POZNAN UNIVERSITY OF TECHNOLOGY

Filip Waligórski

Rozgłaszanie danych w grafach dużej skali

Praca magisterska

Promotor: dr Anna Kobusińska

Poznań, 2017

Spis treści

Rozdział 1

Wstęp

Rozdział 2

Istniejące rozwiązania

W tym rozdziale zaprezentowano istniejące komunikatory dla dwóch osób oraz komunikatory grupowe. Ich cele i funkcjonalność są zbliżone choć realizują je z różnymi założeniami oraz bazując na różnych architekturach i koncepcjach. Poniżej opisane zostały wybrane rozwiązania z naciskiem na cechy wyróżniające je spośród konkurencyjnych aplikacji.

Facebook Messenger [1]

Jest to jeden z najpopularniejszych obecnie komunikatorów. Oferuje zarówno czat dla 2 osób jak i grupowy. Wspiera wysyłanie wszelkich multimediów i plików oraz dostarczanie wiadomości pod nieobecność nadawcy. Dostępny jest na najszerszej gamie platform — jako aplikacja webowa, mobilna oraz desktopowa, czym wyróżnia się na tle konkurencji. Architektonicznie Messenger polega na „centralnym serwerze" przekazującym wiadomości. Cudzysłów wynika z faktu, że pod pojęciem „serwer" kryje się ogromna infrastruktura złożona z wielu maszyn, którą firma musi utrzymywać. Wadami tego komunikatora jest choćby brak wsparcia szyfrowania wiadomości czy fakt, że nie jest to oprogramowanie open-source.

Bleep [2]

Bleep jest komunikatorem zaprojektowanym przez firmę rozwijającą protokół BitTorrent. Do dyspozycji użytkowników oddano aplikację mobilną oraz aplikację desktopową (brak aplikacji webowej). Kod źródłowy aplikacji nie został udostępniony (nie jest to open-source), co oznacza, że użytkownicy nie mogą upewnić się, czy aplikacja działa i została zaimplementowana zgodnie z założeniami. Bleep oferuje rozmowy dla dwóch osób, a w planach twórców jest zaimplementowanie komunikacji grupowej. Wiadomości są szyfrowane przed wysłaniem na urządzeniu nadawcy i odszyfrowywane na urządzeniu odbiorcy — szyfrowanie end to end.

Jednak najważniejszą cechą wyróżniającą ten komunikator jest jego architektura - brak centralnego serwera pośredniczącego w przekazywaniu wiadomości. Komunikaty przesyłane są bezpośrednio między urządzeniami, jeśli oba są dostępne w momencie wysyłania, a w przeciwnym przypadku wiadomość umieszczana jest w DHT (Distributed Hash Table) i przechowywana do czasu odebrania jej. Specjalny mechanizm dba o to, by wiadomość nie

zniknęła z DHT wcześniej. Dane o koncie użytkownika oraz klucze szyfrujące pozostają lokalnie na urządzeniu.

Signal [3]

Twórcy aplikacji Signal skupili się przede wszystkim na bezpieczeństwie i prywatności użytkowników. Wiadomości są szyfrowane na urządzeniach więc pomimo faktu, że architektura zakłada obecność centralnego serwera, wiadomości przechowywane na nim nie mogą zostać odczytane przez osoby trzecie. Kod źródłowy jest dostępny publicznie co oznacza, że każdy może sprawdzić zgodność implementacji z oferowanymi założeniami. Podobnie jak w przypadku aplikacji Bleep, dostępne są natywne aplikacje mobilna i desktopowa. Możliwe jest prowadzenie rozmowy grupowej pomimo zastosowania szyfrowania wiadomości — treść zostaje zaszyfrowana symetrycznie (jedna wersja dla wszystkich odbiorców niezależnie od ich liczby), a następnie sam klucz jest szyfrowany zgodnie z oczekiwaniami każdego z odbiorców z osobna. Dzięki temu mechanizmowi uniknięto sytuacji, w której nadawca musiałby przygotować n wersji całej, potencjalnie dużej wiadomości dla n odbiorców.

Podobne rozwiązania: Wire, WhatsApp, Telegram, Allo

Darkwire [4]

Darkwire to aplikacja open-source oferująca komunikator grupowy z dostępem poprzez stronę internetową (aplikacja webowa). W przeciwieństwie do większości rozwiązań użytkownik nie musi tworzyć konta by skorzystać z programu. W celu skomunikowania się z użytkownikami należy wymienić między nimi identyfikator czatu (link do konkretnego pokoju) za pośrednictwem innego kanału komunikacyjnego (np. e-mail). Takie rozwiązanie zakłada, że identyfikator nie zostanie odgadnięty przez osoby trzecie — w przeciwnym przypadku będą one mogły odczytać wysyłane wiadomości. Architektura zakłada istnienie centralnego serwera uczestniczącego w przekazywaniu wiadomości. Z racji faktu, że aplikacja ma otwarte źródła, każdy może uruchomić swój własny serwer. Komunikaty są szyfrowane na urządzeniu (w przeglądarce) przed wysłaniem, zatem serwer nie zna treści wiadomości. Centralny serwer przesyła wiadomości tylko do tych uczestników, którzy są dostępni w momencie nadania wiadomości (brak wsparcia dla odbierania starszych wiadomości czy wysyłania wiadomości do użytkowników niedostępnych w danej chwili).

Friends [5]

Ten niszowy projekt open-source oferuje aplikację desktopową i umożliwia czat grupowy. Szyfrowanie wiadomości nie zostało do tej pory zrealizowane, ale jest jednym z punktów przyszłego rozwoju. Głównym celem twórców było stworzenie programu niezależnego od centralnego serwera oraz umożliwiającego rozmowę przy użyciu alternatywnych kanałów komunikacyjnych (np. poprzez Bluetooth) w sytuacji gdy połączenie internetowe jest niedostępne. Aplikacja wykorzystuje algorytm plotkowania (gossiping) oraz replikuje wiadomości przy użyciu drzewa skrótów (hash tree, Merkle DAG, DAG - Directed Acyclic Graph). Dzięki temu wiadomości w czacie mogą zostać połączone nawet w przypadku,

gdy ktoś nadał komunikaty będąc odłączonym od sieci — mechanizm podobny do łączenia zmian w repozytorium kodu. Gwarantuje to ostateczną spójność — przykładowy scenariusz dla 3 użytkowników:

1. Wiadomość wysłana przez użytkownika A została odebrana przez użytkownika B, który dołączył ją do swojego drzewa wiadomości.
2. Użytkownik A stał się niedostępny.
3. Użytkownik C stał się dostępny i odebrał od użytkownika B zmienioną wersję drzewa i w ten sposób dowiedział się o wiadomości wysłanej przez użytkownika A pomimo faktu, że ten jest w tej chwili niedostępny.

Tox [6]

Tox jest z założenia rozproszonym i szyfrowanym protokołem do wymiany wiadomości. Powstało kilkanaście implementacji klientów obsługujących go co pozwala na komunikowanie się z użytkownikami różnych aplikacji. Wśród zaimplementowanych aplikacji są programy na komputery stacjonarne oraz smartfony. Wiadomości są przesyłane bezpośrednio między nadawcą i odbiorcą dlatego obie strony muszą być dostępne jednocześnie. Brak wsparcia dostarczania wiadomości gdy jedna strona jest niedostępna to duża wada wszystkich aplikacji implementujących ten rodzaj transmisji P2P. Jednym z rozwiązań tego problemu zaproponowanym przez twórców protokołu jest skorzystanie z serwerów, którym użytkownik ufa i których zadaniem jest przekazywanie wiadomości do odbiorcy pod nieobecność nadawcy. Narusza to jednak założenie o rozproszeniu systemu (braku centralnych węzłów). Wsparcie dla komunikacji grupowej jest jednym z celów rozwoju protokołu.

ZeroChat, ZeroMail, BitMessage [7][8]

Przytoczone aplikacje realizują pomysły na komunikatory oparte o mechanizm podobny do transakcji kryptowalutowych. Wysłanie wiadomości wymaga obliczenia funkcji skrótu z zadanym prefiksem (proof of work) i umieszczenia bloku w łańcuchu (blockchain). Samo tylko wyliczenie funkcji skrótu powinno z definicji zająć około 4 minut [9], podczas gdy pozostałe komunikatory dążą do uzyskania czasu dostarczenia wiadomości bliskiego zeru (rozmowa w czasie rzeczywistym). Mimo tej znaczącej wady należy potraktować te projekty jako próbę stworzenia rozwiązania o innej architekturze niż dotychczas zaprezentowane (centralny serwer lub P2P). Być może w przyszłości wady uda się zminimalizować, a zalety architektury opartej o blockchain okażą się kluczowe.

Rozdział 3

Koncepcja

W niniejszym rozdziale opisano technologie i protokoły wykorzystane do opracowania koncepcji i implementacji rozproszonego komunikatora grupowego. Konkretnie, są to: protokół BitTorrent przedstawiony w punkcie 3.1 oraz jego implementacja w języku JavaScript — WebTorrent — punkt 3.2.

BitTorrent [10]

BitTorrent to protokół komunikacyjny pozwalający na wymianę i dystrybucję plików przez Internet. Jego główną zaletą jest podział plików na części i możliwość pobierania tych części od użytkowników, którzy w danym momencie również uczestniczą w procesie udostępniania. Pozwala to na znaczne odciążenie serwera. W szczególności możliwe jest nawet wyłączenie serwera, a plik pozostanie dostępny do pobrania, jeśli tylko wszystkie jego fragmenty zostały przed wyłączeniem rozesłane do zainteresowanych komputerów (wystarczy, że jedna maszyna posiada daną część i podzieli się nią z pozostałymi).

Podstawowe pojęcia

Poniżej znajduje się lista najważniejszych pojęć związanych z protokołem wraz z krótkim wyjaśnieniem ich znaczenia:

- Torrent — plik lub zbiór plików udostępnionych do pobrania.
- .torrent metafile — dodatkowy plik z metadanymi dotyczącymi udostępnionych plików. Zawiera między innymi:
  - nazwy i rozmiary plików,
  - liczbę i rozmiar fragmentów, na jakie zostały podzielone pliki,
  - listę skrótów SHA-1 fragmentów w celu weryfikacji poprawności,
  - adresy URL trackerów.
- piece, block — podczas przygotowywania torrenta pliki dzielone są na fragmenty (piece), a każdy taki fragment składa się z bloków (block). Blok jest najmniejszą jednostką, którą można przesłać przez sieć między użytkownikami. Aby udostępniać fragment użytkownik musi posiadać wszystkie jego bloki.

- info hash — 160-bitowa wartość będąca wynikiem funkcji skrótu SHA-1, której podawana jest część metapliku .torrent (nazwy plików i lista skrótów fragmentów). Info hash pozwala jednoznacznie zidentyfikować dany torrent.
- tracker — serwer, którego zadaniem jest przechowywanie adresów IP użytkowników pobierających dany torrent. Pozwala użytkownikom na znalezienie siebie nawzajem.
- klient (client) — program uruchomiony na komputerze użytkownika, który pozwala na pobieranie plików z wykorzystaniem protokołu BitTorrent.
- peer — węzeł (komputer, klient) pobierający i wysyłający fragmenty torrenta. Zazwyczaj nie posiada jeszcze wszystkich fragmentów.
- seed — peer posiadający wszystkie fragmenty.
- swarm — grupa peerów pobierających dany torrent.
- peer-to-peer (P2P) — sieć złożona z komputerów, które komunikują się ze sobą np. w celu wymiany plików. Peery tworzą sieć P2P.
- Distributed Hash Table (DHT) — rozproszona tablica mieszająca — sieć składająca się z węzłów, które umożliwiają zapisywanie i odczytywanie rekordów w formie klucz-wartość. Węzły dzielą między sobą zbiór wszystkich kluczy. W kontekście protokołu BitTorrent, DHT może zastąpić rolę trackera.
- magnet link — link pozwalający na uzyskanie metadanych torrenta bez konieczności pobierania metapliku .torrent. Link powinien zawierać przynajmniej info hash torrenta oraz listę trackerów.

Scenariusz pobrania torrenta

W celu pobrania torrenta niezbędne są następujące czynności:

1. Pobranie metapliku .torrent lub poznanie (kliknięcie) magnet linku identyfikującego dany torrent. Zazwyczaj informacje te można uzyskać na stronach internetowych katalogujących istniejące torrenty (wyszukiwarkach torrentów).
2. Uzyskanie listy trackerów z metapliku lub magnet linka.
3. Pobranie z trackera listy peerów uczestniczących w udostępnianiu torrenta.
4. Pobranie fragmentów torrenta od peerów.

Jak komunikują się peery

BitTorrent używa 12 typów wiadomości do prowadzenia komunikacji pomiędzy peerami:

- hand-shake — wiadomość rozpoczynająca połączenie,
- bitfield — wskazuje, jakie fragmenty posiada peer,
- keep-alive — wiadomość podtrzymująca otwarte połączenie,
- port — informuje o zmianie portu,
- choke, unchoke, interested, not interested — 4 wiadomości informujące o zmianie stanu peera (związane z algorytmem z punktu 3.1.4),

- have — wiadomość informująca o tym, że peer otrzymał nowy fragment,
- request — żądanie fragmentu,
- piece — wiadomość zawierająca fragment torrenta,
- cancel — wiadomość anulująca żądanie fragmentu.

Choking algorithm

W idealnej sytuacji wymiana plików za pośrednictwem protokołu BitTorrent jest sprawiedliwa, to znaczy każdy peer może pobierać pliki, ale jednocześnie powinien udostępniać posiadane fragmenty innym. By zapobiec sytuacji, w której peer blokuje wysyłanie posiadanych fragmentów, wprowadzono odpowiedni algorytm. Wykorzystuje on 4 typy wiadomości spośród wspomnianych w punkcie 3.1.4. Polega on na tym, że dany klient pozwala (unchoke) na pobieranie fragmentów od siebie tylko tym peerom, którzy posiadają i udostępnią klientowi swoje fragmenty. Pobieranie wymaga zatem kooperacji i wymiany interesujących, brakujących fragmentów. Bez tej wymiany połączenie zostaje przerwane (choke).

Wady i zalety protokołu

Wśród zalet protokołu znajduje się przede wszystkim wspomniane na początku zmniejszone obciążenie serwera. Prędkość pobierania może osiągnąć wyższą wartość niż limit transferu wychodzącego z serwera — ograniczeniem jest jedynie dostępność pliku wśród peerów oraz limit transferu przychodzącego do danego klienta. Rozesłanie pliku o rozmiarze m do n odbiorców bez użycia protokołu wymagałoby transferu danych o rozmiarze $m \cdot n$ z węzła udostępniającego, natomiast z użyciem protokołu ilość danych wysłanych przez nadawcę mieści się w przedziale $\langle m; m \cdot n \rangle$. Kolejną zaletą jest możliwość pobierania pliku nawet, jeśli oryginalny nadawca (twórca torrenta) jest niedostępny (zakładając oczywiście, że w swarmie rozesłane zostały najpierw wszystkie fragmenty pliku) — cecha ta okaże się przydatna podczas implementacji komunikatora grupowego.

Do wad protokołu należy zaliczyć zwiększone obciążenie oraz narzut komunikacyjny po stronie klienta — konieczność koordynacji pobierania i udostępniania, wysyłanie i odbieranie wiadomości kontrolnych. Wadą może być sam fakt, że klient zobowiązany jest do udostępniania pobieranego pliku. Jedną z ważniejszych kwestii, o jakie należy zadbać jest również dostępność pliku — klient nie pobierze całego torrenta jeśli nie znajdzie w sieci wszystkich jego fragmentów (tak zwany „problem ostatniego fragmentu"). Dodatkowo problematyczne może być wyszukanie metapliku .torrent lub magnet linku odpowiadającego danemu torrentowi.

Protokół z założenia powinien być w pełni rozproszony i nie polegać na żadnych publicznych serwerach — jedynie na bezpośredniej komunikacji użytkowników końcowych (P2P). Niestety kilka usług serwerowych jest wciąż aktywnie wykorzystywanych do prawidłowego działania sieci:

- serwer hostujący metapliki .torrent lub magnet linki pozwalający na wyszukiwanie interesujących plików po nazwie, tagach, innych właściwościach, oferujący statystyki torrenta, komentarze itp. — może zostać zastąpiony wyszukiwarką torrentów wbu-

dowaną w klienta, który wysyła zapytanie do podłączonych peerów, one przekazują je dalej, aż do momentu otrzymania odpowiedzi.

- tracker — istnieją jednak rozwiązania umożliwiające śledzenie swarmu bez użycia zewnętrznego serwera np. DHT, PEX (Peer Exchange).
- serwery pośredniczące w nawiązaniu połączenia dwóch klientów ukrytych w prywatnych sieciach IP (wykorzystujących translację adresów)

WebTorrent [11]

WebTorrent to biblioteka napisana w języku JavaScript, implementująca protokół BitTorrent. Dzięki zastosowaniu tego języka możliwe jest użycie protokołu w skrypcie wykonywanym w przeglądarce po wejściu na stronę internetową. Biblioteki można również użyć jako moduł w programie dla platformy Node.js. Funkcjonalność oferowana w obu przypadkach jest niemal identyczna (poza kilkoma sytuacjami wynikającymi z ograniczeń danej platformy).

Rozdział 4

Architektura

- webrtc
- angular
- python eve jako lekki serwer rest (co robi, że jest też mockiem)
- p2p
- bittorent
- webtorrent
- Stun ICE, NAT traversal, 90% wszystkich połączeń, ogromne liczby - to idzie w tysiące połączeń na minutę
  - https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
- rysunek co jest z czym połączone
- webrtc w webworkers - póki co nie ma tego https://github.com/w3c/webrtc-pc/issues/230

Rozdział 5

Wyniki testów

Rozdział 6

Wnioski

Bibliografia

[1] https://pl-pl.messenger.com/ (document), 2.1

[2] http://www.bleep.pm/ (document), 2.2

[3] https://whispersystems.org/ (document), 2.5, 2.3

[4] https://darkwire.io/ (document), 2.4

[5] http://moose-team.github.io/friends/ (document), 2.5

[6] https://tox.chat/ (document), 2.6

[7] https://zeronet.readthedocs.io/en/latest/using_zeronet/sample_sites/ (document), 2.7

[8] https://bitmessage.org/wiki/Main_Page (document), 2.7

[9] https://bitmessage.org/bitmessage.pdf 2.7

[10] http://www.bittorrent.org/ (document), 3.1, 3.1.5

[11] https://webtorrent.io/

(document), 3.2