieci 2 zostały wycofane i wykopane ponownie, dlatego długość nowego wspólnego łańcucha to 17 (2 wstępne + 7 podsieć 1 + 5 podsieć 2 + 3 sychronizujące). Stany kont sprawdzone po synchronizacji również były zgodne dla obu węzłów i poprawne biorąc pod uwagę skrypt rozsyłający:

  
Stany kont po synchronizacji (te same dla wszystkich węzłów)

Można więc powiedzieć, że sieć jest **dobrze przygotowana** na ten konkretny scenariusz.

### Złośliwy węzeł

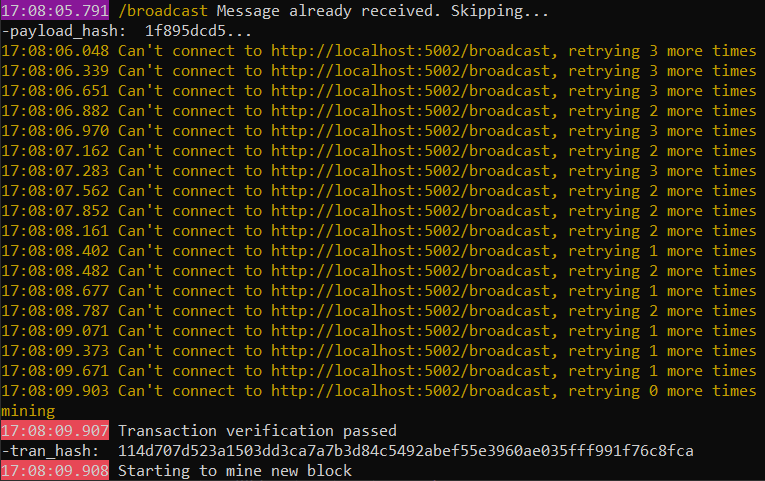
Złośliwość mogłaby polegać na wyłączeniu przekazywania wiadomości broadcast, ale w taki sposób by nie wywołać wykrycia awarii. W ramach testu odpowiedź pauzowanego węzła została zmodyfikowana tak, aby nie powodować błędów. W tym scenariuszu wszystko zadziałało tak samo, ale podsieci nigdy **nie dowiedziały się o awarii 5002**, co oznacza że jego złośliwość mogłaby spowodować **rozerwanie sieci na pomniejsze**.

### Problem nieskończonej odpowiedzi

W przypadku, gdy węzeł nie odpowiada to istnieje ryzyko zablokowania łącza i nieskończone oczekiwanie. Problem ten został napotkany w fazie tworzenia projektu i zabezpieczony parametrem metody fetch:

signal: AbortSignal.timeout(1500)

Parametr mówi, że połączenie zostanie porzucone po 1.5s bez odpowiedzi. Ustawienie to jest jednak dość nadmiarowe i prawdopodobnie mogłoby być mniejsze.

  
Zrzut logów z 5001

Wadą zaimplementowanego loggera jest to, że pokazuje czas wyświetlenia wiadomości, niekoniecznie wydarzenia, a przy wielu asynchronicznych zapytaniach czasem powstaje tłok. To co jest ważne na zrzucie to to, że mimo nieudanych połączeń praca była kontynuowana, czyli **ten problem został zabezpieczony.**

### Propozycja rozwiązania

Przy awarii łącza sytuacja jest dość prosta do rozwiązania. Awarię można rozpoznać poprzez wykrycie braku odpowiedzi na wysyłane do węzła wiadomości broadcast. Można również dla pewności okresowo wysyłać do sąsiadów prosty ping w celu diagnostycznym. W sytuacji utraty siedzi następnie należało by poczekać chwilę i spróbować ponownie, bądź od razu zwrócić użytkownikowi błąd i poprosić o ponowne dołączenie do sieci – tym razem podając inny adres docelowy połączenia.

W przypadku złośliwego węzła sytuacja się komplikuje:

Pierwszą sugestią jest tworzenie określonej liczby nadmiarowych połączeń przy dołączaniu do sieci. Dla poprawnej sieci jedno połączenie jest wystarczające jednak w celu zabezpieczenia można dołączyć np. do 3 węzłów, które już są w sieci. Sprawi to, że powstaną połączenia zapasowe, ale kosztem zwiększonej redundancji ruchu w sieci. Liczbę nadmiarowych połączeń można by było dobrać eksperymentalnie w celu uzyskania balansu. Ważne jest też, aby nadmiarowe połączenia prowadziły do jak najbardziej odległych węzłów, aby zminimalizować ryzyko że będą one w zmowie.

Drugą metodą jest okresowe wysyłanie sygnału przez węzeł główny INIT. Mógłby on rozsyłać raz na np. 5 minut wiadomość zaszyfrowaną kluczem prywatnym asymetrycznym, którą po odbiorze inne węzły rozszyfrowały by w celu weryfikacji, czy na pewno pochodzi od INIT. Jeśli wiadomość by nie docierała przez wyspecyfikowany czas to węzeł wiedziałby, że stracił połączenie. Nie jest to jednak pomysł warty dalszych rozważań, ponieważ daje „pole do popisu” dla złośliwych węzłów, które blokowały by broadcast, ale przepuszczały sygnał kontrolny.

## Nieautoryzowana kontrola nad węzłami

### Endpointy testowe - /pause, /leave\_network i inne

Pauzowanie to mechanizm powodujący zatrzymanie odpowiedzi węzła na broadcast. Opuszczanie sieci to kontrolowany sposób wyłączania węzła. Oba te endpointy są możliwe do użycia przez każdego atakującego – jest to jednak świadome i oznaczone. Służą one ułatwieniu dostępu i skryptowania w ramach testu. W ostatecznej wersji realnego programu (nie tylko proof of concept) należało by je usunąć i ewentualnie zastąpić czymś dostępnym przez GUI bądź konsolę z poziomu użytkownika.

Program zawiera też kilka endpointów testowych do odczytu danych, które np. zwracają listę sąsiadów. Jest to oczywiście naruszenie poufności – sytuacja w tym przypadku jest analogiczna co do poprzedniej.

Podsumowując, istnieją endpointy umożliwiające **nieautoryzowany dostęp**, ale są one **wykorzystywane do testów** i **oznaczone jako niebezpieczne**.

### Kontrolowane odłączenie

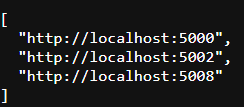
Test polegał na sprawdzeniu jakie skutki można wywołać ręcznie wysylając zapytania na endpointy odpowiedzialne za odłączenie.

#### DELETE /neighbors oraz PUT /neighbors

Poprzez wywołanie Powershell:

Invoke-WebRequest -Uri http://localhost:5001/neighbors -Method Delete -Body "[`"http://localhost:5008`"]" -ContentType "application/json"

Udało się doprowadzić do dodania niestniejącego węzła do listy sąsiadów:

  
Sąsiedzi 5001 po teście

Oznacza to, że endpoint umożliwia **tworzenie nadmiarowych i fałszywych połączeń** złośliwym aktorom. Metoda PUT pozwala na analogiczne działanie, ale trochę innym wywołaniem.

#### Inne

Ważnym faktem opartym o analizę endpointów związanych z kontrolowanym odłączeniem jest to, że **nie da się doprowadzić do usunięcia** żadnego węzła z listy sąsiadów. Jeśli węzeł faktycznie opuszcza sieć to będzie wywoływać awarie połączenia – inny mechanizm usuwający nieaktywne węzły spośród sąsiadów zajął by się tym problemem. **Pod kątem usuwania dostępu innym implementacja jest bezpieczna.**

## Nieuczciwy węzeł

### Akceptacja wszystkich transakcji

Test polegał na wysłaniu trzech niepoprawnych transakcji, a następnie jednej prawdziwej. W teście oszustem był 5001.



Bloki oszusta po teście (blok 4 odpowiada legalnej transakcji, 1-3 niepoprawnym)



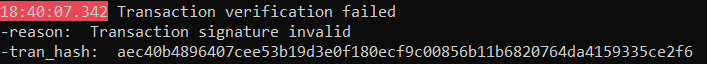
Bloki uczciwego kopacza po teście (blok 1 odpowiada legalnej transakcji)

Można zauważyć, że uczciwi kopacze dzięki weryfikacji bloków i transakcji nie dopuszczają nawet poprawnie wykopanych niespójnych transakcji. Ciekawym zjawiskiem jest to, że oszust nie przyjął poprawnego bloku wykopanego przez kogoś innego – jest tak, ponieważ nie węzły nie akceptują przestarzałych bloków. Można uznać, że **sieć jest odporna na tak prosty scenariusz złośliwości**.

### Podmiana wartości w transakcji

Podmiana odbywała się na etapie przetwarzania transakcji, po jej zaakceptowaniu złośliwy węzeł 5001 podwajał wartość przesyłanej waluty i podmieniał hash transakcji na nowy.

W teście prawdziwa transakcja była rozsyłana z węzła 5001.

  
Zrzut logów z węzła 5002

Jak widać weryfikacja transakcji nie powiodła się i transakcja nie została rozesłana. Błędem jest nieprawidłowość podpisu – oczywiście oszust nie może podpisać transakcji bez dostępu do czyjegoś klucza prywatnego, a jeśli ma do niego dostęp to może robić co chce bez „zabawy” w podwajanie i rehashowanie.

Ze względu na fakt, że sprawdzany jest hash całego zestawu danych i jego podpis to wiadomo, że **system jest odporny na jakiekolwiek podmiany danych transakcji**.

### Podmiana wartości bloku

Proces analogiczny do podmiany w transakcji, ale ze względu na umiejscowienie przetwarzania bloków lepiej było przeprowadzać testy dla węzła 5000 jako kopacza (zamiast 5001, 5004 jako kopacz bez zmian). Sprawdzone zostały scenariusze:

* zmiany danych
* zmiana danych i rehash
* zmiana danych i wstawienie łańcucha znaków z odpowiednią ilością zer jako hasha

Żaden z nich się nie powiódł. Mimo niedużej liczby testów wyczerpują one temat podmiany danych i pokazują **bezpieczeństwo systemu w tym kontekście**.

## Działanie w zmowie

## Ataki na portfel

## Utrata danych logowania/klucza

## Błędne dane i żądania

## Złamanie algorytmów kryptograficznych