Babeş–Bolyai University of Cluj-Napoca
Faculty of Mathematics and Informatics
Specialization: Computer Science

Diploma Thesis

Plan for a protocol for the live transmission of data over distributed networks

Abstract

Requirements for a protocol for the streaming of data from a single authoratative source to multiple destinations in a distributed, peer-to-peer manner, with an emphasis placed on the minimization of needed infrastructure, security, and non-repudiation of data. An early prototype that demonstrates some, but not all, of the ideas is also provided.

2024 Gardner Márk

Advisor:
[Teacher, Tit]

Babeş–Bolyai University of Cluj-Napoca
Faculty of Mathematics and Informatics
Specialization: Computer Science

Diploma Thesis

Plan for a protocol for the live transmission of data over distributed networks

Advisor:
[Teacher, Tit]

Student:
Gardner Márk

2024

Universitatea Babeş–Bolyai, Cluj-Napoca
Facultatea de Matematică şi Informatică
Specializarea Informatică

Lucrare de licenţă

Planificarea unui protocol pentru transmiserea datelor în timp real asupra reţele ditribuite

Conducător ştiinţific:
[Tit. Un profesor]

Absolvent:
Gardner Márk

2024

Babeş–Bolyai Tudományegyetem Kolozsvár
Matematika és Informatika Kar
Informatika Szak

Szakdolgozat

Elosztott hálózatokon történő adatok élő közvetítésére való protokoll tervezése

Témavezető:
[Tit. Egy Tanar]

Szerző:
Gardner Márk

2024

Tartalomjegyzék

1. fejezet

Bevezető

1.1. Motiváció

A peer-to-peer információ-megosztó protokollok, mint például a BitTorrent, IPFS, és mások az adatok integritását helyezik előtérbe. Ez sokszor igenis fontos, de léteznek esetek, amikor más szempont fontosabb lehet. Ezeknek az eseteknek egy nagy részében, az adatok gyors megérkezése fontosabb mint az adatok egyben maradása. Ez a kettős adat-továbbítási prioritás a kliens-szerver modellnél nagyon tisztán látható, abban, hogy a két alap protokoll (a TCP és az UDP) pont ezt a két igényt fedezi.

Ezeknek a protokolloknak gyakran reklámozott része, hogy az adatbiztonság mellett a megosztó fél identitása is védett. Habár ez sok esetben kecsegtető, bizonyos esetekben fontos lehet az, hogy bizonyíthassa valaki, hogy honnan szedte az adatait.

A fentiek ismeretében, ebben a dolgozatban le lesz írva egy protokoll vázlata, ami a következőket teljesíti:

- Egyetlen, hitelességgel rendelkező forrásból eljuttat adatokat gyorsan, számos érdekelt félhez.
- Semelyik fél se legyen túlságosan leterhelve
- Bármelyik fél tudja bizonyítani, hogy az adat amit kapott a forrástól jön, és hiteles
- Az ellenfeles viselkedésű feleket fel tudja ismerni, azonosítani, és minimalizálni befolyásukat
- hatékonyan létrehoz egy hálózatot az érdekelt felek között
- tudjon nagy mennyiségű időtől függő adatokat közvetíteni

1.2. lehetséges felhasználási területek

Olyan adatokról volna szó, amiknek csak a ”jelenlegi” értéke számít. Nem probléma, ha az elejéről akármennyi elvesztődik, sőt, a közepéről sem.

Példák felhasználási területekre lehetnének a műholdak szenzoradatai nyilvános hozzáférési módszere, vagy akár más közérdekű tudományos kísérlet mérései, vagy a szórakoztatás terén az internetes videós élő közvetítéseket is lehetne ezen a hálózaton hordozni.

2. fejezet

Létező komponensek

2.1. Létező peer-to-peer adat-megosztó megoldások

A létező, népszerű, p2p adat-megosztó megoldások az adat fajtájától függően két nagy kategóriába sorolhatóak (amiknek létezik nem üres metszete):

1. Állománymegosztó protokollok (mint például a már említett IPFS [Benet, 2014] vagy BitTorrent, vagy akár a croc nevű nyitott forráskódú applikáció, vagy a Gnutella protokoll, ami a Morpheus és a LimeWire applikációk alapját szolgálta)

2. Instans-üzenet-küldő programok (például a tox protokoll [Wikipedia contributors, 2023], vagy az IRC).

Habár bármilyen adatot amit a számítógép tud kezelni le lehet menteni egy állományba, és bizonyos lépéseken keresztül szöveges üzenetekbe is át lehet alakítani (például base64 encoding), ezek a protokollok egyszerűen nem alkalmasak arra, hogy egy állandóan frissülő adatforrást közvetítsenek.

2.2. létező peer-to-peer ”streaming” megoldások

Apache kafka: igazából kliens-szerver megoldás, és az adatok feldolgozásával/kategorizálásával foglalkozik, nem a kezdeti közvetítéssel.

Strivecast[str2019]: explicit a videó közvetítését szolgálja, a WebRTC technológia segítségével. Szükséges szerver, hogy elintézze a kezdeti csatlakozást. A peer5 ugyancsak. Ezen felül, ez a szoftver a kétirányú kommunikációt helyezi előtérbe, ami sokszor nem fontos.

Seedess[Lalasava, 2017]: Bittorent alapú, és reklámjai szerint csak csökkenti a szerverek terhelését, és ezt is csak 70%-osan. A példa amit adnak egy videóállomány, vagyis nem ”élő adás”.

3. fejezet

A hálózat struktúrája

A protokoll központi eleme a hálózat, amely egy fa típusú konfigurációt vesz majd fel. A gyökérben található az adatforrás, amíg a gyerekek a hallgató közönség.

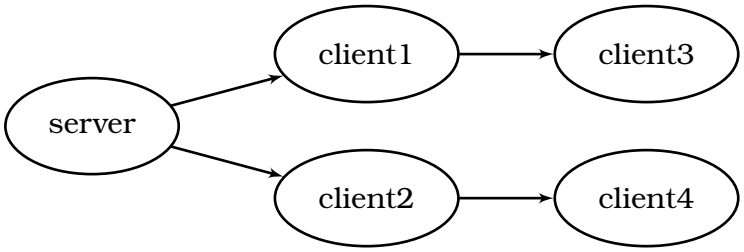
3.1. tervezési lehetőségek

A fa szerkezetére két lehetőség van, amik között még a tervezés folyamán döntést kellene hozni: legyen bináris vagy sem.

Ha a fa bináris, akkor lehet garantálni ennek sok kellemes tulajdonságát, amelyek kihasználásával a hálózat elrendezését és a csatlakozási folyamatot nagy mértékben lehetne egyszerűsíteni, és ezért gyorsítani is.

Ha a fa nem bináris, akkor a felek egyénileg el tudják dönteni mennyire képesek, és így sokkal kevesebb lenne a fa mélysége, vagyis az átlagos késés sokat csökkenne, de ekkor több adatot kell több helyre elküldeni, ami a protokollt is, a program logikáját is bonyolítaná.

A teljes fára nézett átlag késés minimalizálása érdekében, egy meritokratikus rangsorolás szerint, amely a 7 fejezetben lesz leírva a jobb hálózati kapcsolattal rendelkező kliensek közelebb kerülhetnek a gyökérhez, és így gyorsabban továbbadhatják a kapott adatokat az ők klienseiknek.



3.1. ábra. hogyan nézhetne ki a hálózat négy klienssel

a jelenlegi prototípusnak a klienseket tartalmazó adatstruktúrája és az ezt kezelő függvények fejléce

```
with GNAT.Sockets;use GNAT.Sockets;

package Network_Children is
    type Child_Number is range 0 .. 2;
    subtype Child_Index is Child_Number range 1 .. Child_Number'Last;

    type Child_Set is array (Child_Index) of Sock_Addr_Type;

    type Child_Status is (Success, NotExist, Empty);
    protected Children is

        entry Add_Child (Child : Sock_Addr_Type);
        entry Get_Children (N : out Child_Number; C : out Child_Set);
        entry Remove_Child (Child : Sock_Addr_Type; Status : out Child_Status);
        pragma Unreferenced (Remove_Child);
    private
        Number : Child_Number := 0;
        Set    : Child_Set    := (others => No_Sock_Addr);
        Locked : Boolean      := False;
    end Children;
end Network_Children;
```

A jelenlegi prototípusban a következő függvények kezelik a klienseket tartalmazó adatsrtuktúrát

```
with Text_IO;

package body Network_Children is
   protected body Children is
      entry Add_Child (Child : Sock_Addr_Type) when not Locked is
      begin
         pragma Debug
           (Text_IO.Put_Line
              (Text_IO.Standard_Error,
               "Add_Child(" & Image (Child) & ") called."));
         Locked       := True;
         Number       := Number + 1;
         Set (Number) := Child;
         Locked       := False;
      end Add_Child;
```

```
entry Get_Children (N : out Child_Number; C : out Child_Set)
  when not Locked is
begin
   Locked := True;
   N      := Number;
   if Number < 2 then
      Set (2) := No_Sock_Addr;
   end if;
   if Number < 1 then
      Set (1) := No_Sock_Addr;
   end if;
   C := Set;
   pragma Debug
     (Text_IO.Put_Line
        (Text_IO.Standard_Error,
         "Get_Children(" & N'Image & " (out), [" & Image (C (1)) &
           ", " & Image (C (2)) & "] (out)) called."));
   Locked := False;
end Get_Children;

entry Remove_Child (Child : Sock_Addr_Type; Status : out Child_Status)
  when not Locked is
begin
   Locked := True;
   if Number = 0 then
      Status := Empty;
      return;
   end if;
   Status := NotExist;
   for I in Set'Range loop
      if Status = Success then
         if I /= Set'Last then
            Set (I) := Set (I + 1);
         end if;
      else

         if Child = Set (I) then
            Set (I) := No_Sock_Addr;
            Status  := Success;
         end if;
      end if;
```

```
         end loop;
         pragma Debug
           (Text_IO.Put_Line
              (Text_IO.Standard_Error,
               "Remove_Child(" & Image (Child) & ", " & Status'Image &
                 ") called"));
         Locked := False;
      end Remove_Child;
   end Children;
end Network_Children;
```

4. fejezet

Az adatfolyam struktúrája

A hálózat tagjai között a közvetített adatok bizonyos extra metainformációval lesz társítva. Ebben a fejezetben le lesznek ezek írva, azzal együtt, hogy miért vannak ott. A mezők listája, még nem végleges, és a sorrend nagy valószínűséggel nagyon más lesz mint ez.

4.1. A közvetítendő adat

Mivel elméletileg a közvetített adat megszakítás nélküli, de a hálózaton csomagkapcsolás van, az adatforrás által adott adatokat először bizonyos, az adat típusa ismeretében megválasztott méretű csomagokra lesz választva.

4.2. Aláírás

Ezután, az adatok hitelességének garanciája érdekében, a szerver egy digitális aláírást számít az adott csomagra, és esetleg egy hash-et is, ha szükséges. A szerver publikus kulcsa már csatlakozás előtt ismert kell legyen a kliensek számára, ezért ezt nem közvetítjük, de a csomagok méretét már igen.

Megeshet, hogy az adatsebesség miatt nem praktikus minden csomagot aláírni, ezért lehetséges az, hogy csak minden k-adik csomaggal társul aláírás, de ez valahogy mind a k csomagra érvényes kell legyen.

4.3. csomagtípus-azonosító

Mivel több csomag típus is fog létezni, lesz egy mező a csomagokban, amely azonosítja, hogy milyen csomag következik: Aláírt, Aláírás-mentes, rendezési, vagy hálózatfenntartási.

4.4. sorrendjelölő

Mivel a protokoll UDP alapú lesz, nem létezik garancia arra, hogy sorrendben érkeztek-e az csomagok. Egyes esetekben, fontos lehet az adatok sorrendbe-helyezése, és erre a célra szolgál a sorrendjelző. Ez a mező nem fogja számon tartani azt, hogy eddig hány csomagot küldött a szerver, mivel hosszas közvetítés után a mező mérete impraktikusan nagy lenne. Ezért nagyjából csak egy bájt méret lesz erre a szerepre félretéve.

5. fejezet

Új adatforrások keresése

6. fejezet

Fel és le csatlakozás

7. fejezet

A hálózat újrarendezése

8. fejezet

Automatikus fegyelmezés és hálózatbiztonság

9. fejezet

Összefoglalt ismeretlenek és továbbfejlesztési lehetőségek

10. fejezet

Lehetséges sebesség korlátozások

Irodalomjegyzék

Benet, J. Ipfs - content addressed, versioned, p2p file system, 2014. URL https://raw.githubusercontent.com/ipfs/papers/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf.

Lalasava, G. Seedess p2p video streaming network. online, 2017. URL https://web.archive.org/web/20230417210201/https://seedess.com/.

str2019. The first isp friendly peer-to-peer live streaming network. online, 2019. URL https://strivecast.com/isp-friendly-p2p-live-streaming-network/.

Wikipedia contributors, . Tox (protocol) — Wikipedia, the free encyclopedia, 2023. URL https://en.wikipedia.org/w/index.php?title=Tox_(protocol)&oldid=1159250538. [Online; accessed 9-June-2023].