Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Ha Nhat Viet
Nr albumu: 209484

Nadawanie strumieni multimedialnych w sieciach punkt-do-punktu

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Grudzień 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

Zakres tej pracy obejmuje projekt i implementację rozproszonego zbioru nadajników strumienia multimedialnego w sieci punkt-do-punktu. Szczególnym zastosowaniem tego projektu będzie zbudowanie skalowalnego systemu telewizji internetowej.

W ramach analizy omówię założenia oraz architektury najpopularniejszych obecnie systemów telewizji działających na bazie sieci IP i pokażę ogólne możliwości rynku multimediów internetowych. We wprowadzeniu przedstawię wnioski na temat wrażliwości ludzkich zmysłów na różne zakłócenia sygnału audiowizualnego. Zaprezentuję modyfikacje sieci punkt-do-punktu umożliwiające transmisję strumieni multimedialnych w Internecie i zaproponuję dodatkowe mechanizmy, które pozwolą na wydajniejsze wykorzystanie zasobów sieci.

Struktura sieci oraz protokół sterujący oparte są na specyfikacji sieci BitTorrent. Do transmisji danych multimedialnych został użyty protokół Real-time Transport Protocol z pewnymi modyfikacjami. Przy projektowaniu modelu strumienia danych wykorzystałem schemat kodowania Multiple Description Coding. Stworzyłem też własny silnik kontroli łącza korzystający z prostych algorytmów pomiaru przepustowości.

Słowa kluczowe

telewizja, multimedia, sieć punkt-do-punktu, BitTorrent, Multiple Description Coding, Real-time Transport Protocol

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

C. Computer Systems Organization
C.2. Computer - Communication Networks
C.2.2. Network Protocols
C.2.4. Distributed Systems

Tytuł pracy w języku angielskim

Multimedia streaming over peer-to-peer network

Spis treści

Wprowadzenie

Internet na przestrzeni lat bardzo zmienił swoje oblicze. Ten, który znamy dzisiaj, różni się od tego sprzed 20 lat zarówno pod względem popularności, wielkości, jak i zastosowania. W pierwszym etapie swojego istnienia Internet był przeznaczony wyłącznie dla uprzywilejowanych pracowników instytutów naukowych. Pod koniec lat 80 liczebność komputerów podłączonych do sieci nie przekraczała stu tysięcy a same sieci szkieletowe miały przepustowość jedynie 56 Kbps. W owych czasach głównym zastosowaniem Internetu była wymiana statycznych danych tekstowych. Dzisiaj komputer osobowy jest już częścią każdego gospodarstwa domowego, a stałe połączenie z Internetem jest jednym z podstawowych standardów pracy zawodowej. Według danych udostępnionych przez Internet World Stats [11] z 30 czerwca bieżącego roku liczba osób posiadających dostęp do sieci wynosi już prawie półtora miliarda, czyli ponad 20% całej populacji Ziemi. W momencie kiedy Cisco Systems [5] ogłosiło, że obecny ruch w Internecie osiąga szacowaną wielkość około 11 eksabajtów na miesiąc, trudno sobie już wyobrazić świat bez Internetu. Niezaprzeczalnym faktem jest to, iż na dzień dzisiejszy Internet jest naszym głównym źródłem informacji. Jak podała agencja International Data Corporation [27] przeciętny Amerykanin spędza dwa razy więcej czasu surfując w Internecie (32,7 godzin tygodniowo) niż oglądając telewizję (16,4 godzin). Te dane można by interpretować dwojako. Z jednej strony oznacza to, że Internet jest już całkowitym liderem wśród mediów. Z drugiej zaś strony pokazuje, że pomimo dużo bardziej dynamicznego rozwoju względem klasycznych mediów nie stanowi on jeszcze substytutu dla telewizji. Kłopotliwe jest pytanie dlaczego wielu ludzi nadal wybiera telewizję z ograniczoną liczbą programów zamiast Internetu ze swoim oceanem informacji? Czemu nawet takie usługi jak YouTube, które dają nam dostęp do ogromnej bazy filmów i nagranych programów telewizyjnych nie zaspokajają naszej potrzeby oglądania telewizji?

Jest tak ponieważ telewizja dostarcza nam:

- programy na żywo — wiele programów telewizyjnych jest nadawana na żywo;
- wyselekcjonowane informacje — wbrew pozorom człowiek w morzu informacji się topi i nie jest w stanie wybrać dla siebie tych najciekawszych. Audycje autorskie i programy informacyjne to gwarancja pewnej jakości merytorycznej;
- lepsza jakość — jakość przekazu telewizyjnego jest wciąż wyższa niż w Internecie.

Na szczęście w obecnej chwili istnieją już aplikacje spełniające te wymogi. Nawet w skali krajowej mamy takie przedsięwzięcia jak iTVP, która transmitowała tegoroczne igrzyska Olimpijskie w Pekinie, oraz Ipla, dzięki której można było obejrzeć wszystkie mecze Euro 2008. Niestety te inicjatywy mają ograniczone możliwości i nie możemy oczekiwać, że z ich pomocą będziemy mogli obejrzeć w Internecie każdy program jaki zechcemy. Smutnym faktem jest również to, że jeśli mamy wolne łącze internetowe lub dana transmisja cieszy się zbyt dużą popularnością, to wymienione rozwiązania stają się dla nas niedostępne.

Gdzie leży problem?
Jak go można rozwiązać?

Problem

Koszty

Przeszkodą w osiągnięciu standardów telewizji kablowej jest wciąż zbyt duży koszt łączy internetowych. Tradycyjny model z jednym nadawcą i wieloma odbiorcami, czyli klient-serwer pochłania spore koszty. Jako przykład przedstawię tutaj projekt Gadu Radia, który współtworzę. Gadu Radio ma średnią słuchalność (wg danych firmy Gemius) wynoszącą około dwudziestu tysięcy osób. Nawet dla strumienia 24 Kbps całkowite obciążenie łącza takiego przedsięwzięcia wynosi pomiędzy 600 a 700 Mbps i przekłada się na kwoty rzędu kilkudziesięciu tysięcy złotych. Są to koszty duże w porównaniu z jakością jaka jest zapewniana. Problemem jest również mała elastyczność usług tego rodzaju. Przeważnie znaczna część wykupionego pasma pozostaje niewykorzystana, a przypadki przekroczenia dozwolonej przepustowości najczęściej wiążą się z dodatkowymi opłatami dla dostawcy.

Zawodność

Kolejnym utrudnieniem Internetu jest jego wysoka zawodność. Każdy użytkownik Internetu jest przeświadczony o tym, iż od czasu do czasu mogą zdarzyć się problemy z połączeniem sieciowym. Lecz jako telewidzowie nie jesteśmy jeszcze przyzwyczajeni do takiej zawodności. Reakcją najpopularniejszego protokołu w Internecie — TCP — na wszelkie sytuacje krytyczne jest przedłużony czas transmisji. W przypadku strumieni multimedialnych oznacza to spowolnienie odbieranego sygnału. Efekt ten jest dla człowieka bardzo nienaturalny i jest odbierany jako odczuwalny spadek jakości.

Rozwiązanie

Żeby znaleźć na to odpowiedź należy zajrzeć nie gdzie indziej, lecz do Internetu. Przyjrzyjmy się dokładniej co mówią statystyki. Z danych z firmy Ellacoya [8] z zeszłego roku wynika, że większość ruchu w Internecie jest generowana przez systemy wymiany plików w sieciach punkt-do-punktu (37%) oraz strumieniowanie multimedialne (21,9%). Podobne wyniki ogłosiła w maju tego roku firma Sandvine [23], mówiąca o 43,5% udziale sieci punkt-do-punktu oraz 14,8% udziale strumieniowania. Szczególnie interesującą informacją jest też to, iż Unia Europejska opowiada się za zwiększeniem wsparcia dla projektów punkt-do-punktu. Przykładem jest 19-milionowy projekt o nazwie P2P-Next, którego częścią jest system **Tribler**.

Koszty

Alternatywą dla drogich łączy o dużej przepustowości jest użycie struktury sieci punkt-do-punktu. W sieciach punkt-do-punktu każdy odbiorca jest również nadawcą. Dzięki takiemu rozwiązaniu zmniejszone zostanie obciążenie serwera nadającego na koszt interesantów. Dodatkowo im strumień jest popularniejszy, tym więcej jest odbiorców danego strumienia, czyli również i nadawców. Zyskujemy więc dodatkową skalowalność.

Zawodność

Człowiek źle odbiera sygnał zwolniony lub przyspieszony, lecz dobrze przyswaja sygnały zniekształcone lub o gorszej jakości. Wynika z tego, że lepszy do zastosowań multimedialnych jest zawodny acz szybki protokół UDP. Przy użyciu protokołu bezpołączeniowego nie tracimy niepotrzebnego narzutu na gwarancję dostarczenia. Dodatkową przepustowość otrzymaną z rezygnacji z modelu połączeniowego możemy wykorzystać do korekcji błędów.

Opierając się na tych dwóch ideach zaprojektowałem rozwiązanie, które pozwoli na wykorzystanie zwykłych łączy w Internecie jako medium do przenoszenia sygnału telewizyjnego. Struktura sieci opiera się w dużej mierze na podstawowych koncepcjach sieci BitTorrent. Sam protokół sterujący powstał jednak w całości od nowa — w celu zminimalizowania narzutu na zużycie sieci. Do przesyłania danych został użyty protokół Real-time Transport Protocol, z pewnymi zmianami interpretacyjnymi niepowodującymi niezgodności z bazowym protokołem. Żeby otrzymać efekt zmiennej jakości strumienia uzależnionej od stanu łącza zaadaptowałem kodowanie Multiple Description Coding. Dodałem również do odbiorników moduł monitorujący stan sieci. Dzięki tej własności rozwiązanie minimalizuje zużycie sieci. Aplikacja została zaprojektowana z myślą o rozszerzeniach i dlatego została podzielona na warstwy odtwarzania, przesyłania danych, protokołu sterującego. Podmiana lub zmiany w jednej warstwie nie wymagają zmian w pozostałych.

Struktura pracy

Praca składa się z następujących rozdziałów:

Rozdział 1. Podstawowe pojęcia — w tym rozdziale opiszę koncepcję sieci punkt-do-punktu oraz tematykę multimediów w Internecie.

Rozdział 2. Istniejące rozwiązania — w tym rozdziale zaprezentuję istniejące na rynku systemy telewizji punkt-do-punktu.

Rozdział 3. Wymagania — opiszę tutaj wszystkie wymagania funkcjonalne i niefunkcjonalne projektu.

Rozdział 4. Technologie — ten rozdział będzie zawierał opis użytych w projekcie technologii.

Rozdział 5. Projekt i implementacja — w tym rozdziale zostaną omówione projekt oraz implementacja systemu.

Rozdział 6. Testy — zaprezentuję wyniki z testów przeprowadzonych na stworzonej implementacji.

Rozdział 7. Podsumowanie — w tym rozdziale podsumuję całą pracę i przedstawię możliwe rozszerzenia.

Rozdział 1

Podstawowe pojęcia

1.1. Transmisja multimedialna

Jak pokazały statystyki, multimedia zdążyły już dzisiaj zawojować Internet. Surfując po Internecie możemy spotkać różne ich rodzaje. Na początku tego rozdziału przedstawię rodzaje usług multimedialnych, które są dostępne obecnie w Internecie. Później omówię złożoność całego procesu transmisji multimedialnej, wytłumaczę najważniejsze pojęcia z zakresu sieci transmitujących i opiszę pokrótce różnice między nimi. Na końcu przedstawię kwestie na jakie należy zwrócić uwagę projektując system multimedialny.

1.1.1. Video on Demand

Video on Demand (skrót: *VoD*) to usługa pozwalająca użytkownikowi na wybranie i obejrzenie zawartości wideo przez internet. Video on Demand należy postrzegać jako wirtualny odpowiednik wypożyczalni wideo, w których możemy zdalnie obejrzeć film lub program z udostępnionej bazy danych. Z podobieństwa do kaset wideo wywodzi się również podstawowa funkcjonalność VoD, a mianowicie sterowanie. Korzystając z usługi Video on Demand użytkownik ma dużą kontrolę nad zamówioną zawartością wideo (i audio) — decyduje o tym, kiedy zacząć, przerwać lub zatrzymać oglądanie. Niektórzy usługodawcy VoD udostępniają również wiele różnych poziomów jakości danych multimedialnych — dopasowanych do stanu łącza użytkownika. W Video on Demand nacisk jest kładziony na to, by obraz wideo odtwarzany na komputerze użytkownika był tej samej jakości co obraz wysłany z serwera usługodawcy. Pod tym względem usługa Video on Demand przypomina inne usługi, w których najważniejszą kwestią jest gwarancja dostarczenia, a nie odtwarzanie w czasie rzeczywistym. W Video on Demand obraz przesyłany jest najczęściej jako wiarygodny strumień, co w razie obciążenia sieci przekłada na zatrzymanie przetwarzania obrazu. Najważniejszym celem Video on Demand jest jak najkrótszy czas odtwarzania nieprzerwanego obrazu (przy zachowaniu odpowiednich warunków jakości QoS) licząc od momentu wysłania strumienia z serwera do momentu rozpoczęcia wyświetlania na komputerze odbiorcy.

1.1.2. Video Conferencing

Video Conferencing to komunikacja użytkowników za pomocą strumienia wideo. Konferencje, w przeciwieństwie do Video on Demand, kładą większy nacisk na stronę merytoryczną transmisji. Głównym celem wideokonferencji jest umożliwienie płynnej interakcji między użytkownikami. W tym przypadku wymaga się, aby opóźnienie transmisji było jak najmniejsze, jeśli

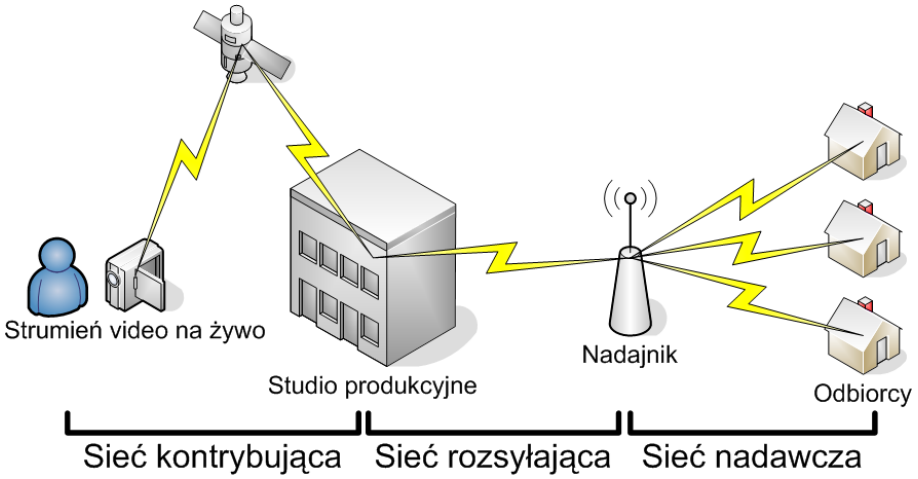
nawet oznacza to słabą jakość. Ważne jest również, aby usługa wideokonferencji jak najmniej obciążała sieć.

1.1.3. Internet Protocol Television

Internet Protocol Television (*IPTV*) jest usługą nadawania obrazu „na żywo". W przeciwieństwie do VoD, IPTV naśladuje telewizję. W usłudze IPTV zakłada się, że użytkownik będzie otrzymywał obraz na żywo, z małym bezwzględnym opóźnieniem. Celem IPTV jest transmisja meczu piłkarskiego lub innego ważnego wydarzenia społecznego na żywo przez Internet. W tym przypadku ważniejsza jest transmisja multimediów w czasie rzeczywistym (zdanie raportu i przekazanie ogólnych informacji) niż dokładne odtworzenie przesłanych danych (wrażenia estetyczne). W tym przypadku nie żądamy gwarancji wiarygodności — pewne fragmenty danych mogą się zgubić, a istotne jest tylko odtwarzanie otrzymanych ramek obrazu w tym samym odstępie czasowym co nagrany obraz. W usłudze IPTV gwarancja jakości oznacza, że obraz jest zrozumiały dla oglądającego go człowieka.

1.1.4. Telewizyjny łańcuch emisyjny

Obraz audiowizualny, który pojawia się w odbiornikach telewidzów to efekt ogromnej liczby pracy. Cały ten proces w przemyśle multimedialnym jest znany pod terminem **łańcucha emisyjnego**. Łańcuch emisyjny z transmisyjnego punktu widzenia dzieli się na trzy etapy:



Rysunek 1.1: Łańcuch emisyjny

Sieć kontrybująca

Sieć kontrybująca służy do przesyłania treści multimedialnej z jej miejsca nagrania do studia. Ponieważ ta treść musi zostać podana obróbce, więc zadaniem sieci kontrybującej jest dostarczenie sygnału w jak najwyższej możliwej jakości. W przypadku transmisji na żywo wymagane jest również minimalne opóźnienie, gdyż każde zakłócenie na tym etapie będzie potem widoczne u widza. Sieci kontrybujące posiadają najczęściej łącza z gwarancją niezawodności o przepustowości rzędu kilkuset megabitów na sekundę.

Sieć rozsyłająca

Sieć rozsyłająca ma na celu rozdystrybuowanie gotowego strumienia multimedialnego ze studia produkcyjnego do stacji nadawczych. Podobnie jak w sieci kontrybującej wymaga się od sieci rozsyłającej by strumień został dostarczony w niezmienionej postaci, ponieważ będzie on potem w tej postaci oglądany przez wszystkich odbiorców. Sygnał ten musi być wysokiej jakości również z tego powodu, że dopiero w stacjach nadawczych zostanie on przekodowany zgodnie z konfiguracją odbiorników.

Sieć nadawcza

Sieć nadawcza to ostatni etap łańcucha emisyjnego, gdzie gotowy zakodowany strumień jest transportowany ze stacji nadawczej do odbiorników. W przeciwieństwie do dwóch poprzednich sieci, sieć nadawcza to już nie tylko niezawodne łącze o wysokiej przepustowości, ale cała infrastruktura połączeń o najróżniejszych topologiach. Koszty utrzymania tej sieci przewyższają wielokrotnie koszty pierwszych dwóch sieci. Właściwe zarządzanie tą infrastrukturą w decydującym stopniu wpływa na zyskowność całego procesu transmisji.

Celem tej pracy magisterskiej jest stworzenie sieci nadawczej operującej w Internecie o minimalnych kosztach utrzymania. Problematyka sieci kontrybującej i rozsyłającej nie znajdują się w zakresie tej pracy.

1.1.5. Projektowanie usług multimedialnych

Przy projektowaniu narzędzi do odtwarzania sygnałów audiowizualnych, należy uważnie poznać wyniki dotychczasowych badań i wnioski na temat funkcjonowania ludzkich zmysłów, w tym przypadku wzroku i słuchu. W przeciwieństwie do komputera, człowiek o wiele lepiej razi sobie z uszkodzonymi lub niepełnymi informacjami z otoczenia. Za ostateczne wrażenie jakie odnosi widz nie odpowiada wyłącznie jego wzrok, ale w dużej mierze mózg. Człowiek w przypadku wtórnego przyswajania bodźców nie potrzebuje już tyle informacji co za pierwszym razem. Z drugiej strony bodźce, z którymi spotyka się pierwszy raz wywołują dużo silniejszy efekt.

Dla umysłu ludzkiego odbieranie niewyraźnego sygnału jest naturalnym zjawiskiem, ponieważ nasze zmysły nie są idealnymi receptorami. Lekkie stłumienie odgłosu z radia bądź bardziej mglisty obraz nie wywołuje w naszym umyśle silnej reakcji. Za to przyspieszenie bądź zwolnienie obrazu w telewizorze albo muzyki w radiu skłania nas do natychmiastowej reakcji. Z tego też powodu dla zapewnienia najlepszego wrażenia odbiorcy należy zadbać o spójność prezentowanego obrazu, kosztem jego dokładności. W mojej pracy przy kwestiach dotyczących jakości odbieranego strumienia, główny nacisk będzie kładziony na synchroniczne wyświetlanie sekwencji obrazów, a nie na przedstawianiu ich w najwyższej jakości.

1.2. Sieci punkt-do-punktu

Termin sieci punkt-do-punktu stał się popularny pod koniec ubiegłego stulecia za sprawą nieistniejącej już sieci Napster. W rzeczywistości idea punkt-do-punktu była zawsze obecna w strukturze i rozwoju Internetu. Zaskoczeniem dla niektórych będzie fakt, że protoplasta dzisiejszego Internetu sieć ARPANET była pierwszą siecią punkt-do-punktu. Koncepcją tej sieci było połączenie różnych struktur sieciowych w jedną spójną sieć, gdzie każdy element będzie traktowany na równi. Obecnie nastąpił powrót do korzeni i sieci punkt-do-punktu znowu cieszą się dużą popularnością.

Koncepcja sieci punkt-do-punktu bazuje na równości. W przeciwieństwie do modelu klient-serwer spopularyzowanego przez strony WWW, sieci punkt-do-punktu nie zakładają istnienia centralnego elementu bardziej uprzywilejowanego od innych. Poszczególne elementy sieci komunikują się każdy z każdym, bez zbędnych pośredników. Przewagą sieci punkt-do-punktu nad strukturą klient-serwer jest większa niezawodność z ogólnego punktu widzenia (nie ma wąskiego gardła ani pojedynczego punktu awarii) oraz teoretycznie nieskończona skalowalność.

W tym podrozdziale omówię rodzaje sieci punkt-do-punktu oraz ich klasyfikację. W szczególności opiszę najpopularniejszy w tym momencie protokół BitTorrent (wg firmy Ipoque stanowi 66% całego ruchu punkt-do-punktu w środkowo-wschodniej Europie).

1.2.1. Klasyfikacja

Sieci punktu-do-punktu mogą być sklasyfikowane według kilku kryteriów [22].

Centralizacja

Pierwsze kryterium dotyczy stopnia scentralizowania sieci. Rozróżniamy:

- zdecentralizowane — określane jako prawdziwe sieci punkt-do-punktu. W modelu zdecentralizowanym wszystkie węzły są równouprawnione. Awaria dowolnego elementu w sieci nie powinna znacząco wpływać na działanie reszty sieci.

- scentralizowane — występuje wyróżniony węzeł w sieci, pełniący zupełnie inną rolę niż pozostałe. Węzeł ten najczęściej dostarcza pewną usługę, kluczową dla całej sieci i w ten sposób stanowi pojedynczy punkt awarii. Jednak w przeciwieństwie do modelu klient-serwer nie jest on pośrednikiem w komunikacji pomiędzy innymi węzłami w sieci.

- hybrydowe — jest to model pośredni pomiędzy scentralizowanym a zdecentralizowanym. W topologii hybrydowej występują elementy uprzywilejowane, będące niezbędnym elementem infrastruktury sieci. Nie jest to jednak jeden węzeł, lecz wiele węzłów. Odpowiednio, liczba uprzywilejowanych elementów rośnie wraz z wielkością sieci.

Struktura

Druga kategoria podziału klasyfikuje sieci punkt-do-punktu według jej struktury [4]:

- niestrukturyzowane — sieć niestrukturyzowana to sieć, w której połączenia pomiędzy węzłami są nawiązywane losowo, bez odgórnie narzuconego kryterium. Węzły w sieci mogą dobierać sąsiadów bardziej wybiórczo, lecz decyzja jest podejmowana na podstawie informacji lokalnych, a nie zebranych z całej sieci. Połączenia pomiędzy węzłami w sieci dynamicznie się zmieniają, dlatego też każdy węzeł posiada informacje wyłącznie o położeniu węzłów, z którymi sąsiaduje.

- ustrukturyzowane — sieci ustrukturyzowane to sieci wyspecyfikowane do określonego zadania. W celu uzyskania lepszej wydajności takie sieci posiadają ograniczenia dotyczące zarówno struktury grafu węzłów, jak i danych umieszczanych na poszczególnych węzłach. W sieci ustrukturyzowanej węzły posiadają pewną wiedzę o położeniu innych węzłów w sieci i mogą w sposób deterministyczny namierzyć dowolny inny węzeł.

Tabela 1.1 ukazuje klasyfikację najbardziej znanych protokołów punkt-do-punktu do wymiany plików.

	Zdecentralizowane	Scentralizowane	Hybrydowe
Niestrukturyzowane	Gnutella 0.4 [9] KaZaA [20]	Napster BitTorrent [6]	Gnutella 0.6 [12]
Ustrukturyzowane	Chord		

Tabela 1.1: Klasyfikacja protokołów punkt-do-punktu

Rozdział 2

Inne rozwiązania

W chwili obecnej istnieje już wiele działających systemów na rynku telewizji punkt-do-punktu. Przeglądając listę wszystkich produktów można odnieść wrażenie, że ta branża została już zdominowana przez duże firmy i nie ma miejsca dla konkurencji. W rzeczywistości, pomimo długiego czasu istnienia takich systemów jak PPLive czy Joost, żaden z nich nie osiągnął niezachwianej pozycji dominującej. Te systemy wydają się wręcz być zacofane i stać w miejscu względem nowych rozwiązań, takich jak Tribler. W tym rozdziale opiszę specyfikację najbardziej znanych produktów. W niektórych przypadkach przedstawię również architekturę sieci i specyfikację protokołów.

2.1. PPLive

PPLive [19] należy do systemów telewizji punkt-do-punktu pierwszego pokolenia, czyli tych, które powstały na przełomie lat 2004-2005. Praktycznie wszystkie aplikacje z tego okresu powstały w Chinach kontynentalnych z powodu luźnego egzekwowania ustaw dotyczących praw autorskich. PPLive szybko zyskał popularność i do dnia dzisiejszego jest liderem w klasyfikacji liczby oglądających. PPLive jako pierwszy również przekroczył próg 500 tysięcy oglądających w jednym momencie. Według danych podanych na stronie z 2006 roku aplikacja daje dostęp do ponad dwustu kanałów i posiada średnio 400 tysięcy widzów dziennie.

2.1.1. Wymagania

PPLive jest aplikacją Windowsową. Nie posiada on własnego odtwarzacza lecz korzysta z programów Windows Media Player bądź RealPlayer. Kanały są nadawane w tempie od 250 Kbps do 400 Kbps. Istnieje również kilka kanałów nadawania w wysokiej jakości 800 Kbps.

2.1.2. Architektura

PPLive jest siecią nieustrukturyzowaną. Cała komunikacja w sieci PPLive jest oparta na protokole TCP. Sama aplikacja rozróżnia dwie warstwy protokołów:

- protokół plotkujący do uzgodnienia listy sąsiadów i kanałów,
- protokół transmitujący strumień multimedialny o wysokiej jakości.

2.1.3. Odtwarzanie

Transmitowany strumień multimedialny jest dzielony na małe paczki (wielkości około 1200 bajtów) i przesyłany pomiędzy klientami w sieci. Aplikacja klienta PPLive przetrzymuje dane w dwóch buforach:

- Sieciowy — szereguje paczki, które przyszły z sieci. Te same paczki są przesyłane dalej kiedy inny klient o nie poprosi. Kiedy ten bufor się napełni, wszystkie dane są przerzucane do bufora odtwarzania.
- Odtwarzania — kiedy ten bufor się wypełni zaczyna się właściwe odtwarzanie

2.1.4. Podsumowanie

PPLive wymaga często okresu około kilku minut dla wstępnego buforowania. Rozmiar buforów wynosi w granicach od 10 do 30 MB. PPLive jest wrażliwy na różne anomalie sieciowe i w takich sytuacjach odtwarzany strumień może być przerywany lub skokowy. Zaletą jest duża popularność aplikacji, dzięki czemu praktycznie zawsze można obejrzeć żądany strumień.

2.2. Joost

W 2006 roku świat obiegła plotka o nowej platformie multimedialnej twórców Skype i Kazaa, która ma wprowadzić Internet w nową erę. Chodziło oczywiście o Joosta [14], który nosił wtedy jeszcze miano The Venice Project. Joost jest jednym z nielicznych produktów w branży informatycznej, który był wyprzedzony przez własną sławę. Na początku 2007 roku kiedy Joost był oznakowany wersją alfa, kolejki do bycia beta-testerem produktu pękały już w szwach. Do dnia dzisiejszego Joost nadal znajduje się w wersji beta.

2.2.1. Wymagania

Obecnie Joost jest dostępny tylko na platformę Windows XP oraz Mac OS X.

2.2.2. Architektura

Na architekturę Joosta składają się trzy logiczne elementy sieci:

- serwery nadawcze — dedykowane maszyny (zbierane w farmy) będące wyłącznymi źródłami treści w sieci Joost,
- super-węzły — elementy sieci nie biorące bezpośredniego udziału w wymianie danych. Służą wyłącznie do zarządzania strukturą sieci,
- węzeł — aplikacja kliencka.

Joost również opiera się na dwóch protokołach sieciowych:

- protokół kontroli — w tej warstwie węzły uzgadniają strukturę połączeń w sieci. Ten protokół jest zbudowany na bazie protokołu TCP.
- protokół transmisji — warstwa wymiany danych, oparta na protokole UDP z użyciem protokołu STUN do omijania blokad nałożonych przez Tłumaczy Adresów Sieciowych (ang. Network Address Translator). Pakiety mają stałą wielkość wynoszącą 1 KB.

2.2.3. Odtwarzanie

Joost używa kodeka CoreAVC firmy CodeCodec zgodnego z standardem H.264 (MPEG-4 p.10).

2.2.4. Podsumowanie

Treść nadawana w sieci Joost pochodzi z dedykowanych farm serwerów. Trudno jest z tego powodu porównać ten system do innych, które są w pełni typu punkt-do-punktu. Joost zapowiada dodanie dodatkowych rozwiązań, które mają umożliwić tworzenie bardziej wydajnych struktur sieciowych. Na chwilę obecną żadne z tych rozszerzeń nie zostało jednak jeszcze zaimplementowane.

2.3. Tribler

Tribler [16] jako jedyny w odróżnieniu od pozostałych produktów jest projektem otwartym. Tribler powstał i jest rozwijany jako wspólna inicjatywa Delft University of Technology oraz Vrije Universiteit Amsterdam. Tribler sam określa się jako społeczność umożliwiająca wymianę plików. W ramach projektu powstał jednak dodatkowo odbiornik strumieni wideo.

2.3.1. Wymagania

Tribler jest dostępny pod systemem Windows, Mac OS oraz Linux Ubuntu w wersji paczek do zainstalowania. Oprócz tego twórcy udostępnili swoje repozytorium z kodem. Całość aplikacji jest napisana w języku Python.

2.3.2. Architektura

Sieć Triblera to w praktyce sieć BitTorrentowa z rozproszonym serwerem śledzącym. W modelu sieci Triblera nie istnieje jeden centralny serwer śledzący, za to każdy węzeł oprócz zasobów danych posiada również metadane o innych węzłach w sieci. W logice węzła sieci powstały również nowe pojęcia jakich nie ma w innych sieciach:

- pożyczanie łącza — inny węzeł może pomóc nam w ściąganiu pewnego zasobu,
- przyjaciele — wybór świadomie zatwierdzony przez użytkownika, który powoduje, że węzeł jest bardziej uprzywilejowany.

Tribler w odróżnieniu od tradycyjnego klienta BitTorrent używa mechanizmu **daj i weź** (and. *give-and-get*) zamiast **wet za wet** (ang. *tit-for-tat*) do zwalczania tzw. jazdy na gapę (objaśnię znaczenie tych terminów w części opisowej protokołu BitTorrent).

2.3.3. Odtwarzanie

Tribler nie posiada własnego odtwarzacza i korzysta z odtwarzacza dostępnego na danej platformie.

2.3.4. Podsumowanie

Tribler jest bardzo ciekawym projektem z powodu swojego niekomercyjnego charakteru. Celem projektu jest stworzenie innowacyjnej technologii oraz propagowanie rozwiązań sieciowych w architekturze punkt-do-punktu. Dzięki dużemu wsparciu ze strony władz unijnych oraz rozgłosowi w środowisku open-source ten projekt rozwija się bardzo dynamicznie.

Rozdział 3

Wymagania

W tym rozdziale przedstawię wszystkie wymagania dotyczące systemu. Na początku postawię cel całego projektu. Później opiszę niezbędne właściwości jakie będzie musiały posiadać protokół sterujący oraz protokół transmisji danych. Na końcu określę wymagania funkcjonalne i niefunkcjonalne implementacji.

3.1. Wymagania systemu

W tej części zostaną przedstawione wszystkie wymagania względem tworzonego systemu.

Termin "system" należy interpretować jako zbiór wszystkich maszyn i aplikacji biorących udział w ogłaszaniu, transmisji i odtwarzaniu tej samej treści multimedialnej. W skład systemu wchodzą następujące elementy:

- Kanał — byt logiczny, określający jeden lub więcej powiązanych ze sobą strumieni multimedialnych.
- Strumień multimedialny — dynamicznie zmieniający się w czasie ciąg danych dźwiękowych bądź wizualnych.
- Węzeł — jednostka sieci, która aktywnie uczestniczy w wymianie danych:
  - Aplikacja kliencka (odbiornik) — węzeł, aplikacja na maszynie użytkownika, która odbiera, nadaje i odtwarza strumień multimedialny.
  - Aplikacja nadająca (nadajnik) — węzeł, aplikacja, która jest źródłem strumienia multimedialnego.
- Połączenia — połączenia sieciowe pomiędzy węzłami, przez które są przesyłane komunikaty sterujące lub dane multimedialne.

3.1.1. Prosta instalacja

Instalacja systemu powinna być możliwa przy minimalnych wymaganiach sprzętowych. Jedynym warunkiem do postawienia i uruchomienia systemu jest posiadanie treści multimedialnej, którą można ogłosić. Aplikacja nadająca może zostać uruchomiona na dowolnej maszynie klasy PC. Maszyna musi mieć możliwość płynnego odtwarzania nadawanego strumienia oraz posiadać pamięć zdolną do przechowania od kilku do kilkudziesięciu minut strumienia.

Przepustowość łącza wyjściowego powinna być kilkukrotnością wielkości rozgłaszanego strumienia i niezależna od liczby zainteresowanych odbiorców.

3.1.2. Elastyczny

System nie posiada żadnej narzuconej z góry struktury. Decyzje o połączeniach między poszczególnymi węzłami sieci są podejmowane lokalnie i mogą się dynamicznie zmieniać w czasie. Każdy węzeł może dołączyć się do sieci i opuścić ją w dowolnym momencie.

Węzły, które dłużej przebywają w sieci same reorganizują się w celu polepszenia transmisji danych. Dobór połączeń powinien zależeć od przepustowości oraz opóźnienia pomiędzy węzłami.

3.1.3. Sprawiedliwy

System nie zakłada istnienia żadnych uprzywilejowanych elementów. Węzły w sieci nie są rozróżniane na żadne klasy i każdy jest w momencie dołączenia do sieci traktowany równorzędnie.

Dodatkowo odbiorcy, którzy będą jeździli na gapę — czyli będą pobierać strumień, ale nie będą go nadawać, będą karani przez swoich sąsiadów. Z czasem ich żądania o strumień będą ignorowane. Jeśli ich zachowanie się zmieni, to w przyszłości będą lepiej traktowani.

3.1.4. Skalowalny

System nie powinien mieć górnego ograniczenia co do liczby klientów mogących odbierać dany kanał. Jeśli popularność kanału wzrośnie, to moc nadawcza systemu również musi wzrosnąć.

3.1.5. Optymalne zużycie łącza

Wykorzystanie przepustowości łącza przez poszczególne węzły musi być tak wydajne, jak jest to tylko możliwe. System w każdej chwili działania powinien starać się maksymalizować następujące wartości:

- całkowite zużycie łącza względem dozwolonej przepustowości,
- udział transmisji strumienia w całym ruchu.

3.1.6. Nieprzeciążanie łącza

Pożądanym zachowaniem systemu jest przestrzeganie ograniczeń przepustowości ustanowionych przez użytkownika. Aplikacja kliencka nie powinna więc przekraczać dozwolonego poziomu zużycia sieci. W sytuacjach awarii sieci aplikacja musi dodatkowo zmniejszyć transmisję danych, by nie generować niepotrzebnego zatoru w sieci.

3.1.7. Strumień czasu rzeczywistego

Dane multimedialne przesyłane w systemie są strumieniem czasu rzeczywistego. Pojęcie czasu rzeczywistego określa zjawisko, kiedy czas trwania akcji wykonywanej na danym zasobie trwa tyle samo co zasób. W tym przypadku oznacza to, że nadawanie dziesięciominutowego strumienia wideo przez sieć będzie trwało dziesięć minut. Najważniejsza własność transmisji czasu rzeczywistego polega na tym, że odtwarzany strumień jest synchroniczny z nagranym strumieniem. Nie występuje więc efekt przyspieszenia czy też zwolnienia charakterystyczny dla nośników taśmowych.

Systemy nadawania czasu rzeczywistego dzielą się na dwa rodzaje:

- twarde — kiedy strumień jest odtwarzany od razu w momencie przyjścia, bez potrzeby zapamiętywania w buforach. Takie rozwiązanie jest wykorzystywane w systemach interaktywnych, jak np. wideokonferencje.

- miękkie — strumień odbierany jest najpierw zapisany na nośnik, a dopiero później z tego nośnika jest nadawany dalej. Jeśli sygnał zostanie przerwany lub nośnik się zepsuje przy zapisie, może nastąpić retransmisja.

Ten system będzie realizował rozwiązanie pośrednie pomiędzy twardym a miękkim nadawaniem czasu rzeczywistego. System powinien wykorzystywać bufor do odbierania strumienia. Ten bufor powinien być jednak mały i służyć tylko w sytuacjach awarii sieci.

Bufor powinien być w stanie przechować kilkadziesiąt sekund strumienia.

3.1.8. Odtwarzanie na żywo

Odtwarzanie na żywo ma miejsce wtedy, kiedy strumień jest odtwarzany w każdym kliencie w tym samym momencie. Ze względu na opóźnienia związane z przesyłaniem w sieci, czasy odtwarzania i wysyłania nie będą idealnie zsynchronizowane.

W przypadku Internetu opóźnienie w wielkości od kilku do kilkudziesięciu sekund jest akceptowalne.

3.1.9. Podzielna jakość strumienia

Kluczowe dla systemu jest założenie, że strumień może mieć różną jakość. Taka własność ma umożliwić odbiorcom dostosowanie jakości odtwarzanego obrazu do przepustowości swojego łącza. Użytkownicy dysponującymi łączami o mniejszej przepustowości powinni otrzymywać płynny strumień aczkolwiek słabszej jakości. Dla klientów o bardzo dużej przepustowości powinny być dostępne strumienie o najwyższej jakości.

Trudno jest jednak uzyskać płynną skalę jakości strumienia. Satysfakcjonującym rozwiązaniem będzie implementacja, w której strumień ma próg jakości wynoszący kilkadziesiąt kilobitów na sekundę.

3.1.10. Korekcja błędów

System musi zakładać awaryjność sieci i być przygotowany na otrzymanie niepełnego bądź wadliwego strumienia. Korekcja błędów składa się z dwóch etapów:

- Zapobieganie — polega na reakcji na błędy strumienia przed fazą odtwarzania.

- Poprawianie — polega na zatuszowaniu wadliwego strumienia.

W przypadku tego projektu rozwiązanie implementujące tylko fazę zapobiegania jest wystarczające.

3.2. Protokół sterujący

3.2.1. Wymagania funkcjonalne

Protokół sterujący musi udostępniać następujące funkcjonalności:

Węzły

Za pomocą protokołu sterującego nowy węzeł w sieci musi być w stanie podłączyć się do systemu. Węzeł staje się odbiornikiem w systemie, kiedy jakiś aktywny odbiornik zarejestruje go jako swojego sąsiada. Protokół powinien dostarczyć węzłom zestaw komunikatów, umożliwiający uzyskanie informacji na temat innych węzłów i nawiązywanie połączenia pomiędzy nimi.

Kanał

Węzeł zainteresowany danym kanałem musi mieć możliwość pobrania informacji na temat aktywnego kanału. Informacje o kanale to między innymi:

- nazwa kanału — nazwa identyfikująca kanał,
- opis kanału — opis określający treść danego kanału,
- czas transmisji — jak długo kanał jest aktywny.

Strumień

W warstwie tego protokołu będzie się również odbywać negocjacja wymiany danych multimedialnych pomiędzy sąsiadami. Oprócz samej negocjacji odbiorniki będą również ogłaszać swoim sąsiadom wieści o otrzymaniu nowego fragmentu strumienia oraz inne metadane o strumieniu.

3.2.2. Wymagania niefunkcjonalne

W tej części przedstawię wszystkie cechy dotyczące architektury protokołu sterującego zgodnie ze standardem projektowania protokołów warstwy aplikacji [18]. Cytowany dokument omawia najważniejsze kwestie, jakie należy wziąć pod uwagę przy projektowaniu protokołu.

Własności sieci

Według założeń protokół będzie działał w Internecie, czyli będzie wykorzystany zarówno w sieci WAN do przesyłania pakietów pomiędzy różnymi autonomicznymi systemami, jak i w sieci lokalnej. W ramach tego protokołu należy zwrócić uwagę na następujące własności sieci (Internetu):

- duże opóźnienie,
- dynamiczne zmiany trasy,
- gubienie się pakietów,
- zmiana kolejności dostarczenia pakietów,
- blokowanie pakietów przez rutery.

Symetryczny

Protokół należy do klasy protokołów punkt-do-punktu, dlatego komunikacja następuje pomiędzy równorzędnymi elementami. Jeśli dwa elementy komunikują się ze sobą, to zestaw wiadomości po obu stronach powinnien być taki sam.

Prosty

Protokół w założeniu ma powstać całkowicie od początku. Z tego powodu projekt tego protokołu będzie dotyczył wyłącznie własności tego systemu. Rozpatrywanie scenariuszy użycia tego protokołu w innych systemach jest zbędne. Prostota protokołu pozwoli na elastyczniejsze modyfikacje w przyszłości.

Oszczędny

W tym projekcie protokół sterujący jest wsparciem dla protokołu transmisji danych. Dlatego im mniejsze będzie wykorzystanie łącza przez komunikację, tym większa będzie przepustowość dla strumienia danych. Należy rozsądnie kalkulować narzut warstwy kontrolnej względem warstwy danych.

Odporny

Ze względu na zawodność Internetu protokół musi zakładać obsługę wszystkich sytuacji awaryjnych. Przez awarię należy rozumieć:

- zgubione pakiety,
- złą kolejność dostarczanych pakietów,
- zniszczone lub fałszywe pakiety,
- spóźnione pakiety,
- przyjście wielu pakietów w jednym momencie.

3.3. Protokół transmisji danych

Najważniejszym kryterium oceny wydajności systemu jest jakość ostatecznie odtwarzanego strumienia. Na to jakiej jakości będzie odtwarzany strumień będzie głównie wpływała wydajność protokołu transmisji danych. W przypadku rozwiązań z zakresu telewizji internetowej protokół transmisji musi być protokołem czasu rzeczywistego.

Drugą własność jaką musi posiadać protokół transmisji danych w tym projekcie to skalowalność względem przepustowości.

Zgodnie z projektem systemu zachowanie węzłów jest obserwowane i odnotowywane przez innych. Na zachowanie danego węzła składa się głównie jego aktywność w wymianie danych. Aby można było to zachowanie zaobserwować, protokół transmisji musi umożliwić zbieranie statystyk.

3.3.1. Protokół czasu rzeczywistego

W przypadku protokołów czasu rzeczywistego gwarancją jakości jest dostarczenie jak największej liczby pakietów na czas, a nie dostarczenie wszystkich w odpowiedniej kolejności.

Podstawowe własności protokołu czasu rzeczywistego to:

Synchronizacja pakietów z czasem

Pakiety danych muszą być oznaczone dokładnym znacznikiem czasowym, celem właściwego uporządkowania w czasie po stronie odbiorcy.

Minimalizacja czasu dostarczenia

Im szybciej pakiet dotrze do klienta, tym istnieje większa szansa, że klient zdąży go rozprzestrzenić dalej do innych zainteresowanych.

Odrzucanie nieważnych pakietów

Pakiet, który dotrze po czasie staje się bezużyteczny. Klient musi jak najszybciej porzucić ten pakiet, by nie zajmował on miejsca pakietom, które są jeszcze ważne.

3.3.2. Skalowalny względem przepustowości

Przez skalowalność rozumiemy możliwość przesłania przez protokół strumienia o dużej jakości. W razie wystąpienia awarii i zmniejszenia przepustowości łącza, do odbiorcy powinna trafić tylko część wyjściowego strumienia, o gorszej jakości.

Takie założenie wymaga, by protokół transmisji był świadomy jakości nadawanego strumienia i potrafił go dzielić na mniejsze.

3.3.3. Statystyki

Protokół powinien umożliwić zbieranie następujących statystyk:

- średnie opóźnienie łącza,
- stratność w transmisji,
- informacje o awarii lub zatorze sieci.

3.4. Implementacja

W ramach implementacji projektu telewizji internetowej bazującej na sieci punkt-do-punktu ma powstać:

- aplikacja nadajnika,
- aplikacja odbiornika z uproszczonym odtwarzaczem wideo.

3.4.1. Nadajnik

Podstawowy scenariusz działania dla aplikacji nadawczej wygląda następująco:

- stworzenie nowego kanału,
- wybieranie zasobu do transmisji na danym kanale,
- start kanału.

W ramach każdej z tych funkcjonalności nadajnik realizuje pewien zestaw zachowań, które omówię szczegółowo w dalszych punktach.

Tworzenie nowego kanału

Postanowienie o stworzeniu nowego kanału jest decyzją biznesową. Jako, że jest to czynność inicjująca uruchomienie całego systemu, musi być podjęta z należytym poprzedzającym planowaniem. W chwili tworzenia kanału jego autor powinien móc zdefiniować takie niezbędne informacje jak:

- nazwa kanału — krótki napis, który powinien być rozpoznawalny i charakterystyczny dla nadawanej treści. Napis ten będzie później rozgłaszany wśród wszystkich odbiorców i wyświetlany w liście kanałów oraz w tytule okna odtwarzania.
- opis kanału — dłuższy napis, określający bardziej szczegółowo treść nadawaną na kanale. Podobnie jak tytuł będzie rozgłaszany wśród odbiorców, ale będzie wyświetlany tylko na żądanie.
- port sterujący — port, na którym nadajnik będzie obsługiwał protokół sterujący.
- port strumienia — port, z którego nadajnik będzie nadawał strumień multimedialny.
- przepustowość — ograniczenia wyjściowe łącza, czyli ile danych może nadajnik w sumie wysyłać.
- liczba odbiorców — do ilu odbiorników może nadajnik wysyłać strumień.

Po wykonaniu czynności tworzenia kanału aplikacja nadajnika powinna zwrócić twórcy plik z meta-informacjami dotyczącymi kanału. Przy użyciu tego pliku aplikacje klienckie będą w dalszej fazie w stanie namierzyć nadajnik. Nadajnik ma ograniczenie na liczbę odbiorców, do których wysyła strumień, ale może komunikować się z dowolną liczbą zainteresowanych.

Wybór zasobu multimedialnego

Zasób multimedialny to strumień wideo, który jest główną treścią danego kanału. W wersji podstawowej dane multimedialne będą znajdować się w pliku statycznym na komputerze nadajnika.

Po wybraniu strumienia wejściowego aplikacja nadajnika powinna sprawdzić czy dany strumień jest poprawny. Jeśli poprawność została potwierdzona, to operacja przebiega pomyślnie.

Start kanału

W momencie wystartowania kanału nadajnik staje się aktywny w sieci. Daje to początek życia całemu systemowi. Nadajnik, będąc w stanie aktywnym, zaczyna reagować na komunikaty z sieci. Może rejestrować klientów do kanału oraz nadawać do nich strumień. Wraz z momentem uaktywnienia kanału zaczyna się proces transmisji strumienia. Strumień w nadajniku jest przetwarzany nawet, jeśli nie ma żadnego aktywnego odbiornika w sieci. Oznacza to tyle, że jeśli po pewnym czasie do kanału zgłosi się jakiś klient, to otrzyma strumień nie od początku, lecz od tego momentu przetwarzania, w którym się znajduje nadajnik.

3.4.2. Odbiornik

Jak w każdej innej sieci punkt-do-punktu najważniejszym elementem tego systemu jest odbiornik. To działanie każdego pojedynczego węzła oraz oddziaływanie między nimi określa strukturę całej sieci.

Schemat działania każdego odbiornika wygląda następująco:

- wprowadzenie danych o nadajniku,
- określenie ograniczeń łącza,
- rejestracja do kanału,
- rozpoczęcie odtwarzania,
- zatrzymanie odtwarzania.

Wprowadzenie danych o nadajniku

Zanim zarejestrujemy się do kanału musimy najpierw namierzyć sieć nadajników, które transmitują dany kanał. Informacje o systemie oraz o kanale posiada dowolny odbiornik/nadajnik będący węzłem w systemie. Informacje o nadajniku można wprowadzić podając:

- adres i port sterujący nadajnika — aplikacja powinna dostarczyć okno do wprowadzenia tych informacji,
- plik wygenerowany przez nadajnik — przez okienko ładowania pliku możemy wczytać ten plik.

W pierwszym przypadku połączymy się dowolnym węzłem w systemie, niekoniecznie będzie to nadajnik. W drugim przypadku zainteresowany zgłosi się prosto do węzła nadającego.

Ustawienia łącza

W opcjach aplikacji użytkownik może ustawić jakie mają być ograniczenia łącza dla danej aplikacji. Ograniczenia jakie można ustawić dla odbiornika to:

- przepustowość wejściowa — z jaką sumaryczną prędkością aplikacja będzie mogła ściągać dane,
- przepustowość wyjściowa — z jaką sumaryczną prędkością aplikacja będzie mogła wysyłać dane,
- liczba sąsiadów — z iloma węzłami w sieci aplikacja może uczestniczyć w wymianie danych.

Zmieniając ustawienia łącza użytkownik wpłynie na jakość odbieranego strumienia.

Rejestracja do kanału

Kiedy użytkownik decyduje się na zarejestrowanie się do kanału, aplikacja wysyła prośbę o rejestrację do podanego nadajnika i oczekuje na akceptację. Jeśli otrzyma odpowiedź odmowną, to użytkownik zobaczy okienko z odpowiednim komunikatem. W przypadku gdy nadejdzie zgoda na rejestrację, aplikacja przełączy się na okno odtwarzania w stanie zatrzymanym. Cały proces negocjacji strumienia, łączenia się z innymi węzłami sieci zachodzi automatycznie.

Rozpoczęcie odtwarzania

Funkcja rozpoczęcia odtwarzania przenosi odtwarzacz ze stanu zatrzymania w stan odtwarzania. Odtwarzacz w stanie odtwarzania wyświetla strumień wideo zsynchronizowany z zegarem odtwarzania. Ta czynność nie wpływa w żaden sposób na komunikację ani wymianę danych w jakiej uczestniczy odbiornik.

Zatrzymanie odtwarzania

Funkcja zatrzymania odtwarzania zatrzymuje odtwarzanie wideo. Podobnie jak "rozpoczęcie odtwarzania" ta akcja nie wpływa na działanie odbiornika.

3.4.3. Wymagania niefunkcjonalne

Wieloplatformowy

Aplikacja powinna być wieloplatformowa, co umożliwi szerszej rzeszy użytkowników korzystanie z niej. Możliwość uruchomienia na różnych systemach operacyjnych urozmaici również testy systemu oraz być źródłem ciekawych wniosków.

Podział na różne warstwy

Niezależność implementacji protokołu sterującego, protokołu transmisji oraz aplikacji odtwarzania oznacza większą elastyczność systemu. Przy ewentualnych zmianach projektowych wymiana jednej warstwy będzie niekosztowna.

Statystyki

Podczas działania aplikacja kliencka nieustannie zbiera statystyki o stanie sieci oraz na temat swoich sąsiadów. Te statystyki stanowią dodatkowe źródło informacji na temat działania systemu. Analiza tych informacji może dać ciekawe rezultaty i pomóc w procesie ulepszenia aplikacji.

Rozdział 4

Technologie

W tym rozdziale opiszę tło technologiczne projektu. Przedstawię koncepcje, na których oparty jest system i szczegółowo omówię każdą wykorzystaną technologię. Opis zawarty w tym rozdziale ma wymiar teoretyczny i nie dotyczy szczegółów implementacyjnych, które zostaną omówione dopiero w następnym rozdziale.

Każda sekcja w tym rozdziale opisuje oddzielną technologię. Na początku każdej sekcji przedstawię cel i założenie wykorzystanej koncepcji. Później omówię specyfikację i najistotniejsze rozwiązania danej technologii. Na końcu pokażę jak ta technologia będzie wykorzystana w moim projekcie.

4.1. BitTorrent

Protokół BitTorrent [6] został zaprojektowany w 2001 roku przez Brama Cohena w celu dystrybucji i wymiany plików przez Internet. W tym samym roku autor stworzył również pierwszą implementację tego protokołu wydaną pod tą samą nazwą. Proporcjonalnie do popularności protokołu wzrasta również liczba jego implementacji, których w tym momencie jest ponad 60 (wg Wikipedii).

Sukces protokołu BitTorrent pokazał, że ta technologia jest wydajna i efektywna. Najpopularniejsze udostępnione zasoby potrafią być rozpowszechniane w sieciach z dziesiątkami tysięcy węzłów. Często jedynym ograniczeniem dla prędkości ściągania zasobu jest przepustowość łącza ściągającego.

W tym rozdziale przedstawię główne zalety protokołu BitTorrent i wyjaśnię jak wykorzystałem wiedzę o protokole BitTorrent do opracowania własnego protokołu.

4.1.1. Cele

- rozpowszechnianie dużych porcji danych,
- minimalizacja czasu pobrania całego zasobu,
- redundancja danych i eliminacja pojedynczego punktu awarii,
- zmniejszenie wymagań sprzętowych i łącza osoby rozpowszechniającej zasób,
- optymalne wykorzystanie przepustowości całej sieci,
- odporność na awarie sieci.

4.1.2. Założenia

- pliki wchodzące w skład zasobu są danymi statycznymi — nie zmieniają się podczas całego procesu rozpowszechniania,
- użytkownicy mają dobre intencje — mało jest osób, które będą pobierać a nie udostępniać,
- sieć obsługuje protokół TCP.

4.1.3. Specyfikacja

Terminologia

- Serwer śledzący (ang. *tracker*) — serwer śledzący wszystkie węzły w ramach jednej sieci punkt-do-punktu.
- Węzeł (ang. *peer*) — element w sieci punkt-do-punktu.
- Klient — aplikacja pozwalająca na wymianę plików w protokole BitTorrent.
- Torrent — plik z informacjami określający jakiś zasób, może być też rozumiany jako jednostka zasobu w sieci BitTorrent.
- Ławica (ang. *swarm*) — zbiór węzłów w sieci punkt-do-punktu zainteresowanych tym samym zasobem.
- Siewnik (ang. *seeder*) — węzeł w sieci punkt-do-punktu, który pobrał całość danych i już tylko je udostępnia, nic nie pobierając
- Pijawka (ang. *leech*) — węzeł w sieci punkt-do-punktu, który ściąga znacznie więcej danych niż wysyła.
- Fragment (ang. *piece*) — fragment pliku, opisany w metapliku torrent.
- Blok — element fragmentu pliku, minimalna jednostka danych wymieniana pomiędzy węzłami.
- Jazda na gapę (ang. *freeriding*) — zachowanie określające wymianę danych, kiedy interesant tylko ściąga, a nic nie wysyła.
- Wet-za-wet (ang. *tit-for-tat*) — strategia w teorii gier.

Scenariusz użycia

1. Użytkownik podaje klientowi na wejściu plik torrent opisujący zasób.
2. Klient wybiera pewien fragment pliku, który chciałby najpierw pobrać.
3. Klient pyta serwer śledzący podany w pliku torrent o uczestników posiadających ten fragment.
4. Serwer śledzący odsyła listę klientów posiadających jakikolwiek blok danego fragmentu.
5. Klient łączy się z niektórymi klientami z podanej listy.

6. Klient wysyła prośbę o blok do innego klienta.
7. Klient otrzymuje blok od innego klienta.
8. Kiedy klient otrzyma wszystkie bloki, łączy je w całość.
9. Klient powiadamia węzeł śledzący, że ściągnął cały fragment.
10. Klient może kontynuować pobieranie fragmentów powtarzając proces od 2. kroku.

Komunikacja

Komunikat może być w jednym z następujących stanów:

- zablokowany (ang. *choke*) — wiadomość oznacza, że adresat został zablokowany przez nadawcę i żadne pakiety nie będą do niego wysyłane.
- odblokowany (ang. *unchoke*) — powiadamia o wznowieniu wysyłania z nadawcy do adresata.
- zainteresowany (ang. *interested*) — nadawca jest zainteresowany danymi, które posiada adresat.
- niezainteresowany (ang. *not interested*) — nadawca nie jest już zainteresowany stanem zasobów adresata.
- posiadany (ang. *have*) — nadawca powiadamia adresata, że znalazł się w posiadaniu całego fragmentu.
- mapa (ang. *bitfield*) — mapa bitowa fragmentów skompletowanych przez nadawcę. komunikat ten jest wysyłany na początku konwersacji.
- żądanie (ang. *request*) — prośba o przesłanie danego bloku.
- fragment (ang. *piece*) — pakiet zawierający proszony blok.
- anulowanie (ang. *cancel*) — anulowanie prośby o blok.

W dalszej części opisuję najważniejsze cechy protokołu BitTorrent.

Niestrukturyzowana

Według klasyfikacji sieci punkt-do-punktu, sieć BitTorrentowa należy do kategorii sieci niestrukturyzowanych. Podobnie jak w innych sieciach niestrukturyzowanych protokół BitTorrent nie wymaga, by węzły w sieci ustanawiały połączenia wedle określonej struktury. Każdy klient łącząc się do serwera śledzącego otrzymuje tylko listę innych węzłów. Decyzja o tym, z którymi węzłami klient się połączy jest dokonywana lokalnie przez każdego klienta i może się zmieniać w czasie.

Fragmentacja

W odróżnieniu od tradycyjnego protokołu HTTP, który wysyła dane w postaci strumienia, BitTorrent używa fragmentacji. Wszystkie pliki wchodzące w skład nadawanej treści są dzielone na kawałki. Każdy z tych fragmentów jest opisywany w pliku torrent wraz z sumami kontrolnymi. Każdy plik torrent ma zdefiniowaną stałą wielkość fragmentu (najczęściej 256 KB lub 1 MB). W późniejszej fazie transmisji te fragmenty są dzielone na jeszcze mniejsze części zwane blokami (zazwyczaj 16 KB). Blok to jednostka, którą węzły posługują się podczas wymiany danych. Po otrzymaniu wszystkich bloków danego fragmentu, klient wylicza sumę kontrolną fragmentu i porównuje ją do sumy zapisanej w pliku torrent.

Kolejność pobierania kawałków może być dowolna, aczkolwiek nieoficjalna specyfikacja protokołu [1] zaleca używanie strategii **losowy na początku** i **najpierw najrzadszy**. Mechanizm **losowy na początku** dotyczy sytuacji kiedy nowy klient dołącza do wymiany i nie posiada jeszcze żadnego fragmentu, którym mógłby się wymieniać. Praktyka ta mówi, że wybór pierwszego fragmentu pliku do pobrania powinien być dokonany w sposób całkowicie losowy. Klient stosujący strategię **najpierw najrzadszy** wybierając fragment do pobrania decyduje się na taki, który znajduje się w posiadaniu najmniejszej liczby węzłów. Jeśli jest wiele kawałków, które spełniają powyższe kryterium, to ostateczny wybór zostanie dokonany w drogą losowania. Dzięki zastosowaniu takiej strategii przez klientów, system BitTorrent zwiększa entropię dostępnych fragmentów.

Freeriding

Do zwalczania zjawiska **jazdy na gapę** protokół BitTorrent wykorzystuje koncepcję pochodzącą z podstaw teorii gier — wet za wet. Strategia **wet za wet** jest znana jako najkorzystniejsze rozwiązanie dylematu więźnia [17]. Składa się ona z dwóch prostych zasad:

- na początku współpracuj,
- potem rób to co przeciwnik zrobił w poprzedniej turze.

Mechanizm wet za wet w klientach BitTorrentowych jest adaptowany do kontroli wymiany danych pomiędzy dwoma węzłami w sieci. Według tej taktyki przy początkowym etapie wymiany danych, klient implementujący wet za wet będzie starał się spełniać żądania drugiej strony. Jeśli jego żądania będą również spełniane, to wymiana powinna przebiegać płynnie. W przypadku kiedy druga strona przestanie reagować na prośby danego klienta, wtedy dany klient zablokuje żądania od nieuczciwego sąsiada. Gdy żądania danego klienta znowu będą spełniane, nastąpi również odblokowanie sąsiada.

4.1.4. Podsumowanie

W moim projekcie postanowiłem wykorzystać wiedzę zebraną na podstawie protokołu BitTorrent do zbudowania własnego protokołu sterującego. Głównym powodem, dla którego wzoruję się na protokole BitTorrent jest właśnie wydajność wspomnianego protokołu. Jako użytkownik sieci BitTorrentowych przekonałem się, że klient obsługujący protokół BitTorrent potrafi optymalnie wykorzystać przepustowość dowolnego łącza. Inaczej niż w innych klientach punkt-do-punktu prędkość ściągania danych wzrasta błyskawicznie. Z perspektywy użytkownika można również zauważyć, że klient protokołu BitTorrent potrafi dynamicznie dostosować się do zmian w sieci, poprzez ciągłą wymianę sąsiadów.

System opisany w tej pracy będzie wykorzystywał wiele koncepcji powstałych w protokole BitTorrent. Główna idea zaczerpnięta z protokołu BitTorrent to fragmentacja danych. W

przypadku tego systemu oznacza to podzielenie strumienia multimedialnego na małe kawałki i przesyłanie ich oddzielnie zamiast używania ciągłego strumieniowania. Drugie podobieństwo to topologia sieci, która pozostanie niestrukturyzowana, tak by była najlepiej przystosowana do dynamicznego przyłączania i odłączania węzłów. Wykorzystam również mechanizm wet za wet do zwalczania nieuczciwych użytkowników sieci. W moim systemie podobnie jak w BitTorrent punktem łączącym różnych użytkowników będzie ich wspólne zainteresowanie rozgłaszaną treścią. W tym przypadku będzie to kanał telewizyjny.

Rodzaj wymienianych danych to jednak kluczowa różnica pomiędzy BitTorrentem a opisanym w tym projekcie systemem. Strumień multimedialny „na żywo" z pewnością nie spełnia już podstawowych założeń protokołu BitTorrent. W odróżnieniu od przesyłania plików statycznych, w strumieniach czasu rzeczywistego kolejność dotarcia fragmentów ma kluczowe znaczenie. Z drugiej strony przesyłając dane multimedialne, nie musimy gwarantować dostarczenia. Z tej obserwacji wynikają dwie istotne zmiany:

- znaczniki czasowe — każdy pakiet danych przesyłanych w sieci będzie oznaczony znacznikiem czasowym, tak by można było uporządkować je w czasie i w razie potrzeby odrzucać przeterminowane.

- protokół UDP — zamiast używania niezawodnego protokołu TCP, należy wykorzystać lżejszy i szybszy protokół UDP. W szczególności do przesyłania strumienia multimedialnego będzie użyty protokół RTP.

4.2. Multiple Description Coding

Obecne techniki kodowania strumieni multimedialnych nie zostały stworzone z myślą o transmisji sygnału telewizyjnego po Internecie. Większość współczesnych systemów kodowania sygnału audiowizualnego zakłada wysoką niezawodność nośnika danych. Również nośniki, używane obecnie do przechowania i transportowania danych multimedialnych zostały stworzone w celu minimalizacji utraty danych. Przy takich założeniach najlepiej się sprawują nieprogresywne kodowania, które dają największą kompresję. Często są używane również kodowania progresywne, które dają większą elastyczność w doborze jakości, ale nadal zakładają niezawodność warstwy transportu.

Te założenia w przypadku tworzenia telewizji internetowej są nieodpowiednie. W realizacji systemów czasu rzeczywistego ważniejsze są założenia związane z synchronizacją i dynamicznym dostosowaniem transmisji danych do warunków sieciowych. W obecnym stanie Internetu, wszelkie założenia dotyczące niezawodności transmisji są nierealistyczne i wymagają rewizji. W miejscu gdzie technika kodowania multimediów styka się z Internetem [10] istotną rolę może odegrać Multiple Description Coding.

4.2.1. Cele

- skalowalna jakość strumienia multimedialnego — strumień ma wiele poziomów jakości. W zależności od jakości strumień wymaga różnej przepustowości transmisji.

- duża niezależność względem błędów transmisji — błędy w transmisji powodują wyłącznie zmniejszenie jakości strumienia, a nie brak strumienia.

- dynamiczna zmiana jakości strumienia — przy awarii łącza spadek jakości strumienia jest natychmiastowy. Strumień dopasowuje się do przepustowości łącza zamiast je zatykać.

4.2.2. Założenia

- łącze jest zawodne — w przeciwnym przypadku użycie MDC jest zbędne.
- strumień może być stratny — nie wymaga się, by strumień był dostarczany w całości. Zakładamy, że odbierany strumień o gorszej jakości względem oryginalnego jest akceptowalny.

4.2.3. Specyfikacja

Multiple Description Coding to kodowanie należące do rodziny Scalable Video Coding. Koncepcja kodowania **Scalable Video Coding** powstało na potrzebę nadawania jednego strumienia audiowizualnego do wielu odbiorców o zróżnicowanych łączach. Głównym celem SVC jest zakodowanie danych multimedialnych do takiego formatu, który będzie miał różne poziomy jakości. Chodzi zatem o to, by strumień wyjściowy był podzielony na części i otrzymanie większej liczby części oznaczało wyższą jakość. W ramach SVC rozróżniamy dwie metody kodowania: Layered Coding (wykorzystywany między innymi w MPEG-2 i MPEG-4) i Multiple Description Coding. Layered Coding udostępnia poziomy kodowania i użytkownik w ramach zwiększenia jakości odbieranego strumienia dekoduje kolejne warstwy.

W przeciwieństwie do Layered Coding, Multiple Description Coding nie rozróżnia warstwy kodowania. Cała istota MDC polega na podzieleniu strumienia multimedialnego na podstrumienie, z których dekodowanie dowolnego podzbioru podstrumieni daje odtwarzalny obraz. Oczywiście zgodnie z koncepcją SVC, im większy jest podzbiór podstrumieni tym jakość jest wyższa. MDC w wielu sytuacjach może okazać się gorsze od standardowych rozwiązań.

4.2.4. Formalny problem

Dany mamy strumień nadawany S_0, oraz strumień odbierany E_0. Oznaczamy R_0 jako wielkość przesyłanego strumienia. Określamy również miarę zakłócenia — D_0, jako odległość pomiędzy S_0 i E_0 według dowolnej metryki (najczęściej euklidesowa).

Dodatkowo definiujemy dwie funkcje:

- R(D) — dla danego zakłócenia wynikiem jest minimalna możliwa wielkość strumienia, przy którym można uzyskać taki poziom zakłócenia.
- D(R) — analogicznie dla danej wielkości strumienia określamy minimalne zakłócenie jakie można osiągnąć nadając strumień.

Multiple Description Coding to podzielenie nadawanego strumienia na n podstrumieni, $S_0 = S_1 + ... + S_n$. Następnie każdy podstrumień i przesyłamy zajmując R_i przepustowości łącza i otrzymując wynikowy strumień E_i oraz zakłócenie D_i. Celem jest odpowiedni podział na podstrumieni, by zminimalizować następujące wartości:

- $\sum_{i=1}^{n} R_i$ — minimalne sumaryczne zużycie łącza. Optymalna wartość to $R(D_0)$.
- $\forall X \subseteq \{E_1, ..., E_n\} : D(\sum X)$ — strumień złożony z każdego podzbioru strumieni był jak najbliższy oryginalnemu.

Najczęściej nie da się jednocześnie zoptymalizować wartości obu powyższych wskaźników. Każdy projekt ma inne wymagania względem przepustowości i jakości, dlatego też to projektujący musi zadecydować, który warunek jest ważniejszy.

4.2.5. Podsumowanie

Wykorzystywanie Multiple Description Coding to adaptacja pomysłu fragmentacji danych do kodowania danych audiowizualnych. Dzięki użyciu koncepcji MDC, transmitując podstrumienie wideo, możemy rozdzielić logikę przesyłania danych od mechanizmu kodowania. Warstwa transportu przy nadawaniu i odbieraniu danych multimedialnych, nie musi sprawdzać ich zawartości, a jedynie skupić się na przetransportowaniu jak największej liczby danych.

Multiple Description Coding jest rozwiązaniem kompromisowym, dlatego też jego użycie jest dużym wyzwaniem i wymaga dokładnego rozpatrzenia. Żeby ta metoda dawała maksymalną wydajność, specyfikacja sieci musi być dobrze rozpoznana.

W ramach tego projektu nie zostanie zaimplementowany wydajny zestaw koder/dekoder MDC. Do zaprezentowania systemu stworzę minimalny prototyp kodowania MDC. W ramach testów zbadam działanie całego systemu na różnych ustawieniach sieciowych i omówię jak wpływa to na efektywność kodowania MDC.

4.3. Real-time Transport Protocol

Pierwotne koncepcje Internetu zakładały iż jego głównym celem będzie transportowanie danych statycznych. Jeszcze zanim został otwarty do celów komercyjnych, jego jedyne użycie polegało na przesyłaniu e-maili i statycznych dokumentów. Do zastosowań takiego typu został też stworzony wiarygodny, strumieniowy protokół Transmission Control Protocol (w skrócie TCP).

Niemniej, pomimo niezwykłych możliwości jakie daje nam protokół TCP, nie jest on optymalnym odpowiedzią na wszystkie pytania Internetu. Znając lepiej ten protokół uświadamiamy sobie, że jest on doskonale dopasowany do takich zadań jak przesyłanie e-maili, pobieranie stron WWW, tworzenie i utrzymywanie połączeń ze zdalną maszyną i wiele innych zadań wczesnego Internetu. Niestety próby użycia protokołu TCP do innych celów, takich jak transmisja strumienia multimedialnego na żywo, pokazują, że nie jest to rozwiązanie efektywne.

Okazuje się, że nie do każdego zastosowania zawsze potrzebujemy niezawodnego kanału transmisji. Zresztą jak pokazały radio i telewizja, multimedia mogą być nadawane przy użyciu zawodnego systemu transmisji. I wydaje się to być rozwiązanie całkowicie wystarczające. Przy takim założeniu powstał protokół Real-time Transport Protocol (w skrócie RTP). RTP jest to protokół warstwy aplikacji (według modelu TCP/IP), zbudowany na protokole User Data Protocol (w skrócie UDP) — bracie protokołu TCP, który w przeciwieństwie do TCP jest protokołem bezpołączeniowym i stratnym.

4.3.1. Cele

- transport mediów czasu rzeczywistego — dostarczenie usługi do transmisji wideo i audio po sieci IP.
- uporządkowanie według czasu — w przypadku pomieszania kolejności dojścia pakietów protokół porządkuje je chronologicznie według znacznika czasowego.
- wykrywanie i korekcja błędów — wykrycie błędów w otrzymanych danych i ich korekcja.
- zarządzanie członkostwem — zarządzanie zbiorem użytkowników biorących udział w wymianie danych.

- raport z jakości danych — zbieranie informacji o jakości otrzymanych danych i przekazanie ich do nadawcy.

4.3.2. Założenia

- dane czasu rzeczywistego — przesyłane dane muszą mieć charakter strumienia czasu rzeczywistego, czyli muszą być znakowalne stemplem czasowym.
- stratność — nie wszystkie dane od nadawcy muszą dotrzeć do odbiorcy.
- zawodne łącze — łącze wykorzystywane do transmisji danych jest awaryjne i nie gwarantuje dostarczenia danych w tej samej postaci w jakiej zostały nadane. W szczególności dane mogą zostać całkowicie utracone.
- zasada z końca do końca — odpowiedzialność za wiarygodność danych należy do punktów końcowych (komunikujących się), a nie do systemu transportującego. Aplikacje korzystające z protokołu RTP muszą implementować całą logikę zarządzania połączeniem i siecią. Protokół RTP tylko zdaje zwrotne informacje na temat stanu sieci i danych, a nie dokonuje retransmisji w sposób automatyczny.

4.3.3. Specyfikacja

Pierwsza specyfikacja [25] protokołu RTP powstała w styczniu 1996 roku. Protokół RTP jest zbudowany na protokole UDP/IP, urozmaicając warstwę transportu o:

- segregowanie i synchronizację w czasie,
- identyfikację danych i nadawców,
- znakowanie ważnych zdarzeń w strumieniu.

oraz dostarcza dodatkowych informacji w postaci:

- wykrywanych strat,
- raportów o jakości odbioru.

Protokół Real-time Transport Protocol składa się z dwóch nieodłącznych pod-protokołów:

- protokół przesyłania danych — zarządza dostarczaniem danych czasu rzeczywistego, takich jak strumienie audio-wideo.
- protokół kontrolny — udostępnia raporty i informacje na temat jakości otrzymanych danych, uczestników biorących udział w wymianie oraz synchronizacji pomiędzy różnymi strumieniami.

Podstawowym pojęciem w protokole RTP jest to **sesja**. Sesja określa grupę uczestników, którzy komunikują się ze sobą przy użyciu protokołu RTP. Każda sesja dotyczy jednego strumienia. Na przykład podczas nadawania treści audiowizualnej potrzebne są przynajmniej dwie sesje — jedna dla dźwięku, druga dla obrazu. Każdy uczestnik identyfikuje sesję po adresie, parze portów, z którego nadaje, oraz parze portów, z którego odbiera dane. Każda para portów składa się odpowiednio z portu dla protokołu danych i dla protokołu kontrolnego. Najczęściej porty nadawania i odbierania są takie same. W pierwszej specyfikacji wymagano, by port protokołu danych miał numer parzysty, a port protokołu kontrolnego wartość o jedną wyższą. W ostatniej wersji specyfikacji zrezygnowano z takiego wymagania. Każdy uczestnik w sesji jest identyfikowany przez unikatową dla danej sesji 32-bitową liczbę całkowitą. Ta wartość jest generowana w sposób losowy przez uczestnika przy wstępowaniu do sesji.

Protokół danych

Schemat pakietu wymiany danych wygląda następująco:

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
V			P	X		CC		M			PT					Numer sekwencyjny																
Znacznik czasowy																															
Identyfikator źródła																															
Dodatkowe źródła ...																															
Rozszerzony nagłówek																															
Ładunek																															

Tabela 4.1: Struktura pakietu danych protokołu Real-time Transport Protocol

Objaśnienie pól:

- **V** — numer wersji, wymagane 1.
- **P** — obicie (ang. padding), ustawione jeśli dany pakiet przekracza normalną długość i zawiera dodatkowe dane.
- **X** — rozszerzenia. Jest ustawiony jeśli pakiet zawiera rozszerzony nagłówek.
- **CC** — liczba dodatkowych źródeł. Określa ile jest innych uczestników, którzy wzięli udział w nadawaniu pakietu. Ta wartość jest niezerowa kiedy strumień został zmiksowany z kilku innych strumieni.
- **M** — oznakowanie, ustawione jeśli dana paczka jest specyficzna (np. oznacza koniec strumienia).
- **PT** — typ ładunku (ang. *payload type*). Wartość określająca typ mediów (np. kodowanie). Przy użyciu tego pola aplikacja podejmuje decyzję o tym jak potraktować otrzymane dane.
- **Numer sekwencyjny** — te pola pakietów są znakowane kolejnymi wartościami. Przy użyciu tego pola odbiorca może stwierdzić, że brakuje pewnych pakietów, bądź że doszły one w złej kolejności. To pole nie służy do synchronizacji danych w czasie.
- **Znacznik czasowy** — 32 bitowa liczba naturalna. Określa kolejność pakietów w czasie według z góry narzuconego sposobu odtwarzania. Jeśli na przykład mamy strumień telewizyjny z częstotliwością odświeżania 50 Hz, to pakiety z znacznikami czasowymi 120 i 121 będą wyświetlane w odstępie 20 ms.
- **Identyfikator źródła** — 32-bitowy identyfikator nadawcy pakietu w ramach tej sesji.
- **Dodatkowe źródła** — identyfikator dodatkowych źródeł.
- **Rozszerzony nagłówek** — jeśli bit **X** został ustawiony, to te pola nie są puste. Rozszerzony nagłówek zawiera na początku 16-bitowy typ rozszerzenia oraz 16-bitową długość rozszerzonego nagłówka.
- **Ładunek** — dane binarne strumienia multimedialnego.

Protokół kontrolny

Protokół kontrolny (ang. *RTP Control Protocol*) jest uzupełnieniem dla protokołu danych. Służy on do informowania uczestników sesji o stanie danych przesyłanych w protokole danych. Dostarcza również wiadomości o zmianach w członkostwie w sesji oraz informacje potrzebne do synchronizacji strumieni.

Implementacja protokołu RTCP składa się z trzech części:

- formaty pakietów — rodzaje pakietów kontrolnych i ich interpretacja. RTCP zawiera wiele rodzajów pakietów, takie jak raporty odbiorcy, raporty nadawcy, opis źródła, komendy związane z uczestnictwem w sesji.
- reguły czasowe — pakiety RTCP są wysyłane okresowo w celu dostarczenia informacji wszystkim uczestnikom sesji. Dokładne zasady jakie komunikaty mają być wysłane i w jakich odstępach czasowych, są definiowane w ramach tych reguł. Ruch na potrzeby protokołu RTCP jest najczęściej utrzymywany na poziomie 5% całego ruchu w sesji.
- baza danych uczestników — w ramach protokołu RTCP utrzymywane są dane o zbiorze wszystkich uczestników sesji, wraz z zebranymi o nich informacjami.

Protokół specyfikuje dokładnie wiele pakietów do komunikacji oraz posiada złożone schematy czasowe. Opisanie ich nie leży w zakresie tej pracy.

4.3.4. Podsumowanie

Real-time Transport Protocol jest obecnie najlepszym protokołem do przesyłania danych czasu rzeczywistego po sieciach IP. W ramach tego projektu użyję implementacji protokołu RTP do przesyłania danych multimedialnych oraz do zbierania informacji na temat wydajności łącza. Dzięki bezstanowości protokołu RTP transmisja danych jest wydajniejsza. RTP daje również aplikacjom końcowym większą kontrolę nad siecią.

Dzięki elastyczności specyfikacji protokołu RTP dostosowanie go do tego projektu obyło się bez większych modyfikacji.

Rozdział 5

Projekt i implementacja

Opiszę w tej części projekt i implementację systemu przekaźników strumienia multimedialnego. Na samym początku wymienię decyzje projektowe, jakie podjąłem w związku z pracą nad tym systemem. W dalszej części omówię działanie całego systemu jako sieci punkt-do-punktu. W ostatniej części tego rozdziału przedstawię implementację najważniejszego elementu systemu — aplikacji odbiorcy.

5.1. Decyzje projektowe

Mianem dobrego produktu informatycznego nie należy określać każdego oprogramowania, które spełnia wyznaczone w specyfikacji wymagania. Branża informatyczna, jak żadna inna nakłada na twórców oprogramowania dodatkową odpowiedzialność utrzymywania, naprawiania i rozwijania swoich produktów. Dobre oprogramowanie to takie, które powstało w myśli wieloletniego rozwoju oraz w zgodzie z dobrymi praktykami programistycznymi. Projekty robione „na czas", tylko na początku wydają się być udanymi, a w dłuższej perspektywie okazują się być zmorą dla twórcy. Dlatego też w procesie tworzenia dobrego oprogramowania niezwykle istotne, jeśli nie najistotniejsze, są decyzje projektowe. Decyzje projektowe to zbiór zasad, które twórca projektu zobowiązuje się przestrzegać w procesie tworzenia całego produktu.

W tym rozdziale wymienię i opiszę wszystkie decyzje projektowe jakie nałożyłem na ten projekt. Podjęcie takich a nie innych decyzji opieram na swoich doświadczeniach związanych z pracą zawodową, podczas której poznałem różne praktyki dobrego programowania i przede wszystkim wyciągnąłem kluczowe wnioski na podstawie popełnionych błędów.

5.1.1. Zwinna metodologia

Pierwsza i najważniejsza decyzja projektowa to postanowienie zastosowania metodologii **zwinnej** (ang. agile) [28] do tworzenia tego systemu. Procesy tworzenia oprogramowania określane jako zwinne metodologie powstały jako opozycja powszechnie stosowanych ciężkich procesów takie jak Rational Unified Process. W przeciwieństwie do ciężkich metodologii zwinne nie zakładają tworzenia dokładnej specyfikacji produktu we wczesnej fazie projektu. W zamian kładą większy nacisk na stopniowe tworzenie działającego oprogramowania. Proces tworzenia w metodologii zwinnej składa się z małych iteracji, w których po każdej powstaje gotowy produkt. Gotowy produkt to taki, który jest przetestowany, posiada dokumentację i może być dostarczony do klienta, ale nie musi implementować wszystkich funkcjonalności. Przy takim systemie pracy ocena postępu projektu jest ściślejsza i lepiej odzwierciedla rzeczywisty stan.

Pracując nad tym projektem tworzyłem stopniowo implementację, dodając w każdej iteracji nowe funkcjonalności i od razu testując je. Po każdej iteracji system był w stanie działającym.

5.1.2. Wieloplatformowość

Chciałem również stworzyć aplikację, która nie będzie zależna od platformy ani systemu, dzięki czemu będzie dostępniejsza oraz umożliwi wykonywanie testów na różnych systemach operacyjnych. Program niezależny od platformy charakteryzuje się tym, że kod źródłowy tego programu bez żadnych modyfikacji może zostać uruchomiony na różnych platformach i działać tak samo.

W realizacji tego założenia wykorzystałem język Java, który nie jest kompilowany do kodu natywnego, lecz do kodu pośredniego, który potem jest uruchamiany na maszynie wirtualnej Javy, dostępnej dla różnych platform. Nowsze maszyny wirtualne Javy oprócz opcji interpretera udostępniają również możliwość kompilacji kodu pośredniego do kodu natywnego w locie — w czasie działania aplikacji, co znacznie przyspiesza aplikację. Java często w opinii publicznej jest uważana za język wolny w porównaniu z językami kompilowanymi. Nie jest to jednak całkowicie zgodne z rzeczywistością. Java przegrywa z C++ pod względem szybkości operacji arytmetyczno-logicznych, jednakże większość testów związanych z dostępem do wejścia/wyjścia oraz operacjami zarządzania pamięcią daje wyniki faworyzujące Javę [13]. Z tego też powodu wybór języka Java wydaje się być w tym zastosowaniu trafny.

5.1.3. Modularność

Bardzo ważną praktyką programistyczną jest zaprojektowanie i stworzenie systemu informatycznego w sposób modularny. Kod całego programu jest przez to bardziej przejrzysty i łatwiejszy w zarządzaniu. Również ułatwia to pracę w metodologii zwinnej, ponieważ funkcjonalności mogą zostać oddzielone do różnych modułów i implementowane stopniowo. W mojej implementacji spróbuję utrzymać jak największą modularność, tak by każda logiczna część aplikacji została wydzielona.

5.1.4. Interfejs programowania aplikacji

Podział projektu na moduły służy nie tylko lepszemu zarządzaniu kodem, lecz również logicznemu odseparowaniu od siebie usług. Ważne jest w takim przypadku, by usługi udostępnione przez moduły zostały wyraźnie zdefiniowane. Taka definicja usługi jest nazywana interfejsem programowania aplikacji i powinna być jedynym sposobem komunikacji modułu z światem zewnętrznym. Co więcej przy projektowaniu takich usług interfejs udostępnionych funkcjonalności powinien być niezależny i przezroczysty względem leżącej pod spodem implementacji.

Projekt obejmuje wiele modułów, które będą się komunikować przez interfejs programowania aplikacji. Projektując i pisząc te interfejsy postaram się postępować zgodnie z zasadami tworzenia dobrych interfejsów [2].

5.1.5. Programowanie sterowane testami

Ważne doświadczenie jakie nabyłem podczas pracy nad tworzeniem oprogramowania to podejście znane jako **programowanie sterowane testami** (ang. *test-driven development*), w skrócie TDD. TDD powstało jako jeden z fundamentalnych mechanizmów **programowania ekstremalnego** (ang. extreme programming), w skrócie XP. TDD zakłada, że oprogramowanie jest dobre tylko jeśli jest dokładnie przetestowane. Jednakże, testowanie oprogramowania

po jego powstaniu często nie jest wyczerpujące i nie wykrywa wszystkich błędów stworzonego kodu źródłowego. Podejście TDD proponuje odwrócić tą kolejność i rozpoczynać proces od przygotowania testów, a dopiero w następnym kroku pisać kod. W tej metodologii testy przejmują rolę specyfikacji i dlatego są tworzone tak, by w pełni określały działanie produktu. Mając gotowy test, programista próbuje stworzyć kod, który przechodzi wszystkie wymagane testy. Po spełnieniu wszystkich testów danej funkcjonalności, programista może rozpocząć implementację nowej funkcjonalności — zaczynając znowu od pisania testów do niej.

5.1.6. Wykorzystywanie gotowych rozwiązań

Przemysł informatyczny to doskonałe środowisko do wymiany know-how i rozpowszechniania wiedzy. Jedynie w tej branży obok produktów wartych kolosalnych kwot możemy znaleźć rozwiązania, którego są całkowicie za darmo. Uważam, że nie należy odkrywać koła na nowo, dlatego też w swojej pracy będę wykorzystywał tak dużo gotowych rozwiązań, jak to tylko jest możliwe.

5.2. Architektura systemu

W tym rozdziale opiszę architekturę całego systemu, czyli wszystkich składników, które złożą się na to, że w wyniku ostatecznym użytkownik będzie mógł obejrzeć strumień wideo na swoim komputerze. W pierwszej części zaprezentuję strukturę sieci i określę jej podział na poszczególne elementy. Dla każdego elementu podam jego zadania i usługi jakie dostarcza. Następnie omówię system pod względem używanych protokołów sieciowych i podziału na warstwy. Na końcu przedstawię scenariusz działania całego systemu.

5.2.1. Ogólna wizja

Cała idea tego projektu opiera się na użyciu znanych mechanizmów rozpowszechniania danych z punktu-do-punktu (zaczerpniętych z BitTorrent) do transmisji fragmentów strumienia multimedialnego, powstałych po zakodowaniu metodą Multiple Description Coding [21]. Problem komplikuje się ponieważ, w przeciwieństwie do danych statystycznych należy tu jeszcze uwzględnić aspekt czasowy przesyłanych danych. Dla optymalizacji działania całego systemu węzły sieci zbierają informacje o opóźnieniu i przepustowości sieci, i wykorzystują je do kontroli transmisji danych.

5.2.2. Rozpowszechniana treść

Zanim przedstawię architekturę całego systemu opiszę dokładnie specyfikację treści, która będzie w tej sieci transmitowana. Celem całego systemu jest dostarczenie strumienia multimedialnego danego kanału do jak największej liczby odbiorców z opóźnieniem nie większym niż ustalone w specyfikacji kanału.

Podstawowe i jedyne założenia dotyczące nadawanej treści to, że:

- jest strumieniem czasu rzeczywistego,
- jest zakodowana schematem MDC.

Korzystając z tych założeń strumień danych można przedstawić za pomocą struktury pokazanej w tabeli 5.1.

Jak widać z tabeli fragmentacja strumienia jest dwuwymiarowa — ze względu na czas wyświetlenia oraz podział na podstrumienie. **Fragmenty** z danego strumienia będą dlatego

	Czas 1	**Czas 2**	**Czas ...**
Deskryptor 1	Fragment	Fragment	...
Deskryptor 2	Fragment	Fragment	...
Deskryptor ...	...	...	...

Tabela 5.1: Tablica deskryptorów MDC

oznaczone numerem deskryptora oraz znacznikiem czasowym w celu jednoznacznej identyfikacji.

Transmisja w systemie będzie więc polegała wyłącznie na wymianie takich fragmentów. Jest to jedyne założenie dotyczące mechanizmu kodowania strumienia. Wszelkie inne szczegóły na temat kodowania i dekodowania nie wpływają na działanie systemu.

5.2.3. Struktura sieci

Struktura węzłów systemu stanowi zdecentralizowaną i niestrukturyzowaną sieć punkt-do-punktu. Projekt zakłada, że system musi działać w całym Internecie i w dowolnej jego podsieci. Jedynym wymaganiem dla topologii sieci jest to, by każdy węzeł w tej sieci był w stanie zobaczyć dowolny inny element (brak obsługi protokołów omijania tłumaczy adresów sieciowych).

Decentralizacja sieci oznacza, iż sieć nie wymusza istnienia żadnego elementu (pojedynczego punktu awarii), z którym każdy inny węzeł musi być połączony. Brak narzuconej z góry struktury sieci z kolei pozwala każdemu węzłowi wybrać z kim zechce nawiązać połączenie. W architekturze sieci nie ma również podziału na elementy bardziej i mniej uprzywilejowane. Każdy węzeł w sieci, na podstawie zebranych danych sam decyduje o tym jaką politykę stosować wobec swoich sąsiadów.

5.2.4. Węzeł

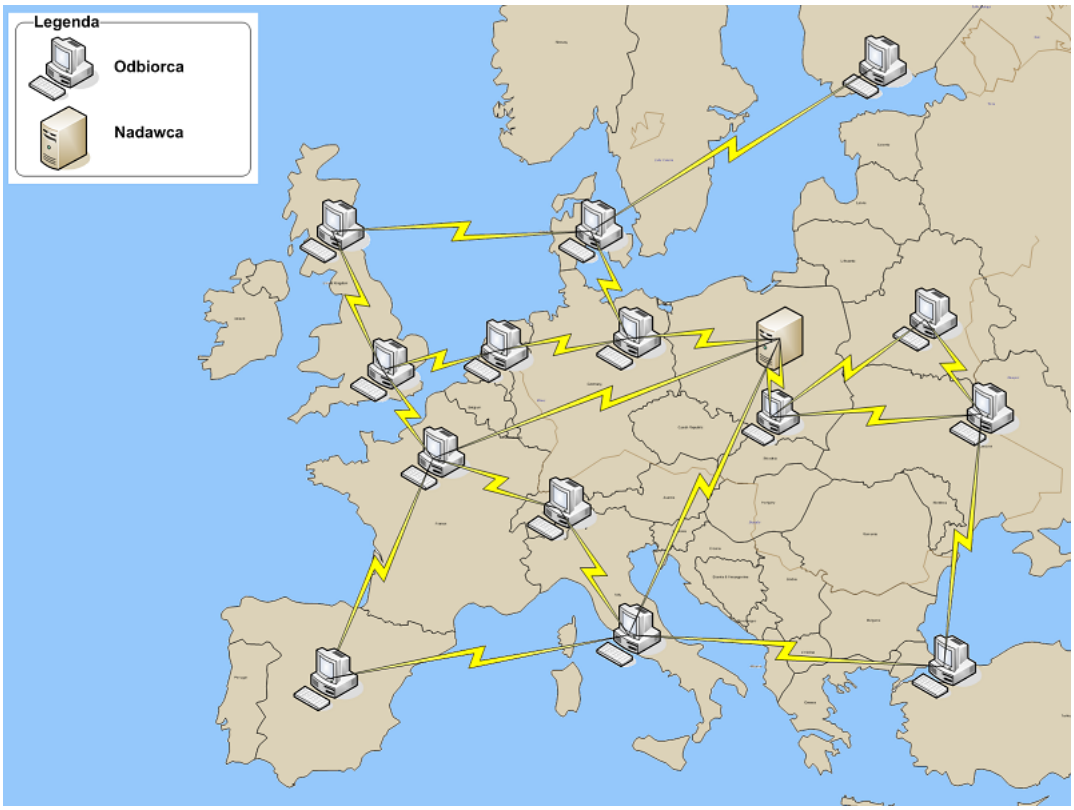
Każdy węzeł w sieci jest rozpoznawany za pomocą identyfikatora, który uzyskuje w momencie przyłączenia do sieci. Oprócz tego musi również posiadać stałe otwarte gniazda sieciowe dla protokołu sterującego oraz protokołu transmisji danych. Z punktu widzenia systemu każdy węzeł utrzymuje dwie struktury danych:

- Baza węzłów — jest to szczegółowa baza innych węzłów w systemie. Węzeł nie musi znać wszystkich innych elementów w sieci, ale powinien utrzymywać informacje na temat swoich bezpośrednich sąsiadów.
- Tablica deskryptorów — węzeł musi również posiadać reprezentację strumienia zapisaną zgodnie z tabelą 5.1.

Z ogólnego punktu widzenia możemy rozróżnić dwa rodzaje węzłów (jak widać na rysunku 5.1):

Odbiorca

Odbiorca w systemie to węzeł, który odbiera strumień multimedialny do własnego użytku oraz dodatkowo nadaje ten strumień do innych odbiorców. Do odbiorcy będzie podłączona aplikacja odtwarzacza, która pobrawszy strumień z bufora odbiorcy wyświetli go użytkownikowi. Usługi jakie dostarcza odbiorca w ramach systemu to:



Rysunek 5.1: Struktura sieciowa systemu

- wysłanie informacji o kanale — odbiorca posiada informacje o kanale, który jest nadawany w systemie i może te informacje przesłać.
- lista sąsiadów — odbiorca może być poproszony o listę swoich sąsiadów.
- powiadomienie o strumieniu — jest to usługa dostarczona z inicjatywy odbiorcy (wyłącznie do swoich autoryzowanych sąsiadów, co zostanie objaśnione w następnej części). Odbiorca po otrzymaniu całego deskryptora strumienia powiadamia swoich sąsiadów o swoim nowym stanie posiadania.
- transmisja strumienia — jeśli jest to możliwe, odbiorca w odpowiedzi na prośbę o fragment strumienia powinien go wysłać do zainteresowanego.

Odbiorca nie musi każdemu węzłowi w sieci dostarczać powyższych usług. Logika każdego odbiorcy odpowiada za to, czy prośba o daną usługę zostanie wysłuchana czy nie.

Nadawca

Nadawca to taki węzeł w systemie, który jest źródłem multimedialnego strumienia czasu rzeczywistego w systemie. Nadawca jest uważany za aktywnego jeśli posiada gotowy strumień czasu rzeczywistego i może go wysłać do zainteresowanych odbiorców. W przeciwnym wypadku Nadawca jest w stanie pasywnym. Nadawca posiadając aktywny strumień na bieżąco powiadamia swoich odbiorców o nowych fragmentach strumienia tak, aby ci mogli wysyłać żądania o nie. W tym rozwiązaniu możliwe jest działanie wielu nadawców na raz. W dodatku

dla lepszej efektywności systemu każdy z nadawców może nadawać inną część strumienia, jak np. system z czterema nadawcami:

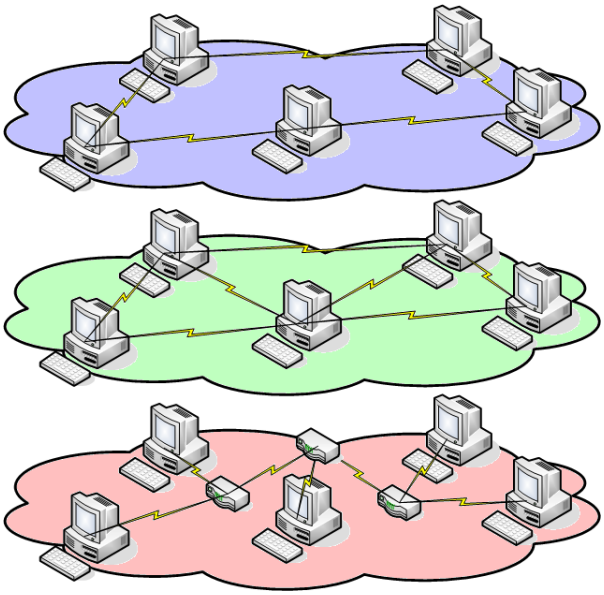
- dwóch nadawców strumienia wideo, każdy strumień stanowiący połowę całego strumienia wideo,
- jeden nadawca strumienia audio,
- jeden nadawca dodatkowych informacji tekstowych — teletekstu.

Jedyny wymóg dla nadawców to konieczność używania spójnych znaczników czasowych, tak by strumienie mogły zostać zsynchronizowane. Idealna sytuacja jest wtedy kiedy nadawcy są sterowani przez pewną centralną logikę (której zakres tej pracy nie obejmuje).

Nadawca dostarcza te same usługi co odbiorca i dla innych węzłów jest nierozróżnialny. Oznacza to, że struktura sieci jest przezroczysta.

5.2.5. Połączenia w sieci

Postępując zgodnie z koncepcją modularności, również połączenia w sieci węzłów dzielą się na warstwy. W tym przypadku powstały trzy warstwy sieciowe, spośród których każda następna to podzbiór poprzedniej. Implementacje poszczególnych warstw nie są jednak od siebie zależne i podmiana dowolnej nie wpłynie na działanie pozostałych.



Rysunek 5.2: Podział na warstwy sieciowe

Podział na warstwy w pewnym stopniu jest związany z podziałem na protokoły. Aczkolwiek nie jest to tożsame, co zostanie wyjaśnione poniżej.

- Warstwa fizyczna (czerwona na rysunku 5.2) — modeluje dokładną topologię sieci fizycznej, w której działa system.
- Warstwa struktury sieci (zielona na rysunku 5.2) — w tej warstwie uwzględnione są tylko węzły należące do systemu, a połączenie pomiędzy dwoma węzłami występuje jeśli

te węzły komunikują się ze sobą, wymieniając się komunikatami protokołu sterującego. Komunikaty w ramach tej warstwy dotyczą informacji na temat struktury sieci i węzłów.

- Warstwa transmisji danych (niebieska na rysunku 5.2) — tutaj połączenia się pojawiają tylko pomiędzy autoryzowanymi sąsiadami. Komunikacja w tej warstwie to albo prośby i zapytania o strumień, albo pakiety z samym strumieniem.

5.2.6. Działanie systemu

System podobnie jak nadawca może być pasywny bądź aktywny. System określamy jako aktywny, jeśli którykolwiek z jego nadawców jest aktywny. W systemie pasywnym jedynie komunikaty warstwy struktury sieci mają znaczenie — czyli węzły mogą nawiązywać połączenia, zrywać je, szukać lepszych sąsiadów, ale nie mogą uzyskać żadnych informacji na temat strumienia danych. System w stanie aktywnym może już korzystać z komunikatów warstwy transmisji danych.

Dla zaprezentowania działania systemu przedstawię podstawowy scenariusz życia systemu.

1. Pojawia się pierwszy nadawca, który ma zaplanowany do nadania kanał multimedialny, lecz nie jest on jeszcze aktywny. Ten moment zapoczątkowuje życie systemu w stanie pasywnym.

2. Odbiorcy podłączają się do systemu komunikując się z nadawcą.

3. Nadawca autoryzuje odbiorców lub przekierowuje ich do innych węzłów w systemie. W taki sposób sieć odbiorców się powiększa. Odbiorcy mogą już w tym momencie próbować optymalizować odległość od nadawców, poprzez mierzenie opóźnienia pakietów.

4. Informacje o kanale są w rozpowszechnione do odbiorców.

5. Nadawcy się uaktywniają, poprzez rozgłoszenie do wszystkich bezpośrednich sąsiadów informacje o dostępności strumienia.

6. Odbiorcy, którzy dostali powiadomienie o dostępności strumienia wysyłają prośbę do nadawcy.

7. Nadawca przesyła zamówione fragmenty strumienia do odbiorców, a ci otrzymawszy cały fragment wysyłają powiadomienie o dostępności do swoich sąsiadów, którzy danego fragmentu nie posiadają. Ten mechanizm powoduje, że strumień rozpropaguje się dalej, aż każdy odbiorca w sieci otrzyma strumień.

5.3. Protokół sterujący

W ramach tego projektu stworzyłem protokół sterujący w sieci punkt-do-punktu. Zadaniem protokołu sterującego jest dostarczenie węzłom zbioru komunikatów do porozumiewania się między sobą. Model tego protokołu w dużej mierze został oparty na mechanizmach protokołu BitTorrent. Sama specyfikacja pakietów została jednak stworzona od nowa. Ten protokół, w przeciwieństwie do protokołu BitTorrent, realizuje również ideę jak najlżejszej komunikacji i został zbudowany na bazie bezpołączeniowego protokołu UDP, a nie TCP. Cała komunikacja opiera się na modelu bezstanowych zapytań i odpowiedzi.

Rysunek 5.3: Schemat komunikacji pomiędzy węzłami

5.3.1. Identyfikacja w sieci

Każdy węzeł komunikujący się przy użyciu protokołu sterującego musi udostępniać stałe otwarte gniazdo (adres maszyny oraz port) sieciowe do komunikacji UDP. Specyfikacja protokołu wymaga, by jedna aplikacja kliencka posiadała tylko jedno gniazdo protokołu sterującego i należała wyłącznie do jednego systemu. Aplikacja nie może więc ani używać wielu gniazd ani też używać jednego gniazda do komunikowania się z węzłami z różnych systemów.

Gniazdo identyfikujące dany węzeł jest adresem docelowym dla komunikatów nadanych do danego węzła oraz adresem wychodzącym komunikatów, które zostały wysłane przez dany węzeł.

5.3.2. Komunikaty

W tej sekcji opiszę znaczenie wszystkich komunikatów protokołu sterującego oraz opiszę strukturę pakietu każdego z komunikatów.

Każdy komunikat wyspecyfikowany w tym protokole jest przesyłany oddzielnie w jednym pakiecie UDP. Wszystkie pakiety posiadają stały pięciobajtowy nagłówek, gdzie pierwsze cztery bajty to identyfikator nadawcy, a piąty to liczba naturalna oznaczająca typ komunikatu. Większość pakietów z komunikatem ma stałą wielkość.

User Datagram Protocol	
Identyfikator	
Typ	Reszta

Tabela 5.2: Podstawowa struktura pakietu komunikatu sterującego

Żądanie autoryzacji

Ten komunikat jest przesyłany w momencie, kiedy nadający komunikat chce nawiązać połączenie z odbierającym w celu rozpoczęcia wymiany danych. Odbierający komunikat reaguje na komunikat na dwa sposoby:

- akceptując żądanie — wysyła komunikat ze **zgodą autoryzacji**.
- odrzucając połączenie — odpowiada komunikatem **odmowy autoryzacji**.

Pakiet (datagram UDP) z komunikatem żądania połączenia ma długość 9 bajtów, z których 5 pierwszych bajtów to identyfikator i typ pakietu. Opiszę poniżej wszystkie pola w tym pakiecie, pomijając te wspólne dla wszystkich komunikatów.

- **typ** (5. bajt) — pakiety z żądaniem połączenia mają wartość typu równą 1.
- **port RTP** (6.-7. bajt) — 16-bitowa liczba naturalna oznaczająca numer portu protokołu RTP chcącego nawiązać połączenie.
- **port RTCP** (8.-9. bajt) — 16-bitowa liczba naturalna oznaczająca numer portu protokołu RTCP chcącego nawiązać połączenie.

Zgoda autoryzacji

Ten komunikat występuje jako odpowiedź na **żądanie autoryzacji**. Komunikat oznacza, że nadawca komunikatu zgodził się autoryzować odbiorcę.

Struktura pakietu komunikatu zgody autoryzacji wygląda podobnie jak pakiet z żądaniem autoryzacji:

- **typ** (5. bajt) — pakiety ze zgodą autoryzacji mają wartość typu ustawioną na 5.
- **port RTP** (6.-7. bajt) — 16-bitowa liczba naturalna oznaczająca numer portu protokołu RTP proszonego o autoryzację. Ten port powinien zostać zapisany w informacjach danego węzła w bazie węzłów proszącego.
- **port RTCP** (8.-9. bajt) — 16-bitowa liczba naturalna oznaczająca numer portu protokołu RTCP proszonego o autoryzację. Ten port powinien zostać zapisany w informacjach na temat danego węzła w bazie węzłów proszącego.

Odmowa autoryzacji

Komunikat z odmową połączenia jest wysyłany, wtedy gdy węzeł proszony o połączenie nie zgodził się autoryzować odbiorcy. Węzły nieautoryzowane nie mogą wymieniać się fragmentami strumienia.

Pakiet odmowy połączenia nie ma żadnych dodatkowych pól poza standardowymi.

- **typ** (5. bajt) — pakiety z odmową połączenia mają typ z wartością 6.

Żądanie listy sąsiadów

Ten komunikat to prośba o zbiór węzłów zapisanych w bazie węzłów adresata komunikatu.

Specyfikację komunikatu z żądaniem listy węzłów stworzyłem wzorując się na protokole BitTorrent. Żądanie może w tym przypadku dotyczyć całej bazy węzłów jaka znajduje się w posiadaniu adresata komunikatu lub tylko węzłów, które posiadają wyszczególniony w żądaniu fragment danych. Węzeł, otrzymując żądanie listy sąsiadów, wysyła w odpowiedzi pewną listę węzłów w postaci komunikatu **lista sąsiadów**.

Cały pakiet ma długość 11 bajtów.

- **typ** (5. bajt) — żądanie listy sąsiadów ma typ z numerem 2.

- **numer żądania** (6.-7. bajt) — identyfikacja żądania u wysyłającego komunikat. Za pomocą tego numeru wysyłający będzie mógł powiązać odpowiedź z listą sąsiadów z tym żądaniem.
- **numer deskryptora** (8.-9. bajt) — jeśli ta wartość jest różna od zera, to oznacza, że wysyłający komunikat jest zainteresowany węzłami, które posiadają deskryptor o danym numerze i danym znaczniku czasowym (poniżej). W przeciwnym przypadku oznacza to prośbę o listę dowolnych węzłów.
- **znacznik czasowy** (10.-13. bajt) — to pole ma znaczenie jeśli wartość deskryptora jest różna od zera. Prośba ta dotyczy węzłów, o których adresat komunikatu wie, że posiadają dany deskryptor w tablicy deskryptorów wyznaczonego przez numer deskryptora i znacznik czasowy.

Lista sąsiadów

Lista sąsiadów to odpowiedź na komunikat z żądaniem listy sąsiadów. Jest to jeden z dwóch komunikatów w tym protokole, który nie posiada stałej długości i w praktyce jest największym pakietem przesyłanym w ramach tego protokołu. Rozmiar pakietu z tym komunikatem jest zależny od wielkości listy węzłów oraz od struktury reprezentującej węzeł w aplikacji. W celu zmniejszenia obciążenia łącza przez ten komunikat postanowiłem użyć gotowego rozwiązania, które pomogłoby mi zserializować tę listę węzłów przed wysłaniem i zdeserializować do listy po otrzymaniu pakietu.

Spośród wszystkich rozwiązań zdecydowałem się użyć biblioteki stworzonej i wykorzystywanej przez firmę Google — Protocol Buffers. Protocol Buffers w przeciwieństwie do XML i JSON posiada możliwość przekodowania wiadomości do postaci binarnej. Według twórców rozwiązania dane zserializowane za pomocą Protocol Buffers są kilka razy mniejsze, a przetwarzanie ich jest o kilkadziesiąt razy szybsze od znanego XML-a.
Struktura pakietu:

- **typ** (5. bajt) — odpowiedź z listą sąsiadów ma to pole z wartością 7.
- **numer żądania** (6.-7. bajt) — identyfikator żądania, którego odpowiedzią jest ten komunikat.
- **lista węzłów** (8.-ostatni bajt) — strumień binarny spakowane przy użyciu Protocol Buffers.

Dane binarne to lista węzłów, w której każdy węzeł ma strukturę:

```
message PeerMessage{
        required int64 id;
        required string address;
        required int32 metaPort;
        required int32 rtpPort;
        required int32 rtcpPort;
}
```

Prośba o informacje o kanale

Węzeł wysyła ten komunikat gdy chce dostać informacje o kanale nadawanym w danym systemie. Nie mając informacji o kanale, węzeł nie może zacząć procesu wymiany danych

multimedialnych w ramach systemu. Na żądanie informacji o kanale adresat żądania może odpowiedzieć komunikatem:

- brak kanału — sam proszony węzeł nie ma żadnych informacji o kanale i dlatego nie może nic o nim powiedzieć.
- informacje o kanale — wysyła te informacje jakie sam posiada.

Pakiet z tym komunikatem nie ma żadnych dodatkowych pól:

- **typ** (5. bajt) — pole typu tego komunikatu ma wartość 10.

Brak kanału

Ten komunikat świadczy o tym, iż system jest jeszcze nieaktywny i nie zawiera żadnych informacji na temat kanału.
Struktura pakietu:

- **typ** (5. bajt) — pole typu tego komunikatu ma wartość 11.

Informacje o kanale

Ten komunikat powinien zawierać informacje identyfikujące kanał, opis struktury strumienia multimedialnego (liczba deskryptorów) oraz czas kiedy odtwarzanie strumienia ma zostać rozpoczęte. Podobnie jak i komunikat z listą węzłów ten komunikat nie ma stałego rozmiaru i wykorzystuje kompresję Protocol Buffers.
Struktura pakietu:

- **typ** (5. bajt) — pole typu z wartością 12.
- **aktualny znacznik czasowy** (6.-9. bajt) — ta wartość to znacznik czasu fragmentu, który jest aktualnie odtwarzany.
- **opóźnienie odtwarzania** (6.-9. bajt) — liczba milisekund, mówiąca z jakim opóźnieniem strumień ma być odtwarzany. Ma to szczególny sens, jeśli strumień nie zaczął być jeszcze nadawany.
- **opis kanału** (6.-ostatni bajt) — dane binarne spakowane przy użyciu Protocol Buffers.

Struktura opisu kanału w schemacie Protocol Buffers:

```
message ChannelMessage{
        required string name;
        required string description;
        required float framerate;
        required int32 descriptorsCount;
}
```

Ogłoszenie dostępności deskryptora

Ogłoszenie dostępności deskryptora również powstało na podobieństwo protokołu BitTorrent. Ten komunikat informuje węzeł otrzymujący wiadomość, że nadający znalazł się już w posiadaniu danego fragmentu strumienia.

Struktura pakietu (11 bajtów):

- **typ** (5. bajt) — ogłoszenia dostępności deskryptora mają zawsze typ z wartością 9.
- **numer deskryptora** (6.-7. bajt) — to pole z numerem deskryptora fragmentu.
- **znacznik czasowy** (8.-11. bajt) — ta wartość to znacznik czasu identyfikujący kolumnę danego fragmentu w tablicy deskryptorów.

Żądanie fragmentu

Żądanie danych powinno być wysłane wyłącznie w przypadku kiedy wysyłający węzeł nie posiada danego fragmentu, a adresat komunikatu tak. Wynikiem tego żądania może być odpowiedź odmowna w postaci komunikatu **odmowy danych** albo pakiet z danymi.

Struktura pakietu (11 bajtów):

- **typ** (5. bajt) — ogłoszenia dostępności deskryptora mają zawsze typ z wartością 3.
- **numer deskryptora** (6.-7. bajt) — to pole z numerem deskryptora żądanego fragmentu.
- **znacznik czasowy** (8.-11. bajt) — ta wartość to znacznik czasu identyfikujący kolumnę żądanego fragmentu w tablicy deskryptorów.

Odmowa wysłania fragmentu

Poprzez ten komunikat nadawca powiadamia odbiorcę, że nie otrzyma on zażądanego fragmentu.
Struktura pakietu:

- **typ** (5. bajt) — pole typu tego komunikatu ma wartość 8.

Odłączenie

Ten komunikat jest wysyłany kiedy nadawca chce odłączyć się od kanału. Wysyła wówczas ten komunikat do wszystkich węzłów, jakie zna.

Struktura pakietu:

- **typ** (5. bajt) — komunikat o odłączeniu się z systemu ma typ o wartości 13.

5.3.3. Mechanizmy

Oprócz reagowania na każdy komunikat w odpowiedni sposób, protokół posiada również dodatkowe mechanizmy sprawdzania poprawności wszystkich pakietów:

- identyfikacja węzła — każdy komunikat który przychodzi od węzła zawiera identyfikator i adres gniazda protokołu sterującego nadawcy. Jeśli węzeł z danym identyfikatorem już występuje w bazie to sprawdzana jest zgodność gniazda protokołu sterującego.

- poprawność transakcji — niektóre komunikaty wchodzą w skład dłuższej konwersacji pomiędzy dwoma węzłami. Często pewne komunikaty mogą zostać wysłane tylko w szczególnych okolicznościach i wysłanie ich inaczej jest złamaniem protokołu.

Każde wykroczenie przeciwko powyższym mechanizmom jest odnotowane. W przypadku kiedy pewien węzeł będzie wyróżniał się dużą liczbą zachowań niezgodnych ze specyfikacją protokołu, wówczas może zostać zapisany na **czarną listę** przez pozostałe węzły.

5.4. Protokół transmisji danych

Drugi protokół wykorzystywany w tym projekcie to protokół transmisji danych. Jak to zostało napisane w rozdziale z wymaganiami, ważniejsze dla tego protokołu jest to, by miał jak najmniejsze opóźnienie, a nie by dawał gwarancję dostarczenia.

Przy takich założeniach postanowiłem wykorzystać właśnie protokół Real-time Transport Protocol. Według założeń protokół RTP ma służyć wyłącznie przenoszeniu danych, nie zaglądając w głąb przenoszonego strumienia, ani jego kodowania. Zaletą protokołu RTP jest też jego popularność — protokół ten jest przez większość ruterów rozpoznawany i przepuszczany.

Do użytku w tym projekcie dokonałem reinterpretacji pewnych pól i cech protokołu RTP. Są to jednak zmiany dozwolone w specyfikacji protokołu.

Dokonane modyfikacje zostaną opisane w kolejnych punktach.

5.4.1. Sesja

Komunikacja w protokole RTP odbywa się w ramach sesji. Sesję można określić jako grupowy proces rozpowszechniania danych. W ramach sesji każda porcja danych jest rozsyłana do wszystkich uczestników sesji. Każdy uczestnik sesji jest zaś identyfikowany przez jego otwarte gniazda wymiany danych RTP i RTCP. Według specyfikacji protokołu RTP pakiety mogą być transmitowane unikastowo bądź multikastowo. W tym rozwiązaniu uwzględnię jedynie transmisję unikastową.

Oryginalnie specyfikacja protokołu RTP zakłada, że strumień wysłany w ramach sesji powinien zostać rozpropagowany do wszystkich jego członków. W przypadku transmisji unikastowej jest to realizowane tak, że wysyłający nadaje pakiety z danymi do wszystkich innych uczestników w sesji. Do transmitowania różnych strumieni protokół RTP proponuje użycie wielu sesji.

W projekcie tego systemu implementacja takiego mechanizmu sesji jest niewydajna i mało elastyczna. Pomimo, iż wszystkie węzły w sieci są zainteresowane jednym strumieniem multimedialnym, to jednak nie tworzą one modelu każdy do każdego (kliki). Ścisłe zastosowanie się do specyfikacji protokołu wymagałoby stworzenia pojedynczej sesji dla każdego połączenia pomiędzy dwoma węzłami, co jest według mnie niepotrzebnym wysiłkiem, biorąc pod uwagę nietrwały charakter połączeń pomiędzy węzłami. Z tego też powodu wprowadziłem do implementacji protokołu RTP małą modyfikację, która umożliwia węzłom w sesji wysłanie pakietu do pojedynczego uczestnika sesji. To usprawnienie powoduje, że każdy węzeł potrzebuje tylko jednego gniazda RTP do komunikacji z innymi węzłami. Połączenia pomiędzy uczestnikami sesji ma wówczas topologię grafu spójnego, a niekoniecznie kliki.

5.4.2. Typ ładunku

Pakiety z danymi w protokole RTP mają bardzo rozbudowany nagłówek (jak to zostało opisane w rozdziale o RTP). W tej części będzie nas interesowało pole **typu ładunku** (ang.

payload type). Jest to ciąg siedmiu bitów począwszy od dziesiątego bitu nagłówka pakietu. Jego zadaniem jest określenie typu danych przenoszonych przez pakiet. Wraz z pierwszą wersją protokołu RTP powstał również dokument [24], określający format tego pola. Zgodnie z tą specyfikacją połowa zakresu tego parametru została przypisana popularnym kodowaniom znanych z lat dziewięćdziesiątych, a reszta pozostała w zakresie dyspozycji implementującego. Jednakże wraz z wzrostem popularności protokołu RTP oraz powstaniem nowych schematów kodowania multimediów ten atrybut stracił na aktualności. W tym momencie ta siedmiobitowa wartość okazuje się być niewystarczająca do dokładnego określenia typu kodowania danych i zawsze potrzebne jest dodatkowe uzgadnianie przy użyciu zewnętrznych mechanizmów (poza RTP).

W tym projekcie dokonam zupełnej reinterpretacji tego pola. Wartość tego pola w pakietach przesyłanych w tym systemie nie będzie zgodna z dokumentem [24] tylko będzie zrozumiała w ramach tego systemu. Pole w protokole transmisji danych będzie oznaczać numer deskryptora przesyłanego fragmentu strumienia. Dzięki takiej zmianie semantycznej pakiet RTP zawiera już niezbędne dla tego systemu pola: numer deskryptora i znacznik czasowy.

5.5. Model danych aplikacji odbiorcy

Przedstawię w tej części model danych, stanowiący fundament logiki całej aplikacji. W zakres modelu wchodzą klasy, odzwierciedlające podstawowe pojęcia w projekcie oraz struktury danych zarządzające obiektami tych klas. Zdefiniuję i opiszę następujące struktury:

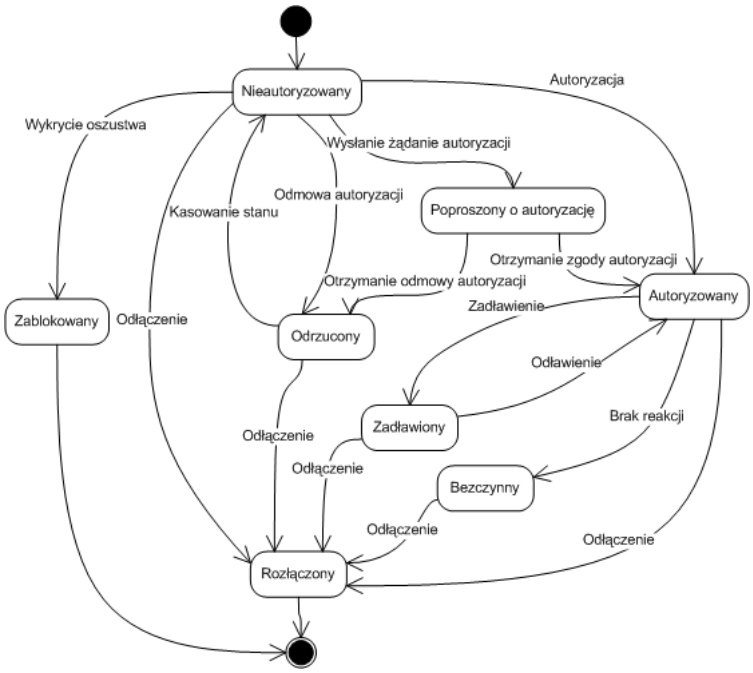
- Węzeł — klasa opisująca sąsiada danego klienta.
- Kanał — klasa, która zawiera wszystkie dane na temat danego kanału.
- Fragment — klasa opisująca fragment strumienia.
- Segment — klasa odzwierciedlająca strukturę ramki w systemie
- Komunikat — klasy odzwierciedlające rodzaje komunikatów protokołu sterującego.
- Kolejki komunikatów — kolejki, w którym znajdują się wychodzące i wchodzące komunikaty.

Wszystkie klasy zdefiniowane w tej sekcji mają interfejs „fasolek Javy” (JavaBean).

5.5.1. Węzeł

Obiekt klasy **Węzeł** to instancja reprezentująca sąsiadów danego klienta w aplikacji. Ściślej ta klasa definiuje połączenie pomiędzy daną aplikacją odbiorcy a aplikacją wskazanego węzła. Struktura klasy **Węzeł** wygląda następująco:

- identyfikator — 32 bitowa liczba naturalna, będąca unikatową reprezentacją danego węzła w całym systemie.
- adres — adres maszyny, na którym działa aplikacja węzła.
- port sterujący — wartość określająca port gniazda protokołu sterującego danego węzła.
- port RTP — wartość określająca port gniazda protokołu RTP danego węzła.
- port RTCP — wartość określająca port gniazda protokołu RTCP danego węzła.



Rysunek 5.4: Diagram stanów węzła

- status — obiekt mówiący o tym jaki jest stan połączenia pomiędzy daną aplikacją a określonym węzłem. Możliwe statusy to (rysunek 5.4):
  - nieautoryzowany — węzeł występuje w systemie, ale nie został jeszcze poproszony o autoryzację ani też nie wysłał żądania autoryzacji do aplikacji.
  - zablokowany — węzeł znajduje się w tym stanie, jeśli występują podejrzenia, że nie przedstawia się on prawdziwymi informacjami podczas komunikacji.
  - poproszony o autoryzację — sąsiad został poproszony o autoryzację, ale nie ma jeszcze odpowiedzi.
  - odrzucony — oznacza iż żądanie autoryzacji zostało odrzucone.
  - autoryzowany — nastąpiła obustronna autoryzacja. Autoryzacja jest warunkiem **koniecznym** dla węzłów do rozpoczęcia wymiany strumieniem multimedialnym.
  - zadławiony — aplikacja zablokowała żądania o dane od danego węzła i nie reaguje na nie.
  - bezczynny — węzeł nie odpowiada na żadne żądania, nawet zapytania o żywotność.
  - rozłączony — węzeł jest uważany za niedostępny.

5.5.2. Kanał

Kanał to klasa, która zawiera wszystkie niezbędne informacje na temat kanału, który w danym momencie jest odtwarzany przez aplikację.

Struktura klasy:

- nazwa — nazwa a jednocześnie identyfikator kanału.
- opis — opis zawartości merytorycznej kanału.

- częstotliwość — częstotliwość wyświetlania obrazu, w tym przypadku oznacza liczbę fragmentów, mieszczących się w jednej sekundzie.
- liczba deskryptorów — na ile podstrumieni dzieli się dany kanał. To ta wartość ustala szerokość okna bufora (liczbę deskryptorów).

5.5.3. Fragment

Ta klasa ma reprezentować jednostkę **fragmentu** zgodnie z koncepcją Multiple Description Coding.

Struktura klasy **Fragment**:

- numer deskrypcji — numer podstrumienia, do którego należy dany fragment.
- znacznik czasowy — 32-bitowa liczba naturalna, określająca pozycję danego fragmentu w strumieniu według porządku chronologicznego.
- stan — pole określające stan dostępności danego fragmentu.
  - niedostępny — ani dany odbiorca, ani żaden z jego sąsiadów, nie posiada danego fragmentu.
  - dostępny — jakiś sąsiad danego klienta posiada dany fragment. Aplikacja nie posiada jednak jeszcze tego fragmentu.
  - zażądany — żądanie przekazania fragmentu zostało wysłane do któregoś z sąsiadów posiadających dany fragment, ale pakiet z danymi jeszcze nie dotarł.
  - zapisany — cały fragment jest dostępny w aplikacji.
  - przeterminowany — czas odtwarzania tego fragmentu już minął.
- zbiór posiadaczy — zbiór węzłów, które mają dany fragment. Bazując na tej liście aplikacja decyduje, do którego sąsiada wysłać zapytanie o fragment.
- dane — dane binarne fragmentu, o ile te istnieją.

5.5.4. Sekwencja

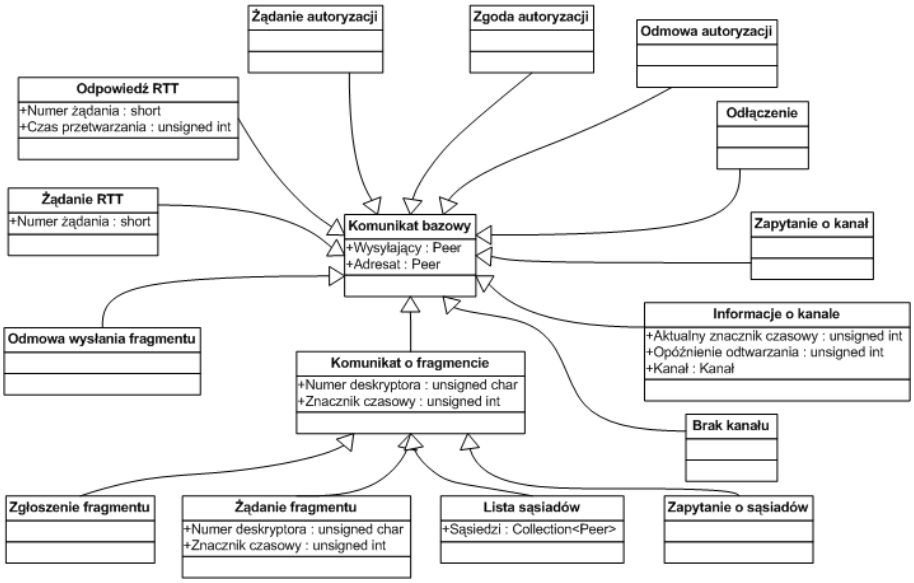
Sekwencja to zbiór wszystkich fragmentów z tym samym znacznikiem czasowym — czyli wszystkie które powinny zostać zdekodowane i odtworzone w tym samym momencie. Sekwencja może zawierać maksymalnie tyle fragmentów co strumień deskryptorów. W terminologii przetwarzania obrazu i dźwięku sekwencja odpowiada jednej ramce. Przykład struktury sekwencji można zobaczyć w tabeli 5.3.

Sekwencja
fragment 1
brak
brak
fragment 4

Tabela 5.3: Przykład sekwencji w cztero-deskryptowym strumieniu

5.5.5. Komunikat

Hierarchia klas komunikatów implementuje specyfikację komunikatów protokołu sterującego. Każdy komunikat opisany w protokole to oddzielna klasa w implementacji. Na rysunku 5.5 znajduje się hierarchia klas i opisy wszystkich atrybutów jakie występują.



Rysunek 5.5: Hierarchia klas komunikatów sterujących

Struktura:

- wysyłający — węzeł, który jest nadawcą komunikatu,
- adresat — węzeł, do którego zostanie/został wysłany komunikat,
- numer deskryptora — 7 bitowa wartość naturalna, opisująca numer deskryptora fragmentu wskazanego w komunikacie.
- znacznik czasowy — 32 bitowa liczba naturalna, oznaczająca sekwencję wskazanego w komunikacie fragmentu w czasie.
- kanał — informacje dotyczące kanału, który aktualnie jest nadawany w systemie. Węzeł dołączając do aktywnego kanału powinien otrzymać te informacje od któregoś ze swoich sąsiadów.
- lista sąsiadów — podzbiór węzłów z bazy węzłów nadawcy komunikatu. Ten zbiór nie jest jednak kolekcją obiektów klasy **Węzła**, ale uproszczonych węzłów wygenerowanych ze schematu protokołu Protocol Buffers. Są to obiekty klasy **Węzeł** uproszczone o status.

5.5.6. Kolejki komunikatów

Niezbędnymi strukturami w aplikacji odbiorcy są kolejki komunikatów. Blokujące kolejki FIFO przechowujące komunikaty umożliwiły nie tylko podzielenie poszczególnych elementów aplikacji na moduły, ale również pozwoliły na zrównoleglenie czynności, tym samym przyspieszając aplikację.

W implementacji wyróżniam cztery kolejki:

- przychodzące komunikaty — kolejka datagramów UDP, którą muszą zostać przetworzone, a potem obsłużone. Ponieważ gniazdo odbierające komunikaty jest tylko jedno, dlatego przetworzenie i obsłużenie niezależnie przychodzących pakietów w jednym wątku byłoby niewydajne.

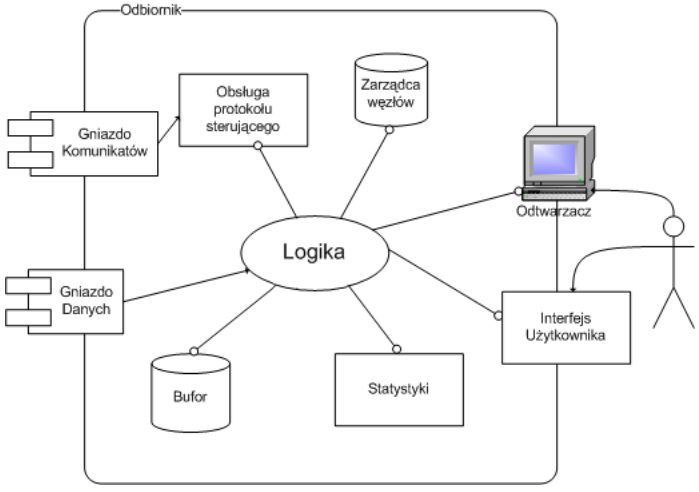
- wychodzące komunikaty — kolejka datagramów UDP, które muszą zostać wysłane. Ponieważ wszystkie komunikaty są nadawane z jednego gniazda UDP, więc najlepszym rozwiązaniem jest by jeden wątek był odpowiedzialny za nadawanie tych pakietów. Z tego też powodu do tej kolejki wiele wątków będzie mogło wkładać pakiety, ale będzie wyłącznie jeden pobierający.

- przychodzące dane — gniazdo danych protokołu RTP to miejsce, do którego będą dochodziły setki kilobitów, a nawet megabity danych na sekundę. Dlatego też rozproszenie obsługi tych danych jest jak najbardziej potrzebne.

- wychodzące dane — podobnie jak wychodzącymi pakietami UDP, obsługą wychodzących pakietów RTP powinien się zajmować również jeden wątek.

5.6. Podział aplikacji na moduły

Podział projektu na moduły polega na rozdzieleniu odpowiedzialności i usług do oddzielnych części. Moduły w implementacji tej aplikacji są albo od siebie całkowicie niezależne albo komunikują się ze sobą jedynie poprzez ustalony interfejs, abstrahując od implementacji poszczególnych modułów. W tym rozdziale wymienię wszystkie moduły jakie występują w implementacji aplikacji odbiornika oraz opiszę ich interfejs i logikę.



Rysunek 5.6: Podział aplikacji ze względu na moduły

W pierwszej części wymienię moduły, które są najbardziej oczywiste i których wyodrębnienie wynika z wykorzystanych technologii. W następnej kolejności będą opisane moduły, które ukazują moją wizję (rysunek 5.6) podziału projektu.

5.6.1. Interfejs użytkownika

W swojej aplikacji wykorzystuję Javową bibliotekę graficzną Swing, do tworzenia interfejsu użytkownika. Biblioteka Swing należy do standardu bibliotek Javowych dostarczonych przez firmę Sun dla aplikacji biurowych i pozwala na rysowanie komponentów graficznych oraz tworzeniu interfejsu komunikującego się z użytkownikiem niezależnie od docelowej platformy, na której będzie uruchamiana aplikacja. Architektura biblioteki Swing opiera się na strukturze model-widok-kontroler (w skrócie MVC od angielskiego Model-View-Controller), przy czym widok i kontroler są ze sobą mocniej powiązane niż zakłada konwencja MVC.

W projekcie aplikacji odbiornika do modułu interfejsu użytkownika będą należały wszystkie klasy widoku i kontrolerów odpowiedzialne za interakcje z użytkownikiem. Model danych dla tych struktur będzie oddelegowany do innych modułów.

Projektowanie interfejsu graficznego w tym projekcie jest kwestią drugoplanową, dlatego też nie będę opisywał jego przebiegu w tej pracy.

5.6.2. Odtwarzacz

Nie ma wątpliwości, że część aplikacji odpowiedzialna za odtwarzanie dostępnego strumienia powinna zostać wydzielona jako oddzielny moduł. W tym projekcie dostarczę przykładowy moduł odtwarzający, który będzie implementował wszystkie usługi interfejsu, aczkolwiek nie będzie tego robił w najwydajniejszy sposób. Implementacja odtwarzacza zakłada jedynie wyświetlenie obrazu wizualnego bez możliwości odtwarzania dźwięku.

Zadaniem odtwarzacza jest wyświetlenie strumienia multimedialnego zapisanego w buforze aplikacji. Najważniejsze założenie odtwarzacza to wyświetlenie obrazu według jego kryteriów czasowych — każda klatka jest pokazana dokładnie w wyznaczonym przez jej znacznik czasowy momencie. Interfejs odtwarzacza dostarcza następujące metody:

- odtwarzanie — metoda, która powoduje wyświetlenie obrazu wideo w oddzielnym oknie odtwarzacza.
- zatrzymanie — zatrzymuje wyświetlany obraz. Po wywołaniu tej metody odtwarzacz powinien znaleźć się w stanie zatrzymanym i nie powinien zajmować czas procesora. Należy pamiętać jednak, że operacja wprowadzania odtwarzacza w stan zatrzymany nie wpływa w żaden sposób na mechanizm strumieniowania i komunikacji.
- dodaj klatkę — za pomocą tej metody bufor odtwarzacza jest wypełniany sekwencjami danych pochodzącymi z okienka bufora.

5.6.3. Protokół sterujący

Ten moduł zajmuje się obsługą całego protokołu sterującego. W skład modułu protokołu sterującego wchodzi:

- gniazdo protokołu sterującego,
- parser komunikatów.

Gniazdo

Do zadań gniazda protokołu sterującego należy odbieranie pakietów przychodzących z sieci i przekazanie ich do dalszej obsługi oraz wysyłanie pakietów z danego gniazda. Samo gniazdo

nie przetwarza przychodzące pakiety, ponieważ jest to operacja zbyt czasochłonna. Każde gniazdo protokołu sterującego zawiera w sobie **gniazdo UDP**, na którym operuje.

Działające gniazdo protokołu sterującego uruchamia dwa wątki:

- odbiorca — wątek, który czeka na gnieździe UDP na przychodzące pakiety i wkłada je w tej samej postaci do **kolejki przychodzących datagramów**.
- nadawca — czeka na **kolejce datagramów wychodzących**. Każdy datagram wyjęty z tej kolejki jest wysyłany przez ten wątek z gniazda UDP.

Do skonstruowania obiektu tej klasy potrzebny jest numer portu docelowego gniazda oraz dwie kolejki pakietów — kolejka datagramów przychodzących i wychodzących.

Parser

Drugi element tego modułu zajmuje się przetwarzaniem datagramów z sieci na komunikaty i odwrotnie.

Obsługą komunikatów zajmują się wątki, które oczekują na kolejce datagramów przychodzących, tłumaczą je na komunikaty i obsługują. Oprócz obsługi przychodzących komunikatów moduł protokołu sterującego udostępnia również interfejs do wysyłania wychodzących komunikatów. Jest to zbiór metod, które w wyniku wywołania przekazują gotowe komunikaty, które następnie są tłumaczone na pakiety i wkładane do kolejki pakietów do wysłania.

Protokół transmisji danych

Moduł protokołu transmisji funkcjonuje na zbliżonych zasadach do modułu protokołu sterującego.

Przepływ danych w tym module również został podzielony na dwie części:

- gniazdo danych,
- kontroler danych.

Gniazdo

Podobnie jak w protokole sterującym to gniazdo wyłącznie zajmuje się odbieraniem pakietów z sieci i przekazaniem ich dalej do obsługi. W moim projekcie wykorzystałem bibliotekę jlibrtp do obsługi protokołu RTP. Biblioteka ta dostarcza zupełną implementację protokołu RTP i RTCP. Częścią tej klasy są 2 wątki:

- odbiorca — klasa implementująca interfejs selektora gniazda RTP. Pakiety otrzymane za pomocą tej klasy przychodzą w postaci pakietów RTP. Podobnie jak w protokole sterującym te pakiety są wkładane do dedykowanej **kolejki pakietów RTP**.
- nadawca — czeka na **kolejce fragmentów do wysłania**. Wyjmuje z niej fragment strumienia i tłumaczy go na pakiety RTP, które potem wysyła.

Utworzone gniazdo protokołu transmisji danych otwiera na stałe (czas działania aplikacji) dwa porty, odpowiednio dla protokołu RTP i RTCP.

Kontroler danych

Kontroler danych to wątek, który czeka na kolejce przychodzących pakietów RTP, wyjmuje je i skleja w fragment. Gotowy fragment jest zapisywany do bufora.

5.6.4. Zarządca węzłów

Ten moduł jest w odpowiedzialny za przechowanie całej wiedzy aplikacji na temat pozostałych węzłów w sieci. Oprócz samych węzłów w tym module znajdują się również informacje o strukturze sieci, jego połączeniach. To również w tym module jest zawarta logika określająca politykę i zachowania danego odbiornika względem innych węzłów. Działanie tego modułu jest wspierane przez informacje zebrane w module **Statystyki**.

Struktura

W ramach tego modułu występują takie struktury:

- zbiór sąsiadów — zbiór wszystkich sąsiadów,
- czarna lista — zbiór węzłów, które należy ignorować.

W module zarządcy jest też przechowywany parametr z maksymalną liczbą autoryzowanych sąsiadów.

Interfejs

Interfejs tego modułu udostępnia takie usługi jak:

- czy autoryzować węzeł — ten moduł decyduje o tym czy na żądanie o autoryzację odpowiedzieć zgodą czy odmową. W ramach tej operacji moduł może odautoryzować inne węzły, by zwolnić połączenie dla nowego sąsiada.
- z jakim prawdopodobieństwem spełnić żądanie danego węzła — nie wszyscy sąsiedzi będą traktowani tak samo.
- do którego węzła wysłać żądanie o fragment — spośród zbioru węzłów, które posiadają dany fragment danych moduł wybiera ten, do którego zostanie wysłane żądanie,
- czy zablokować wymianę danych z którymś węzłem — jeśli któryś z sąsiadów będzie ignorował żądania danej aplikacji, to ten moduł decyduje czy ten węzeł powinien zostać zablokowany.

Wątki

Oprócz biernie dostarczonego interfejsu, w ramach tego modułu działa jeden wątek, służący do optymalizacji połączeń w sieci. Wątek nawiązujący nowe połączenia działa w tle i okresowo budzi się, sprawdzając stan połączeń w sieci. Decyduje on o tym czy powinien nawiązać nowe połączenie oraz czy powinien któregoś ze swoich starych sąsiadów wymienić na nowego.

W obecnej implementacji moduł zarządcy węzłów przechowuje wyłącznie podzbiór węzłów aktywnych w systemie. Nie posiada on jednak żadnych informacji na temat połączeń pomiędzy węzłami oraz ich odległości od nadajników.

5.6.5. Bufor

Moduł **bufora** służy do zarządzania strukturą danych strumienia multimedialnego w aplikacji. Zgodnie z projektem systemu, strumień jest reprezentowany w postaci kodowania Multiple Decription Coding.

Struktura

Okno bufora to struktura określająca jakie fragmenty strumienia są w danym momencie aktualne. Odzwierciedla ono tablicę deskryptorów MDC (tabela 5.1).

Okno bufora nie wizualizuje jednak stanu całego strumienia, ale tylko pewien aktualnie przetwarzany jego podciąg. Okno bufora ma stałą wielkość — niezmienną liczbę deskryptorów i stałą rozpiętość czasową, liczoną w znacznikach czasowych. Każda kolumna to sekwencja fragmentów. W aplikacji jest reprezentowana za pomocą listy sekwencji.

	1	2	3	4	5	6	7	8
1	D	4	2	1	U	A	U	U
2	D	1	A	A	A	U	A	U
3	5	3	D	5	A	U	U	U
4	D	D	A	A	3	U	U	U

Tabela 5.4: Struktura okna deskryptorów z 8 sekwencjami i 4 deskryptorami

W oknie pokazanym w tabeli 5.4 każdy symbol w miejscu fragmentu oznacza jego stan:

- **U** — niedostępny,
- **A** — dostępny,
- **liczba** — zażądany. Liczba oznacza identyfikator węzła, do którego zostało wysłane żądanie,
- **D** — zapisany.

W oknie bufora nie ma fragmentów przeterminowanych. Fragmenty stają się przeterminowane, kiedy są usuwane z okna.

Interfejs

Interfejs udostępnia następujące usługi:

- pobranie jednej sekwencji — ta operacja dostając jako argument znacznik czasowy przekazuje całą sekwencję oznaczoną tą wartością. Jeśli znacznik czasowy dotyczy sekwencji nie objętej oknem bufora, to przekazana sekwencja będzie pusta (null).
- pobranie fragmentu — znając numer deskryptora i znacznik czasowy fragmentu, można z okna bufora pobrać ten pojedynczy fragment. Jeśli fragment nie znajduje się w oknie bufora, to odpowiedzią będzie pusta referencja (null).
- zapisanie danych do danego fragmentu — mając gotowe dane binarne fragmentu można zapisać je do okna bufora używając tej metody. Parametry metody to numer deskryptora, znacznik czasowy i tablica bajtów. Jeśli wskazany fragment nie znajduje się w oknie bufora, to metoda zgłosi wyjątek.
- przesunięcie okna — ta metoda przesuwa okno bufora względem strumienia o jedną sekwencję. W wyniku tej operacji z okna wypada sekwencja fragmentów o najstarszym znaczniku czasowym i powstaje nowa sekwencja z najświeższym znacznikiem czasowym.

Wątki

Oprócz udostępnienia usług dotyczących bufora ten moduł posiada również własną logikę służącą do wypełniania okna bufora, implementowaną przez wątek **żądającego dane** oraz drugi wątek który odpowiada na takie żądania wysłane przez sąsiadów. Do manipulowania oknem bufora jest wykorzystywany wątek **zegara**.

- **Żądający dane** — wątek oczekujący w oknie bufora na pojawienie się dostępnego fragmentu. Mając fragment o stanie **dostępny** pyta moduł **Logiki**, do którego spośród węzłów posiadający dany fragment powinien się zgłosić, po czym inicjuje w module **Logiki** wysłanie żądanie fragmentu od danego węzła. W działającej aplikacji chodzi tylko jeden taki wątek.
- **Wysyłający fragmenty strumienia** — ten wątek kontroluje wysyłanie fragmentów do sąsiadów, którzy o te dane prosili. W zależności od wcześniejszych zachowań danego sąsiada oraz dostępnej przepustowości łącza wychodzącego ten wątek decyduje o tym czy powinien żądanie spełnić czy też odrzucić.
- **Zegar** — zegar służy do synchronizacji czasu w systemie. Jeden kwant czasu w zegarze jest mierzony jako czas odtwarzania jednej sekwencji. Wątek zegara chodzi i w każdej iteracji przesuwa okienko bufora. Przesuwając okienko bufora zegar wyciąga najstarszą sekwencję i wywołuje odpowiednią metodę tak, by ta sekwencja została przekazana do odtwarzania.

5.6.6. Statystyki

Niezbędnym warunkiem do optymalizacji połączeń pomiędzy węzłami w systemie jest to posiadanie dokładnych informacji na temat sieci. To zadanie w implementacji aplikacji odbiornika pełni właśnie ten moduł.

Struktura

W module **statystyk** jest monitorowana cała komunikacja danej aplikacji z każdym sąsiadem. Do tego celu w module powstał słownik, który dla każdego węzła utrzymuje liczniki wszystkich wysłanych i odebranych komunikatów oraz pakietów ze strumieniem.

Interfejs

Informacje jakie moduł będzie dostarczać to:

- liczba żądań wysłanych do każdego węzła,
- liczba pakietów otrzymanych od każdego węzła,
- sumaryczna liczba bajtów otrzymanych od każdego węzła,
- liczba żądań otrzymanych od każdego węzła,
- liczba pakietów wysłanych do każdego węzła,
- sumaryczna liczba bajtów wysłanych do każdego węzła,
- dwustronne opóźnienie przesyłania pakietu dla każdego węzła,

- chwilowa całkowita prędkość ściągania,
- chwilowa całkowita prędkość wysyłania,
- średnia całkowita prędkość ściągania,
- średnia całkowita prędkość wysyłania.

Wątki

Aby z informacji zebranych z innych modułów wygenerować szczegółowe statystyki potrzebny jest wątek, który nieustannie chodzi i przelicza te informacje. W ramach tego modułu powstał również prosty miernik dwustronnego opóźnienia transportu (ang. *Round-trip delay time*) do wszystkich sąsiadów danego odbiornika. W obecnej wersji zarówno logika wyliczająca statystyki, jak i miernik opóźnienia są uruchamiane w tym samym wątku.

5.7. Logika aplikacji

Jednym z celów tego projektu była możliwość łatwej i szybkiej podmiany istniejącego modułu na nowy bez potrzeby modyfikowania pozostałych. Aby zrealizować ten cel, przyjąłem założenie, że moduły w aplikacji nie będą się bezpośrednio ze sobą komunikować. Zamiast tego wszystkie moduły w systemie będą odwoływać się do centralnej logiki, w której zawarte będą schematy współpracy poszczególnych elementów aplikacji. Sama **Logika** nie implementuje żadnych funkcjonalności, tylko oddelegowuje wszystkie wywołania do odpowiednich modułów. W tej części przedstawię różne przypadki użycia aplikacji oraz scenariusze przepływu sterowania.

5.7.1. Przetwarzanie przychodzącego komunikatu sterującego

Podstawową funkcjonalnością logiki odbiornika jest umiejętność komunikowania się z innymi węzłami w sieci. Aplikacja musi zatem potrafić przetwarzać każdy przychodzący komunikat a potem odpowiednio go obsłużyć. Na rysunku 5.7 został opisany proces przetwarzania komunikatu — czyli etap zaczynający się od momentu przyjścia datagramu danych z sieci do momentu stworzenia obiektu komunikatu.

Widać na tym rysunku, że wywołania do poszczególnych modułów są robione za pośrednictwem logiki. Na przykładzie tej usługi widać, że wyłączenie modułu statystyk polega tylko na usunięciu wywołania do tego modułu w logice, które jest całkowicie niezauważalne w innych modułach.

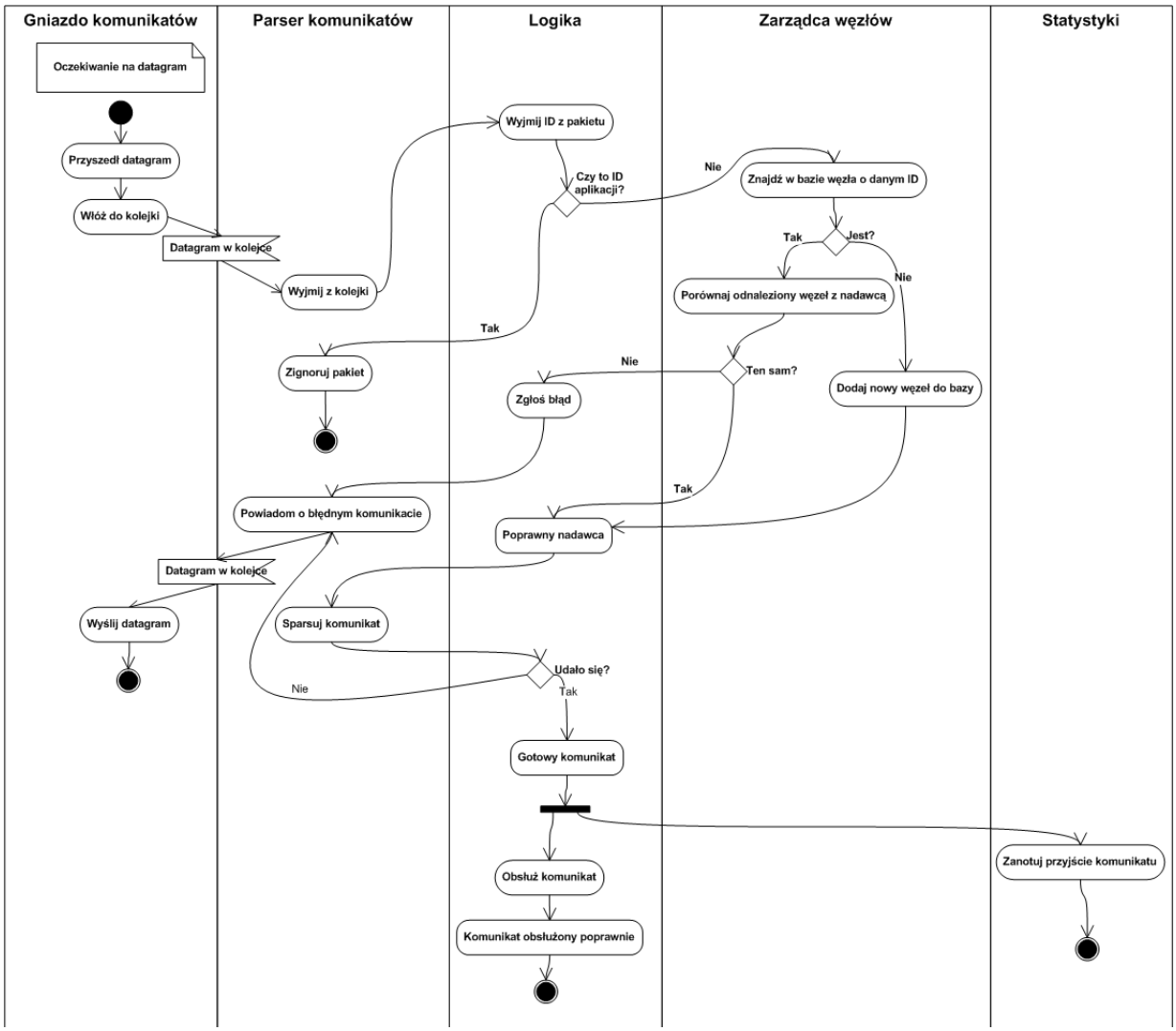
Obsługa poszczególnych komunikatów zostanie przedstawiona w następnych częściach.

5.7.2. Obsługa żądania autoryzacji

Na rysunku 5.8 znajduje się opis scenariusza przyjęcia **żądania autoryzacji**. Proces przetwarzania pakietu pokazany w poprzedniej części, został tutaj skrócony do dwóch pierwszych czynności (**datagram** i **otrzymano żądanie autoryzacji**).

5.7.3. Obsługa zapytania o kanał

Na rysunku 5.9 przedstawiam scenariusz, w którym aplikacja posiadająca informacje o kanale, otrzymuje zapytanie **żądanie informacji o kanale**. W takiej sytuacji konieczne jest pobranie



Rysunek 5.7: Przetwarzanie przychodzącego komunikatu

od zegara znacznika czasowego aktualnie odtwarzanej sekwencji oraz opóźnienie odtwarzania — informacje potrzebna w komunikacie zwrotnym.

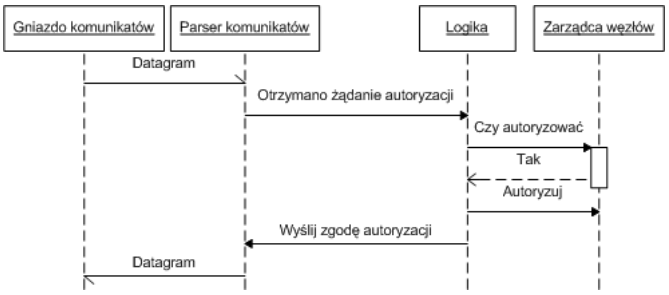
W przypadku kiedy odbiornik nie wie nic jeszcze o kanale, odpowiedzią na takie zapytanie powinien być komunikat z **brakiem kanału**.

5.7.4. Obsługa żądania fragmentu

Otrzymawszy żądanie fragmentu, zapisujemy żądanie do kolejki żądań. Żądanie zostanie wyciągnięte z drugiej strony przez **wysyłającego fragmenty strumienia** (rysunek 5.10).

5.7.5. Obsługa pakietu z fragmentem strumienia

Diagram sekwencji z rysunku 5.11 pokazuje co się dzieje z przychodzącymi pakietami RTP, które zawierają fragment strumienia.



Rysunek 5.8: Obsługa żądania autoryzacji

Rysunek 5.9: Obsługa zapytania o kanał

5.8. Uruchomienie aplikacji

W ramach projektu powstała aplikacja nadawcy i odbiornika. Opiszę tutaj sposób uruchamiania ich oraz podam krótką instrukcję obsługi każdej z aplikacji.

5.8.1. Nadawca

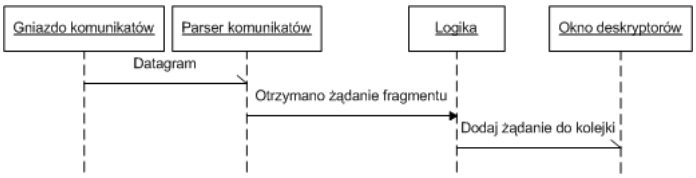
W tej wersji nadawca został zaimplementowany do obsługi slajdu obrazów, bez możliwości zakodowania prawdziwego strumienia wideo. Aplikacja nadawcy nie posiada interfejsu graficznego i może zostać uruchomiona z podaniem pliku konfiguracyjnego. Plik konfiguracyjny musi być w postaci XML i zawierać następujące informacje:

- channelName — nazwa kanału,
- channelDescription — opis kanału,
- path — ścieżka do katalogu z sekwencją obrazów,
- framerate — częstotliwość wyświetlania obrazów,
- maxPeers — maksymalna liczba odbiorców,
- maxUpload — maksymalna prędkość wysyłania.

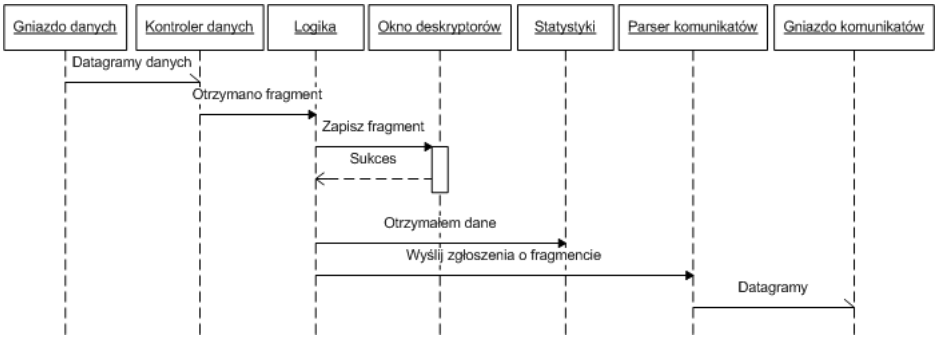
5.8.2. Odbiornik

Aplikacja odbiornika może zostać uruchomiona w trybie nieinteraktywnym lub graficznym.

Pierwszy tryb był wykorzystywany do celów testowych i nie posiada możliwości wyświetlania ściągniętego strumienia. W tym trybie podczas uruchomienia z linij komend należy



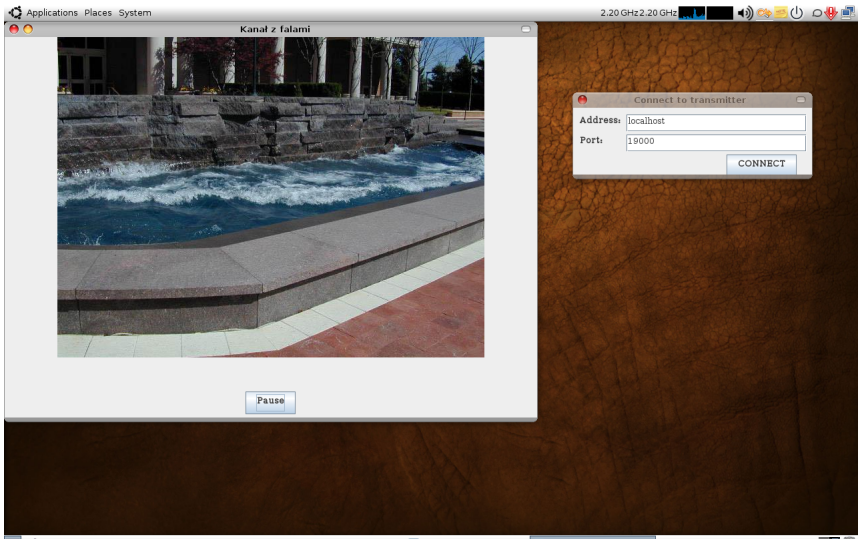
Rysunek 5.10: Obsługa żądania fragmentu



Rysunek 5.11: Reakcja na pakiety z fragmentem strumienia

podać plik konfiguracyjny aplikacji oraz adres nadawcy/odbiorcy, z którym aplikacja ma się połączyć.

W trybie graficznym po uruchomieniu użytkownik zobaczy okno nawiązania połączenia (jak to widać na rysunku 5.12), gdzie musi podać adres nadawcy/odbiornika, z którym chce się połączyć. Po zatwierdzeniu adresu użytkownik połączy się z systemem nadającym kanał jeśli adres był poprawny albo aplikacja zamknie się z błędem w przeciwnym przypadku. Po połączeniu się z systemem pojawi się okno odtwarzania, na którym będzie ekran wideo oraz jeden przycisk sterowania **play/pause**. Przycisk jest aktywny jeśli odbiornik będzie posiadał informacje o kanale. Kiedy odtwarzacz znajduje się w trybie odtwarzania na przycisku powinien znajdować się napis **pause** — oznaczający możliwość zatrzymania odtwarzania. Po naciśnięciu na ten przycisk **pause** odtwarzanie wideo powinno zatrzymać się na ostatnim obrazie, a przycisk zmieni się na **play**. Na rysunku 5.13 widzimy przykład z czterema uruchomionymi instancjami odbiornika, gdzie trzy są w stanie odtwarzania, a czwarty (w prawym górnym rogu) został wstrzymany.



Rysunek 5.12: Jeden uruchomiony odbiornik i okno połączenia drugiej instancji

Rysunek 5.13: Przykład czterech uruchomionych odbiorników

Rozdział 6

Testy

W tym rozdziale przedstawię wyniki przeprowadzonych testów. Ich celem jest pokazanie, iż system sprawdza się w praktycznym zastosowaniu i może być używany do transmisji strumieni multimedialnych. Na podstawie różnych rodzajów testów uzyskałem również ważne informacje na temat ograniczeń i niedociągnięć w całym projekcie. W pierwszej części tego rozdziału opiszę środowisko testowe, które wykorzystałem do przeprowadzenia tych testów. Później przedstawię plan testów i omówię po kolei każdy scenariusz testowy.

6.1. Środowisko

Testy przedstawione w tym rozdziale zostały przeprowadzone w warunkach hermetycznych, stworzonych specjalnie do tego celu. W celu wykonania tych testów otrzymałem dostęp do laboratorium komputerowego, należącego do Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego.

Laboratorium o nazwie **Red**, gdzie testy miały, miejsce składa się z szesnastu maszyn. Każda z maszyn w tym laboratorium ma konfigurację:

- procesor — INTEL CORE 2 DUO 6600 2.4GHz,
- pamięć — 2 GB,
- karta sieciowa — 100 Mbps PCI,
- system operacyjny — PLD Linux i686 z jądrem 2.6.22.18,
- Java — Java(TM) SE Runtime Environment (build 1.6.0_05-b13).

Komputery były połączone w sieć lokalną przy użyciu przełącznika o przepustowości 1 Gbps.

Podczas działania testów maszyny nie były obciążane przez żadne inne procesy.

6.2. Plan testów

Dla lepszego zrozumienia wyników uzyskanych w tym rozdziale, przedstawię tutaj cały plan testów. W tym zakresie zaprezentuję cele procesu testowania, opiszę ogólną procedurę testu i omówię strukturę danych będących rezultatem tych testów. Wszystkie testy charakteryzują się powtarzalnością i czynnik losowości został zmniejszony do minimum.

Cel

Celem testów jest sprawdzenie systemu pod względem:

- poprawności,
- wydajności,
- obciążenia,
- skalowalności,
- odporności na awarie.

Struktura

W testach zostały wykorzystane:

- Nadajnik — aplikacja konsolowa, wyświetlająca cykliczne sekwencję 10 obrazów, każdy o rozmiarze 60 KB.
- Odbiornik — aplikacja w trybie konsolowej bez odtwarzacza, która zapisuje na bieżąco statystyki do pliku.

W każdym teście obciążenie było równomiernie rozkładane na wszystkie maszyny, poprzez sprawiedliwy podział liczby uruchomionych aplikacji odbiorników na pojedynczą maszynę. We wszystkich testach do generowania strumienia multimedialnego służyła wyłącznie jedna instancja aplikacji **Nadajnika**.

Każdy test polega na włączeniu jednej instancji Nadajnika do nadawania strumienia oraz wielu instancji Odbiorników do odbierania tego strumienia i rozpropagowania go dalej. Wszystkie scenariusze można uogólnić przy pomocy następującego schematu:

1. uruchomienie nadajnika na jednej z maszyn,
2. równoległe wykonanie na wszystkich maszynach skryptu uruchamiającego odbiorniki (z równym podziałem liczby instancji na poszczególne maszyny). Na każdej maszynie odbiorniki są uruchamiane w łańcuchu — każdy podłącza się do poprzedniego. Skrypt wykonuje czynności:
   (a) uruchamia odbiornik, wskazując mu nadajnik do podłączenia się,
   (b) uruchomia następne odbiorniki. Każdy w miejsce nadajnika otrzymuje adres uprzednio uruchomionego odbiornika.

Testy powtarzają ten sam schemat działania, lecz z różnymi parametrami. Każdy z testów charakteryzuje się parametrami:

- czas trwania transmisji — wartość wyrażona w sekundach,
- częstotliwość strumienia — liczba sekwencji wyświetlonych w ciągu jednej sekundę,
- opóźnienie odtwarzania — okres czasu, określający z jakim wyprzedzeniem strumień powinien być dostępny w systemie zanim nastąpi jego czas odtwarzania,
- sąsiedzi nadajnika — maksymalna liczba węzłów jaką nadajnik może autoryzować,

- odbiorniki — liczba wszystkich odbiorników, biorących udział w wymianie danych,
- czas życia odbiornika — rama czasowa, określająca moment przyłączenia się węzła do systemu i moment jego odłączenia.
- sąsiedzi odbiornika — maksymalna liczba węzłów jaką każdy odbiornik może autoryzować.

Rezultat

Podczas trwania testu każdy z odbiorników nieustannie zapisuje do pliku swój stan, z których później są generowane sumaryczne statystyki. Dane są zapisywane do pliku raz na kwant czasu zegara (długość trwania jednej sekwencji). W ramach każdego zapisu utrwalane są takie informacje jak:

- rodzaje wysłanych i odebranych komunikatów,
- liczba odebranych i wysłanych fragmentów strumienia,
- wejściowe i wyjściowe zużycie łącza,
- opóźnienie łączy do poszczególnych sąsiadów.

Spośród tych wszystkich danych najistotniejszą rolę w analizie wydajności systemu odgrywa informacja na temat fragmentów strumienia. Liczba otrzymanych na czas fragmentów danych dokładnie odzwierciedla jakość strumienia wyświetlonego w końcowym odtwarzaczu.

6.3. Scenariusze

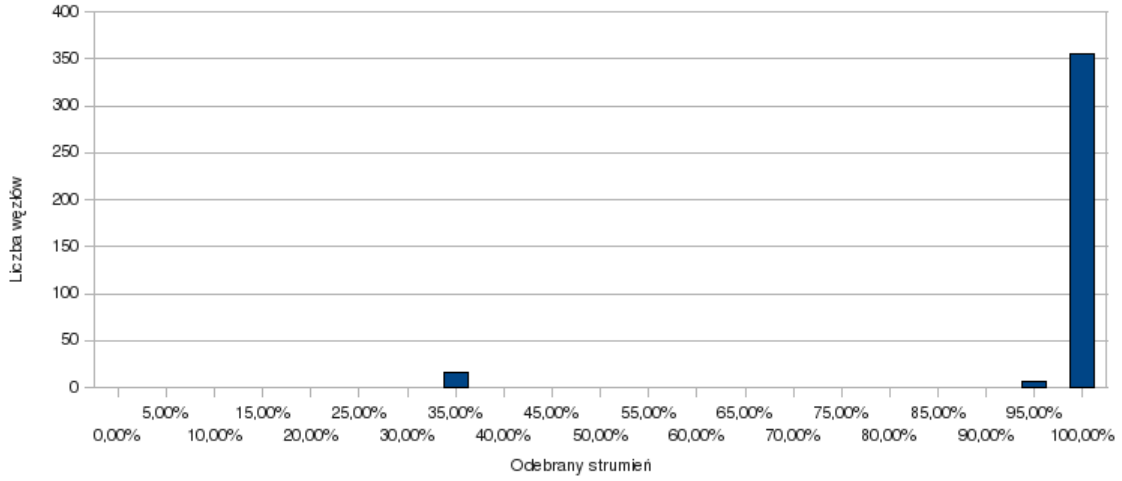
W ramach każdego scenariusza przedstawię parametry, przy jakich dany test został uruchomiony. Parametry zostały dobrane tak by testy jak najefektywniej wykorzystywały możliwości dostępnego środowiska, zachowując przy tym płynność działania.

6.3.1. Testy statyczne

Testy statyczne zawierają zestaw scenariuszy testowych, gdzie struktura sieciowa węzłów jest statyczna w trakcie całej transmisji. Wszystkie operacje nawiązywania połączenia są dokonywane zanim jeszcze zaczyna się transmisja i do momentu jej zakończenia nie dochodzą ani nie odchodzą żadne węzły. W ramach tego przypadku przedstawię trzy scenariusze.

Mały strumień — 500 Kbps

- czas transmisji — 60 sekund,
- częstotliwość strumienia — 1 sekwencja na sekundę,
- opóźnienie odtwarzania — 10 sekund,
- sąsiedzi nadajnika — 16,
- liczba jednoczesnych węzłów w systemie — 375,
- liczba wszystkich węzłów w systemie — 375,



Rysunek 6.1: Jakość odebranego strumienia dla systemu z 375 odbiornikami i strumieniem 500 Kbps

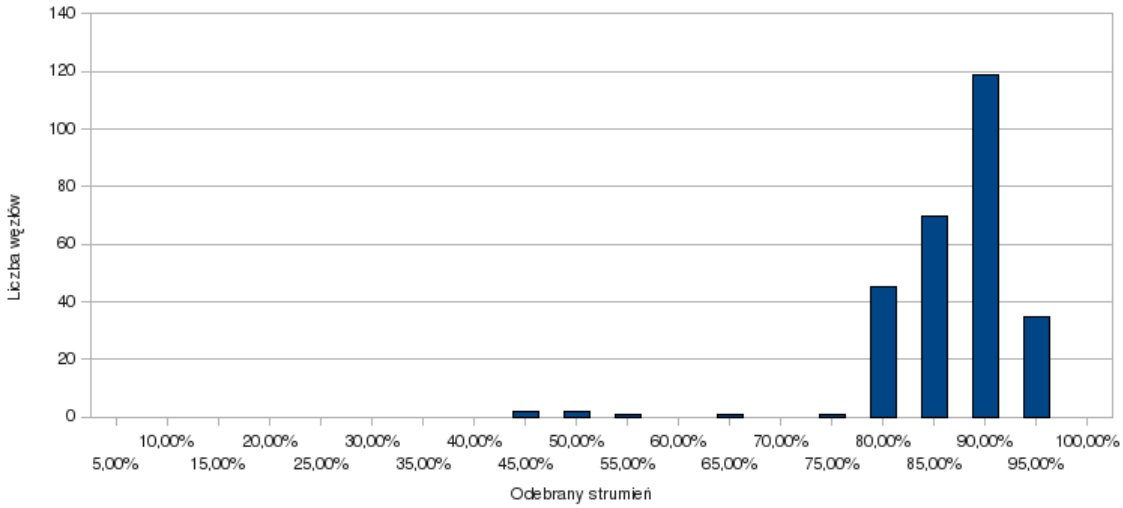
- czas życia odbiornika — 60 sekund,
- sąsiedzi odbiornika — 6,

Przy małym strumieniu obciążenie jednego odbiornika była nieznaczne — 0,2% procesora i współczynnik load wynoszący 0,5 dla każdej maszyny. Dzięki temu możliwe było uruchomienie aż 375 instancji odbiorników. Przy próbie uruchomienia większej liczby, część instancji nie otrzymywała dostępu do procesora. Jak widać (na rysunku 6.1) w tym scenariuszu tylko kilka węzłów nie otrzymało wszystkich 60 sekwencji. Taka wydajność w odbieraniu strumienia przekłada się na 500 Kbps zużycie łącza wejściowego dla każdej instancji, czyli w sumie około 175 Mbps ruchu dla całej sieci.

Średni strumień — 2,5 Mbps

- czas transmisji — 60 sekund,
- częstotliwość strumienia — 5 sekwencji na sekundę,
- opóźnienie odtwarzania — 10 sekund,
- sąsiedzi nadajnika — 16,
- liczba jednoczesnych węzłów w systemie — 280,
- liczba wszystkich węzłów w systemie — 280,
- czas życia odbiornika — 60 sekund,
- sąsiedzi odbiornika — 6,

Przy pięciokrotnym zwiększeniu obciążenia łącza, liczba odbiorników umożliwiająca płynne działanie systemu zmniejszyła się do 280. Jak widać na wykresie 6.2 przy przetwarzaniu



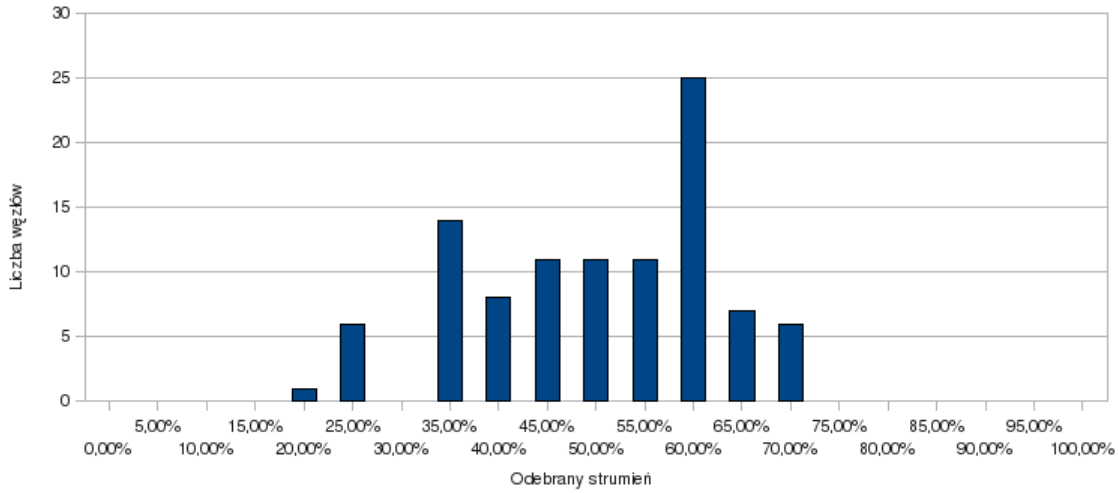
Rysunek 6.2: Jakość odebranego strumienia dla systemu z 280 odbiornikami i strumieniem 2,5 Mbps

pięciu sekwencji na sekundę, większość węzłów otrzymuje już tylko 90% strumienia. Należy jednak pamiętać, że jest to 90% z 2,5Mbps. To znaczy, że każdy węzeł otrzymuje średnio 2,2 Mbps, co daje w sumie 590 Mbps ruchu całej sieci.

Duży strumień — 5 Mbps

- czas transmisji — 60 sekund,
- częstotliwość strumienia — 10 sekwencji na sekundę,
- opóźnienie odtwarzania — 10 sekund,
- sąsiedzi nadajnika — 16,
- liczba jednoczesnych węzłów w systemie — 100,
- liczba wszystkich węzłów w systemie — 100,
- czas życia odbiornika — 60 sekund,
- sąsiedzi odbiornika — 6,

Niestety w momencie zwiększenia liczby sekwencji do 10 na sekundę wyniki (rysunek 6.3) znacznie się pogorszyły. Obciążenie (wskaźnik load) maszyny przy transmisji wzrosło z 0,5 (w poprzednich testach) do 2, a zużycie procesora na proces wynosił około 3%. Dużym problemem była również ograniczona wielkość bufora UDP na komputerach testowych (domyślnie 128 KB), który w przypadku przepełnienia zaczyna odrzucać pakiety. W tym scenariuszu próba uruchomienia więcej niż 100 węzłów w systemie powodowało, że duża część odbiorników w ogóle nie otrzymywała żadnych danych.



Rysunek 6.3: Jakość odebranego strumienia dla systemu z 100 odbiornikami i strumieniem 5 Mbps

Wysyłanie pakietów

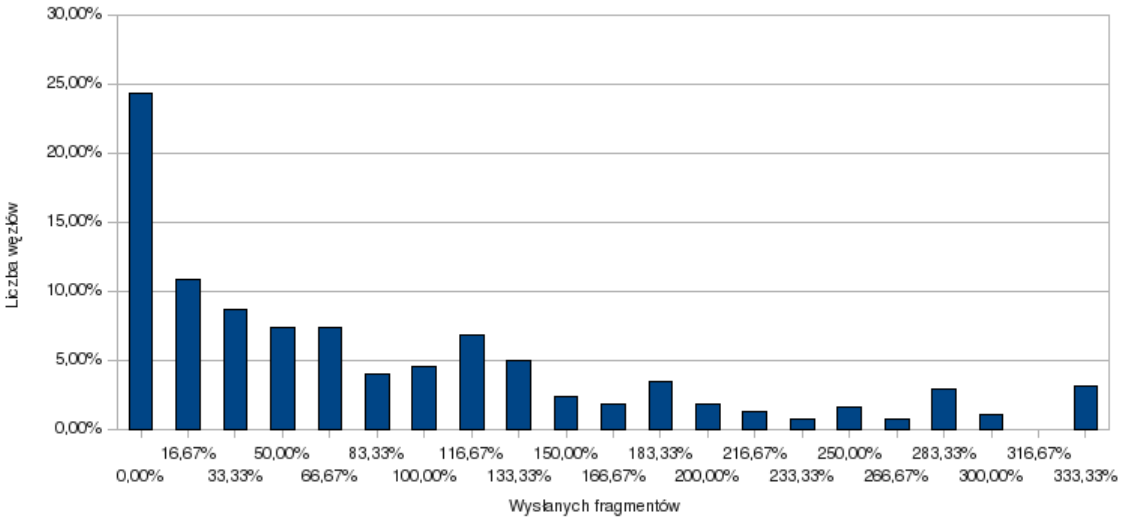
Powyżej przedstawiłem wykresy mówiące o jakości odebranego strumienia dla każdego węzła. Główna zaleta sieci punkt-do-punktu polega jednak na wydajniejszym wykorzystaniu wyjściowego łącza każdego węzła. W przypadku wysyłania pakietów z danymi wszystkie powyższe scenariusze zachowały się podobnie jak pokazuje wykres 6.4. Na tym wykresie oś pozioma przedstawia średnią przepustowość wyjściową każdego węzła w porównaniu z wielkością strumienia nadawanego w systemie. To znaczy, że w odniesieniu do pierwszego scenariusza 100% to 500 Kbps, w drugim 100% to 2,5 Mbps, a w trzecim 5 Mbps. W tych wszystkich trzech przypadkach podczas transmisji około 25% węzłów praktycznie nic nie wysyła. Zaś ponad jedna trzecia węzłów wysyła więcej niż może odebrać (100%). Po zliczeniu całego ruchu w sieci podczas transmisji okazało się, że około 94% pakietów z danymi było wysłane przez Odbiorniki, a tylko 6% przez sam Nadajnik.

6.3.2. Testy dynamiczne

W tej części przedstawię testy, w których do sieci będą nieustannie przyłączać się i rozłączać węzły. Jak pokażą wyniki, zmieniające się systemy charakteryzują się mniejszą wydajnością niż sieci statyczne.

Odbiornik z 6 sąsiadami

- czas transmisji — 240 sekund,
- częstotliwość strumienia — 1 sekwencja na sekundę,
- opóźnienie odtwarzania — 10 sekund,
- sąsiedzi nadajnika — 16,
- liczba jednoczesnych węzłów w systemie — 160,



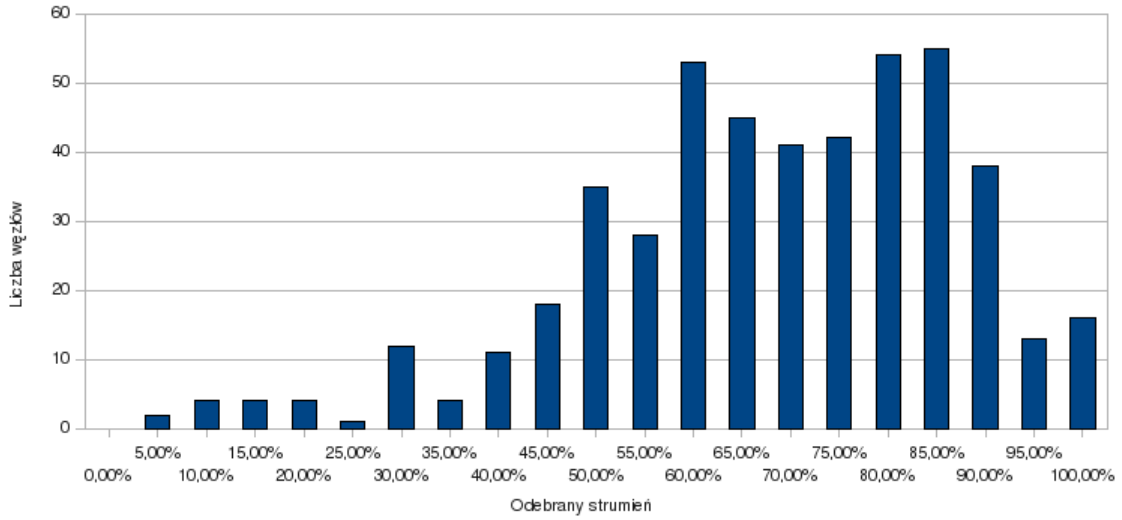
Rysunek 6.4: Liczba wysłanych pakietów na każdy węzeł

- liczba wszystkich węzłów w systemie — 480,
- czas życia odbiornika — 60 sekund, na każdej maszynie co 6 sekund jest uruchamiana nowa instancja odbiornika,
- sąsiedzi odbiornika — 6,

Jak widać na wykresie 6.5 efektywność tego systemu w porównaniu ze statyczną siecią węzłów o tej samej jakości strumienia, jest znacznie mniejsza. W tym przypadku próba uruchomienia jednocześnie 375 węzłów, a w sumie 1125 węzłów spowodowała, że były grupy węzłów, które odłączały się od systemu tworząc oddzielną sieć, nie połączoną z Nadajnikiem. To zachowania wynika z faktu, iż węzły w tej implementacji losowo wybierają sąsiadów, nie biorąc pod uwagę położenie sąsiada względem Nadajnika. Można zobaczyć, że w tym scenariuszu jakość otrzymanego strumienia waha się między 50% a 90%, co nie jest zadowalającym rezultatem.

Odbiornik z 10 sąsiadami

- czas transmisji — 240 sekund,
- częstotliwość strumienia — 1 sekwencja na sekundę,
- opóźnienie odtwarzania — 10 sekund,
- sąsiedzi nadajnika — 16,
- liczba jednoczesnych węzłów w systemie — 160,
- liczba wszystkich węzłów w systemie — 480,
- czas życia odbiornika — 60 sekund, na każdej maszynie co 6 sekund jest uruchamiana nowa instancja odbiornika,
- sąsiedzi odbiornika — 10,



Rysunek 6.5: Jakość odebranego strumienia dla systemu z 160 odbiornikami i strumieniem 500 Kbps

Jak napisałem w poprzednim scenariuszu, problemem dla systemów ze zmieniającymi się węzłami, jest duża częsta rotacja sąsiadów dla każdego węzła. W przypadku, kiedy odbiornik może mieć mało sąsiadów, wymiana sąsiada bardziej wpływa na całkowity ruch danego odbiornika. Jeśli jednak zmienimy konfigurację odbiorników i pozwolimy by każdy mógł mieć nie 6, lecz 10 sąsiadów, to okazuje się, że potrafi on ściągać znacznie lepszy strumień (jak na rysunku 6.6). Jest to potwierdzeniem, że mając więcej sąsiadów węzeł staje się stabilniejszy.

6.3.3. Testy awarii

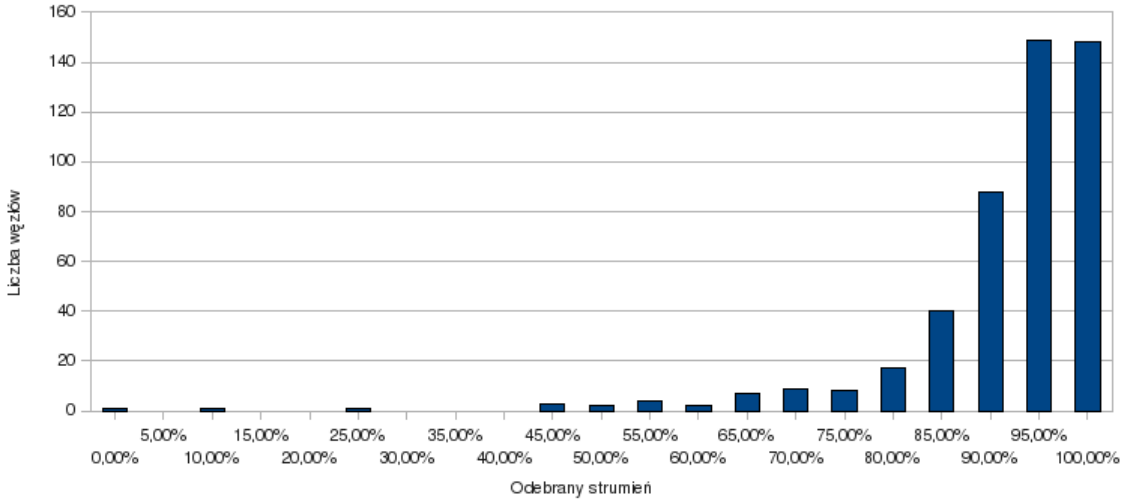
W tej części przedstawię działanie systemu w sytuacji awaryjnej. Podczas tych testów zabiję część węzłów działających w systemie i sprawdzę jak system reaguje na to. Scenariusz tego testu jest taki jak test z rysunku 6.6, przy czym podczas 240 sekundowej transmisji wystąpiły trzy sytuacje awaryjne:

- w 36 sekundzie wyłączono maszynę red01,
- w 72 sekundzie wyłączono maszynę red02,
- w 108 sekundzie wyłączono maszynę red03.

Jak się okazuje (wykres 6.7) te awarie nie spowodowały znacznego spadku jakości na pozostałych maszynach, na których działało 425 pozostałych instancji. Jak widać, nawet przy awarii 10% odbiorników w sieci, pozostałe potrafią szybko się od nich uniezależnić i nadal utrzymać wysoką jakość strumienia.

6.4. Wnioski

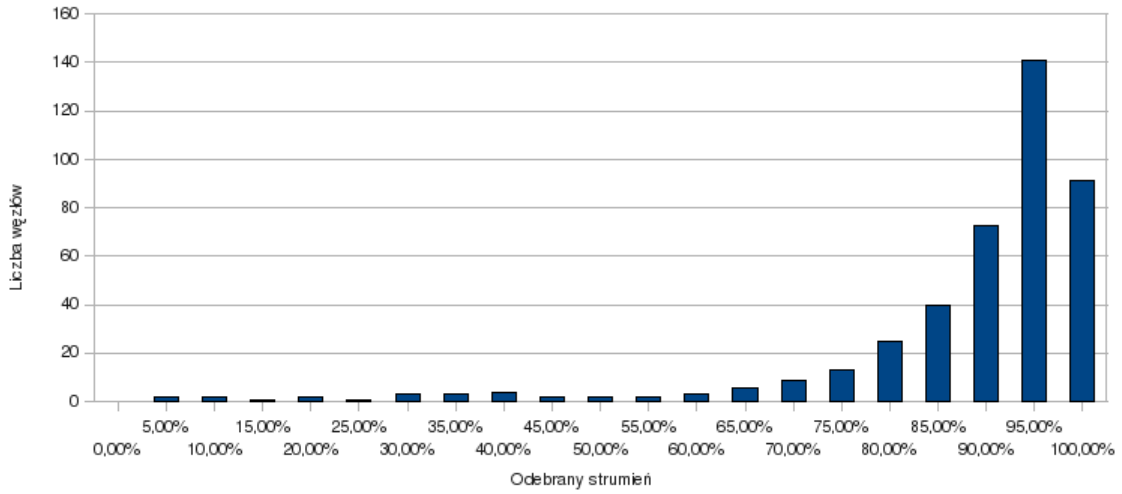
W celu sprawdzenia wydajności i poprawności tego systemu przeprowadziłem kilkadziesiąt testów dla kilkunastu różnych scenariuszy. Z wszystkich wybrałem i omówiłem 6 scenariuszy, które są najwyraźniejsze i najlepiej ukazują zachowanie systemu. Spośród wielu parametrów w



Rysunek 6.6: Strumień 500 Kbps z 160 jednoczesnymi odbiornikami. Każdy węzeł może mieć 10 sąsiadów

systemie, na przekroju testów modyfikowałem tylko parametry związane z jakością strumienia i liczbą sąsiadów dla odbiornika, gdyż mają one największy wpływ na działanie w systemu.

Przedstawione w tej części wyniki testów z pewnością nie wyczerpały wszystkich możliwych przypadków działania systemu. Rezultat otrzymany w danym środowisku nie odzwierciedla również działania sytemu w Internecie. Jak pokazały testy, udało się jednak osiągnąć podstawowy cel tego projektu, polegający na zmniejszeniu obciążenia Nadajnika na koszt Odbiorników.



Rysunek 6.7: Strumień 500 Kbps z 160 jednoczesnymi odbiornikami (z 10 sąsiadami), sytuacja z 3 awariami

Rozdział 7

Podsumowanie

W ramach pracy przygotowałem projekt i prototypową implementację systemu przekaźników strumienia multimedialnego. Powstałe rozwiązanie spełnia wszystkie wymagania założone na początku projektu. Przy pomocy dostarczonej aplikacji nadawczej możliwe jest wytworzenie strumienia multimedialnego „na żywo" i nadanie go do wszystkich zainteresowanych. Korzystając z aplikacji odbiornika użytkownik może obejrzeć nadawany strumień.

Dzięki strukturze punkt-do-punktu sieci przekaźników system jest teoretycznie nieograniczenie skalowalny. W praktyce, jak pokazały testy, nawet w sieci z setkami odbiornikami jakość otrzymanego strumienia dla każdego poszczególnego odbiornika jest nadal bardzo wysoka.

Udało się w tym rozwiązaniu zrealizować koncepcję odtwarzania w czasie rzeczywistym, gdzie nacisk kładziony jest na płynne wyświetlanie strumienia, a nie na utrzymanie najwyższej jakości obrazu. Nadawany strumień w przypadkach awarii sieci potrafi zmniejszyć swoją jakość utrzymując nadal płynny obraz. W momencie kiedy usterka znika jakość jest błyskawicznie przywracana.

Zdecentralizowany i niestrukturyzowany charakter sieci odbiorników sprawia, że zachowanie całego systemu jest zdeterminowane przez logikę poszczególnych węzłów i ich wzajemną współpracę. Odbiorniki, dzięki umiejętności mierzenia opóźnienia transmisji oraz kontrolowania przepustowości łącza, nawiązują wydajniejsze połączenia z sąsiadami oraz dynamicznie adaptują się do warunków w sieci. To wszystko powoduje, że system podczas działania nieustannie zmienia i optymalizuje się.

Obecny wynik tej pracy jest tylko implementacją szkieletu systemu przekaźników strumienia multimedialnego. Stworzenie efektywnego i praktycznego produktu z tej dziedziny wymaga znacznie więcej czasu i pracy. Ten projekt pokazał jednak, że stworzenie działającej sieci przekaźników punkt-do-punktu jest możliwe.

7.1. Możliwe rozszerzenia

Struktury aplikacji odbiornika i nadajnika zostały zaprojektowane z myślą o rozszerzeniach i modyfikacjach w przyszłości. W obecnej wersji aplikacji nie wszystkie moduły zostały zaimplementowane w najwydajniejszy sposób. Poprawki i refaktoryzacje polegające na wykorzystaniu szybszych struktur danych oraz lepszym zrównolegleniu pewnych procesów dałyby z pewnością lepszy efekt.

Spośród funkcjonalnych rozszerzeń, chciałbym zrealizować następujące:

7.1.1. Mechanizm plotkowania

Wzbogacenie obecnego protokołu sterującego o własności **mechanizmu plotkowania** [3] dałyby możliwość wykorzystania znanych własności procedury plotkowania do optymalizacji struktury sieci. Przy istniejącej specyfikacji protokołu sterującego, dane na temat węzłów obejmują wyłącznie ich adresy. Jeśli mechanizm plotkowania zostałby wprowadzony, do wymienianych informacji mogłyby zostać dodane informacje statystyczne o węzłach, które już w obecnym rozwiązaniu są zbierane. Jak wynika z badań systemów plotkujących, dzięki temu procesowi informacje na temat np. oszukujących węzłów, odległości pomiędzy węzłami albo przynależności do podsieci byłyby bardzo szybko rozpropagowane po całej sieci. Z takim rozwiązaniem baza informacji o węzłach w każdym odbiorniku miałaby charakter globalny, a nie tylko lokalny.

7.1.2. Algorytm nawiązywania połączeń

W tym rozwiązaniu decyzje o nawiązaniu połączeń i ich zrywaniu są dokonywane wyłącznie na podstawie opóźnienia połączenia pomiędzy dwoma węzłami oraz liczbie danych przesłanych pomiędzy nimi. Sam węzeł nie przetrzymuje żadnych informacji na temat sieci jako całości. W przypadku takiego systemu nadawczego sieć jest typowym modelem grafu przepływów i algorytmy heurystyczne dla sieci przepływowych mogą wpłynąć na lepszą wydajność takiego systemu. Dodatkowa optymalizacja systemu może zostać uzyskana poprzez wprowadzenie do aplikacji odbiornika mechanizmu mapującego strukturę sieci węzłów na graf.

7.1.3. Kodowanie Multiple Description Coding

Istniejąca implementacja nadajnika i odbiornika pozwala wyłącznie na przesyłanie sekwencji zdjęć i wyświetlenie ich w postaci pokazu slajdów. Pomimo, iż cały system korzysta z modelu Multiple Description Coding do transmisji danych, to jednak strumień nie jest zakodowany ani dekodowany przy użyciu metody MDC. Jeśli do implementacji zostanie dołączony koder i dekoder MDC, wówczas rozwiązanie będzie miało już charakter prawdziwej telewizji internetowej. Na chwilę obecną nie istnieje jeszcze jednak efektywna otwarta implementacja schematu Multiple Description Coding.

Dodatek A. Zawartość płyty CD

doc/praca.pdf	Praca tekstowa w formacie PDF
src/Receiver	Katalog z kodem źródłowym odbiornika
src/Transmitter	Katalog z kodem źródłowym nadajnika
bin/Receiver.jar	Skompilowana aplikacja odbiornika
bin/Transmitter.jar	Skompilowana aplikacja nadajnika
bin/lib	Katalog z bibliotekami potrzebnymi do uruchomienia aplikacji
conf/receiver.xml	Przykładowy plik konfiguracyjny dla odbiornika
conf/transmitter.xml	Przykładowy plik konfiguracyjny dla nadajnika
script	Katalog z skryptami do uruchamiania testów

Bibliografia

[1] *Bittorrent Protocol Specification v1.0*, http://wiki.theory.org, 2006.

[2] Joshua Bloch, *How to Design a Good API and Why it Matters*, Google Presentation, 2007.

[3] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, *Gossip Algorithms: Design, Analysis and Applications*, Proceedings IEEE Infocomm, 3:1653-1664, Miami, March 2005.

[4] Miguel Castro, Manuel Costa and Antony Rowstron, *Peer-to-peer overlays: structured, unstructured, or both?*, Microsoft Research, Cambridge, CB3 0FB, UK, 2004.

[5] Cisco Systems, *Approaching the Zettabyte Era*, Cisco Visual Networking Index, 2008.

[6] Bram Cohen, *The BitTorrent Protocol Specification*, BitTorrent.org, 2008.

[7] Mark Crovella , Balachander Krishuamurthy, *Internet Measurement*, John Wiley and Sons, 200and

[8] Ellacoya, *Ellacoya Data Shows Web Traffic Overtakes Peer-to-Peer (P2P) as Largest Percentage of Bandwidth on the Network*, 2007.

[9] *The Gnutella Protocol Specification v0.4*, 2000.

[10] Vivek K. Goyal, *Multiple Description Coding: Compression Meets the Network*, IEEE Signal Processing Magazine, 2001.

[11] www.internetworldstats.com, *Internet Usage in Europe*, Internet World Stats, 2008.

[12] Tor Klingberg, Raphael Manfredi, *The Gnutella Protocol Specification v0.6*, http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, 2002

[13] Keith Lea, *The Java is Faster than C++ and C++ Sucks Unbiased Benchmark*, http://www.kano.net/javabench, 2008.

[14] Colm Maccarthaigh, *Joost Network Architecture*, 2007.

[15] Colin Perkins, *RTP: Audio and Video for the Internet*, Addison Wesley Professional, 2003.

[16] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker2, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen and H. J. Sips, *TRIBLER: a social-based peer-to-peer system*, Wiley InterScience, 2007.

[17] *Prisoner's Dilemma*, Stanford Encyclopedia of Philosophy, 2007.

[18] Marshall T. Rose, *On the Design of Application Protocols*, RFC 3117, 2001.

[19] Keith W. Ross, Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, *Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System*, 2006.

[20] Keith W. Ross, Jian Liang, Rakesh Kumar, *Understanding KaZaA*, http://citeseer.ist.psu.edu/liang04understanding.html, 2004.

[21] Keith W. Ross, Zhengye Liu, Yanming Shen, Shivendra S. Panwar, Yao Wang *P2P Video Live Streaming With MDC: Providing Incentives For Redistribution*, 2007.

[22] Salvadori Baptista do Carmo, Gustavo Salem Barbar, Jamil Coelho, Fernanda Barbosa, *A New Classification for Peer-to-peer Networks Architectures*, Latin America Transactions, IEEE (Revista IEEE America Latina), 2006.

[23] Sandvine, *2008 Analysis of Traffic Demographics in North-American Broadband Networks*, 2008.

[24] H. Schulzrinne, *RTP Profile for Audio and Video Conferences with Minimal Control*, RFC 1890, 1996.

[25] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, 1996.

[26] Wes Simpson, *Video Over IP: A Practical Guide to Technology and Applications*, Focal Press, 2005.

[27] Karsten Weide, *U.S. Online Consumer Behavior Survey Results 2007, Part I: Wireline Internet Usage*, International Data Corporation, 2007.

[28] Wikipedia, *Agile Software Development*, http://en.wikipedia.org/wiki/Agile_software_development, 2008.