BAB I
PENDAHULUAN

1.1 Latar Belakang

Internet telah berkembang menjadi media penyebaran informasi yang banyak digunakan oleh pengguna 32% dari populasi dunia (World Bank). Salah satu bentuk informasi yang banyak digunakan oleh pengguna internet adalah informasi berupa suara. Proses pengantaran berkas suara digital melalui internet disebut *audio streaming*.

Audio streaming umumnya dilakukan secara terpusat yaitu terdapat satu *node* peladen (*server*) yang melayani permintaan dari *node* yang bertindak sebagai pengguna (*client*). Pada metode ini beban akan terpusat pada satu peladen, dengan demikian untuk melayani pengguna dalam skala besar dibutuhkan sumberdaya yang besar pada sisi peladen. Sebagai contoh situs Youtube membutuhkan dana sekitar US $ 6 Juta sebagai biaya bandwith (Garbacki:1).

Salah satu masalah yang banyak dihadapi dalam *audio streaming* terpusat adalah beban yang makin besar seiring dengan jumlah pengguna yang semakin besar. Beban yang tersebut dapat berupa pemakaian *bandwidth* ataupun beban komputasi dan media penyimpanan.

Saat ini mulai berkembang model hubungan secara *peer to peer* (P2P). Menurut C. Huang, dkk (dalam Garbacki, Paweł) *peer to peer* adalah :

"*Model berbagi sumber daya yang menyediakan sebuah solusi arsitektur yang atraktif untuk aplikasi dengan bandwith terbatas. peer-peer menggunakan bandwith unggah mereka untuk meredistribusikan konten yang telah diunduh sebelumnya, mengurangi kebutuhan dan beban pada peladen (server).*"

Model P2P dapat dimanfaatkan untuk memindahkan beban yang terpusat pada peladen kepada pengguna sehingga implementasi model P2P pada sistem *audio streaming* diharapkan dapat meningkatkan kualitas pemutaran tanpa memperbesar sumberdaya dari sisi peladen. Dalam konteks keterbatasan *bandwith* P2P dapat dimanfaatkan untuk mengurangi kebutuhan bandwith di sisi peladen, karena sesuai dengan sifat jaringan P2P yang dapat membagikan kebutuhan beban *bandwith*.

Salah satu model implementasi *peer to peer* yang saat ini dikembangkan menggunakan metode Block Scheduling. Dalam Block Scheduling konten media dibagi dalam beberapa blok dan setiap *node* dalam jaringan P2P membentuk sebuah hubungan saling berbagi dengan tetangganya (Huang), sehingga dalam mengunduh suatu konten media tidak terpusat pada satu *node* tetapi tersebar sesuai dengan jumlah *node* yang tergabung dalam jaringan P2P.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka rumusan masalah dijabarkan dalam uraian berikut:

1) Bagaimana sistem *peer to peer* metode *Block Schedulling* pada On-demand *Audio Streaming*?
2) Bagaimana penyebaran beban pada pada sistem *peer to peer* metode Block Schedulling ?

1.3 Batasan Masalah

Beberapa hal yang menjadi batasan masalah dalam perancangan dan pembuatan sistem antara lain:

1) sistem dirancang pada Layer Applications pada OSI Layer,
2) tidak menerapkan metode NAT *Traversing* untuk komunikasi dengan *node* dibelakang NAT,
3) proses *decode* dan pemutaran berkas suara menggunakan pustaka Bass.NET,
4) pengerjaan dan pengujian dilakukan di jaringan Laboratorium Informatika dan Komputer Jurusan Teknik Elektro Universitas Brawijaya.

1.4 Tujuan

Tujuan penyusunan skripsi ini adalah

Merancang dan mengimplementasikan sistem *Audio Streaming* dengan menggunakan prinsip *peer to peer* metode *block scheduling*.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh melalui pengerjaan skripsi ini dijelaskan dalam uraian berikut.

a. Bagi penyusun
   1) membangun sistem *peer to peer* dengan metode *block scheduling*.
   2) menerapkan ilmu yang telah diperoleh dari Teknik Elektro Konsentrasi Rekayasa Komputer.
   3) menambah wawasan mengenai metode Block Schedulling.

b. Bagi pengguna
   1) melakukan pemutaran audio lewat jaringan dengan lebih cepat
   2) meningkatkan utilisasi sumber daya komputer.

1.6 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini dijelaskan dalam uraian berikut.

BAB I Pendahuluan

Menjelaskan latar belakang, rumusan masalah, ruang lingkup, tujuan dan sistematika penulisan skripsi.

BAB II Dasar Teori

Menjelaskan kajian pustaka dan dasar teori yang digunakan.

BAB III Metodologi

Menjelaskan metodologi yang digunakan dalam pengerjaan skripsi.

BAB IV Perancangan

Menjelaskan konfigurasi perangkat keras dan desain perangkat lunak.

BAB V Implementasi

Menjelaskan sistem *peer to peer* metode *block scheduling* dan pengembangan program *tracker* dan *peer* sebagai bentuk implementasi sistem.

BAB VI Pengujian

Menjelaskan langkah pengujian sistem dan pembahasan hasil pengujian.

BAB VII Kesimpulan dan Saran

Berisi kesimpulan dan saran.

BAB II
DASAR TEORI

2.1 Streaming

Tren multimedia dalam penggunaan perangkat komputasi khususnya *Personal Computer* (PC) merujuk pada konsumsi dan produksi berbagai jenis bentuk media antara lain teks, gambar, suara, animasi, dan video. Produk multimedia awalnya dikonsumsi dengan cara mengakses berkas yang tersedia dalam media penyimpanan yang tersedia dalam *Personal Computer*. Dengan berkembangnya teknologi jaringan komputer mulai dilakukan pengaksesan berkas-berkas multimedia memanfaatkan hubungan jaringan komputer.

Pengaksesan berkas audio maupun video melalui jaringan dapat dilakukan seperti memperlakukan berkas teks yaitu dengan mengunduh dan membuka; tetapi kebanyakan pengguna lebih suka langsung memutar berkas bersamaan dengan proses mengunduh berlangsung, hal ini biasa disebut *streaming* (Hafeeda: 2). *Te*rdapat tiga pendekatan dalam implementasi *streaming*, yaitu berbasis *Unicast* dan berbasis *Multicast* selain itu juga ada metode implementasi *peer to peer.*

2.1.1 Streaming Berbasis Unicast

Dalam pendekatan ini setiap pengguna yang mengakses akan membuat satu hubungan (*stream*) *unicast* baru. Saluran *unicast* adalah sambungan dari titik ke titik dimana terdapat satu penerima dan satu pengirim dalam saluran itu (Tanenbaum: 17). Implementasi tersentralisasi adalah implementasi berbasis *unicast* yang banyak digunakan.

Streaming tersentralisasi dicapai menggunakan *server* yang hebat dengan sebuah koneksi ber-*bandwitdh* tinggi. Pendekatan ini mudah untuk dibuat dan dimanajemen. Tetapi kekuatiran akan skalabilitas dan reliabilitas adalah hal yang biasa. Masalah reliabilitas terjadi karena hanya ada satu entitas yang menangani setiap permintaan. Sedangkan permasalahan skalabilitas terjadi karena untuk mencukupi kebutuhan *streaming* dengan jumlah pengguna yang bertambah maka harus dilakukan penambahan kemampuan *server* yang sepadan(Hafeeda: 354).

2.1.2 Streaming Berbasis Multicast

Penggunaan pendekatan multicast menghasilkan pemakaian sumberdaya yang lebih baik dalam melayani permintaan beberapa pengguna menggunakan saluran (*stream*) yang sama. Hal ini dapat dilakukan dengan membuat pohon distribusi multicast.

Multicast *streaming* lebih mudah diterapkan pada *live streaming* di mana setiap pengguna tersinkronisasi: setiap pengguna mendapatkan bagian yang sama dari sebuah saluran dalam satu waktu. Untuk *On-Demand streaming* di mana setiap pengguna tidak tersinkronisasi, beberapa metode telah diusulkan. Salah satu ide utama untuk menggunakan pendekatan multicast adalah *patching*. Dalam metode ini pengguna diperbolehkan bergabung dalam sesi multicast yang telah berjalan. Sebagai tambahan, pengguna juga membuat hubungan unicast dengan *server* untuk mendapatkan bagian dari berkas yang telah terlewati dalam sesi multicast. Penggunaan metode *patching* mengharuskan pengguna membuat beberapa saluran dalam satu periode *patching*, berarti pengguna harus memiliki paling tidak dua kali lebih besar dari pada laju *streaming* (*streaming rate*) (Hafeeda: 356).

2.1.3 Streaming Berbasis *peer*-to-*peer*

Selain implementasi berbasis unicast dan multicast yang lebih umum, terdapat juga satu implementasi yang saat ini mulai mendapat perhatian yaitu *peer-to-peer*. Ide kunci dari arsitektur ini adalah pengguna (disebut *peer*) membagikan sebagian sumberdayanya kepada sistem, sehingga kapasitas sistem secara keseluruhan meningkat dan dapat melayani pengguna lebih banyak (Hafeeda: 357).

Dalam sistem *peer-to-peer* setiap permintaan yang dilakukan oleh *peer* dilayani oleh *supplying peer* tertentu. Selanjutnya setelah menerima data, *peer* tersebut menyimpannya pada media penyimpanan lokal sehingga dapat menjadi *suppluing peer* baru untuk *peer* lain (Cheng Kao: 1).

Secara konsep teoritis implementasi *streaming* menggunakan *peer-to-peer* dapat menyelesaikan permasalahan skalabilitas dan reliabilitas yang dihadapi oleh implementasi berbasis *unicast* yang tersentralisasi. Penggunaan beberapa *peer* untuk menangani permintaan pengguna akan membuat sistem tidak bergantung pada satu

entitas. Sedangkan kebutuhan akan penambahan sumberdaya *server* untuk mengakomodasi penambahan pengguna dapat diminimalisir atau bahkan tidak perlu dilakukan.

2.2 Socket

Untuk mengimplementasikan suatu program aplikasi yang berkomunikasi menggunakan jaringan dibutuhkan suatu antarmuka yang menjembatani antara jaringan dan program aplikasi. Di dalam sistem operasi umumnya dijumpai sebuah API (*Applications Programming Interface*) yang memberikan akses yang mudah antara pemrogram ke komunikasi jaringan.

Sesungguhnya pada setiap sistem operasi dibebaskan untuk mendefinisikan API masing-masing, semakin lama beberapa API yang ada mulai didukung secara luas; dan dipindahkan ke sistem operasi lain selain asalnya. Ini yang terjadi pada antarmuka *socket* yang asalnya diberikan distribusi Unix dari *Barkeley* (Larry: 31).

Implementasi antarmuka socket umumnya memiliki pendekatan sesuai tradisi Unix dimana setiap hal dalam sistem dianggap sebagai file. Sehingga secara umum operasi dalam socket dilakukan dengan cara menulis maupun membaca dari suatu *file descriptor.*

Berikut adalah 3 tipe antarmuka socket :

1) *Stream socket* mengijinkan proses untuk berkomunikasi menggunakan TCP. Sebuah *stream socket* menyediakan aliran data dua arah, handal, berurutan, tidak terduplikasi dengan tanpa batasan.
2) *Datagram socket* mengijinkan proses untuk berkomunikasi menggunakan UDP. Sebuah *datagram socket* mendukung aliran data dua arah. Sebuah proses dalam *datagram socket* dapat mendapatkan pesan dalam urutan yang berbeda daripada urutan saat mengirim dan menerima data terduplikai.
3) *Raw socket* memberikan akses ke ICMP. (oracle)

2.3 Algoritma Block Schedulling

peer-to-peer pada dasarnya adalah konsep hubungan langsung antara entitas dalam jaringan komputer di mana posisi keduanya sama yaitu saling melayani dan

dilayani. Namun dalam implementasi *streaming* berbasiskan *peer-to-peer* digunakan prinsip *block scheduling*. Pada *streaming* berbasiskan *peer-to-peer* digunakan *block scheduling* untuk menangani pengiriman *block* dalam sistem secara efisien dan optimal.

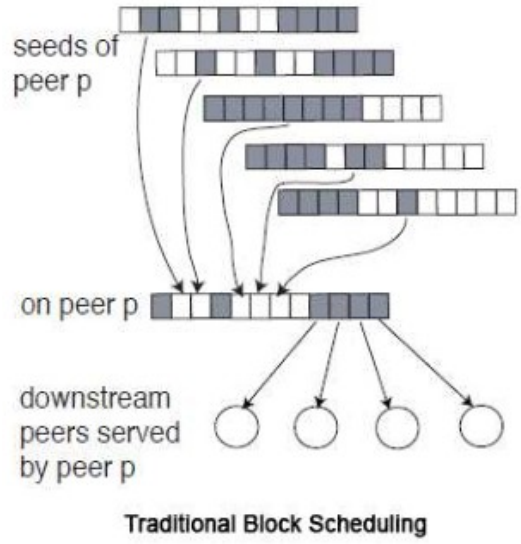
Dalam *block scheduling* berkas media dipartisi menjadi beberapa blok dengan ukuran yang sama, setiap *block* memiliki nomor urutan yang unik. Selanjutnya setiap *peer* memutar blok-blok yang ada sesuai dengan urutan dan dalam kecepatan sesuai dengan laju *streaming*[1] *(Huang: 1).*

Sebuah *peer* mengunduh suatu blok dari sebuah berkas dari *peer* lain melalui sebuah *pull method*[2]. *peer* yang membutuhkan blok mengirim permintaan ke *peer* lain, permintaan itu dikirim dengan beberapa pertimbangan untuk menentukan bagian mana yang akan diunduh terlebih dahulu. Pertimbangan tersebut antara lain :

1) Urutan : Metode ini memilih bagian sesuai dengan urutan pemutaran.
2) Paling jarang : Cara lain adalah dengan memilih bagian paling jarang dalam sistem. Dengan memilih bagian paling jarang dalam sistem memungkinkan memperbaiki kualitas dari sistem seperti dibahas Y. Zhou dalam A Simple Model for Analyzing P2P *Streaming* Protocols.(le Chang: 5)

1 Laju *streaming* : Atau *streaming rate* adalah jumlah data yang di kirim dalam satu satuan waktu.

2 *Pull method :* mengacu pada cara mendapatkan berkas/pesan dengan terlebih dahulu mengirim permintaan. (motorolla mobility llc 2012) https://motorola-global-portal.custhelp.com/app/answers/detail/a_id/15859



Gambar 2.1: Ilustrasi prinsip kerja block scheduling

2.4 MP3

Format berkas Audio MP3 merujuk pada standar MPEG 1/2 *layer* 3 yaitu standar kompresi berkas audio. MP3 merupakan standar internasional untuk berkas audio terkompresi yang dibuat untuk MPEG-1 yang merupakan standar kompresi gambar bergerak. Rasio kompresi berkisar antara 1/10 sampai 1/12 dapat dicapai tanpa penurunan kualitas (Sdcard:2013).

2.4.1 Struktur data MP3

Berkas suara MP3 merupakan berkas berbentuk biner, dimana strukturnya secara umum secara berurutan adalah informasi Tag V2, frame-frame dan informasi Tag V1. Informasi Tag V2 dan Tag V1 adalah informasi tentang berkas suara yang tersimpan, secara umum berisi informasi tentang judul, artis, tahun dan genre dari sebuah berkas audio (DanONeill). Sedangkan *frame* adalah adalah representasi digital dari potongan audio berdurasi 0,026 detik.

Suatu *frame* MP3 terdiri dari 4 byte informasi dan representasi digital dari audio. 4 byte informasi tersebut terdiri :

1) 11 bit *Frame* synchronizer yang merupakan tanda awal dari suatu frame.
2) 2 bit informasi versi MPEG.
3) 2 bit informasi layer.
4) 1 bit informasi ada atau tidak proteksi CRC[3].
5) 4 bit informasi laju bit.
6) 2 bit informasi laju pencuplikan.
7) 1 bit padding.
8) 1 bit private bit yang dapat dimanfaatkan untuk informasi tambahan sesuai keinginan.
9) 2 bit infomasi *channel* yang berisi tentang jenis kanal yang digunakan berkas audio.
10) 2 bit *mode extension* yang memberi informasi tambahan bila kanal yang digunakan adalah *join stereo*.
11) 1 bit informasi tentang ada atau tidak hakcipta pada berkas MP3.
12) 1 bit informasi tentang orisinilitas berkas.
13) 2 bit informasi keberadaan penguatan frekuensi.

Secara digital suatu blok *frame* akan memiliki panjang bit yang berbeda-beda walaupun secara audio durasinya tetap 0,026 detik. Panjang *frame* ditentukan oleh laju bit[4], laju pencuplikan[5] dan 1 bit *padding*. Panjang bit suatu *frame* berkas dapat dihidung dengan rumus :

Panjang Bit = (144 x laju bit / laju pencuplikan) + *Padding*

3 CRC : kependekan dari *cyclic redundancy check* yaitu skema pemeriksaan galat yang memanfaatkan bit paritas. http://www.digikey.com/us/en/techzone/wireless/resources/glossary.html

4 Laju bit : Atau *bitrate* dalam bahasa inggris adalah banyak bit dalam satu satuan waktu, direpresentasikan dalam satuan bps atau bit per second.

5 Laju pencuplikan : atau *sample rate* dalam bahasa inggris adalah banyaknya pencuplikan dalam satu satuan waktu, biasa direpresentasikan dalam satuan Hz.

BAB III
METODE PENELITIAN

Dalam penyusunan skripsi ini, dirancang suatu sistem jaringan *peer to peer* dengan metode *block scheduling* untuk melakukan *Audio Streaming*. Metode penelitian yang digunakan pada penyusunan skripsi ini dijelaskan dalam uraian berikut.

3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

1. *audio streaming*,
2. *peer to peer*,
3. *block scheduling*,
4. pemrograman socket,
5. memainkan berkas audio.

3.2 Pengujian Sistem

Pada tahap ini dilakukan pengujian terhadap hasil implementasi perangkat lunak yang ditujukan untuk mengukur performansi dari sistem *peer to peer* yang dibangun. Beberapa hal yang digunakan untuk mengukur kualitas pemutaran berkas adalah sebagai berikut :

1. *Total delay per playback* di ukur dengan mencatat berapa lama penundaan (*delay*) yang terjadi dalam pemutaran suatu berkas suara.
2. Total ignored transmit session per playback di ukur dengan mencatat berapa kali suatu sesi pengiriman dibatalkan karena *timeout*.
3. Pembagian beban pada saat streaming dijalankan.
4. *Throughput* jaringan pengujian.
5. Beda antara berkas yang diunduh dengan berkas asli

3.3 Perancangan dan Implementasi

Pada tahap ini dilakukan analisis kebutuhan perangkat lunak, dari hasil analisis tersebut akan dibangun solusi permasalahan yang memenuhi kebutuhan-kebutuhan

perangkat lunak yang akan digunakan sebagai dasar untuk melakukan implementasi. Adapun tahap-tahap dalam perancangan adalah sebagai berikut :

1. Analisis kebutuhan perangkat lunak
2. Membuat alat bantu perancangan berupa DFD (Data Flow Diagram)
3. Melakukan perancangan perangkat lunak

Selanjutnya dilakukan implementasi berdasarkan analisis dan perancangan yang telah dilakukan. Aplikasi yang telah dirancang dibangun menggunakan beberapa bahasa pemograman yaitu Javascript dan C#.

3.4 Kesimpulan dan Saran

Pada tahap ini, kesimpulan dan analisis dari pengujian dipaparkan. Tahap selanjutnya adalah pembuatan saran untuk perbaikan dan pengembangan penelitian selanjutnya.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini membahas mengenai perancangan perangkat lunak untuk mengimplementasikan *audio Streaming peer to peer* metode *block scheduling*. Dalam perancangan akan dilakukan langkah-langkah berikut :

4.1 Analisis Kebutuhan Perangkat Lunak

4.1.1 Analisis kerja *peer to peer* Block Schedulling

4.1.2 Abstraksi Sistem

4.1.3 Diagram Konteks Sistem dan Alur Komunikasi

4.1.5 Analisis Kebutuhan Perangkat Lunak

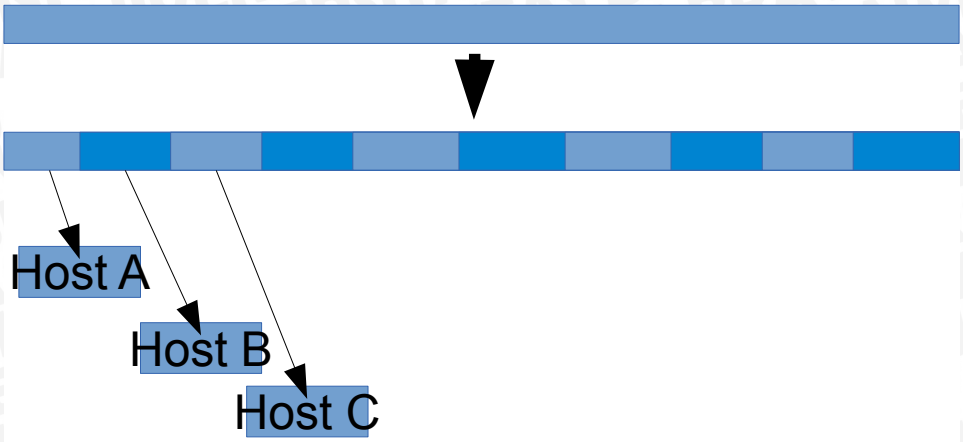
4.1.6 Analisis Kebutuhan Perangkat Keras

4.1 Analisis Kebutuhan Perangkat Lunak

Untuk membuat perangkat lunak yang digunakan untuk implementasi sistem *Streaming Audio* dengan *peer to peer* metode *block scheduling* dibutuhkan analisis terlebih dahulu. Analisis kebutuhan perangkat adalah proses yang digunakan untuk mendapat, menganalisis dan memvalidasi kebutuhan-kebutuhan sistem (Sommervielle, 2001).

4.1.1 Analisis kerja *peer to peer* Block Schedulling

Prinsip dasar dari *block Schedulling* adalah memecah proses pengunduhan berkas menjadi beberapa proses yang mengunduh ke beberapa *host.* Dengan memecah proses pengunduhan menyebabkan beban kerja dapat terbagi.

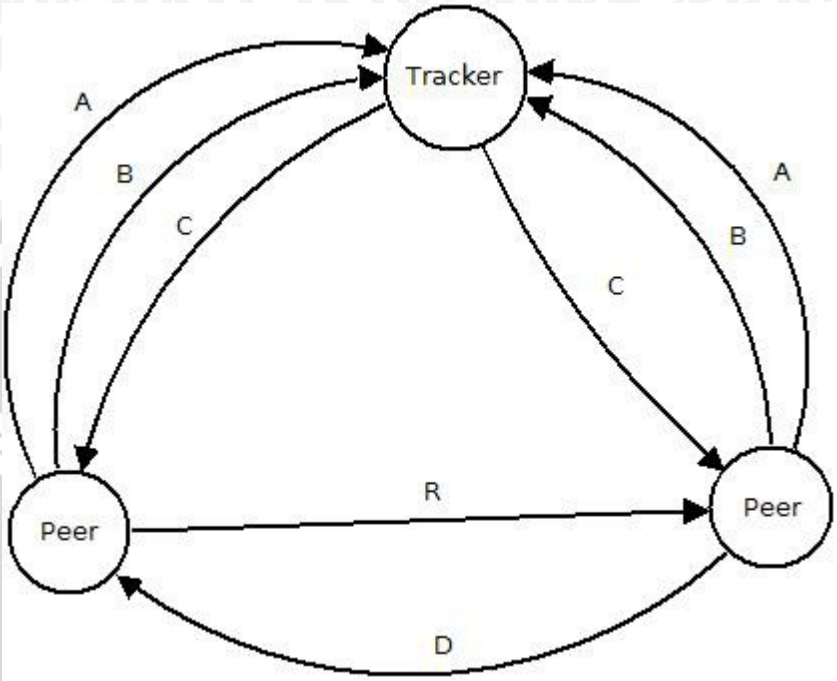


Gambar 4.1 : Prinsip kerja peer to peer Block Schedulling

Dari gambar 4.1 dapat dilihat bagaimana suatu berkas utuh dibagi menjadi beberapa blok untuk selanjutnya dilakukan pengunduhan dari beberapa *host* yang ada. Dalam sistem ini diperlukan lebih dari satu *host* yang tersedia. Selain melakukan permintaan terhadap suatu berkas, *host* juga memiliki tanggung jawab melayani permintaan berkas dari *host* lain bila berkas tersedia dalam *host* tersebut. Sehingga terbentuk jaring-jaring yang terdiri dari *host* yang saling melayani, *host* kemudian disebut dengan *peer*.

Selain *peer* yang melayani dan dilayani juga dibutuhkan mekanisme untuk menjaga dan memelihara informasi tentang berkas yang tersedia dalam jaringan antar *peer*, untuk menjaga dan memelihara informasi tentang berkas yang tersedia dalam jaringan maka dibutuhkan satu *host* bertindak sesuai fungsi tersebut yang disebut *tracker*. Informasi tentang berkas antara lain berisi daftar berkas yang tersedia, *host* yang tersedia untuk masing-masing berkas dan ukuran berkas.

4.1.2 Abstraksi Sistem



Gambar 4.2: Abstraksi sistem audio streaming metode peer to peer block scheduling

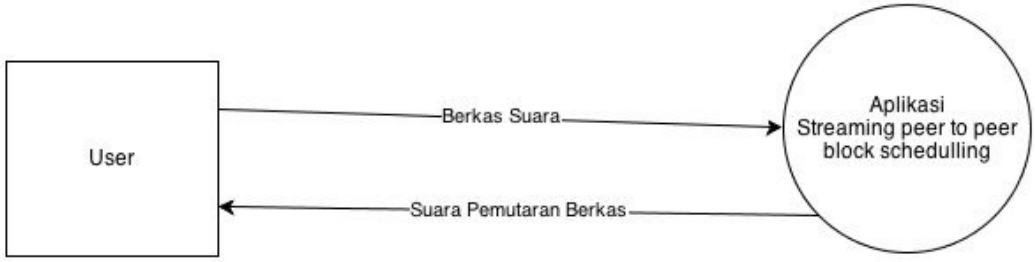
Dari analisis sebelumnya telah disimpulkan bahwa diperlukan dua jenis *host* yaitu *peer* dan *tracker*. Sistem jaringan *peer to peer* yang digunakan nantinya akan memiliki struktur dan cara kerja logis seperti digambarkan pada ilustrasi pada gambar 4.2. Langkah kerja sistem menurut gambar 4.2 adalah sebagai berikut :

1) Terdapat dua jenis *host* yaitu *tracker* dan *peer, tracker* bertugas untuk mengkonsolidasi *peer* yang ada, sehingga dapat terjadi hubungan antar *peer*. Dalam sistem terdapat satu *host tracker* dengan beberapa *peer*.
2) *Tracker* memiliki daftar yang berisi informasi berkas yang tersedia dalam jaringan dan *peer* yang aktif dalam dalam jaringan.
3) *Peer* memiliki informasi berkas yang tersedia dalam dirinya (lokal).
4) *Peer* mengirim pesan (garis A) untuk mengabarkan kepada *tracker* bahwa dirinya dalam keadaan siap .

5) *Peer* memperbarui informasi tentang berkas yang dimiliki kepada *tracker* (garis B).
6) *Peer* mendapatkan informasi tentang berkas yang tersedia dalam jaringan dari *tracker* (garis C).
7) Dengan memanfaatkan daftar yang tersedia, *peer* yang ada dapat berkomunikasi untuk saling berbagi berkas audio.

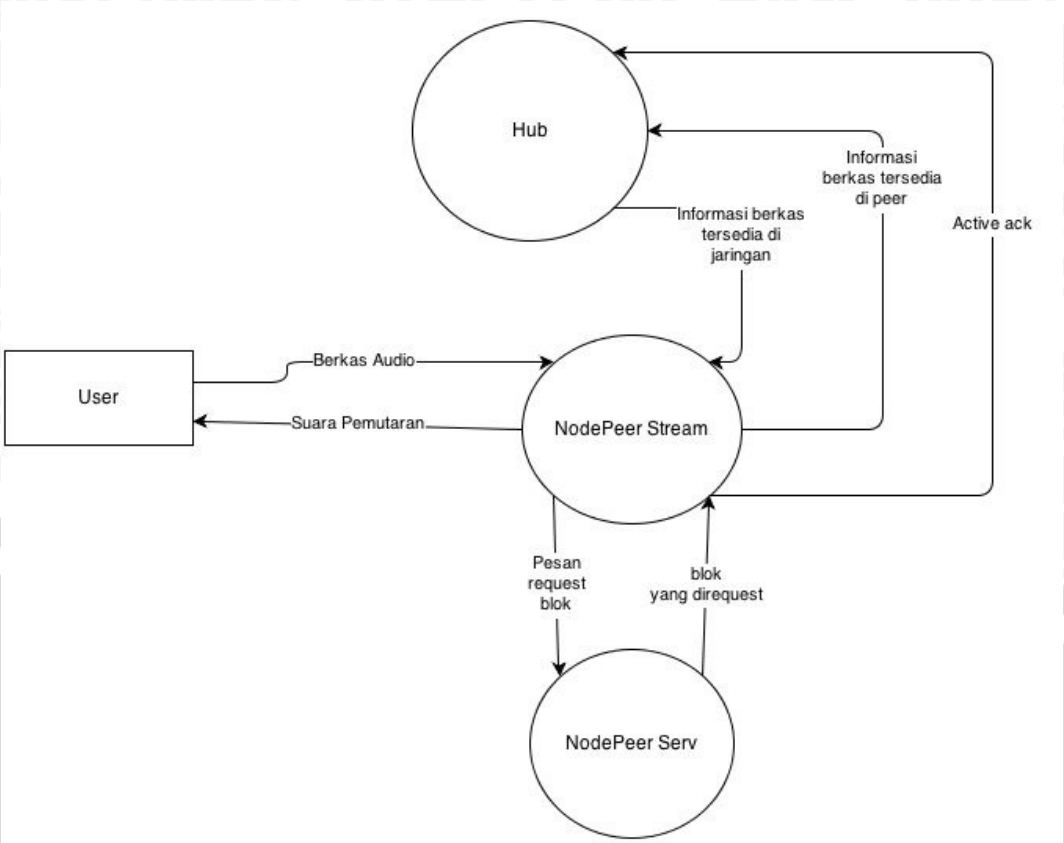
4.1.3 Diagram Konteks Sistem dan Alur Komunikasi

Untuk menggambarkan sistem secara umum dan entitas-entitas yang terlibat di dalamnya, perlu dibuat diagram konteks. Diagram konteks atau disebut juga DFD0 merupakan diagram yang menggambarkan aliran data sistem secara keseluruhan. Diagram konteks Level 0 sistem *audio streaming peer to peer metode block scheduling* dapat dilihat pada gambar 4.3.



Gambar 4.3 Diagram Konteks level 0 Sistem peer to peer

Diagram konteks level 0 sistem mempunyai dua komponen pembentuk yaitu *User* dan Sistem *peer to peer*. Sistem *peer to peer* berfungsi sebagai sarana pengguna untuk melakukan *streaming* berkas audio. Untuk melakukan *streaming* maka salah satu *user* perlu menyediakan berkas audio terlebih dahulu sehingga tersedia untuk dilakukan *streaming* oleh *user lain*.



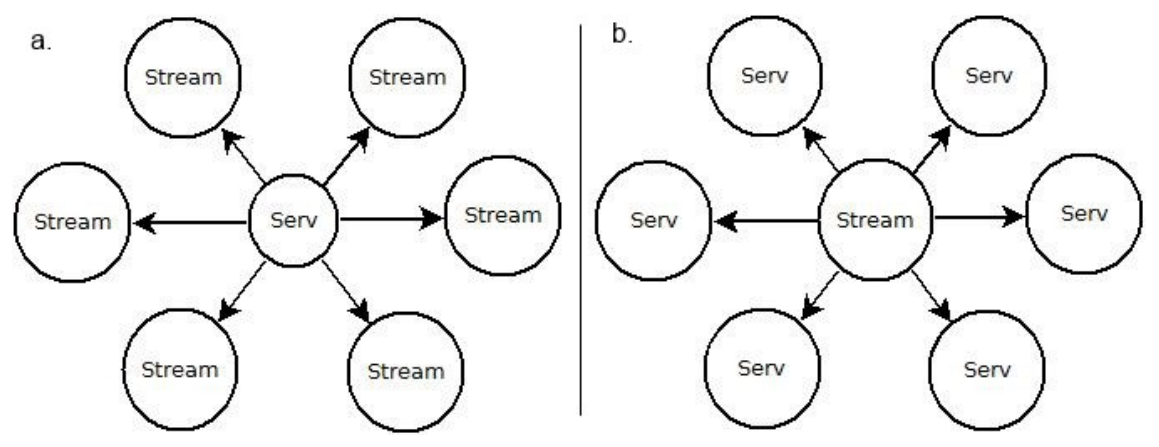
Gambar 4.4 Diagram Konteks Level 1 Sistem peer to peer

Selanjutnya dari Abstraksi Sistem maupun diagram konteks Level 0 maka diperlukan 3 komponen utama yang diperlukan, yaitu :

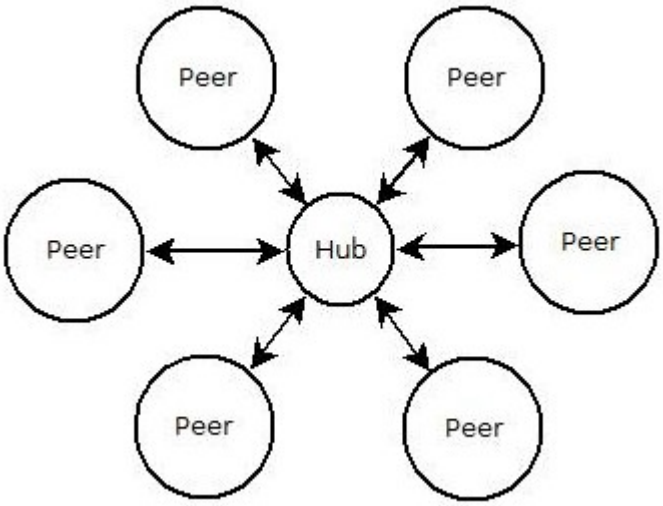
1. Komponen yang bertindak sebagai *tracker.*
2. Komponen yang melayani permintaan data dari *peer* lain.
3. Komponen yang melakukan permintaan data pada *peer* lain.

Dari komponen tersebut disusun Diagram konteks level 1 seperti pada gambar 4.4. Pada Diagram konteks level 1 terdapat 3 komponen yaitu Hub yang bertindak sebagai *tracker*, NodePeer Stream yang melakukan permintaan *streaming* dan NodePeer Serv yang melayani permintaan *streaming*.

Hub menghimpun dan menyediakan informasi tentang berkas yang tersedia dalam jaringan dan *peer* lain yang aktif. NodePeer Stream melakukan proses *streaming* audio berdasarkan informasi berkas yang didapat dari Hub, sedangkan NodePeer Serv menerima dan melayani permintaan berkas. NodePeer Serv dan NodePeer Stream menjadi satu kesatuan sebagai *peer.*



Gambar 4.5. a. NodePeer Serv melayani beberapa NodePeer Stream. b. NodePeer Stream melayani beberapa NodePeer Serv



Gambar 4.6 Hub berkomunikasi dengan beberapa peer dalam satu waktu

Interaksi NodePeer Serv dan NodePeer Stream antar *peer* ditunjukan pada Gambar 4.5. Pada Gambar 4.5 a satu NodePeer Serv melayani permintaan dari beberapa NodePeer Stream dalam satu waktu, sedangkan pada Gambar 4.5 b satu NodePeer Stream melakukan permintaan ke beberapa NodePeer Serv dalam satu waktu. Begitu pula Hub berkomunikasi dengan beberapa *peer* dalam satu waktu seperti ditunjukan pada Gambar 4.6.

4.1.4 Analisis Kebutuhan Perangkat Lunak

Untuk membuat 3 (tiga) komponen sistem yang telah dijabarkan sebelumnya yaitu Hub, NodePeer Stream dan NodePeer Serv dibutuhkan bahasa pemrograman yang memenuhi kebutuhan berikut :

1. Kemampuan *multithreading*, diperlukan untuk melakukan permintaan dan peladenan lebih dari satu dalam satu waktu.
2. Dukungan terhadap TCP socket *programming.*
3. Dukungan terhadap penggunaan basis data.
4. Dukungan terhadap operasi pembacaan dan penulisan berkas.
5. Dukungan terhadap pemutaran berkas suara.

NodePeer Stream dan NodePeer Serv berada pada satu program aplikasi sedangkan Hub merupakan program aplikasi berbeda, sehingga dimungkinkan untuk diimplementasikan dengan bahasa pemrograman maupun lingkungan bekerja yang berbeda. Hub menggunakan Node.Js sebagai *platform* implementasi. Selain mendukung 4 syarat utama untuk melakukan Implementasi sistem, Node.Js dapat digunakan untuk membuat program aplikasi dengan lebih cepat dan mudah karena menggunakan Javascript sebagai bahasa pemrogramannya. Selain itu juga digunakan pustaka basis data SQLite untuk melakukan operasi basis data. Node.Js dan komponen pendukung lainnya dapat dijalankan pada beberapa sistem operasi, maka dipilih sistem operasi Ubuntu untuk menjalankan Hub.

NodePeer dikembangkan menggunakan bahasa pemrograman C# dengan memanfaatkan pustaka .NET yang menyediakan dukungan komunkasi TCP lewat socket API dan operasi berkas. Untuk basis data menggunakan basis data berorientasi objek bernama db4object. Untuk melakukan pemutaran berkas suara digunakan pustaka Bass. Untuk menjalankan program aplikasi *peer* diperlukan sistem operasi Windows dengan .Net Framework.

4.1.5 Analisis Kebutuhan Perangkat Keras

Berdasarkan kebutuhan perangkat lunak yang digunakan, didapatkan kebutuhan minimum perangkat keras untuk menjalankan sebagai berikut :

1. Perangkat keras NodePeer:
    a) Prosesor sekelas pentium 4 1 Ghz
    b) Hardisk 10 Gb
    c) RAM 512 MB
    d) Sound Card
    e) Ethernet 100 Mbps
2. Perangkat keras Hub
    a) Prosesor sekelas pentium 4 1 Ghz
    b) Hardisk 5 GB
    c) RAM 256 MB
    d) Ethernet 100 Mbps

4.1.6 Perancangan Perangkat Lunak

4.1.6.1 Perancangan Perangkat Lunak Hub

Merujuk pada diagram konteks level 1, komponen Hub memiliki beberapa fungsi yaitu :

1. Menerima Informasi berkas yang tersedia dari NodePeer.
2. Menerima Informasi status keaktifan NodePeer.
3. Memberikan Informasi berkas yang tersedia pada tiap NodePeer kepada NodePeer.

Komunikasi yang dilakukan antara NodePeer dan Hub selalu diawali oleh NodePeer. Perintah-perintah nantinya dikirim oleh NodePeer, dimana setiap perintah merupakan implementasi dari 3 fungsi utama dari hub. Perintah yang dikirim merupakan teks dengan format sebagai berikut :

[Perintah]::[Keterangan1]::[Keterangan2]

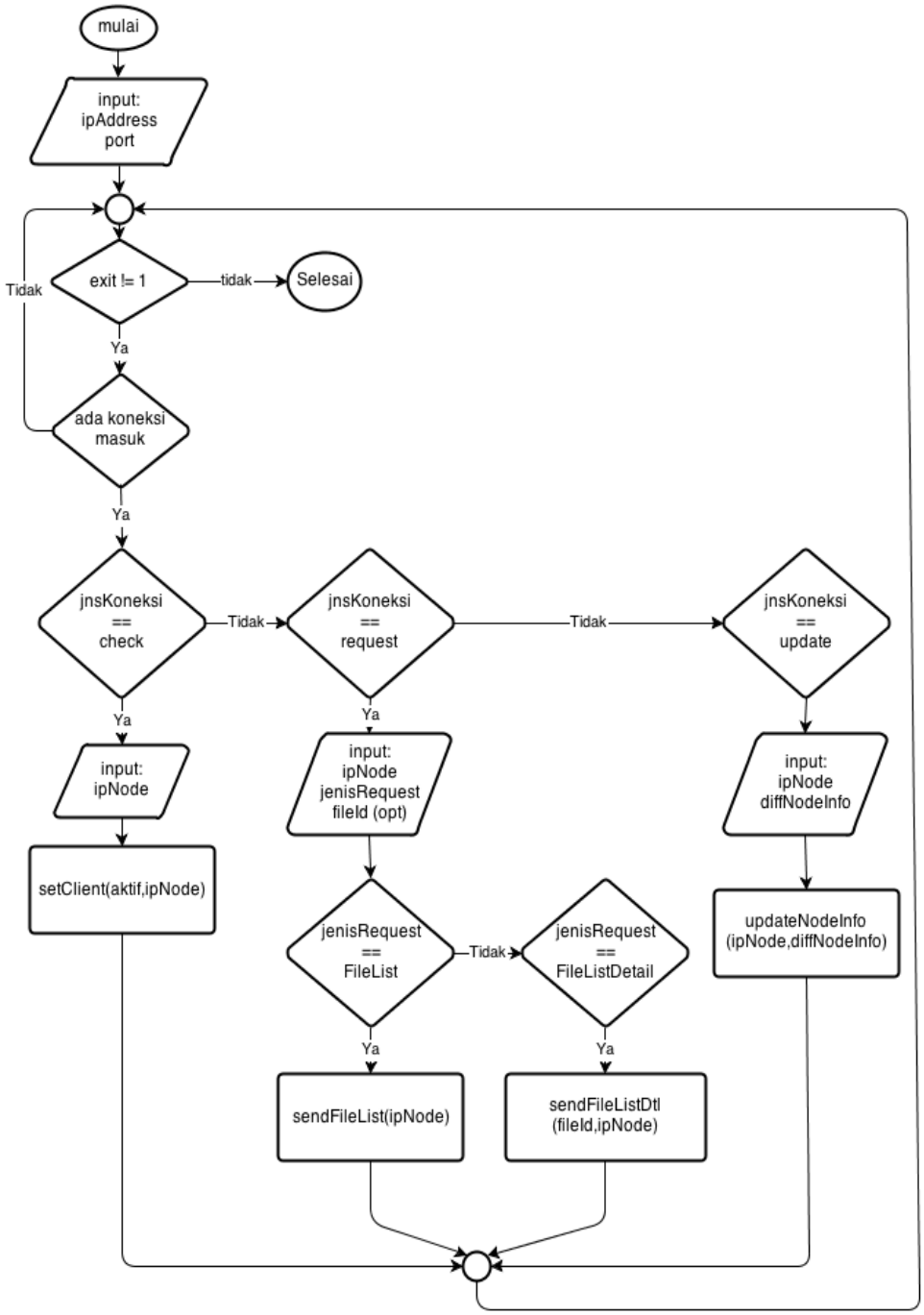
Pada Gambar 4.7 dapat terlihat bagaimana proses hub dalam menanggapi perintah yang masuk. Menggunakan rutin kondisional untuk menentukan aksi apa yang dilakukan untuk setiap perintah yang datang dari NodePeer.

Sedangkan informasi balasan yang dikirim oleh Hub kepada NodePeer dalam format JSON. JSON dipilih karena selain merupakan format data bawaan dari Javascript juga tersedia *parser* yang mudah digunakan pada lingkungan .NET.

Sesuai gambar 4.6 terlihat bahwa Hub harus dapat melayani benyak komunikasi dengan NodePeer dalam satu waktu, sehingga dibutuhkan kemampuan untuk menangani perintah dalam satu waktu. Node.Js memiliki kemampuan penanganan fungsi tak serempak (*asynchronous*) yang baik, asalkan setiap fungsi yang dibuat adalah fungsi asynchronous. Berikut contoh fungsi *asynchronous* pada NodeJs :

```
var server = net.createServer(function (socket) {
  socket.write('Echo server\r\n');
  socket.pipe(socket);
});
```

Kode di atas merupakan socket *server* sederhana pada NodeJs, setiap terdapat koneksi baru dari *client* maka Node.Js akan membuat alokasi pada *Heap Memory* sendiri sehingga *server* dapat menangani banyak *client* dalam satu waktu.



Gambar 4.7 Diagram alir pada Hub saat menangani perintah

4.1.6.2 Perancangan Perangkat Lunak NodePeer

Komponen NodePeer memiliki dua komponen utama yaitu NodePeer Stream dan NodePeer Serv. Keduanya memiliki fungsi utama sebagai berikut :

Untuk NodePeer Stream :

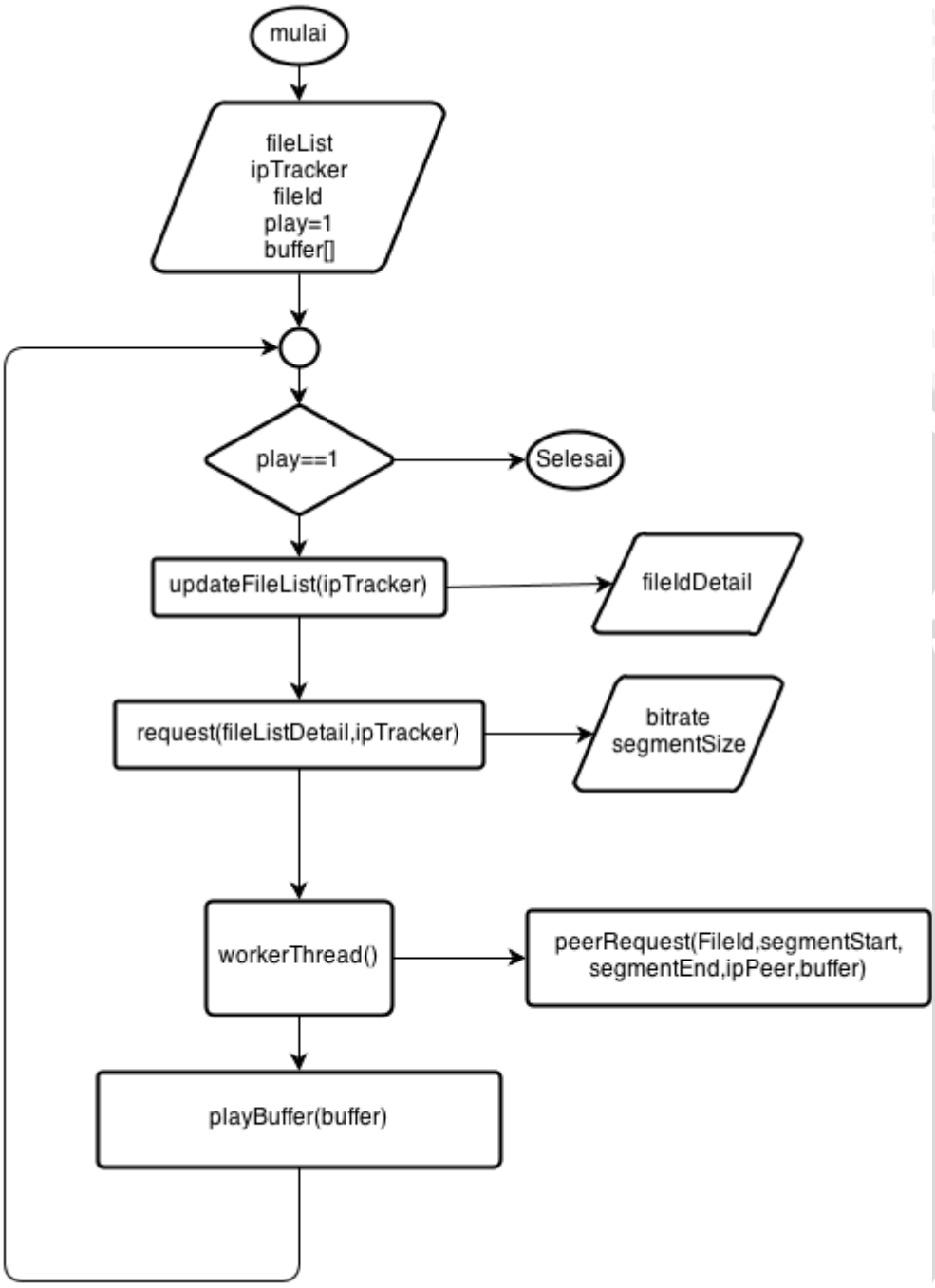
1. Melakukan komunikasi dengan NodePeer Lain dan hub.
2. Melakukan pengunduhan berkas lewat TCP socket dari NodePeer lain.
3. Melakukan penyusunan berkas yang diunduh.
4. Memainkan berkas audio.
5. Melakukan permintaan lebih dari satu dalam satu waktu.

Untuk NodePeer Serv :

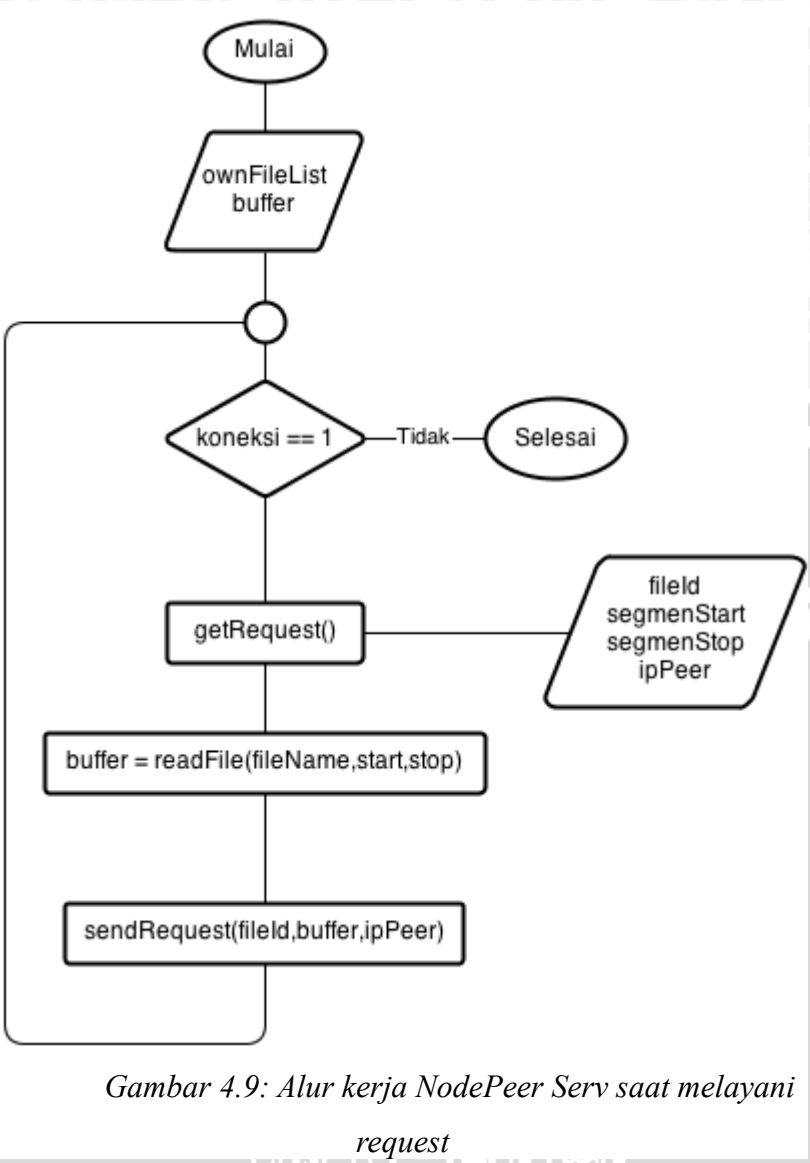
1. Melakukan komunikasi dengan NodePeer Lain.
2. Melakukan pengiriman berkas lewat TCP socket ke NodePeer lain.
3. Melayani permintaan dalam satu waktu.

Prinsip kerja NodePeer Stream digambarkan pada Gambar 4.8. Dari gambar tersebut terlihat alur kerja yang dilakukan oleh NodePeer Stream saat melakukan proses s*treaming*. Langkah pertama yang dilakukan adalah melakukan komunikasi dengan Hub untuk mendapatkan informasi tentang NodePeer lain yang memiliki berkas yang akan di-*streaming*, selanjutnya melakukan permintaan blok berkas kepada *peer* lain yang memiliki berkas yang dimaksud untuk selanjutnya ditulis ke buffer dan dimainkan oleh pemutar berkas.

Sedangkan prinsip kerja NodePeer Serv digambarkan pada Gambar 4.9, dari gambar tersebut terlihat apa yang dilakukan NodePeer Serv saat ada permintaan datang. Setelah menerima pesan, NodePeer Serv mengartikan pesan tersebut lalu kemudian mengirim berkas sesuai dengan permintaan.



Gambar 4.8 Alur kerja NodePeer Stream saat memutar audio



Gambar 4.9: Alur kerja NodePeer Serv saat melayani request

Fungsi PeerRequest pada NodePeer Stream (lihat Gambar 4.8) merupakan fungsi yang menginisiasi permintaan secara *asynchronous* dan *multithreading* , NodePeer tidak perlu menunggu permintaan blok oleh NodePeer Stream selesai dilakukan dan dapat melakukan beberapa permintaan dalam satu waktu. Sedangkan alur kerja pada Gambar 4.9 dilakukan dalam fungsi *asynchronous* dan terpisah dari thread utama untuk setiap permintaan baru dari *peer* lain, sehingga memungkinkan untuk melayani lebih dari satu permintaan dalam satu waktu.

4.2 Implementasi

4.2.1 Implementasi Desain Perangkat Lunak

Implementasi dari desain *audio streaming* menggunakan *peer to peer* metode *Block Scheduling* memerperlukan dua perangkat lunak yang berbeda. Perangkat lunak pertama sebagai *tracker* untuk kemudian disebut Hub dan yang kedua adalah perangkat lunak untuk melakukan *streaming audio* selanjutnya disebut NodePeer.

Hub berguna untuk mengumpulkan dan merawat data yang digunakan NodePeer untuk mengetahui daftar berkas dan alamat *peer* yang memiliki berkas tersebut. NodePeer berfungsi untuk mengunduh potongan-potongan berkas dari beberapa komputer yang tersedia untuk selanjutnya memainkan berkas tersebut, selain itu berfungsi juga sebagai peladen permintaan berkas bagi NodePeer lain.

4.2.1.1 Implementasi desain Hub

4.2.1.1.1 Daftar Perintah dan Fungsinya

Pada dasarnya Hub menerima perintah dari NodePeer dan membalasnya sesuai dengan perintah yang diterima. Komunikasi dilakukan lewat *socket* TCP berupa pesan teks polos (*plain text*). Berikut ini adalah daftar perintah yang ada beserta keterangan fungsinya :

1. FL

   Penggunaan : FL

   Fungsi : Digunakan untuk mendapat kan daftar file (FileList) yang tersedia di dalam jaringan. Informasi yang dikembalikan berisi Id, Nama Berkas, Laju bit (*bitrate*), Laju Pencuplikan (*Sample Rate*), Ukurang file.

   Contoh keluaran :

   ```
   [{"id_file":1,"nama":"Misread.mp3","bitrate":"192",
   "samplerate":"44100","size":5011748}]
   ```

2. FD

   Penggunaan : FD;;ID_FILE

Fungsi : Digunakan untuk mendapatkan daftar *host* yang memiliki berkas beserta besar berkas yang tersedia tiap *host*. Informasi yang dikembalikan antara adalah Nama Berkas, Alamat IP, Besar blok yang dimiliki.

Contoh Keluaran :

```
[{"nama":"Misread.mp3","ip":"192.168.0.8","block_av
ail":7917},
{"nama":"Misread.mp3","ip":"192.168.0.7","block_ava
il":8018},
{"nama":"Misread.mp3","ip":"192.168.0.4","block_ava
il":8018},
{"nama":"Misread.mp3","ip":"192.168.0.5","block_ava
il":8018},
{"nama":"Misread.mp3","ip":"192.168.0.6","block_ava
il":8018},
{"nama":"Misread.mp3","ip":"192.168.0.9","block_ava
il":8018}]
```

3. NA

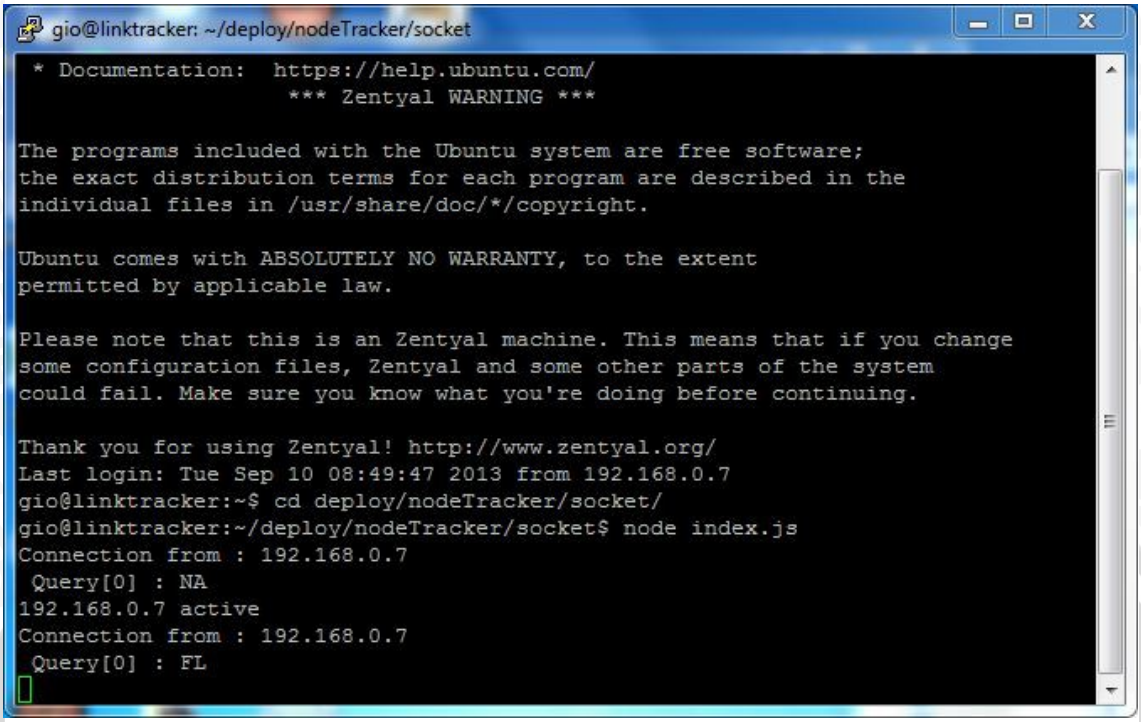
   Penggunaan : NA

   Fungsi : Memberi tahu bahwa *host* dalam kondisi aktif.

4. UP

   Penggunaan : UP;;ID_FILE;;Jumlah Blok yang tersedia

   Fungsi : Digunakan *host* untuk memberitahu Hub tentang besar blok yang dimiliki.



Gambar 4.10 Tampilan SSH yang sedang membuka sesi remote untuk menjalankan hub

4.2.1.1.2 Koneksi dan operasi basis data menggunakan SQLite

Untuk melakukan penyimpanan informasi digunakan basis data SQLite. Pada Node.Js digunakan driver sqlite bernama sqlite3. Berikut kode yang digunakan untuk menginisiasi koneksi ke SQLite :

```
var sqlite3 = require("sqlite3");
var db = new sqlite3.Database('./trackerdb',
sqlite3.OPEN_READWRITE);
```

Berbeda dengan model pemrograman struktural nodejs mensyaratkan semua fungsi untuk bekerja dalam mode *asynchronous*, sehingga memiliki struktur fungsi yang berbeda untuk menjalankan operasi basis data.

Berikut contoh eksekusi perintah SQL pada nodejs :

```
db.serialize(function () {
db.all("SELECT * FROM host WHERE ip='" + nodeId + "'", function
(err, row) { })
});
```

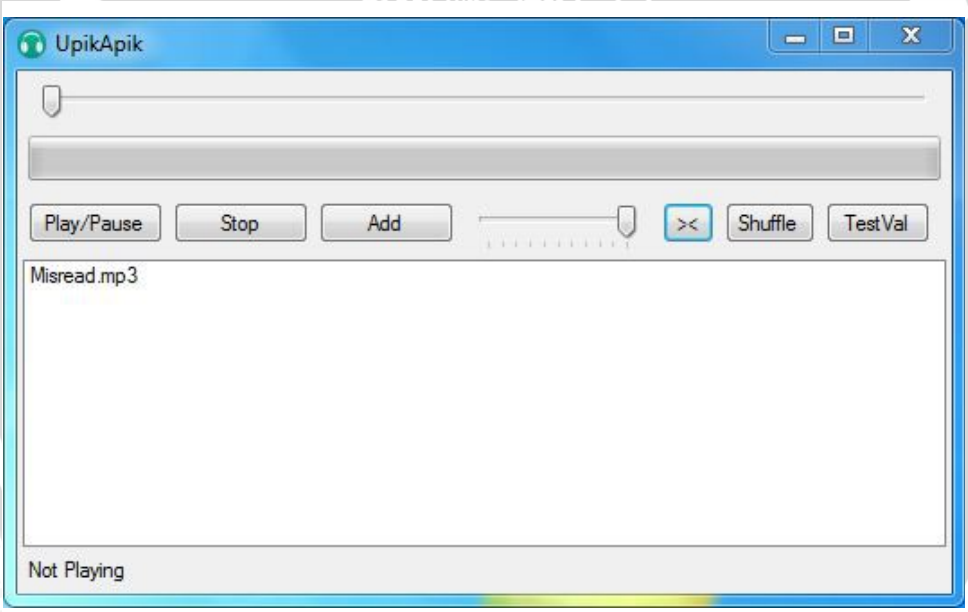
4.2.1.1.3 Penanganan Banyak Koneksi Dalam Satu Waktu

Untuk memulai penanganan *asynchronous* pada Node.Js menggunakan mekanisme *event driven*, yaitu terdapat kejadian (*event*) yang memicu dijalankannya satu fungsi. Pada penanganan koneksi *socket* digunakan *event* ON DATA yang berarti suatu fungsi dijalankan setiap terdapat koneksi masuk (data masuk). Seperti terlihat pada kode dibawah ini :

```
socket.on("data", function (data) {
   [seleksi kondisi] // lihat Gambar 4.7
}
```

Pada kode diatas terlihat penanganan seleksi kondisi perintah data yang masuk di-lakukan secara *asynchronous*.

4.2.1.2 Implementasi Desain NodePeer



Gambar 4.11 Tampilan aplikasi NodePeer

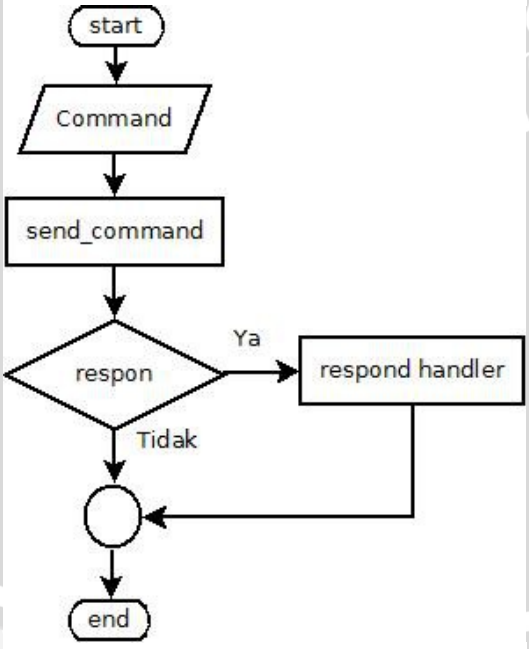
NodePeer terdiri dari empat komponen yang memiliki fungsi masing-masing, yaitu :

1. RedToHub

Komponenen RedToHub menghubungkan antara NodePeer dengan Hub. RedToHub menjaga informasi antara lain daftar berkas yang tersedia di jaringan *peer to peer* dan daftar *host* yang siap melayani permintaan suatu file. Informasi tersebut di dapat dari Hub.

```
public void command(string command)
{
    ThreadPool.QueueUserWorkItem(new WaitCallback(connectionHandler));
    this.comToHub = command;
    parsedCommand = comToHub.Split(';');
}
```

Gambar 4.12 : Code yang bertanggung jawab mengirim perintah dan menerima balasan dari Hub



Gambar 4.13 Alur pengiriman pesan ke Hub

Pada RedToHub fungsi command bertugas untuk mengirim dan melakukan penanganan informasi, baik informasi yang dikirim maupun informasi yang diterima dari Hub. Pada Gambar 4.12 terlihat bahwa perintah akan dikirimkan kepada Hub melalui *thread* baru, bila terdapat respon dari hub maka akan diberikan kepada *respond handler* seperti terlihat pada Gambar 4.13.

2. AsynchRedServ

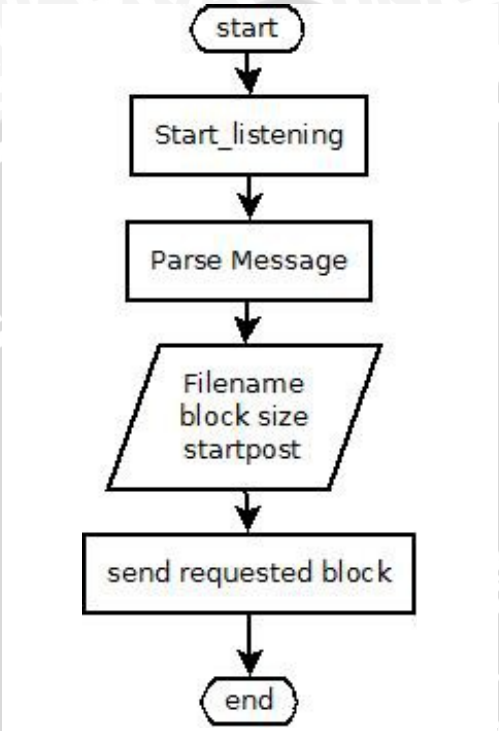
Komponen NodePeer Serv diimplementasikan sebagai class bernama AsynchRedServ. Berfungsi untuk melayani permintaan blok berkas suara dari NodePeer lain. Ketika suatu NodePeer mengirimkan permintaan blok berkas maka AsynchRedServ akan melayani permintaan itu.

```
public void startListening()
{
    try
    {
        server = new TcpListener(ipServer);
        server.Server.ReceiveTimeout = 8000;
        server.Start();
        while (enable)
        {
            allDone.Reset();
            server.BeginAcceptTcpClient(acceptCallback, server);
            allDone.WaitOne();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
}
```

Gambar 4.14 Fungsi startListening

Pada AsynchRedServ fungsi startListening merupakan pusat dari aktivitas yang ada dalam proses melayani permintaan *block* file dari *peer* lain. Gambar 4.14 memberikan gambaran umum bagaimana fungsi startListening bekerja. Setiap terdapat koneksi baru dari NodePeer lain, maka akan dibuat satu *thread* baru lewat yang men-

jalankan fungsi acceptCallback. *Callback* inilah yang merupakan implementasi dari rancangan *asynchronous* dan *multithreading*.



Gambar 4.15 Alur peladenan permintaan blok berkas

Pada fungsi *callback* acceptCallback pesan yang masuk akan di-*parser* untuk mendapatkan informasi permintaan. Informasi yang didapat adalah nama berkas, posisi bit pada berkas dan besar blok. Secara garis besar alur peladenan dapat dilihat pada Gambar 4.15.

3. AsynchRedStream

Komponen NodePeer Stream diimplementasikan sebagai *class* bernama AsynchRedStream. Berfungsi sebagai pengatur proses pengunduhan berkas dari berbagai *host* nodePeer yang siap untuk mendapat permintaan. AsynchRedStream memastikan berkas yang diunduh untuk berjalan dengan lancar sehingga tidak terjadi penundaan dalam proses *streaming*.

```
public void startStream(int id_file, file_list fileinfo)
{
    this.fileinfo = fileinfo;
    this.id_file = id_file;
    string filename = getFilename();
    blocksize = getBlockSize();
    int filesize = getFileSize();
    bassBuffer = new byte[filesize];

    file = new FileStream(AppDomain.CurrentDomain.BaseDirectory + "music/"
        + filename, FileMode.OpenOrCreate, FileAccess.Write,FileShare.ReadWrite);

    enable = true;
    enStream = true;
    createStartPostQueue(blocksize, filesize);
    startTimer = new Timer(
        x => { startTimerCallback(filename, blocksize, filesize); }, null, 0, 500);
}
```

Gambar 4.16 Fungsi startStream

Pada AsynchRedStream terdapat fungsi startStream (Gambar 4.16) yang merupakan rutin yang mengatur proses *streaming* berkas, mulai dari menentukan jumlah blok dalam satu berkas hingga menyusun tiap blok yang didapat dari *peer* lain. Fungsi startStream akan menjalankan fungsi startTimerCallback yang merupakan implementasi desain asynchronous dan multithreading dari NodePeer Stream.

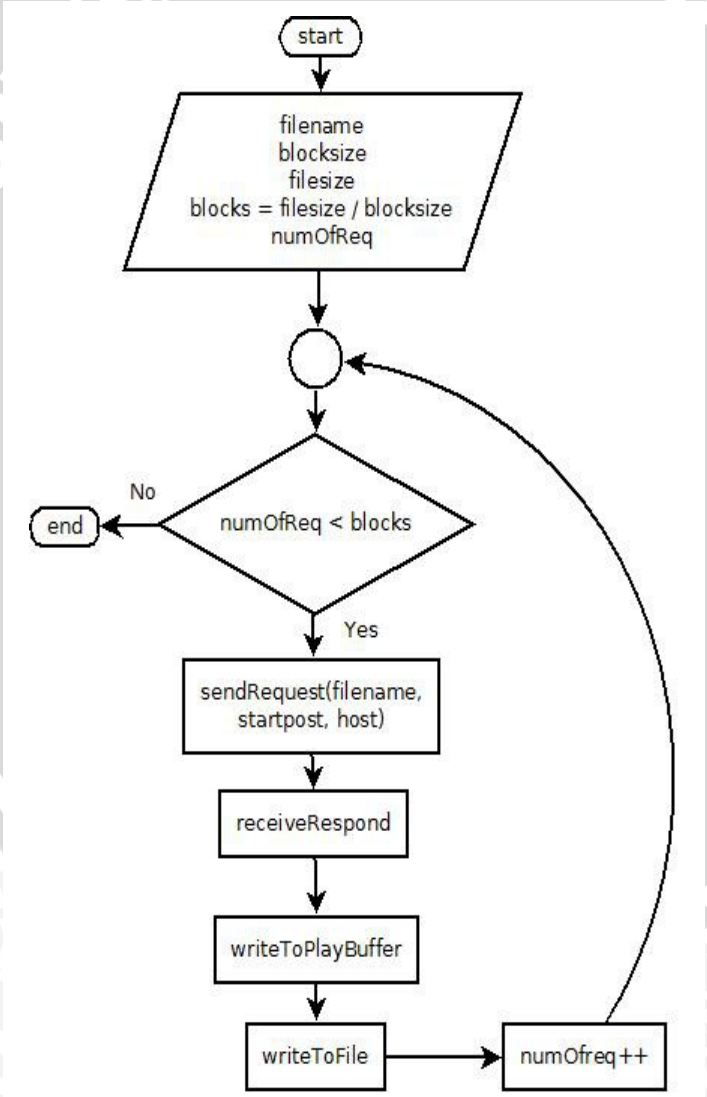
Fungsi startTimerCallback membutuhkan tiga paramater untuk berjalan yaitu filename, *block*size dan filesize. Filename adalah nama berkas yang akan dimainkan, *block*size adalah besar blok yang diminta dan filesize adalah besar berkas yang akan dimainkan.

Sebelum dilakukan permintaan terlebih dahulu dilakukan pembuatan antrian permintaan lewat fungsi createStartPostQueue. Tiap item dalam antrian berisi besar blok,

nama berkas, dan alamat NodePeer yang memiliki berkas. Satu berkas utuh akan dibagi-bagi menjadi beberapa blok dengan besar yang ditentukan dengan rumus :

Blocksize = (1200 / ukuran frame berkas mp3) * ukuran frame berkas mp3

Besar *blocksize* merupakan hasil perkalian antara ukuran frame berkas mp3 dengan suatu nilai acak. Besar *block*size akan mempengaruhi persebaran dan pembagian peladenan dari nodePeer yang tersedia. Formula di atas dibuat dengan dasar tujuan keacakan untuk tiap besar blok berkas bukan tiap streaming, hal ini disebabkan besar ukuran frame berkas mp3 akan sama pada berkas yang sama.



Gambar 4.17 Alur permintaan blok ke node lain

Seperti terlihat pada Gambar 4.17 setelah antrian selesai dibuat maka permintaan dikirim kepada NodePeer yang memiliki berkas sesuai dengan informasi dari item dalam antrian. Setelah blok yang diminta telah diterima maka dilanjutkan dengan menuliskannya pada buffer pemutaran berkas suara dan ditulis ke berkas dalam media penyimpanan. Fungsi startTimerCallback akan menjalankan beberapa permintaan sekaligus.

4. BassPlayer

BassPlayer memiliki fungsi untuk memainkan *stream* yang sedang berjalan sesuai dengan kondisi stream. Bila suatu blok berkas suara yang seharusnya dimainkan belum tersedia maka BassPlayer akan memberi waktu jeda hingga AsynchRedStream menyelesaikan proses pengunduh. Jeda akan dilakukan bila blok yang tersedia kurang dari T+3 detik waktu permainan, dimana T adalah posisi waktu permainan saat ini.

BassPlayer menggunakan pustaka Bass yang dihubungkan melalui binding C# Bass.Net. Untuk memainkan berkas *streaming* BassPlayer memainkan berkas yang di muat ke buffer, sehingga dibutuhkan pembuatan Bass *stream* dari alamat memory. Berikut adalah potongan kode implementasi nya :

```
public void play_buffer(IntPtr buffer, int length){
  local = false;
  if(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_PLAYING) {
    Bass.BASS_StreamFree(stream);
  }
  if ((stream = Bass.BASS_StreamCreateFile(buffer, 0L, length,
  BASSFlag.BASS_DEFAULT)) != 0) {
    Bass.BASS_ChannelPlay(stream, false);
  }
}
```

BAB V
PENGUJIAN DAN ANALISIS

5.1 Lingkungan Pengujian

5.1.1 Berkas Pengujian

Dalam pengujian ini digunakan dua berkas musik MP3 . Berkas yang digunakan diuraikan pada Tabel 5.1 .

Tabel 5.1 Berkas yang digunakan dalam pengujian

No	Nama	Bitrate (Kbps)	Samplerate (KHz)	Panjang (menit)	Size (Byte)
1.	Misread.mp3	192Kbps	44.100	03:29	5011748
2.	Homesick.mp3	256 Kbps	44.100	03:13	6204885

5.1.2 Lingkungan Pengujian

Pengujian jaringan *peer to peer* akan menggunakan 11 komputer: 1 untuk Hub dan 10 untuk nodepeer. Setiap komputer yang tersedia dihubungkan menggunakan *switch* Fast Ethernet serta dihubungkan dengan cabel Cat 5e.

5.1.2.1 Konfigurasi perangkat keras Hub

Dalam kerjanya komponen Hub tidak terlalu membutuhkan sumberdaya pemrosesan maupun penyimpanan yang besar, sehingga dalam implementasi digunakan komputer virtual. Spesifikasi perangkat keras *Host* mesin virtual diuraikan pada Tabel 5.2 sedangkan spesifikasi perangkat keras mesin virtual diuraikan pada Tabel 5.3.

Tabel 5.2 Spesifikasi Perangkat Keras Host Mesin Virtual

Prosesor	Intel Core I3 3,3 Ghz
Memori	2 X 4 GB
Hardisk	500 GB
NIC	Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Gigabit Ethernet controller

Tabel 5.3 Spesifikasi Perangkat Keras Mesin Virtual

Memori	995 MB
Hardisk	7 GB
NIC	Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Gigabit Ethernet controller

5.1.2.2 Konfigurasi perangkat keras Node *peer*

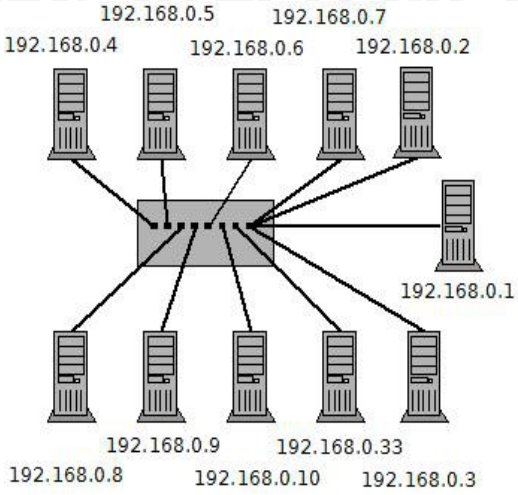
Perangkat keras Node *peer* sejumlah 10 komputer yang kemampuan multimedia untuk memutar berkas audio. Spesifikasi perangkat keras yang digunakan masing-masing Node *peer* diuraikan pada Tabel 5.4.

Tabel 5.4 Spesifikasi perangkat keras komputer NodePeer

Prosesor	Intel E7500 (2.93 GHz, 1066 MHz FSB)
Board	HP H-I41-uATX
Memori	2 x 1GB DDR3 10600
Hardisk	320 GB
NIC	Realtek Semiconductor Co., Ltd. RTL8101E/RTL8102E PCI Express Fast Ethernet controller (rev 02)

5.1.2.3 Konfigurasi jaringan

Jaringan *peer to peer* menggunakan jaringan lokal dengan topologi bintang. Setiap kartu jaringan pada masing-masing *node* terhubung pada *switch* DES 1024R yang memenuhi standar *Fast Ethernet* (10/100 MBPS) dengan media kabel UTP Cat 5e. Gambar 5.1 menunjukan topologi jaringan yang digunakan beserta alamat IP masing-masing node.



Gambar 5.1 Topologi jaringan beserta Alamat IP komputer

5.2 Pengujian Total Delay per Playback

Total Delay per Playback adalah jumlah waktu penundaan yang terjadi pada saat pemutaran / *streaming* berkas suara. Penundaan terjadi karena blok berkas yang seharusnya dimainkan belum sampai, sehingga dibutuhkan waktu untuk menyelesaikan proses pengunduhan berkas tersebut. Penundaan pemutaran berkas suara dilakukan oleh komponen Bass Player bila blok yang tersedia dalam *buffer* kurang dari T+3 detik, dimana T adalah posisi pemutaran suara saat itu.

Pengujian dilakukan dengan melakukan *streaming* berkas audio, selama *streaming* dilakukan pencatatan *delay*. Pencatatan *delay* dilakukan menggunakan timer pada *thread* yang berbeda dari program utama dan berjalan dengan interval 100 ms. Timer akan mencatat bila terjadi penundaan saat proses *streaming*. Host dengan alamat 192.168.0.8 tidak dilakukan test karena merupakan *host* yang mengunggah berkas pertama kali.

Tabel 5.5 Lama penundaan pada tiap *host* nodePeer

No	Alamat IP	Delay pada berkas (ms)	
		Misread.mp3	Homesick.mp3
1.	192.168.0.1	0	0
2.	192.168.0.2	0	0
3.	192.168.0.3	0	0
4.	192.168.0.4	0	0
5.	192.168.0.5	0	0
6.	192.168.0.6	0	0
7.	192.168.0.7	0	0
8.	192.168.0.9	0	0
9.	192.168.0.10	0	0

Nilai *delay* rata-rata didapat dari akumulasi *delay* dari tiap nodePeer dibagi jumlah nodePeer.

Tabel 5.6 Lama penundaan rata-rata yang terjadi dalam proses *streaming*

No	Nama	Bitrate (Kbps)	Samplerate (Hz)	Panjang (menit)	Delay Rata-rata
1.	Misread.mp3	192Kbps	44.100	03:29	0
2.	Homesick.mp3	256 Kbps	44.100	03:13	0

Selain menguji penundaan juga dilakukan pengujian terhadap *throughput* dari aplikasi, pengujian dilakukan dengan mengnangkap paket-paket yang lewat pada saat pengujian berlangsung menggunakan program Wireshark. *Throughput* didapat dari penghitungan yang dilakukan oleh program Wireshark.

Tabel 5.7 Rata-rata throughput jaringan

No	Rata-rata paket per detik	Rata-rata besar paket (byte)	Througput (Mbit/s)
1.	845,156	700,490	4,736

Pada hasil pengujian pada tabel 5.6 tidak terjadi penundaan pemutaran berkas suara di setiap pengujian yang dilakukan, disebabkan oleh setiap blok yang dibutuhkan untuk melakukan pemutaran berkas sudah sampai sebelum dimainkan, sehingga tidak terjadi penundaan. Hal ini terjadi karena *throughput* jaringan yang digunakan untuk pengujian serperti terlihat pada tabel 5.7 memiliki laju yang lebih cepat dari pada yang laju bit (*bitrate*) berkas suara yaitu.

5.3 Pengujian Total Ignored Transmit Session per Playback dan pembagian beban saat streaming

Pengujian *Total Ignored Transmit Session per Playback* dilakukan dengan cara mencatat tiap sesi yang gagal setelah melewati waktu yang ditentukan (*time out*), *time out* yang digunakan adalah 100 ms. Host dengan alamat 192.168.0.8 tidak dilakukan test karena merupakan *host* yang mengunggah berkas pertama kali.

Pengujian penyebaran pembagian beban dilakukan dengan mencatat dari mana sebuah peer mendapatkan blok dan jumlah blok yang diminta dari tiap peer. Pencatatan dilakukan dengan membuat sebuah struktur data dalam program yang menampung data-data yang dibutuhkan.

Tabel 5.8 Jumlah sesi yang diabaikan pada tiap *host* nodePeer

No	Alamat IP	Ignored Session	
		Misread.mp3	Homesick.mp3
1.	192.168.0.1	0	0
2.	192.168.0.2	0	0
3.	192.168.0.3	0	0
4.	192.168.0.4	0	0
5.	192.168.0.5	0	0
6.	192.168.0.6	0	0
7.	192.168.0.7	0	0
8.	192.168.0.9	0	0
9.	192.168.0.10	0	0

Tabel 5.9 Penyebaran beban pada saat streaming, sample peer ke-10

No	Nama	Jumlah blok										
		Peer 1	Peer 2	Peer 3	Peer 4	Peer 5	Peer 6	Peer 7	Peer 8	Peer 9	Tota l	SD
1.	Homesick .mp3	66	66	67	67	66	66	-	67	66	531	0.48
2.	Misread. mp3	-	-	71	70	70	70	-	71	70	422	0.47

Dari hasil pengujian pada Tabel 5.8 terlihat bahwa tidak ada permintaan dari *peer* yang diabaikan, hal ini berarti tidak ada *peer* yang terlalu sibuk untuk melayani permintaan *peer* lain. Ketidakadaan *peer* yang terlalu sibuk melayani permintaan dikarenakan penyebaran beban yang merata pada *peer* yang ada dalam sistem seperti terlihat pada Tabel 5.9.

5.4 Pengujian Pembandingan Berkas Suara Unduhan

Untuk mengetahui adanya perbedaan antara berkas sumber dengan berkas yang telah diunduh maka dibandingkan kedua berkas tersebut. Pengujian menggunakan aplikasi WinMerge untuk melihat perbedaan pada berkas *binary.*

Tabel 5.10 Perbandingan bit berkas hasil *streaming* dengan berkas asli

No	Alamat IP	Perbedaan			
		Misread.mp3		Homesick.mp3	
		Byte	Persen (%)	Byte	Persen(%)
1.	192.168.0.1	422	0.0084	531	0.0855
2.	192.168.0.2	422	0.0084	529	0.0852
3.	192.168.0.3	420	0.0083	521	0.0853
4.	192.168.0.4	420	0.0083	531	0.0855
5.	192.168.0.5	422	0.0084	531	0.0855
6.	192.168.0.6	422	0.0084	530	0.0854
7.	192.168.0.7	420	0.0083	521	0.0853
9.	192.168.0.9	422	0.0084	530	0.0854

10.	192.168.0.10	424	0.0084	534	0.0856

Angka pada kolom perbedaan berarti banyaknya titik perbedaan dalam satu berkas binary. Merujuk pada tabel 5.10 dapat terlihat bahwa perbedaan hampir sama dengan jumlah blok yang diminta. Perbedaan disebabkan oleh proses pengiriman berkas lewat jaringan dengan tidak dilakukan rutin koreksi error setelahnya.

Perbedaan yang terjadi berkas yang terjadi menimbulkan glitch atau suara yang tidak diharapkan. Karena perbedaan yang terjadi menyebar secara acak di dalam berkas dan jumlahnya sedikit sehingga glitch yang terjadi tidak terlalu terasa saat berkas lagu dimainkan.

BAB VI
PENUTUP

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujinan audio *streaming peer to peer* metode *block scheduling* didapat kesimpulan antara lain :

1. *Audio streaming peer to peer* metode *block scheduling* diimplementasikan menjadi 2 bagian aplikasi yaitu Hub sebagai *tracker* dan NodePeer sebagai *peer*. Tracker dibutuhkan untuk mengkonsolidasi informasi tentang berkas dan *peer* dalam jaringan. NodePeer terdiri dari 2 komponen utama yaitu Nodepeer Stream dan Nodepeer Serv, Nodepeer Stream melakukan proses *streaming* berkas suara sedangkan Nodepeer Serv melakukan proses peladenan *streaming* berkas suara.
2. Berdasarkan hasil pengujian tidak terjadi penundaan yang mengganggu pemutaran berkas suara karena *troughput* lebih besar dari pada *bitrate* kedua berkas yang diuji yaitu 4,736 Mbit/s , beban terbagi hampir sama rata antar peer yang ada dalam sistem dengan SD = 0.48 untuk pengujian pada berkas pertama dan SD = 0.47 untuk pengujian pada berkas kedua, selain itu terjadi perbedaan antara berkas sumber dengan berkas hasil *streaming* sebesar 0.0083% sampai 0.0084% untuk berkas pertama dan 0.0854% - 0.0856% untuk berkas kedua.

6.2 Saran

Saran yang diberikan untuk pengembangan sistem *audio streaming peer to peer* metode *block scheduling* antara lain :

1. Mengimplementasikan Error Checking sehingga dapat menghindari perbedaan berkas suara.
2. Penggunaan banyak *tracker* atau penyebaran dan penghimpunan informasi dilakukan di setiap *peer* dapat meningkatkan kehandalan sistem.
3. Mengimplementasikan penanganan kesalahan yang lebih baik.
4. Menggunakan algoritma peladenan antrian yang lebih baik.
5. Pengujian dilakukan di lingkungan pengujian yang lebih luas

DAFTAR PUSTAKA

Peterson , Larry L. dan Bruce S. Davie. “Computer Networks, A Systems Approach”. Morgan Kaufmann Publishers. 2007

Sommerville, Ian. “Sofware Engineering”. 2001

Tanenbaum, Andrew Stuart dan David J. Wetherall. “Computer Networks, fifth edition”. Pearson Education ltd. (2011)

Chang, Le. “A Comparison Study of Block Schedulling Algorithms in P2P-VoD Systems”. Departement of Computer Science University of Victoria

Garbacki, Paweł, dkk. "Offloading Servers with Collaborative Video on Demand"

Huang , Guowei dan Liang He. "Optimizing the Playback Quality of Data-Driven peer-to-peer Streaming". International Conference on Computational and Information Sciences. (2012).

Kao, Yung-Cheng dkk. “A Network Coding Equivalent Content Distribution Scheme for Efficient *peer*-to-*peer* Interactive VoD Streaming”. IEEE Transactions On Parallel and Distributed System Vol. 23. (2012)

M. Hafeeda, dkk. “A hybrid architecture for cost-effective on-demand media streaming”. Computer Networks 44. (2004)

M. Hafeeda, Mohamed dan Barat K Bhargava. “On-Demand Media Streaming over the Internet”.

DanONeill. “Home – Id-tag3.org”. http://id3.org/Home. Web. 13 Juli 2013

“Overview of Sockets (Network Interface Guide)”. Oracle. 2010. Web. 20 April 2013

SD Association. “Glossary – SD Associaion”. https://www.sdcard.org/join/glossary/#mp3. Web. 13 Juli 2013

The World Bank. "Internet users as percentage of population". The World Bank Group. Web. 9 April 2013.

LAMPIRAN