Zastosowanie technologii blockchain oraz rozproszonego systemu plików do budowy aplikacji zdecentralizowanych

Michał Sieczkowski

Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Abstrakt. Aplikacje zdecentralizowane mogą zrewolucjonizować sposób w jaki prowadzonych jest wiele działalności sektora usług. Mają potencjalne zastosowania w bankowości, ubezpieczeniach, administracji publicznej, handlu internetowym, kształceniu na odległość czy nawet hazardzie. Pozwalają budować systemy nie wymagające centralnych zarządców i zaufanych pośredników, często obniżając przy tym konieczne do zapłacenia prowizje i chroniąc użytkowników przed cenzurą.

W tej pracy przedstawiłem technologię blockchain oraz opartą na niej platformę Ethereum umożliwiającą tworzenie inteligentnych kontraktów, wykorzystywanych w aplikacjach zdecentralizowanych. Omówiłem rozproszony system plików IPFS oraz koncepcje zastosowania go do zmniejszenia ilości danych przechowywanych w łańcuchu bloków. Przeanalizowałem również zalety aplikacji zdecentralizowanych względem standardowych aplikacji internetowych w architekturze klient-serwer.

Słowa kluczowe: Blockchain · Smart Contract · Ethereum · Decentralised Application · DApp · InterPlanetary File System · IPFS

1 Wprowadzenie

Opracowanie kryptowaluty Bitcoin wywołało przełom w dziedzinie systemów rozproszonych. W swoim słynnym artykule [10] Satoshi Nakamoto przedstawił algorytm Proof-of-Work, będący rozwiązaniem problemu bizantyjskich generałów [5] opartym na teorii prawdopodobieństwa. Dzięki temu pomysłowi udało się osiągnąć zdecentralizowany konsensus w sieci Peer-to-peer i możliwe stało się stworzenie pierwszej elektronicznej waluty, niezależnej od żadnej centralnej instytucji emisyjnej. Technologia blockchain, na podstawie której działa Bitcoin znalazła zastosowanie w wielu innych dziedzinach. W 2013 r. programista Vitalik Buterin zaproponował koncepcję rozproszonej platformy obliczeniowej Ethereum [3]. Pozwala ona na uruchamianie skryptów operujących na łańcuchu bloków, napisanych w dedykowanym, zupełnym w sensie Turinga języku programowania. W praktyce oznacza to możliwość tworzenia aplikacji korzystających z blockchaina jak ze swoistej bazy danych.

W niniejszej pracy starałem się zaprezentować koncepcję aplikacji zdecentralizowanych oraz narzędzia, które umożliwiają ich tworzenie. Rozpocząłem od ogólnego spojrzenia na technologię blockchain oraz przedstawienia najważniejszych, wspólnych dla większości jej implementacji właściwości. Na bazie tych informacji w kolejnym punkcie omówiłem platformę Ethereum, obsługującą inteligentne kontrakty. Następnie przybliżyłem tytułowy rozproszony system plików - InterPlanetary File System. Na koniec zdefiniowałem aplikacje zdecentralizowane oraz opisałem ich zalety względem aplikacji internetowych w architekturze klient-serwer.

2 Technologia blockchain

Blockchain jest to rozproszona baza danych oparta na sieci P2P. Dane zapisywane są w blokach tworzących rosnącą listę, która replikowana jest na wielu węzłach sieci. Nie ma w niej centralnego zarządcy, a każdy węzeł pełni równocześnie rolę klienta i serwera. Wspomniana lista bloków, zwana popularnie łańcuchem, jest odporna na modyfikacje bloków dodanych w przeszłości. Jest to możliwe dzięki zastosowaniu kryptograficznej funkcji skrótu[1]. Każdy blok ma w swoim nagłówku specjalne pole zawierające skrót bloku poprzedniego. Dzięki temu, jakakolwiek zmiana danych w bloku N, spowoduje zmianę wyniku funkcji skrótu, który jest zapisywany w nagłówku bloku N+1 i konieczność jego ponownego zapisania. To z kolei spowoduje zmianę skrótu bloku N+1, który jest przechowywany w nagłówku bloku N+2 itd. Blockchain można sobie również wyobrazić jako maszynę stanową, w której dołączenie kolejnego bloku powoduje przejście do nowego stanu. Bloki grupują transakcje, które w przypadku Bitcoina reprezentują przelewy kryptowaluty z adresu na adres.

2.1 Algorytm konsensusu

Dowolny węzeł sieci P2P może zgłosić transakcje, które chce żeby zostały zapisane w następnym bloku. Może także zbierać transakcje od innych węzłów i brać udział w procesie tworzenia nowego bloku. Aby wszystkie węzły mogły uzgodnić wspólną wersję łańcucha, potrzebny jest algorytm, zgodnie z którym dojdą do konsensusu. Powinien on również uniemożliwiać pojedynczemu węzłowi zyskanie dominującej pozycji przy budowaniu nowych bloków, ponieważ byłoby to sprzeczne z ideą decentralizacji.

Proof-of-Work W tym algorytmie węzeł, który chce dołączyć nowy blok musi dostarczyć dowód, że wykonał pracę polegającą na rozwiązaniu problemu obliczeniowego [6]. Inne węzły akceptują nowy blok po sprawdzeniu poprawności transakcji w nim zawartych oraz zweryfikowaniu dowodu pracy. Za udane dołączenie bloku do łańcucha węzeł otrzymuje nagrodę w postaci środków w kryptowalucie. Mało prawdopodobne jest, że jeden węzeł wyprodukuje dwa bloki

[1] W dalszej części tej pracy przez *skrót* należy rozumieć wynik kryptograficznej funkcji skrótu.

pod rząd, ponieważ konkuruje on z mocą obliczeniową wszystkich pozostałych węzłów biorących udział w konkursie. W przypadku, gdy dwóm różnym węzłom uda się zbudować nowy blok w tym samym czasie, w łańcuchu pojawi się rozgałęzienie. Po sieci P2P będą krążyły dwie konkurujące wersje łańcucha. Kolejne węzły zaczną budować nowe bloki na podstawie tej wersji, która dotrze do nich szybciej. W końcu jedna z gałęzi łańcucha stanie się dłuższa i zostanie uznana za obowiązującą, ponieważ wykonano na niej najwięcej pracy. Proces tworzenia nowych bloków nazywany jest popularnie kopaniem, a węzły biorące w nim udział - górnikami.

2.2 Problem podwójnego wydawania

Przy próbie zaimplementowania cyfrowej waluty pojawia się tzw. problem podwójnego wydawania [7]. Załóżmy, że mamy łańcuch o długości 50 bloków i akceptujemy wszystkie transakcje w nim zawarte. Nieuczciwy kupujący tworzy transakcje transferującą 1 BTC[2] na adres sprzedawcy, która zostaje uwzględniona przez sieć w bloku 51. W tym samym momencie rozpoczyna kopanie alternatywnej wersji łańcucha, w której zamiast do sprzedawcy przelewa środki na inny adres, będący pod jego kontrolą. Nie udostępnia jej na razie pozostałym węzłom. Sprzedający po zweryfikowaniu, że na jego adresie pojawiły się środki przesyła kupującemu dobro cyfrowe. Kupującemu udaje się sprawić, że jego gałąź zawiera więcej bloków niż ta aktualnie obowiązująca. Po otrzymaniu dobra cyfrowego udostępnia swoją wersję łańcucha, a sieć ją akceptuje ponieważ jest najdłuższa. Dzięki zastosowaniu algorytmu Proof-of-Work, taki scenariusz jest bardzo trudny do powtórzenia, ponieważ wymagałoby to posiadania więcej niż połowy mocy obliczeniowej całej sieci.

3 Ethereum

Projekt Ethereum powstał w odpowiedzi na rosnące zainteresowanie wykorzystaniem technologii blockchain w innych obszarach niż kryptowaluty. Zamiast budować osobne łańcuchy bloków do różnych zastosowań, Vitalik Buterin zasugerował uniwersalną platformę do tworzenia aplikacji zdecentralizowanych [3]. Jest ona oparta na jednym łańcuchu bloków i algorytmie konsensusu typu Proof-of-Work. Ethereum można traktować jako wirtualny komputer, w którym instrukcje kodu maszynowego są wykonywane z taką samą niezawodnością co transakcje Bitcoinowe. Platforma wyposażona jest w kryptowalutę Ether, którą płaci się za przechowywane dane oraz wykonywane obliczenia. Jednostką miary wysiłku jaki ponosi sieć jest *Gas*. Jej nazwa nie bez powodu przywodzi na myśl benzynę. Tak jak silnik samochodu potrzebuje do działania paliwa, tak transakcje w Ethereum "palą" Gas.

[2] BTC - skrótowe oznaczenie kryptowaluty Bitcoin.

3.1 Konta

W Ethereum wyróżniamy dwa rodzaje kont: zewnętrzne (ang. externally owned) oraz kontraktowe. Stan globalny sieci jest zbiorem stanów wszystkich kont [11].

Konto zewnętrzne. Ma adres stanowiący ostatnie 20 bajtów skrótu Keccak-256 klucza publicznego i jest kontrolowane przez tego kto posiada odpowiadający mu klucz prywatny. Struktura konta zewnętrznego zawiera dwa pola: saldo w walucie Ether oraz licznik wykonanych transakcji.

Konto kontraktowe. Może powstać w wyniku transakcji wykonanej za pomocą konta zewnętrznego. Po utworzeniu jego działaniem steruje kod inteligentnego kontraktu. Posiada pamięć, w której przechowywane są zmienne reprezentujące jego stan. Jego adres wyznaczany jest na podstawie adresu oraz licznika transakcji konta zewnętrznego, które je utworzyło. Struktura konta kontraktowego, oprócz salda i licznika transakcji, zawiera dodatkowo skrót kontrolującego kodu oraz skrót korzenia drzewa pamięci.

3.2 Transakcje

Transakcje w Ethereum można podzielić na trzy kategorie: przelewy kryptowaluty Ether, transakcje tworzące konta kontraktowe oraz wywołania funkcji inteligentnych kontraktów [11]. Przelewy mogą być wysyłane i odbierane przez oba typy kont. Dowolne konto może również wykonać transakcję utworzenia konta kontraktowego oraz wywołania funkcji. Struktura transakcji zawiera pola reprezentujące: adres odbiorcy, podpis cyfrowy (identyfikujący zarazem nadawcę), przesyłaną kwotę w walucie Ether, numer transakcji, dane, *Gas Limit* oraz *Gas Price*. Transakcja utworzenia konta kontraktowego ma dodatkowo pole *Init* zawierające kod inteligentnego kontraktu. Pole danych wykorzystywane jest przy transakcjach wywołujących funkcje. Zapisywane są w nim identyfikator wywoływanej funkcji oraz przykazywane argumenty. Gas Limit to maksymalna liczba jednostek paliwa jaką chcemy aby transakcja spaliła. Jest to mechanizm chroniący nadawcę przed nieskończonymi pętlami w inteligentnych kontraktach, które mogłyby pozbawić go wszystkich środków. Gas Price to cena jaką nadawca transakcji chce zapłacić za jednostkę paliwa. W teorii im wyższa cena, tym chętniej górnik dołączy transakcję do kolejnego bloku. Maksymalna prowizja jaką zapłaci nadawca jest iloczynem Gas Limit i Gas Price, a nadwyżka za niewykorzystane jednostki paliwa zwracana jest na jego konto.

3.3 Drzewo Merkle Patricia

Strukturą danych wielokrotnie wykorzystywaną w Ethereum jest *drzewo Merkle Patricia* [12]. Jest ono połączeniem dwóch innych struktur dziedziczącym ich zalety: drzewa skrótów (ang. Merkle Tree) [8] oraz skompresowanego drzewa trie (ang. Patricia Trie) [9]. Drzewo Merkle Patricia używane jest do sprawdzania

integralności danych wymienianych między węzłami sieci P2P. Pozwala w szybki i bezpieczny sposób udowodnić, że pewne dane należą do konkretnego bloku, bez konieczności posiadania wszystkich pozostałych danych w nim zawartych. Dzięki tej właściwości węzły mogą pobierać tylko te dane które są dla nich istotne i wciąż mieć pewność, że zostały one dołączone do łańcucha bloków.

3.4 Struktura nagłówka bloku

Nagłówek bloku w Ethereum zawiera 15 różnych elementów [11]. Najważniejsze z nich to: skrót poprzedniego bloku, znacznik czasu określający moment powstania bloku oraz skróty korzeni trzech drzew: stanu, transakcji i pokwitowań. Drzewo stanu przechowuje struktury wszystkich kont w Ethereum, drzewo transakcji - struktury wszystkich transakcji dodanych w ramach bloku, a drzewo pokwitowań - struktury pokwitowań odpowiadających tym transakcjom. Pokwitowania zawierają takie informacje jak liczba zużytych przez transakcję jednostek paliwa oraz logi generowane przez inteligentne kontrakty.

3.5 Maszyna Wirtualna Ethereum

Środowiskiem uruchomieniowym dla inteligentnych kontraktów jest Maszyna Wirtualna Ethereum. W jej specyfikacji [13] opisany jest zestaw obsługiwanych instrukcji maszynowych. Na ich podstawie, tworzone są języki wysokiego poziomu ułatwiające zrozumienie działania kodu inteligentnych kontraktów. Obecnie najpopularniejszym z nich jest Solidity, którego autorzy wzorowali się na językach Python, C++ oraz JavaScript. Solidity jest statycznie typowane, wspiera dziedziczenie i ponowne wykorzystanie kodu oraz umożliwia tworzenie własnych typów złożonych.

4 InterPlanetary File System

IPFS to protokół oparty o sieć P2P, której węzły tworzą rozproszony system plików. Z założenia ma być trwałą i zdecentralizowaną metodą na przechowywanie i udostępnianie plików. Dziedziczy wiele sprawdzonych rozwiązań z projektów takich jak BitTorrent, Git czy SFS [2].

4.1 Tożsamość węzłów sieci

Węzeł przed przyłączeniem do sieci generuje parę kluczy: publiczny i prywatny. Identyfikator węzła *NodeId* jest skrótem klucza publicznego. Aby zapobiec atakom typu Sybil [4], IPFS wykorzystuje mechanizm spowalniający proces tworzenia nowego identyfikatora zaczerpnięty z protokołu S/Kademlia [1]. Akceptowany jest tylko taki NodeId, którego skrót posiada d zerowych bitów na najbardziej znaczących pozycjach. W praktyce oznacza to szukanie dopuszczalnej wartości identyfikatora, poprzez generowanie kolejnych par kluczy, na zasadzie prób i błędów. Parametr d (*difficulty*) dobierany jest tak, aby nie odbywało się to zbyt szybko. Operacja znalezienia NodeId ma złożoność obliczeniową $O(2^d)$.

4.2 Adresowanie na podstawie treści

Każdy plik zapisany w IPFS, posiada adres będący jego skrótem [2]. Dzięki temu, węzeł sieci P2P po pobraniu pliku, może sprawdzić jego integralność wyznaczając skrót i porównując go z adresem. Adresowanie na podstawie treści ma jeszcze jedną ważną zaletę: pozwala zmniejszyć redundancję danych przechowywanych przez węzły. Dwa pliki o jednakowej treści, posiadają ten sam, niezmienny adres i są traktowane w sieci jako jeden plik. Ze względu na powyższe cechy, IPFS znajduje zastosowanie do przechowywania danych wykorzystywanych w aplikacjach zdecentralizowanych. W pamięci inteligentnego kontraktu, można rejestrować adresy do plików z IPFS, zawierających duże ilości danych, których zapisanie w łańcuchu bloków byłoby kosztowne.

4.3 Niezmienne ścieżki do zmiennych danych dzięki IPNS

Adresy będące skrótami danych, w niektórych sytuacjach mają również wady. Załóżmy, że chcemy udostępnić w IPFS aplikację internetową. Wraz z wprowadzaniem poprawek do jej kodu, zmianie ulegać będzie jej adres. Udostępnianie użytkownikom nowego adresu z każdą kolejną wersją aplikacji jest niepraktyczne. InterPlanetary Name System został stworzony aby rozwiązać ten problem [2]. Każdy węzeł sieci ma własną przestrzeń nazw z adresami zaczynającymi się od `/ipns/<NodeId>`. Może w niej udostępniać pliki i katalogi podpisane swoim kluczem prywatnym. Inne węzły przy pobieraniu spod adresów z IPNS, sprawdzają autentyczność danych weryfikując zgodność podpisu cyfrowego z kluczem publicznym oraz NodeId.

5 Aplikacje zdecentralizowane

Pojęcie aplikacji zdecentralizowanych nie zostało jeszcze precyzyjnie określone w literaturze. Większość definicji jako cechę wyróżniającą takie aplikacje od innych podaje wykorzystanie technologii blockchain. Często przytaczanym atrybutem jest fakt posiadania otwartego, niezależnie rozwijanego przez społeczność użytkowników kodu źródłowego. W moim odczuciu do powyższych charakterystyk należy dodać jeszcze jedną cechę. Aby aplikacja była w pełni zdecentralizowana, powinna być udostępniona w rozproszonym systemie plików. W przypadku hostingu na centralnym serwerze istnieje podatność na ataki odmowy dostępu, które mogą uniemożliwić użytkownikom korzystanie z aplikacji. Inteligentny kontrakt na podstawie, którego działa aplikacja zdecentralizowana będzie nadal dostępny w łańcuchu bloków, ale przeciętny użytkownik nie będzie potrafił ręcznie wysyłać transakcji. Można sobie wyobrazić scenariusz w którym atakującemu zależy na tym, aby jakiś użytkownik nie podjął akcji, w wyniku czego inteligentny kontrakt przejdzie do stanu korzystnego dla atakującego.

5.1 Zalety

Aplikacje zdecentralizowane tworzone na Ethereum oraz inne platformy wspierające inteligentne kontrakty, w porównaniu z aplikacjami w architekturze klient-serwer mają szereg istotnych zalet:

- Brak pojedynczego, wrażliwego na awarie punktu w postaci centralnego serwera (ang. single point of failure). Dane zreplikowane są na wielu węzłach sieci P2P i dopóki któryś z nich działa, można uzyskać do nich dostęp.
- Większa odporność na ataki hakerskie - uzyskanie kontroli nad dużą liczbą węzłów, wymaga więcej pracy niż zaatakowanie pojedynczego podmiotu. Ważne jest, aby węzły korzystały z różnych implementacji protokołu blockchain, tak, aby podatność wykryta w jednej z nich, nie umożliwiała przejęcia kontroli nad całą siecią.
- Automatyczne uwierzytelnianie - każdy użytkownik aplikacji zdecentralizowanych posiada konto w postaci pary kluczy: publicznego i prywatnego. Za ich pomocą potwierdza swoją tożsamość i podpisuje transakcje w sieci blockchain. Tworzenie dodatkowego mechanizmu logowania jest więc zbędne, a jedno konto może być jednocześnie wykorzystywane w różnych aplikacjach.
- Wbudowana kryptowaluta - nie trzeba integrować aplikacji z zewnętrznym systemem płatności internetowych aby umożliwić użytkownikom przesyłanie środków.
- Transparentność - aplikacje zdecentralizowane składają się z kodu warstwy prezentacji (JavaScript), do którego jest dostęp z poziomu przeglądarki internetowej oraz kodu bajtowego inteligentnego kontraktu, który jest publicznie widoczny w łańcuchu bloków. Użytkownicy mogą więc samodzielnie sprawdzić, czy twórca aplikacji nie zawarł w niej luk w zabezpieczeniach, które mogłyby być wykorzystane w przyszłości.

6 Podsumowanie

Technologie zaprezentowane w tej pracy pozwalają na tworzenie nowego typu aplikacji internetowych. Działają one dzięki kolektywnemu wysiłkowi korzystających z nich osób: poprzez współtworzenie kodu źródłowego, udostępnianie mocy obliczeniowej potrzebnej do osiągnięcia rozproszonego konsensusu oraz dzielenie się plikami w sieci P2P. Taki model zasadniczo różni się od tradycyjnych scentralizowanych rozwiązań i ze względu na przedstawione wcześniej zalety, powinien być brany pod uwagę przy tworzeniu nowych systemów.

Literatura

1. Baumgart, I., Mies, S.: S/kademlia: A practicable approach towards secure key-based routing. In: 2007 International Conference on Parallel and Distributed Systems. pp. 1–8 (Dec 2007). https://doi.org/10.1109/ICPADS.2007.4447808
2. Benet, J.: IPFS - content addressed, versioned, P2P file system. CoRR **abs/1407.3561** (2014), http://arxiv.org/abs/1407.3561

3. Buterin, V.: "A Next-Generation Smart Contract and Decentralized Application Platform" — Ethereum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
4. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems. pp. 251–260. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_24
5. Fedotova, N., Veltri, L.: Byzantine generals problem in the light of P2P computing. In: 2006 3rd Annual International Conference on Mobile and Ubiquitous Systems - Workshops. pp. 2–3 (July 2006). https://doi.org/10.1109/MOBIQW.2006.361758
6. Gramoli, V.: From blockchain consensus back to Byzantine consensus. Future Generation Computer Systems pp. 2–3 (2017). https://doi.org/10.1016/j.future.2017.09.023
7. Karame, G.O., Androulaki, E., Roeschlin, M., Gervais, A., Čapkun, S.: Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. ACM Trans. Inf. Syst. Secur. **18**(1) (May 2015). https://doi.org/10.1145/2732196
8. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology — CRYPTO '87. pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
9. Morrison, D.R.: PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric. J. ACM **15**(4), 514–534 (Oct 1968). https://doi.org/10.1145/321479.321481
10. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf (2008)
11. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf pp. 3–5 (2014), Byzantium Version, commit: 12779ac - 2018-11-27
12. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf pp. 19–20 (2014), Byzantium Version, commit: 12779ac - 2018-11-27
13. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf pp. 25–36 (2014), Byzantium Version, commit: 12779ac - 2018-11-27