VOICE GATEWAY UNTUK INTERNET TELEPHONY

TUGAS AKHIR

Disusun Oleh :

JOKO SARASWANTO
5197100107

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2004

VOICE GATEWAY UNTUK INTERNET TELEPHONY

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer Pada
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui

Dosen Pembimbing

Febriliyan Samopa, S.Kom., M.Kom.
NIP. 132 206 858

SURABAYA
Agustus, 2004

kupersembahkan
kepada ibu, bapak, istri, dan putra-putra tercinta, Hamzah dan Azis
serta semua yang telah mendukung dan membantu hingga tugas akhir ini selesai

ABSTRAK

Semakin tingginya biaya komunikasi untuk melakukan percakapan langsung jarak jauh (baik SLJJ maupun SLI) menggunakan jaringan telepon tradisional (PSTN) dan berkembangnya teknologi informasi berbasis protokol internet yang cepat dan relatif lebih murah mendorong pertumbuhan penggunaan internet itu sendiri. Dengan teknologi internet, tidak hanya data teks saja yang bisa ditransmisikan namun lebih dari itu, data suara maupun video pun dapat dikirimkan melalui jaringan berbasis protokol internet. Atas dasar tersebut, penulis berupaya mengembangkan suatu sistem yang dapat menggabungkan antara teknologi PSTN dan internet sehingga penggunaan jaringan telepon tradisional dapat digunakan bersama dengan jaringan internet.

Pada awalnya sistem Voice Gateway ini menerima penggilan telepon dari pelanggan melalui voice modem yang terpasang pada sistem. Kemudian pelanggan PSTN memasukan kode area dan nomor telepon tujuan dengan cara menekan tombol-tombol telepon sesuai dengan format yang ditentukan. Oleh Voice Gateway, masukan sinyal-sinyal DTMF tersebut akan diterjemahkan melalui tone decoder yang selanjutnya dikirim ke aplikasi Voice Gateway dalam bentuk data digital. Kemudian data tersebut akan dipisahkan menjadi kode area dan nomor telepon. Voice Gateway akan menterjemahkan kode area menjadi alamat IP Voice Gateway tujuan dan selanjutnya Voice Gateway melakukan sambungan ke Voice Gateway tujuan, setelah sambungan berhasil dibangun Voice Gateway lokal akan mengirimkan nomor telepon yang harus di-dial oleh Voice Gateway tujuan. Setelah Voice Gateway tujuan berhasil menghubungi nomor tujuan maka sesi percakapan pun dimulai.
Selanjutnya ketika salah satu pihak memutuskan sambungan maka sesi percakapan akan terputus dan kemudian disisi Voice Gateway lokal akan mengeluarkan tagihan berdasarkan lama percakapan yang telah dilakukan.

Dari hasil ujicoba yang telah dilakukan maka metode kompresi/dekompresi yang menghasilkan delay minimum adalah Codec MS PCM 64kbit/s dengan kualitas suara Excellent (menurut standar Mean Opinion Score) dan rata-rata delay adalah 19.55 ms dimana delay maksimal yang direkomendasikan sesuai standar ITU G.114 adalah 150 ms. Sistem ini dapat dikembangkan menjadi lebih komplek lagi, misal : layanan faxmili melalui jaringan internet, aplikasi pengganti sistem PABX dalam suatu LAN dan lain sebagainya.

KATA PENGANTAR

Pertama-tama penulis mengucapkan syukur alhamdulillah kepada Allah SWT yang telah memberikan ridho-Nya sehingga penyusunan buku tugas akhir yang berjudul "**VOICE GATEWAY UNTUK INTERNET TELEPHONY**" ini dapat diselesaikan dengan baik dan lancar. Shalawat serta salam semoga tetap tercurah kepada tauladan dan junjungan mulia, Rasulullah Muhammad SAW, keluarganya, para sahabat, dan umatnya hingga akhir jaman nanti.

Tugas Akhir ini merupakan syarat akademis bagi para mahasiswa dalam rangka menyelesaikan studi kesarjanaan pada Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Penulis menyadari bahwa tugas akhir ini jauh dari sempurna, masih banyak kekurangan-kekurangan. Oleh karena itu sangat diharapkan kritik dan saran yang membangun untuk perbaikan ke depan.

Surabaya, Agustus 2004

Penulis

UCAPAN TERIMA KASIH

Segala puji bagi Allah Swt yang Maha Pengasih lagi Maha Penyayang. Pada kesempatan ini, penulis menyampaikan penghargaan dan rasa terima kasih yang sebesarnya-besarnya atas bimbingan, dukungan, doa, serta bantuan yang diberikan kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini kepada:

1. Ibu, Bapak, Mas Bambang, Mas Toto, Mas Imung dan Ade Retno, yang senantiasa memberi semangat, dukungan, dan doa, sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik.
2. Istriku tercinta, Rhein Astrisandy, yang dengan cubitan sayangnya memberikan kekuatan moril yang tak terhingga sehingga penulis dapat menyelesaikan tugas akhir ini.
3. Belahan jiwaku, Hamzah dan Azis, yang tangis dan candanya telah memenuhi hari-hari sulit penulis sehingga ruang hati penulis kembali lega dan penuh semangat.
4. Keluarga Bapak Akhmad Sariadi, de Dhona, de Ujang, de Firda, terima kasih atas segala dukungan yang telah diberikan. Semoga Allah membalas dengan yang lebih baik lagi.
5. Bapak Febriliyan Samopa, S.Kom, M.Kom selaku dosen pembimbing, yang telah memberikan banyak dukungan dan bimbingan selama pembuatan Tugas Akhir.
6. Bapak dan Ibu dosen, yang dengan keikhlasannya telah mencurahkan ilmu dan pengetahuan kepada penulis.

7. Teman-teman seperjuanganku: Yudie, Dio, Ronie, Nugie, Gemol, Cemphez, Gershom, Yuhana, Wiwik., dan lainnya yang tidak dapat disebutkan satu persatu disini. Ingat, sahabat sejati kemanapun dia pergi, akan selalu dihati.
8. Mas Yudhi, Pak Narno, Pak Kadir, Mas Sugeng dan pegawai TU lain, yang sudah sering membantu penulis dalam urusan administrasi.
9. Teman- teman di P2KB Unesa : Pak Nursalim, Pak Ikhsan Abadi, Pak Arif, Bu Tri, Bu Yulia, Mas Fauzy, Asmunin, Agus, Tauchid, Hari, Syai'ul, Rudi,
10. Semua pihak yang telah mendukung penulis hingga Tugas Akhir ini selesai.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

IP Telephony merupakan teknologi yang memungkinkan suara, data, dan video berkolaborasi menggunakan jaringan berbasis Internet Protokol seperti LAN, WAN, dan Internet. Secara rinci, IP Telephony menggunakan standar IETF dan ITU untuk mengatur lalu lintas data multimedia diatas setiap jaringan yang menggunakan Protokol Internet, menawarkan kepada pengguna baik fleksibilitas media fisik (misal : POTS lines, ADSL, ISDN, Leased Line, Kabel Coaxial, Satelit, dan Twisted Pair) dan fleksibilitas penempatan secara fisik. Sebagai hasilnya, jaringan berbasis Protokol Internet yang ada dimana-mana dapat digunakan untuk menghubungkan individu, bisnis, sekolah, dan pemerintah di seluruh dunia.

Di masa lalu, beberapa perusahaan telah membangun jaringan terpisah untuk menangani lalu lintas tradisional data, suara dan video. Masing-masing membutuhkan layanan transport yang berbeda. Jaringan-jaringan semacam ini membutuhkan harga yang mahal baik untuk pemasangannya, pengelolaan maupun untuk perawatannya. Lagipula, karena secara fisik jaringan ini berbeda satu sama lain, integrasi satu sama lain mnjadi sulit, jika tidak mustahil, membatasi kegunaan potensi masing-masing.

IP Telephony menggabungkan suara, video dan data dengan cara menetapkan jenis transportasi yang umum, yakni IP, yang secara efektif

menggabungkan ketiga jaringan menjadi satu. Hasilnya adalah meningkatkan kemampuan pengelolaan, menurunkan biaya-biaya pendukung, munculnya peralatan-peralatan kolaborasi baru dan meningkatkan produktifitas.

Aplikasi-aplikasi yang mungkin untuk IP Telephony termasuk didalamnya adalah kolaborasi dokumen-dokumen *real-time* , kegiatan belajar-mengajar jarak jauh, pelatihan pegawai, *video conferencing, video mail*, dan *video on demand.*

1.2 Permasalahan

- Bagaimana mengenali penekanan tombol/tuts telepon dan menterjemahkannya menjadi data teks.
- Bagaimana membuat sambungan antara Voice Gateway melalui jaringan berbasis protokol Internet.
- Bagaimana melakukan transmisi suara antara Voice Gateway yang kemudian disalurkan ke pengguna.
- Bagaimana memilih Algoritma CODEC yang tepat untuk digunakan dalam Voice Gateway.
- Bagaimana melakukan kalkulasi biaya percakapan.

1.3 Batasan Masalah

- Topologi jaringan yang digunakan dalam tugas akhir ini adalah topologi *Peer to peer.*
- Pilihan Algoritma Kompresi/Dekompresi yang digunakan merupakan standar yang disediakan oleh Microsoft Windows.

1.4 Tujuan

Tujuan akhir dari Tugas Akhir adalah membuat perangkat lunak yang berfungsi sebagai Voice Gateway untuk *Internet Telephony*.

1.5 Metodologi

Metodologi penelitian yang dilakukan penulis untuk menyusun tugas akhir ini adalah sebagai berikut:

- *Studi Literatur*, mencari, mempelajari dan merangkum berbagai macam literatur yang berkaitan dengan permasalahan, teori-teori yang berkaitan dan bahasa pemrograman yang digunakan.
- *Perancangan Perangkat Lunak*, membuat desain dasar mengenai aplikasi yang akan dibuat termasuk didalamnya desain class dan fungsi-fungsi yang dibutuhkan.
- *Pembuatan Perangkat Lunak*, melakukan pengkodean terhadap desain yang telah dibuat termasuk juga desain antar muka aplikasi yang dibuat.
- *Pengujian dan Evaluasi*, melakukan ujicoba dengan membuat beberapa skenario ujicoba beserta data-datanya dan melakukan evaluasi terhadap hasil ujicoba.
- *Penulisan Tugas Akhir*, menyusun buku Tugas Akhir sebagai dokumentasi dari pelaksanaan tugas akhir ini.

1.6 Sistematika Penulisan Tugas Akhir

Sistematika penulisan tugas akhir ini adalah sebagai berikut:

Bab I **Pendahuluan**, berisi latar belakang, permasalahan, batasan masalah, tujuan dan manfaat, metodologi dan sistematika penulisan tugas akhir.

Bab II **Dasar Teori**, memuat teori-teori yang mendukung penulis untuk menyusun tugas akhir ini. Diantaranya antara lain tentang DTMF *(Dual Tone Multiple Frequency)*, Voice over Internet Protocol, DirectX, DirectPlay, DirectPlayVoice dan lain sebagainya.

Bab III **Desain**, berisi desain arsitektur Voice Gateway yang dibuat, desain perangkat keras yang digunakan, diagram alur kerja Voice Gateway dan diagram alur data Sistem Voice Gateway.

Bab IV **Implementasi**, berisi pseudo code untuk fungsi-fungsi dan class-class module yang dibuat dalam aplikasi Voice Gateway.

Bab V **Ujicoba dan Evaluasi**, berisi skenario dan data ujicoba yang dilakukan oleh penulis beserta evaluasi dari hasil ujicoba.

Bab VI **Kesimpulan dan Saran**, berisi kesimpulan-kesimpulan yang dapat diambil dari penyusunan tugas akhir ini dan saran-saran penulis untuk pengembangan aplikasi ini.

BAB II

DASAR TEORI

BAB II

DASAR TEORI

2.1 Teknologi VoIP

VoIP adalah perpindahan sinyal-sinyal percakapan dengan menggunakan metode yang dapat diterima dari pengirim ke tujuan melalui jaringan internet. Sinyal percakapan merupakan potongan-potongan digital suara percakapan yang di *sampling* pada interval yang teratur. *Sample* ini dikirim melalui jaringan ke tujuan yang diinginkan dan kemudian disusun ulang menjadi sinyal analog yang mewakili suara asli. Didalam sebuah lingkungan jaringan maya, VoIP menawarkan sebuah teknologi yang mengijinkan komunikasi suara secara langsung antara sebagian besar pengguna jaringan.

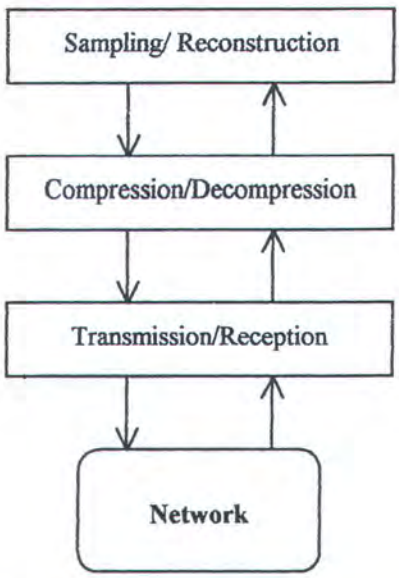
Penggunaan jaringan internet menuntut pemanfaatan protokol IP untuk melakukan pemindahan informasi. Jaringan berbasis IP terkadang mengacu pada jaringan berbasis paket , karena mereka berkomunikasi dengan cara mengirim dan menerima paket-paket data dengan format yang telah disepakati. Dua protokol standar, TCP/IP dan UDP, adalah protokol-protokol yang paling banyak digunakan hingga saat ini. Semua ISP mendukung protokol TCP/IP. Setiap pengguna rumahan dengan koneksi dial-up ke Internet juga menggunakan TCP/IP untuk berkomunikasi dengan internet.

TCP/IP mengacu pada format data yang dikirimkan melalui jaringan dan aturan-aturan berlaku untuk memastikan data sampai ditempat yang diinginkan. Paket-paket yang melalui TCP/IP harus *reliable* , artinya bahwa setiap potongan-potongan paket yang dikirimkan melalui jaringan dianggap valid dan diakui oleh

penerima. Jika data yang dikirimkan lebih besar dari satu potong paket data, data tersebut dapat dipecah menjadi potongan-potongan yang lebih kecil dan masing-masing dikirimkan secara terpisah. Paket-paket tersebut kemudian disusun ulang sesuai dengan urutan yang dikehendaki di tempat tujuan sebelum kemudian digunakan oleh aplikasi pengguna. TCP/IP menjamin bahwa setiap paket yang disusun ulang ditempat penerima tersusun dalam urutan yang sebenarnya. Penyusunan dalam urutan yang benar merupakan hal yang sangat vital untuk sebuah sinyal suara. Kesalahan atau hilangnya paket data akan menurunkan kualitas data suara yang ditransmisikan secara signifikan. Bagaimanapun juga, ongkos pemrosesan dan waktu tunda untuk jaminan ini akan meningkatkan *latency* secara signifikan dalam proses transmisi dan penyusunan kembali sinyal-sinyal suara.

UDP adalah protokol IP kedua yang paling sering digunakan. Tidak seperti TCP/IP, data UDP adalah *unreliable* . Protokol UDP tidak berisi permintaan yang kuat untuk mengakui setiap paket individu. Walaupun hal ini mengurangi ongkos dan waktu tunda dalam pemrosesannya, paket-paket dapat tiba dalam keadaan rusak atau bahkan tidak sampai ke penerima sama sekali. Kedua protokol ini menggunakan jaringan berbasis protokol IP untuk melakukan transmisi. Jaringan IP tidak memberikan jaminan *specific path* untuk melakukan pengiriman paket antara pengirim dan penerima. Setiap paket bisa saja mengambil jalur yang berbeda dalam jaringan. Untuk alasan ini, UDP secara umum tidak disarankan untuk digunakan dalam transmisi suara secara langsung karena tidak cukup memuaskan.

Protokol IP terbaru dikembangkan secara lebih spesifik untuk *streaming audio* dan *video* melalui Internet yang biasa dikenal dengan *Real-Time Transfer Protocol* (RTP). RTP mengolah *packet sequencing* dan *time stamping* pada sebuah *UDP data stream* untuk memastikan rekonstruksi pengurutan paket yang tepat pada penerima dengan tidak memaksakan ongkos pemrosesan transmisi yang tinggi. Komponen inti dari sebuah sistem VoIP untuk lingkungan virtual akan sedikit dibedakan dari sistem VoIP yang didesain untuk kepentingan lain, seperti sebagai alternatif pengganti telepon. Keseluruhan proses inti sistem VoIP diperjelas pada gambar 2.1. di bawah ini.



Gambar 2.1. Komponen-komponen VoIP

Panah yang menunjuk ke bawah menggambarkan arah yang harus diikuti ketika mengirimkan sinyal pembicaraan, panah yang menunjuk keatas menggambarkan urutan proses yang terjadi ketika sinyal pembicaraan diterima. Ketika label dari kotak tersebut terisi dua item, maka bagian kiri berarti data pengiriman sinyal bicara dan bagian kanan berarti hasil penerimaan sinyal

tersebut. Keduanya dikelompokkan bersama karena keduanya dioperasikan pada level yang sama. Item bagian kanan tidak selalu merupakan lawan dari bagian sebelah kiri. Untuk kepentingan mengirimkan suara secara langsung melalui sebuah jaringan komputer, sinyal pembicaraan harus didigitalisasi terlebih dahulu untuk kemudian ditransmisikan. Pada aplikasi VoIP umumnya, *sound card* dari PC melakukan *sampling* pada sinyal pembicaraan dan mengkonversikannya kedalam informasi digital.

Ketika sebuah blok yang telah didigitalisasi diterima, maka blok tersebut harus ditransfer kembali kedalam format sinyal audio. Seperti pada *sampling* dan digitalisasi untuk transmisi, maka transformasi tersebut terjadi pada bagian *receiver* dari *sound card* PC. Output dari proses tersebut ditampilkan melalui speaker maupun headphone untuk kepentingan pendengar. Sehingga dapat disimpulkan, *reconstruction* adalah kebalikan dari operasi *sampling*.

Beberapa hal perlu dipertimbangkan sebelum melakukan digitalisasi sinyal. Pertama, jika banyak orang diijinkan untuk melakukan pembicaraan pada saat bersamaan, seperti pada kasus di sistem *virtual environment training*, sinyal pembicaraan dari orang-orang tersebut harus dicampur secara bersamaan pada *receiver*. Pencampuran juga telah harus selesai pada *receiver* dari *sound card* PC.

Kedua, ketika mengirimkan blok data melalui jaringan, maka akan muncul sebagian kecil variasi pada waktu, sehingga hal tersebut berakibat dibutuhkannya beberapa waktu untuk setiap blok untuk mencapai tujuannya; sayangnya, hal ini dapat berubah menjadi cukup besar. Jalur masing-masing paket yang ditempuh melalui jaringan dari pengirim kepada penerima tidak di kontrol oleh salah satu

dari mereka (diasumsikan sebuah koneksi internet antara pengirim dan penerima). Telah banyak penelitian dilakukan pada paket arus data yang didistribusikan melalui jaringan, yang tidak hanya difokuskan pada hal tersebut. Sehingga dapat dikatakan bahwa sekali sebuah paket meninggalkan pengirimnya, kontrol pada jalur yang dilaluinya menuju penerima dan waktu yang dibutuhkan menuju penerima, adalah tidak terkontrol. Satu-satunya cara untuk mengontrol arus jaringan adalah dengan mengontrol keseluruhan jaringan. Pada kasus internet, hal tersebut tidaklah mungkin. Masalah dengan variasi dapat dilihat pada contoh berikut. Misal sinyal suara di rekonstruksi pada urutan tertentu sehingga paket suara diterima. Karena variasi waktu diterimanya paket pada jalur jaringan yang tidak dapat dikontrol, hal itu memungkinkan bahwa blok berikutnya belum sampai ketika output dari blok pertama selesai atau blok tersebut tiba tetapi diluar urutan. Untuk mengatasi hal tersebut, skenario rekonstruksi ditujukan khusus untuk menciptakan waktu tunda untuk memberikan peluang pada paket yang terkirim terakumulasi. Paket yang terakumulasi dapat diurutkan kembali dengan urutan yang berbeda dengan urutan kedatangannya, namun disamakan dengan urutan pengirimannya.

Lebih jauh tentang permasalahan tersebut, untuk melakukan digitalisasi informasi dibutuhkan sejumlah tertentu *bandwidth* yang disediakan khusus untuk koneksi. Proses *sampling* dan *reconstructing* sinyal pembicaraan pada tingkatan yang sangat tinggi sangat memungkinkan mengingat keunggulan teknologi komputing yang berkembang pada tahun-tahun akhir ini. Akan tetapi, melakukan *sampling* pada tingkatan yang lebih tinggi memiliki kecenderungan untuk

membanjiri koneksi jaringan dengan paket data, padahal setiap jaringan memiliki keterbatasan *bandwidth*. Secara sederhana menaikkan tingkatan *sampling* tidak akan secara nyata mengurangi *latency* dan dapat memberikan efek tertentu pada performance jaringan. Pada sebuah lingkungan jaringan maya sinyal suara bukan satu-satunya data yang ditransmisikan melalui jaringan. Data kondisi jaringan, tindakan yang terlibat dan informasi-informasi kontrol harus dikirimkan secara berkala untuk menjaga status bersama antara pengguna *virtual environment* yang terpisah. Seluruh lalu lintas jaringan, termasuk data suara, harus berbagi *bandwidth* untuk melakukan koneksi.

Skenario kompresi pada umumnya sering digunakan untuk mengurangi kebutuhan *bandwidth* yang besar dalam komunikasi suara. Kompresi berarti melakukan *encoding* dari suatu informasi dengan menggunakan suatu algoritma yang mengurangi data mentah dan memecahnya menjadi bagian-bagian yang lebih kecil. Terdapat beberapa tipe strategi kompresi. Penggunaan teknik kompresi untuk data tertentu, juga digunakan untuk data jenis lain. Beberapa teknik yang lain didesain secara spesifik untuk melakukan *streaming media* melalui Internet. Kompresi tipe ini secara signifikan mengurangi jumlah data yang harus ditransmisikan. Namun, semakin ruwet dan *processor-intensive* yang digunakan dalam suatu algoritma kompresi, semakin besar waktu tunda dalam transmisi dari sinyal suara orisinil.

Sekali blok berisi data pembicaraan yang dikompresi mencapai tujuannya, maka blok tersebut harus didekompresi kembali. Dekompresi sangat berkaitan erat dengan kompresi. Yaitu merupakan operasi kebalikan dari skenario kompresi

yang digunakan. Kompresi dan dekompresi menjadi sangat penting ketika koneksi jaringan lambat, seperti halnya menggunakan koneksi *dial-up* pada Internet.

Sampai saat ini, transmisi paket-paket suara telah direpresentasikan dalam bentuk pengirim tunggal dan penerima tunggal. Dalam lingkungan virtual, banyak skenario latihan yang dilibatkan lebih dari dua yang terlibat. Masalah pertumbuhan *bandwidth* muncul jika pengirim harus secara individu mentransmisikan data voice kepada pihak lain, khususnya jika jumlah pihak yang terlibat sangat besar. Semakin besar pihak yang terlibat, akan semakin banyak jumlah jalur antara pihak yang terlibat untuk mentransmisikan blok informasi suara.

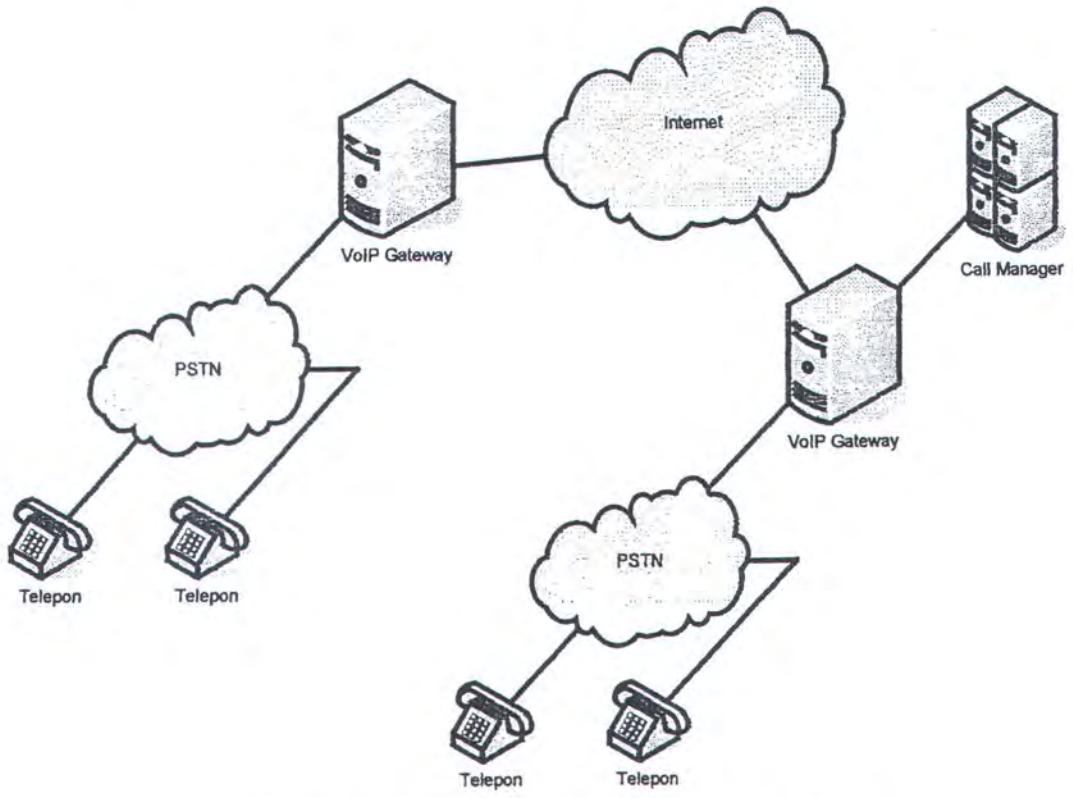
Multicasting adalah protokol berbasis IP dimana mesin secara individu terdaftar ke suatu alamat IP seolah-olah alamat tersebut adalah pengguna lain. Secara individu pengguna mengirim dan menerima dari *multicast address*. Perbedaan skema ini adalah setiap pengguna menerima semua data yang ditransmisikan ke *multicast address*. Hal ini mengijinkan transmisi tunggal sinyal suara dan penerimaan secara simultan oleh semua *user*. *Multicasting* memberikan jaminan peningkatan *streaming media* melalui Internet dengan cara mengurangi jumlah pesan yang dikirim kepada sejumlah besar *subscriber*. Namun, *multicasting* membutuhkan modifikasi perangkat keras pada semua *switch* dan *router* pada jalur jaringan. Perubahan yang mengijinkan *multicasting* melalui Internet belum diterapkan. Jaringan berskala kecil, dimana kontrol dan akses pada hardware jaringan memungkinkan, memberikan penawaran sebagai pilihan satu-satunya dalam penggunaan *multicast*.

2.1.1 Jenis Konfigurasi Jaringan VoIP

Pada umumnya terdapat 3 jenis konfigurasi jaringan VoIP, yaitu :

- Telepon melalui internet

Konfigurasi ini menggunakan fasilitas PSTN atau PABX pada kedua sisi sub sistem terminalnya. Konfigurasi seperti ini akan membutuhkan antar muka berupa gateway yang menghubungkan jaringan VoIP dengan jaringan internet. Untuk konfigurasi seperti ini dibutuhkan satu sistem tambahan lainnya yang dapat memetakan pemanggilan nomor telepon menjadi kode-kode IP, atau lebih dikenal dengan sebutan *Call Manager*. Ilustrasi konfigurasi pertama ini dapat dibuat sebagai berikut :

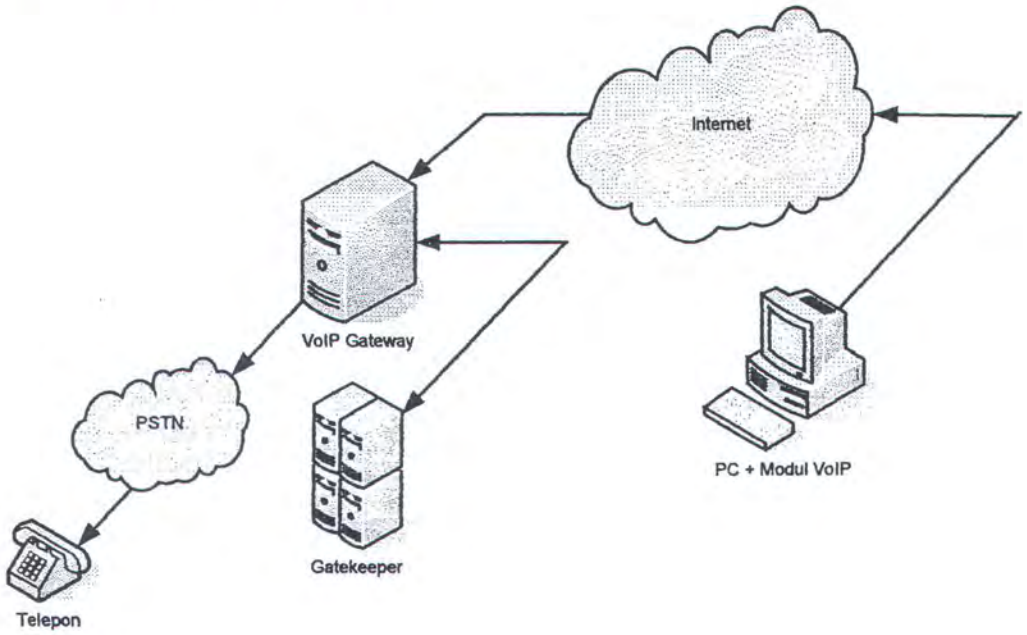


Gambar 2.2. Konfigurasi telepon melalui Internet

- Gabungan perangkat keras telepon dan perangkat berbasis IP (*Hybrid*)

Konfigurasi ini menggunakan sistem *hybrid*, yaitu campuran antara sub sistem terminal menggunakan PC disatu sisi dan sub sistem terminal menggunakan PSTN (telepon analog) di sisi yang lain.

Kelemahannya adalah sistem pemanggilan (pensinyalan) hanya berlaku satu arah dari terminal komputer ke telepon analog, tidak dapat berlaku sebaliknya. Hal ini terjadi karena keterbatasan metode pemanggilan yang ada pada sistem PSTN (telepon analog) dimana kita tidak membuat tabel routing dari PSTN menuju komputer tertentu. Untuk mengatur QoS, diperlukan perangkat tambahan yang disebut sebagai *gatekeeper*.

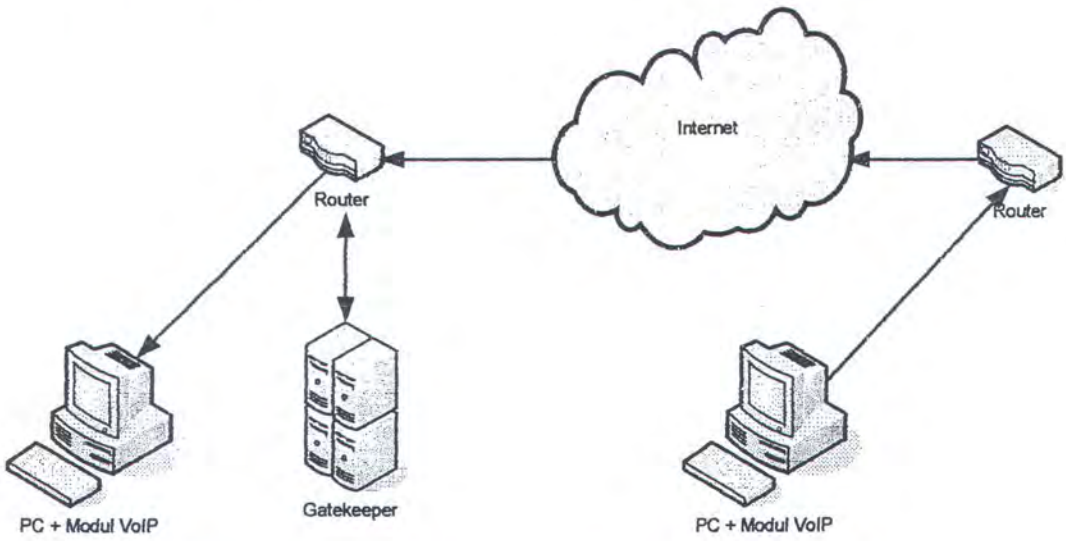


Gambar 2.3. Gabungan perangkat telepon dan perangkat berbasis IP

- Komunikasi antarperangkat berbasis IP

Pada dasarnya, konfigurasi jenis ini lebih banyak pada pengembangan bidang perangkat lunak multimedianya saja, belum memperhatikan masalah pengaturan pada media transmisi. Konfigurasi seperti ini membutuhkan sistem

pensinyalan yang tidak terlalu rumit, sehingga hanya pada kondisi tertentu saja dibutuhkan software manajemen pensinyalan. Sistem ini membutuhkan minimal sebuah *gatekeeper*.



Gambar 2.4. Komunikasi antarperangkat berbasis IP

2.1.2 Kualitas Layanan VoIP

Pengertian QoS (Quality of Service) adalah kemampuan suatu jaringan untuk menyediakan layanan yang lebih baik pada trafik data tertentu pada berbagai jenis platform teknologi. QoS tidak diperoleh langsung dari infrastruktur yang ada melainkan dengan mengimplementasikan pada jaringan yang bersangkutan.

Aplikasi VoIP adalah aplikasi *real time*, sehingga tidak dapat mentolerir *delay* (dalam batasan tertentu) dan *packet loss*. Namun dalam kenyataannya, *delay* internet sangat besar, bahkan melebihi *delay* yang terjadi pada celullar. Untuk mengurangi *delay* ini, banyak cara telah ditempuh. Cisco mengembangkan routernya sehingga dapat memberikan *delay* secara minimal dengan menggunakan

teknologi packet switching sebagai pengganti data switching. Pengurangan *delay*-nya dapat mencapt 20 ms. Selain mengimplementasikan perangkat keras dengan *delay* rendah, cara lain yang dapat ditempuh adalah dengan mengoptimalkan penggunaan bandwith, mengatur metode antrian yang dipakai, dan menggunakan protokol-protokol manajemen untuj mengatur paket-paket data yang dilewatkan. Dengan kata lain mengatur *Quality of Services* pada jaringan VoIP.

Untuk keperluan VoIP, ada syarat-syarat yang harus dipenuhi oleh suatu infrastruktur jaringan internet.

- Jaringan harus mempunya policy pengaturan trafik yang jelas.
- Bandwith jaringan harus memenuhi standar minimal aplikasi.
- Terdapat urutan prioritas paket data pada jaringan tersebut.

Tanpa ketiga hal ini, kita tidak dapat menjamin QoS jaringan tersebut dan akan berakibat menurunnya kualitas suara yang diterima oleh terminal.

Quality of Service pada IP telephony adalah parameter-parameter yang menunjukkan kualitas paket data jaringan. Beberapa parameter yang menyatakan QoS untuk IP Telephony antara lain *latency*, *delay*, *jitter*, *packet loss* dan *sequence error* pada jaringan internet.

2.1.2.1 Latency

Latency adalah waktu yang dibutuhkan oleh suatu perangkat dari meminta hak akses ke jaringan samapa mendapatkan hak akses itu. Ada dua jenis *latency*, yaitu *real* dan *induced*. *Real latency* berhubungan dengan fisikal jaringan dan karakteristik penyambungan dari media pengangkutannya, seperti pensinyalan elektrik dan *clocked speed*. Juga berhubungan dengan RTT (*Round Trip Time*)

selama ditransmisikan dari sumber ke tujuan melalui berbagai perubahan kecepatan transmisi.

Induced latency adalah *delay* yang terjadi akibat *delay* antrian pada peralatan jaringan (misalnya ethernet card, router), *delay* proses pada end system, dan kongesti lain jaringan antara sumber dan tujuan. Pada jaringan yang cukup besar, *delay* antrian tidak dapat ditangani secara baik (misalnya penggunaan metode antrian yang berbeda pada tiap router).

Ada satu lagi *latency* yang tidak terlalu signifikan namun dapat berpengaruh pada perspektif pengguna yaitu *remembered latency*. *Latency* ini adalah *latency* yang terjadi akibat daya ingat manusia. Manusia dapat menyerap begitu banyak informasi dan cenderung sensitif pada *delay* saat pengiriman dan penyampain informasi.

Misalnya, seorang yang menggunakan layanan jaringan akan cenderung mengingat jumlah proses kegagalan yang sedikit dalam penggunaan servisnya dibanding mengingat jumlah keberhasilan penggunaan layanan tersebut. Penanggulangan *latency* yang paling cepat adalah dengan mengatur metode antrian pada tiap router.

2.1.2.2 *Delay*

Salah satu pertimbangan desain dalam implementasi transmisi suara adalah minimisasi *delay* satu arah atau *end-to-end delay*. *Delay* merupakan parameter yang paling menentukan dalam QoS, dan parameter ini juga yang paling dapat diminimalisasikan.

Ada beberapa penyebab terjadinya *delay*, antara lain :

- Kongesti
- Kekurangan pada metode *traffic shaping*
- Penggunaan paket-paket data yang besar pada jaringan berkecepatan rendah.
- Adanya paket-paket data dengan ukuran yang berbeda-beda.
- Perubahan kecepatan antar jaringan WAN.
- Pemadatan bandwith secara tiba-tiba.

Trafik suara merupakan trafik *realtime* sehingga jika *delay* dalam pengiriman paket suara terlalu besar, ucapan yang disampaikan tidak dapat dikenali. *Delay* maksimum yang dapat ditolerir pada transmisi sinyal sesuai dengan standar ITU G.114 yang merekomendasikan bahwa *delay* kumulatif harus lebih kecil atau sama dengan 150 mdetik (*1-way delay*). *Delay* kumulatif ini terdiri atas dua jenis, yaitu :

- Fixed *Delay*, terbagi lagi atas :
  - *Delay* propagasi, adalah *delay* yang ditentukan oleh karakteristik jarak antara sumber dan tujuan, serta media transmisi yang digunakan untuk pengiriman sinyal suara.
  - *Processing delay*, merupakan *delay* yang diakibatkan oleh coding, kompresi, dekompresi dan decoding yang ditentukan oleh algoritma standar Codec.
  - *Delay* paketisasi, merupakan *delay* yang disebabkan pemrosesan pada sampel suara digital yang dibawa untuk ditempatkan pada payload sampai paket terisi penuh. Untuk mengurangi terlalu banyak *delay*

paketisasi, biasanya digunakan beberapa skema kompresi seperti pembagian yang dapat dikirim.

- Variable *delay*, dibagi lagi menjadi :
  - *Delay* antrian, *delay* ini disebabkan oleh waktu tunggu paket yang dilayani pada sebuah *trunk*.
  - *Delay* jitter buffer, digunakan di sisi penerima untuk melicinkan *delay* variable dan untuk memungkinkan decoding dan dekompresi.

2.1.2.3 Jitter

Parameter kedua adalah jitter. Jitter disebabkan oleh bervariasinya waktu penerimaan paket-paket data dari pengirim ke penerima. Parameter ini dapat ditangani dengan mengatur metode antrian pada router saat terjadi kongesti atau saat perubahan kecepatan terjadi. Hanya saja jitter tidak mungkin dihilangkan sebab metode antrian yang paling baik tetap saja tidak dapat menangani semua kasus antrian.

Gambar 2.5. Ilustrasi Jitter

Jitter dapat mempengaruhi kualitas suara yang dikirim. Untuk meminimalisasikan jitter ini, diusahakan agar pengiriman tiap-tiap paket data melalui jalur yang sama dan jangan sampai terjadi *packet loss* atau kongesti jaringan.

2.1.2.4 Packet Loss

Jaringan IP tidak dapat memberikan jaminan bahwa paket akan dikirim semua sesuai dengan pesanan. Paket akan didrop di bawah beban puncak dan selama periode kongesti yang disebabkan oleh beberapa faktor seperti kegagalan link transmisi atau kapasitas yang tidak mencukupi.

Gambar 2.6. Ilustrasi Packet Loss

Packet loss pada jaringan untuk IP telephony sangat besar pengaruhnya, dimana bila terjadi *packet loss* dalam jumlah tertentu, akan menyebabkan terjadi interkoneksi TCP melambat (terlalu banyak pengulangan proses handshake). Biasanyan *packet loss* sebesar 10% tidak bisa ditolerir. Pendekatan yang digunakan untuk mengkompnsasi *packet loss* meliputi interpolasi suara dengan pengulangan paket terakhir, pengiriman informasi redundan, menggunakan metode pembagi bandwith. Beberapa metode yang digunakan untuk membagi bandwith berdasarkan tingkatan trafik, antara lain :

- *Traffic shapping* (pengatur trafik), menggunakan metode *leaky bucket* untuk memetakan trafik menjadi beberapa antrian untuk menyediakan beberapa output dengan perilaku yang mirip. Metode ini diaplikasikan pada lapisan IP.

- *Admission Control*, secara fisik membatasi kecepatan jaringan dengan menggunakan clocking data pada tingkatan tertentu atau menggunakan sebuah token bucket untuk 'mencekik' trafik yang masuk.
- *IP precedence*, menggunakan bit IP precedence dalam header IP untuk membentuk sampai 8 kelas trafik.
- *Differential Congestion Management*, sebuah skema yang menyediakan perlakuan yang sesuai untuk suatu kelas trafik tertentu pada saat terjadi kongesti.

Dengan melakukan penanggulangan – penanggulangan seperti di atas, maka QoS jaringan dapat dikendalikan dengan lebih baik.

2.1.2.5 Sequence Error

Kongesti di dalam jaringan paket switch dapat mengakibatkan paket mengambil route yang berbeda untuk mencapai tujuan yang sama. Akibatnya, paket sampai dengan urutan yang berbeda.

Gambar 2.7. Ilustrasi Sequence Error

Untuk penilaian subjektif kualitas layanan VoIP adalah MOS (Mean Opinion Score) dimana nilai-nilai subjektifnya diambil berdasarkan kepuasan pendengar dan pembicara disaat mengadakan hubungan VoIP. Untuk MOS, secara garis besar dapat dilihat didalam tabel 2.1.

Tabel 2.1. Penilaian MOS terhadap kualitas layanan VoIP

NILAI MOS	Opinion Score untuk tes pembicaraan	Opinion Score untuk tes pendengaran
5	Excellent	Excellent (jauh lebih keras dari yang diisyaratkan)
4	Good	Good (lebih keras dari yang diisyaratkan)
3	Fair	Fair (kejelasan suara sesuai yang diisyaratkan)
2	Poor	Poor (lebih pelan dari yang diisyaratkan)
1	Bad	Bad (jauh lebih pelan dari yang diisyaratkan)

Data kuantitatif yang bisa didapat untuk menentukan MOS adalah dari sisi metode kompresi, yaitu bitrate yang dihasilkan serta *delay* kompresi yang dihasilkan. Tabel penilaian MOS yang didapat dari sisi metode kompresi dapat dilihat dalam tabel 2.2 berikut ini.

Tabel 2.2. Hubungan metode kompresi dengan penilaian MOS

Standard	Algorithm	Bit Rate (Kbit/s)	Typical end-to-end delay (ms) (excluding channel delay)	Resultant Voice Quality
G.711	PCM	48, 56, 64	<<1	Excellent
G.723.1	MPE/ACELP	5.3, 6.3	67-97	Good(6.3), Fair(5.3)
H.728	LD-CELP	16	<<2	Good
G.729	CS-ACELP	8	25-35	Good
G.729 annex A	CS-ACELP	8	25-35	Good
G.722	Sub-band ADPCM	48, 56, 64	<<2	Good
G.726	ADPCM	16,24,32,40	60	Good(40), Fair(24)
G.727	AEDPCM	16,24,32,40	60	Good(40), Fair(24)

2.2 DirectX

Microsoft DirectX adalah salah satu kelompok teknologi yang dibuat oleh Microsoft untuk menghasilkan komputer-komputer berbasis Microsoft Windows

sebagai sebuah platform ideal untuk menjalankan dan menampilkan aplikasi-aplikasi seperti *games* yang sarat dengan *real-time full-color 3-D graphics*, *video*, musik interaktif, dan *surround sound*. Dibuat secara langsung dalam keluarga sistem operasi Windows, DirectX merupakan bagian dari Microsoft Windows 98, Microsoft Windows Millennium Edition, dan Microsoft Windows 2000, sebagaimana halnya dengan Microsoft Internet Explorer. Jika dibutuhkan, komponen-komponen DirectX juga dapat diupdate secara otomatis dalam sistem operasi pada saat melakukan instalasi aplikasi-aplikasi *games* atau multimedia yang terbaru.

DirectX memberikan pengembang perangkat lunak suatu set API (*application programming interfaces*) yang konsisten yang menyediakan mereka peningkatan hak akses pada *feature* peralatan dari suatu *high-performance hardware* seperti *3-D graphics accelerators* dan *sound cards*. *API control* ini yang biasa disebut *low-level functions* termasuk *graphics memory management* dan *rendering*, memberikan dukungan bagi perlengkapan input seperti joystick, keyboards, dan mouse, dan kontrol dari *sound mixing* dan *sound output*. *Low-level functions* dikelompokkan dalam komponen yang mendukung DirectX, yaitu: Microsoft Direct3D, Microsoft DirectDraw, Microsoft DirectInput, Microsoft DirectMusic, Microsoft DirectPlay, Microsoft DirectSound, dan Microsoft DirectShow.

DirectX menyediakan *developers tool* yang membantu mereka mendapatkan kinerja terbaik yang mungkin dari mesin yang mereka gunakan. Yaitu menyediakan mekanisme secara ekplisit untuk aplikasi untuk

mendeterminasi kemampuan terkini dari sistem hardware sehingga mereka dapat mengijinkan performance yang optimal.

Sebelum DirectX, para pengembang yang membuat aplikasi-aplikasi multimedia untuk PC harus meng-kustomisasi produk-produknya sehingga produk-produk tersebut dapat berjalan dengan baik pada beragam jenis perangkat keras dan konfigurasi-konfigurasi pada mesin Windows. DirectX menyediakan sebuah *hardware abstraction layer* (HAL) yang menggunakan *software drivers* untuk membangun komunikasi antara *game software* dan perangkat keras komputer. Hasilnya, para pengembang dapat menggunakan satu paradigma DirectX yang konsisten untuk mengimplementasikan produk-produk mereka pada berbagai macam perangkat keras dan konfigurasi yang sangat beragam.

2.2.1 Komponen-komponen DirectX

Microsoft DirectX 8.1 dibuat dengan dilengkapi komponen-komponen sebagai berikut :

- DirectX Graphics merupakan gabungan dari Microsoft DirectDraw dan komponen Microsoft Direct3D dari Versi DirectX sebelumnya kedalam sebuah satuan *application programming interface* (API) yang dapat digunakan untuk semua pemrograman grafis. Didalamnya termasuk komponen Direct3DX utility library yang menyederhanakan banyak tugas pemrograman grafis.
- DirectX Audio berisi gabungan dari Microsoft DirectSound dan komponen Microsoft DirectMusic dari Versi DirectX sebelumnya ke dalam sebuah single API yang dapat di gunakan untuk semua pemrograman audio.

- Microsoft DirectInput menyediakan support untuk berbagai macam input devices, termasuk full support untuk *force-feedback technology*.
- Microsoft DirectPlay menyediakan support untuk permainan-permainan melalui jaringan dengan banyak pemain..
- Microsoft DirectSetup adalah sebuah simple API yang menyediakan *one-call installation* dari komponen DirectX.

2.2.2 Pemrograman DirectX dengan Visual Basic

Bagian ini memperkenalkan bagaimana melakukan pemrograman aplikasi dengan menggunakan Microsoft DirectX untuk Microsoft Visual Basic.

Topik berikut ini mencakup:

- Pembuatan Obyek DirectX

  Obyek DirectX8 memiliki banyak metode, misal method untuk melakukan penghitungan untuk Microsoft Direct3D. Untuk membuat suatu obyek DirectX8, terlebih dahulu harus dideklarasikannya sebuah variabel dengan tipe obyek DirectX8.

```
Public gObjDX As New DirectX8
```

Gambar 2.8. Kode Pogram deklarasi variabel obyek DirectX

Banyak dari obyek-obyek dasar komponen DirectX, seperti DirectSound8 tidak dibuat secara langsung. Namun, harus dibuat dengan cara memanggil metode DirectX8 yang bersesuaian. Contoh, untuk membuat sebuah obyek DirectSound8, terlebih dahulu buat

sebuah variabel dan *assign* variabel tersebut pada *return value* dari metode DirectX8.DirectSoundCreate.

```
Public gObjDSound As DirectSound8
Set gObjDSound = DirectX8.DirectSoundCreate(vbNullString)
```

Gambar 2.9. Kode Pogram membuat obyek dasar komponen DirectX

- Menggunakan GUID

  Microsoft DirectX menggunakan *Globally Unique Identifiers* (GUID) secara extensif untuk tujuan mengidentifikasi objek. GUID pada umumnya merupakan menggunakan struktur 128-bit, tapi umumnya direpresentasikan oleh *equivalent string* mereka masing-masing, misal `{AC330441-9B71-11D2-9AAB-0020781461AC}`.

- Untuk membuat suatu GUID

  1. Pada folder *\Samples\Multimedia\VBSamples\DXMisc\Bin* dari *DirectX SDK*, *run* Vbguidgen.exe.
  2. *Paste* GUID kedalam aplikasi Microsoft Visual Basic, dan letakkan tanda quotation disekelilingnya.

  GUID juga dapat dihasilkan dengan cara memanggil metode `DirectX8.CreateNewGuid` saat *run*..

- *Passing Array ke Method*

  Beberapa metode, semacam `DirectInputDevice8.Send DeviceData`, mengambil suatu tipe data *array* sebagai suatu *parameter*. Contoh, dalam *code fragment* berikut, *formatArray* adalah suatu array dari tipe `DIDEVICEOBJECTDATA`, *lCount* berisi sejumlah

elemen dalam *array*, dan *lFlags* adalah suatu *flag parameter* yang melakukan control terhadap bagaimana data tersebut dikirimkan.

```
MyDevice.SendDeviceData(lCount, formatArray(), lFlags)
```

Gambar 2.10. Kode Pogram Passing Array ke Metode

- Menggunakan Flags

Metode-metode suatu obyek dalam Microsoft DirectX memiliki suatu parameter dengan tipe data Long yang biasa disebut *flags* atau sesuatu yang serupa. *Type* dapat juga memiliki satu atau lebih *flag members*. *Flag fields* ini adalah cara tercepat untuk memilih dari beberapa variasi dari pilihan yang sering dipergunakan untuk menspesifikasikan perilaku dari suatu method secata tepat. contoh, *flags parameter* dari `DirectInput8.GetDevicesBySemantics` mengijinkan anda untuk mendefinisikan *scope* dari *enumeration* dengan cara memilih salah satu dari beberapa pilihan.

Suatu *flag* adalah sebuah *single yes-no bit* dari suatu informasi. Dalam sebuah *flag field*, masing-masing *bit* dari *integer* terkait dengan sebuah *flag* terpisah yang mewakili sebuah bagian independen dari sebuah informasi. Jika bit tersebut di-set, maka *flag* jadi *enabled*. Jika *bit* bernilai *zero*, *flag* jadi *disabled*. *Flag fields* umumnya berisi suatu *set* dari *flags* terkait yang diwakili oleh suatu *enumeration*. umumnya, *numerical value* dari masing-masing *member* dari suatu *enumeration* mewakili suatu bit yang berbeda dalam sebuah *flag field*. Namun, beberapa *member* dapat mewakili kombinasi dari dua atau lebih *flag*.

- Menggunakan Bitmask

Beberapa tipe Microsoft DirectX, semacam D3DCAPS8 memiliki **Long** member yang bertindak sebagai *bitmask*. *Bitmasks* adalah mirip dengan *flag fields*, namun mereka biasanya dipergunakan sebagai mask dari suatu *bits* tertentu dalam sebuah *field*. Hal itu dapat dikustomisasi dengan menggunakan format *hexadecimal* untuk meng-assign *values* pada *bitmasks* karena hal itu lebih mudah untuk mendeterminasikan, bit yang mana yang akan terpengaruh. Namun, melakukannya dapat mengakibatkan timbulnya suatu error karena Microsoft Visual Basic biasanya meng-convert konstanta hexadecimal ke tipe yang paling pendek. contoh, &HFF akan dikonvert pada suatu Integer dengan suatu value dari -1. jika value ini kemudian di-*passed back* ke *DirectX* untuk *Visual Basic*, dimana sebuah Long yang diharapkan, maka nilainya akan dikonversikan ke &HFFFFFFFF.

Untuk memastikan bahwa suatu *hexadecimal bitmask* telah dikonversikan dengan tepat, tempatkan sebuah ampersand kedua setelah persamaan. contoh:

```
DDSD.ddpfPixelFormat.lRBitMask = &HFF&
```

Gambar 2.11. Kode Pogram Bitmask

- Enumerasi DirectX

Kebanyakan aplikasi *Microsoft DirectX* harus melakukan enumerasi terhadap *resources* yang tersedia, biasanya saat melakukan inisialisasi. Contoh, sebuah aplikasi yang menggunakan *DirectX*

Graphics kemungkinan perlu intuk mencari tahu apakah *display modes available* atau tidak, atau sebuah aplikasi dengan menggunakan Microsoft DirectInput dapat membutuhkan untuk melakukan enumerasi pada *available buttons* dan melakukan akses pada sebuah *joystick.*

Pada DirectX, aplikasi meng-*handle enumeration* dengan tetap menjaga *enumeration object* yang tepat. Ketika sebuah *enumeration object* dibuat, ia membangun suatu koleksi yang ada sebagai *long* dari suatu *object* yang ada. Aplikasi kemudian dapat mengakses koleksi tersebut dengan menggunakan metode enumerasi obyek.

DirectX mendukung banyak enumerasi obyek, masing-masing dibuat untuk suatu tugas yang spesifik. Anda umumnya menjada suatu enumerasi obyek dengan memanggil suatu method. Misal, class `DirectX8` terdiri dari dua metode semacam itu. Salah satu diantaranya, `DirectX8.GetDSEnum` yang berfungsi mengembalikan suatu obyek enumerasi `DirectSoundEnum8`. Obyek ini mengijinkan kita untuk melakukan enumerasi perangkat Microsoft DirectSound untuk menjaga suatu GUID bagi perangkat yang tepat, kemudian dapat menggunakan GUID ini untuk membuat suatu obyek DirectSound8.

Obyek enumerasi yang lain adalah menjaga melalui berbagai variasi obyek-obyek komponen DirectX. Misal, anda dapat memanggil metode `DirectInput8.GetDIDevices` untuk mengambil informasi tentang perangkat input.

Contoh coding yang diilustrasikan pada gambar 2.12. adalah menunjukkan bagaimana melakukan enumerasi semua perangkat input yang terpasang pada sistem. Variabel `di` adalah suatu obyek DirectInput8. Langkah pertama adalah membuat sebuah variabel dengan tipe obyek enumerasi DirectInput.

```
Dim diEnum As DirectInputEnumDevices8
Set diEnum = di.GetDIDevices(0, DIEDFL_ATTACHEDONLY)
```

Gambar 2.12. Kode Pogram Inisialisasi obyek enumerasi

Selanjutnya, gunakan metode DirectInputEnumDevices8 melakukan iterasi melalui perangkat yang terpasang. Eximinasi masing-masing device untuk mendeterminasi apakah ia mempunyai kemampuan particular yang dibutuhkan. Dalam contoh berikut, nama perangkat diletakkan dalam suatu *listbox*:

```
Dim diDevice As DirectInputDeviceInstance
Dim X As Integer

For X = 1 To diEnum.GetCount
    Set diDevice = diEnum.GetItem(X)
    Call List1.AddItem(diDevice.ProductName)
Next X
```

Gambar 2.13. Kode pogram enumerasi alat input

- *Error Handling*

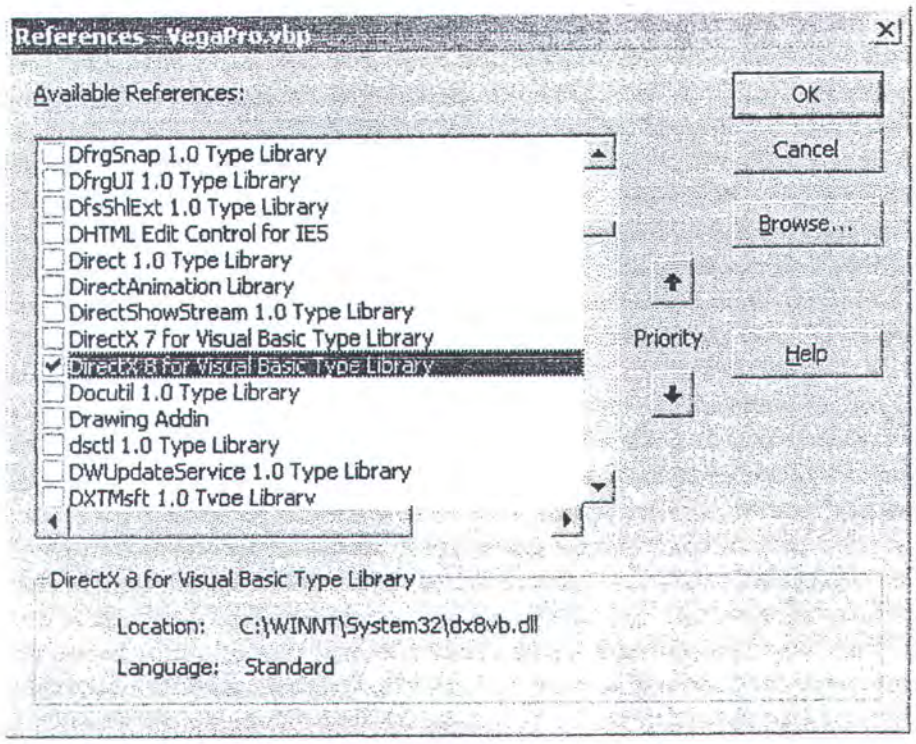
Jika pemanggilan metode suatu obyek gagal, *error* akan muncul, dan suatu *error number* diatur dalam *global Visual Basic Err object*. Aplikasi harus melakukan percabangan pada suatu *error handler* yang mengeksiminasi *Err object* dan berurusan dengan error, atau eksekusi dibatalkan.

2.2.3 Membuat referensi ke Visual Basic Library

Untuk menggunakan Microsoft DirectX untuk suatu proyek Microsoft Visual Basic, pastikan bahwa proyek tersebut telah dapat mengakses *library* DirectX.

Untuk memastikan sebuah proyek telah memiliki akses ke *type library*

1. Di Microsoft Visual Studio, pada menu **Project**, klik **References**.
2. Pilih **DirectX 8 for Visual Basic Type Library**, kemudian klik **OK**.



Gambar 2.14. Jendela References Visual Basic 6.0

2.3 DirectPlay

Microsoft DirectPlay API menyediakan perlengkapan pemrograman untuk mengembangkan aplikasi *multiplayer* semacam *game* atau *chat client*. Lebih sederhananya, bagian ini akan memberikan penjelasan pada seluruh aplikasi semacam *game*. Aplikasi *multiplayer* memiliki dua karekteristik dasar. Dua atau lebih user individu, masing-masing dengan sebuah *game client* pada komputer mereka. Link jaringan yang memberikan hak pada *user* komputer untuk berkomunikasi antara satu dengan yang lainnya, kemungkinan melalui sebuah server yang terpusat.

DirectPlay menyediakan sebuah layer yang secara global mengisolasi aplikasi anda dari *underlying network*. Untuk beberapa kepentingan tertentu, aplikasi dapat secara sederhana menggunakan DirectPlay API, dan mengijinkan DirectPlay untuk mengendalikan komunikasi jaringan secara lebih detail. DirectPlay menyedikanan banyak fitur yang berfungsi untuk menyederhanakan proses implementasi dari berbagai aspek dari aplikasi *multiplayer*, termasuk:

- Membuat dan mengelola baik *peer-to-peer* dan *client/server sessions*
- Mengelola *user* dan *group* antar *session*
- Mengelola pesan antar member dari sebuah *session* melalui link jaringan yang berbeda dan kondisi jaringan yang bervariasi
- Mengijinkan suatu aplikasi untuk berinteraksi dengan *lobbies*
- Mengijinkan *user* untuk berkomunikasi antara satu dengan yang lainnya melalui *voice*

2.3.1 Membuat dan Mengelola *Session*

Game session merupakan bagian inti dari *multiplayer game*. Sebuah *session* memiliki dua atau lebih *user* yang sedang bermain secara simultan, masing-masing berinteraksi dengan *game client* yang sama pada komputer mereka. Seorang pemain adalah sebuah entitas pada game itu sendiri, dan didefinisikan oleh game tertentu. Masing-masing user kemungkinan memiliki lebih dari satu pemain pada sebuah game. Namun, aplikasi *game* harus mengelola pemain itu sendiri, menggunakan *interface* Microsoft DirectPlay atau obyek untuk masing-masing pemain.

Langkah pertama dalam membuat sebuah *session* adalah dengan mengumpulkan kelompok *user*. Terdapat dua pendekatan mendasar. Banyak *game session* diatur oleh sebuah aplikasi *lobby* yang berjalan pada *remote computer*. Pendekatan ini dipergunakan pada umumnya aplikasi *games* yang berbasis internet. Juga memungkinkan untuk mengatur *game* dengan menggunakan *user* individu sedangkan komputer yang menjalankan komunikasi antara satu dengan yang lainnya. Pendekatan ini secara tipikal terbatas pada situasi tertentu, seperti sekelompok user yang berada pada satu LAN yang sama.

Ketika sebuah *session* telah diatur, maka *game* telah dijalankan dan permainan *game* dimulai. Saat *session* dilakukan, pemain kemungkinan dieliminasi dari *session*, atau pemain baru ditambahkan. Secara lebih detail sangat tergantung pada masing-masing *game*.

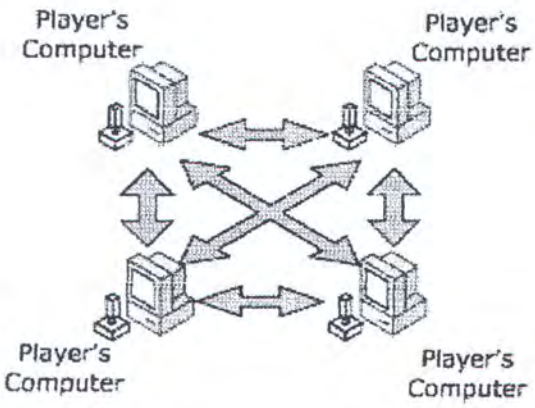
Dalam sebuah *multiplayer game*, masing-masing antar muka pengguna dapat disinkronisasikan dengan seluruh pengguna lain dalam suatu *session*.

Mengelola sebuah *multiplayer session* sangat membutuhkan aliran pesan secara kontinu dari dan ke masing-masing *user*. Sebagai contoh, setiap kali pemain bergerak, sebuah pesan harus dikirimkan untuk meng-*update* posisi dari semua pemain pada *game client* yang lain dalan suatu *session*. Inti dari DirectPlay adalah bagian dari API yang mendukung efisiensi dan fleksibilitas pesan antar semua komputer dalam sebuah *session*.

Terdapat dua cara mendasar untuk mengatur *messaging topology* dari sebuah *session*: *peer-to-peer* dan *client/server*. Kedua topologi tersebut memiliki kekurangan dan kelebihan masing-masing, sehingga perlu dilakukan evaluasi mana yang paling sesuai untuk *game* yang dibuat.

2.3.2 Topologi Peer-to-Peer

Peer-to-peer game terdiri dari komputer pemain individu, yang terhubung pada link jaringan. Dalam skema, topologi dari empat-pemain *peer-to-peer game* berikut:



Gambar 2.15. Topologi Peer-to-peer

Game play dikendalikan dengan masing-masing *user game client* berkomunikasi secara langsung dengan *user client* yang lain. Misal, ketika salah

satu *user* bergerak, *game client* harus mengirimkan tiga pesan *update*, satu untuk masing-masing pengguna komputer yang lain.

Peer-to-peer game secara normal diatur dan diluncurkan melalui sebuah aplikasi *lobby client* yang terdapat pada komputer pengguna. Terdapat dua cara mendasar untuk *lobby client* dapat mengatur sebuah *session*:

- *Lobby client* berkomunikasi secara langsung dengan *user lobby clients* potensial lainnya. Pendekatan ini dapat digunakan, dalam kondisi, untuk mengatur sebuah *game* antara *user* pada *LAN subnet* yang sama.
- *Lobby client* bertindak sebagai sebuah link ke aplikasi *lobby server* yang berjalan pada sebuah *remote computer*. Inilah cara *Internet-based game* secara normal diatur dan dikelola.

Sekali sebuah *session* diatur dan diluncurkan, sebagian besar atau seluruh *messaging* akan terjadi antara *user*. Jika sebuah *lobby server* dilibatkan, maka ia hanya mengendalikan tugas semacam meng-update daftar dari anggota *session* ketika seorang pemain meninggalkan *game*, atau mengijinkan seorang user baru untuk melakukan *request entry* pada *session*. Dengan kata lain, server tetap berjalan di *background*, dan secara tipikal tidak perlu mengendalikan ataupun mengawasi sebagian besar *message* yang telah dikirimkan.

Karena keberadaan server *non-existent* atau setidaknya tidak secara langsung terlibat dalam *game play*, seorang didesain sebagai *game host*. Mereka bertanggung jawab untuk mengendalikan *logistical details*, seperti membawa pemain baru ke dalam sebuah *ongoing session*.

Peer-to-peer games memiliki manfaat dari kesederhanaan. Semua itu diperlukan kumpulan pemain dengan *game clients*, dan sebuah cara untuk mengorganisir sebuah *session*. *Primary drawback* dari topologi *peer-to-peer* adalah *scalability*. Sebanyak peningkatan jumlah *user*, sejumlah *message* diperlukan untuk memfasilitasi peningkatan *game play* secara geometris. Jumlah maksimum dari *user* yang dapat diakomodasi sangat tergantung pada game dan *bandwidth* dari jaringan, tapi secara umum tidak lebih dari 20-30.

Peer-to-peer session terdiri dari suatu koleksi *user* yang terhubung dalam satu jaringan. Ketika suatu *lobby server* digunakan untuk mengatur dan meluncurkan suatu *game*, pengiriman dan pengolahan pesan dibutuhkan untuk menjalankan *game* yang dikirimkan secara langsung dari satu *user* ke *user* yang lain. Segala macam komunikasi dengan *lobby server* adalah untuk kepentingan terbatas semacam meng-*update* list dari beberapa *participants*.

Dengan *peer-to-peer game*, segala hal yang diperlukan untuk menjalankan *game* adalah tugas dari *client software*. Tanpa keterlibatan *server*, semua pemrosesan yang membutuhkan untuk membuat dan mengelola *game* universal harus ditangani oleh aplikasi *client*. Berikut ini merupakan prinsip dasar dari suatu *lobbyable* Microsoft DirectPlay *peer-to-peer game*. sebagai contoh sederhana, sebuah aplikasi *peer-to-peer*.

2.3.2.1 Inisialisasi suatu Peer-to-Peer Session

Peer-to-peer dapat diluncurkan secara langsung oleh *user*, atau *lobby-launched* oleh suatu aplikasi *lobby client* yang tersedia pada komputer user.

Dokumentasi ini mengasumsikan bahwa *game* adalah *lobbyable*, dan dapat berkomunikasi dengan *lobby client*.

Salah satu langkah pertama yang harus diambil untuk mendeterminasikan apakah *game* tersebut adalah *lobby-launched*. Untuk melakukannya, buat sebuah obyek DirectPlay8LobbiedApplication dan obyek DirectPlay8LobbyEvent. Register obyek DirectPlay8LobbyEvent dengan Microsoft DirectPlay dengan memanggil metode DirectPlay8LobbiedApplication.RegisterMessageHandler. Obyek DirectPlay8LobbyEvent adalah esensial sebagai *event handler* yang menerima peringatan secara langsung dari *lobbied application object*, dan secara langsung dari *lobby client* dan *lobby*. Hal itu tidak disediakan oleh DirectPlay dan harus diimplementasikan oleh aplikasi yang dibuat.

Jika suatu aplikasi merupakan *lobby-launched*, DirectPlay akan memanggil metode `DirectPlay8LobbyEvent.Connect`. *dlNotify* parameter akan berisi sebuah tipe `DPL_MESSAGE_CONNECT` dengan informasi koneksi. Seperti obyek *address* untuk *member session*.

Juga harus dibuat sebuah obyek DirectPlay8Peer dan meregistrasikan sebuah DirectPlay8Event *notification handler object* dengan memanggil `DirectPlay8Peer.RegisterMessageHandler`. Object ini akan memberikan makna utama dari komunikasi dengan DirectPlay dan user yang lain dalam *session*.

2.3.2.2 Memilih Service Provider untuk Peer-to-Peer Session

Service provider adalah koneksi jaringan yang digunakan. Kebanyakan *game* menggunakan salah satu service provider seperti TCP/IP atau modem, tetapi

Microsoft DirectPlay juga menyediakan dukungan untuk koneksi secara serial dan IPX .

Jika user terkoneksi pada *session* melalui *lobby client*, *service provider* dapat dideterminasikan yang tepat dengan cara menganalisa tipe `DPL_CONNECTION_SETTING` yang diterima ketika DirectPlay memanggil `DirectPlay8LobbyEvent.Connect`. Selain itu, untuk mendeterminasikan *service provider* yang tepat untuk digunakan, gunakan metode `DirectPlay8Client.GetServiceProvider`.

2.3.2.3 Memilih Host untuk Peer-to-Peer Session

Meskipun sebagian aspek dari *peer-to-peer games* dapat ditangani variasi pengguna komunikasi secara langsung dengan satu sama lain, terdapat beberapa tugas yang harus dimiliki *single owner*. Tugas-tugas ini ditangani oleh *game host*. Untuk bergabung dalam suatu *session*, perlu diketahui *address* dari *host session*. Cara umum untuk memilih sebuah *host* adalah melalui suatu *lobby server*. Dalam kasus tersebut, ketika aplikasi user dikoneksikan pada *session*, seting koneksi yang diterima ketika Microsoft DirectPlay memanggil metode `DirectPlay8LobbyEvent.Connect` termasuk obyek address *hos*. Untuk mengetahui siapa yang bertindak sebagai *session host* adalah dengan cara:

Memeriksa `dwFlags` member dari struktur `DPL_CONNECTION_SETTINGS` yang terdapat dalam `DPL_MESSAGE_CONNECTION_SETTINGS` yang melewati *dlNotify* parameter. Jika member tersebut diset sebagai `DPLCONNECTSETTINGS_HOST`, maka sistem anda adalah sebagai host. Jika

DPLCONNECTSETTINGS_HOST flag tidak di-set, maka *address host* akan didapatkan dari pdp8HostAddress member.

2.3.2.4 Koneksi ke Peer-to-Peer Session

Untuk melakukan koneksi, *session host* harus memiliki obyek *address*. Jika aplikasi terkoneksi melalui suatu *lobby client*, obyek *address* dapat diperoleh dengan cara memanggil DirectPlay8LobbiedApplication. GetConnectionSettings. Host *address* juga dapat diperoleh dengan melakukan enumerasi pada *host* yang tersedia. Informasi dikembalikan oleh enumerasi termasuk masing-masing obyek *address*, dan suatu struktur DPN_APPLICATION_DESC yang digambarkan oleh *session* terkait.

Untuk meminta bergabung dalam suatu *session*, lakukan pemanggilan DirectPlay8Peer.SetPeerInfo untuk men-*set* nama pemain, dan kemudian panggil DirectPlay8Peer.Connect dengan *host's address* yang dipilih untuk melakukan koneksi pada *session*.

Ketika seorang pemain meminta bergabung dalam sebuah *session*, Microsoft DirectPlay memanggil memanggil DirectPlay8Event. IndicateConnect. Untuk menerima pemain kedalam *session*, Atur parameter *fRejectMsg* bernilai False sebelum dikembalikan nilainya. Set *fRejectMsg* bernilai True jika ingin menolak *request*. Dalam kasus lain, metode DirectPlay8Event.ConnectComplete akan dipanggil dengan suatu *response*. Jika host menerima koneksi, member **hResultCode** dari tipe DPNMSG_CONNECT_COMPLETE akan diset ke 0. jika *request* ditolak atau gagal karena beberapa alasan lain, **hResultCode** akan di-set pada suatu error code.

Sekali pemain baru terkoneksi, DirectPlay mengumumkan keberadaan pemain baru dengan memanggil `DirectPlay8Event.CreatePlayer` untuk masing-masing member dari suatu session, termasuk host. Parameter *IPlayerID* berisi ID pemain yang akan digunakan untuk mengirim pesan ke pemain tersebut.

2.3.2.5 Mengelola Session Peer-to-Peer

Session host bertanggung jawab untuk mengelola *session*, termasuk:

- Mengelola daftar anggota *session* dan alamat-alamat jaringannya masing-masing.
- Memutuskan apakah *user* baru diijinkan untuk bergabung dengan *session.*
- Memperingatkan semua *member* ketika *user* baru bergabung dalam *session*, dan menginformasikan kepada mereka alamat user baru.
- Menyediakan *user* baru dengan *game state* yang baru
- Memperingatkan semua *user* ketika seorang *user* meninggalkan *session.*

Ketika pemain bermaksud bergabung pada suatu *session*, Microsoft DirectPlay akan memanggil metode `DirectPlay8Event.IndicateConnect`. Untuk menerima pemain dalam *session* maka set *fRejectMsg* sebagai False. Setting *fRejectMsg* pada nilai lain akan menghasilkan penolakan pada *request.* Dalam kasus lain, DirectPlay memanggil pemain metode `DirectPlay8Event.ConnectComplete` dengan *response* pada suatu *request.*

Host dapat memindahkan seorang pemain dari *session* dengan memanggil `DirectPlay8Peer.DestroyPeer`. Member dari suatu *session* tidak dapat memanggil metode ini secara sukses.

2.3.2.6 Normal Peer-to-Peer Game Play

Dalam *Microsoft DirectPlay* , suatu *message* secara esensial adalah sebuah block dari *game-related data* yang anda kirimkan pada satu atau lebih *members* dari suatu *session. DirectPlay* tidak secara spesifik menggambarkan isi dan format dari data block, ia hanya menyedikan mekanisme untuk mentransmisikan data dari satu *user* ke *user* yang lain. Sekali *game* sebagai *underway*, masing-masing *session member* secara normal mengirimkan suatu *constant stream* dari suatu *message* pada seluruh *members* yang lain dari suatu *session* sebagai durasi dari *game*. Tujuan utama dari *message* ini adalah untuk menjaga *game state* agar tetap sinkron, sehingga masing-masing aplikasi *user* menampilkan UI yang sama. Namun, *message* dapat juga dipergunakan sebagai variasi dari tujuan khusus *game* yang lain.

Untuk banyak *game*, terutama yang memiliki banyak perubahan secara cepat, kita perlu mengelola *messaging* secara hati-hati. *DirectPlay* melanjutkan *outgoing messages* pada suatu *level* yang dapat ditangani oleh *target*. Kita perlu berhati-hati bahwa kita tidak mengirimkan *message* terlalu cepat dan padat, dan pastikan *message* yang terpenting telah lewat.

Untuk mengirimkan sebuah pesan pada session member yang lain, panggil metode `DirectPlay8Peer.SendTo`. DirectPlay akan memanggil *member* metode `DirectPlay8Event.Receive` beserta datanya. Untuk mengirimkan sebuah

message pada pemain tertentu, set *idSend* parameter pada player ID yang dituju ketika metode `DirectPlay8Event.CreatePlayer` dipanggil. Kita juga dapat mengirimkan sebuah *message* pada masing-masing pemain dalam *session* dengan melakukan setting pada *idSend* menjadi `DPNID_ALL_PLAYERS_GROUP`. Kita juga dapat mendefinisikan group dari pemain, dan menggunakan sebuah single `SendTo` untuk menaggil dan mengirim suatu *message* pada semua member pada suatu group.

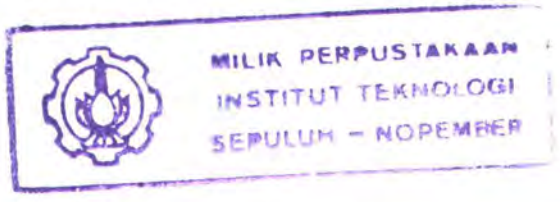
2.3.2.7 Meninggalkan Session Peer-to-Peer

Untuk meninggalkan suatu *session*, perlu dilakukan terminasi pada koneksi dengan memanggil `DirectPlay8Peer.Close`. Microsoft DirectPlay akan memanggil metode `DirectPlay8Event.DestroyPlayer` dengan *IPlayerID* parameter yang diset pada ID pemain..

Jika kita *adalah session host*, juga lakukan *terminates* pada *session* kecuali kita melakukan konfigurasi pada *session* untuk mengijinkan *host migration*. Lihat Host Migration untuk lebih detail.

2.3.2.8 Terminasi Peer-to-Peer Session

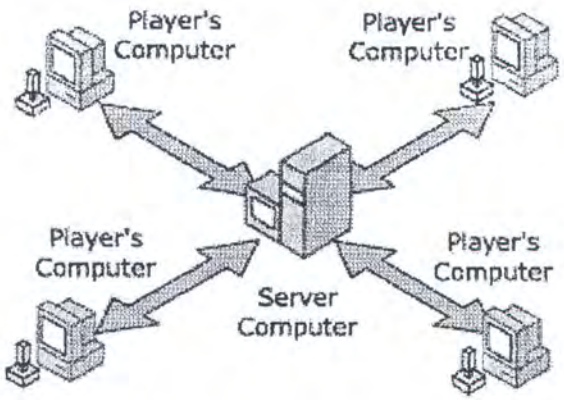
Ketika *session* telah selesai, *host* harus melakukan terminasi pada *session* dengan memanggil metode `DirectPlay8Peer.TerminateSession`. Metode ini melakukan terminasi pada *session* meskipun jika *host-migration* dalam kondisi *enabled*. Microsoft DirectPlay akan memperingatkan semua peserta *session* dengan memanggil metode `DirectPlay8Event.TerminateSession`. Kemudian perlu dilakukan beberapa *cleanup* yang diperlukan. Untuk memulai *session* yang



lain, terlebih dahulu harus dipanggil `DirectPlay8Peer.Close`, dan kemudian `DirectPlay8Peer.RegisterMessageHandler`.

2.3.3 Topologi *Client/Server*

Pemain c*lient/server game* pada komputer secara individu, terhubung pada sebuah komputer server central. Dalam skema, topologi dari empat pemain dalam permainan *client/server* terlihat sebagai berikut:



Gambar 2.16 Topologi Client/Server

Game dikelola dengan adanya komunikasi antara masing-masing user dengan server. Server bertanggung jawab untuk memberikan informasi dari dan ke pengguna yang lain. Misal, ketika suatu *user* bergerak, mereka mengirimkan sebuah *message* kepada *server*. Server mengirimkan *message* tersebut kepada pemainnya yang lain untuk memberikan informasi pada mereka tentang adanya perubahan pada *game state*. Server dapat memiliki sejumlah tanggung jawab:

- Bertindak sebagai *session's messaging hub*. Masing-masing komputer hanya perlu untuk mengirimkan *message* pada *server*. *Server* memegang kendali pada *logistics* dalam mensinkronisasi semua *user* yang lain.

Pengaturan semacam ini secara substansial dapat mengurang lalu lintas *message*, terutama pada *game* yang berukuran besar.

- *Host* bagi *game*. Server secara normal mengurus tugas yang harus dikendalikan oleh *session host* pada *peer-to-peer game*.
- Dukungan banyak aspek dari sebuah *game*. Server sering kali tidak lebih dari *support game logistics*. Pada banyak *game*, terutama *game* yang besar, banyak dari pemrosessan data terutama yang terkait dengan *game universe* dilakukan dalam Server. *Game client* terutama bertanggung jawab dalam menangani UI.

Client/server game secara normal diatur dan diluncurkan melalui sebuah aplikasi *lobby client* yang terdapat pada komputer pengguna. *Lobby client* bertindak sebagai link ke sebuah aplikasi *lobby server* yang secara normal dijalankan pada sebuah *remote computer* yang sama yang merupakan *host* dari *game*. Sekali *game* diluncurkan, Aplikasi *game server* berubah menjadi *host*, dan mengendalikan tugas-tugas semacam menambahkan *user* baru kedalam *game*.

Terdapat sejumlah manfaat dari *client/server game*:

- Lebih efisien, terutama untuk *games* skala besar. Umumnya, jangkauannya lebih baik dari pada *peer-to-peer games*, karena penambahan pemain hanya mengakibatkan peningkatan secara linear dalam *messaging traffic*. Topologi *Client/server* sangat perlu dilakukan untuk *massively-multiplayer games*.
- Sumber daya pemrosesan yang tidak terbatas bagi *user computers*. Dapat dilakukan banyak pemrosesan untuk mengendalikan sebuah

complex game universe pada sebuah komputer tunggal yang *powerful*, dan membiarkan pengguna komputer mengendalikan UI.

- Kontrol pada aspek kunci dari game pada bagian terpusat. Dalam kondisi ini, perlu dilakukannya update yang sering pada *game* atau memperbaiki *bug* dengan modifikasi yang sederhana pada aplikasi *server*, menghindari *update* pada sejumlah besar *game clients*.

Namun, sekali dikembangkan sebuah *peer-to-peer game*, maka secara esensial harus diselesaikan. *Game clients* adalah merupakan sejumlah besar *self-sufficient*. Dengan sebuah *client/server game*, terdapat sebuah komitmen yang berkelanjutan pada user yang lebih dari menyediakan *normal support services*. Sehingga perlu disediakan dan dijaga sebuah *game server computer* dan *software* terkaitnya, bersamaan dengan link jaringan untuk mengendalikan semua pengiriman *message*, untuk keberlangsungan dari aplikasi. Dalam kasus *massively multiplayer games*, maka perlu untuk mengoperasikan server pada periode tertentu dengan sedikit atau tanpa kendali dalam service, atau resiko kemarahan *user* karena gangguan pada permainan *game* mereka.

2.3.4 Komunikasi Jaringan DirectPlay

Fungsi utama dari Microsoft DirectPlay adalah untuk membantu menyediakan *messaging support* yang efektif dan efisien yang secara besar mengisolasi aplikasi dari *underlying network hardware and software*. Jika kita ingin mengirimkan sebuah *status update*, kita hanya perlu melakukan *call* pada DirectPlay API, tergantung pada jenis network link yang dilibatkan. DirectPlay

network service provider mendukung komunikasi melalui TCP/IP, IPX, modem, dan komunikasi *serial*. DirectPlay tidak mendukung *secure communication*.

2.3.4.1 DirectPlay Transport Protocol

Inti dari kemampuan jaringan Microsoft DirectPlay adalah pada *DirectPlay protocol*. *Transport-layer protocol* ini secara keseluruhan telah tercakup dalam DirectPlay8, dan saat ini dipergunakan untuk keseluruhan aktifitas pengiriman pesan. *DirectPlay protocol* difokuskan pada penyederhanaan aktifitas pengiriman data dari aplikasi pengiriman kepada aplikasi target, tanpa perlu mengkhawatirkan apa yang terjadi diantaranya. Protocol ini menawarkan sejumlah fitur yang diperlukan untuk mengembangkan *multiplayer games*, termasuk:

- *Reliable* dan *Unreliable message*. *Reliable message* akan dikirimkan kembali hingga aplikasi target menerima mereka. Kita dapat menggunakan tipe pengiriman dalam sebuah *message-by-message basis*.
- *Sequential* dan *non-sequential* pengiriman dari suatu *message*. *Sequential messages* akan dikirimkan pada aplikasi target application dalam susunan sebagaimanan saat mereka dikirimkan.
- *Message fragmentation* dan *reassembly*. Jika ukuran *message* melampaui kapasitas dari suatu *particular network*, *DirectPlay* secara otomatis melakukan *fragments* dan melakukan *reassembles* pada *message*.
- *Congestion control*. *DirectPlay* secara otomatis melakukan *throttles* pada *outgoing messages* ke level yang dapat dikendalikan oleh *target*.

Feature ini mencegah kita membanjiri *target* dengan sejumlah *message* melebihi kemampuan *target* untuk memproses.

- *Send prioritization.* Untuk memastikan message yang mana yang harus dikirimkan terlebih dahulu, *DirectPlay* mengijinkan kita untuk mendesain message sebagai *low, medium,* atau *high priority.* Untuk *high priority messages* dikirimkan melalui bagian depan dari *output queue,* baru diikuti dengan *medium* dan *low priority messages.*
- *Message timeouts.* Untuk mencegah *outgoing message queue* menjadi berjubel dengan message yang telah dikirimkan, *DirectPlay* mengijinkan kita untuk meng-*assign* sebuah *timeout value* pada seluruh *message.* Ketika sebuah *message times out,* maka ia dipindahkan dari *outgoing message queue,* untuk menandai apakah *message* tersebut telah dikirimkan atau belum.

2.3.4.2 DirectPlay Addresses

Dalam rangka mengirimkan *message,* masing-masing yang terlibat dalam sebuah multiplayer game harus memiliki sebuah *unique address. Address* dapat mengarah pada komputer dimana aplikasi berjalan (*device address*), atau sebuah komputer dimana aplikasi perlu untuk melakukan komunikasi (*host address*).

Microsoft DirectPlay addresses adalah dalam format dari *URL strings.* Strings ini terdiri atas sebuah *scheme, scheme separator,* dan *data string* dalam general format. Data string berisi beberapa elemen yang menspesifikasikan segala hal yang diperlukan untuk mengijinkan terjadinya komunikasi antara pengirim dan target, melalui variasi dari beberapa tipe yang berbeda dari suatu *network link.*

Dalam penggunaan, *URL strings* tercakup dalam sebuah *DirectPlay address object* yang melalui metode DirectPlay API. Terdapat beberapa pilihan dalam memanipulasi URL string, secara langsung atau menggunakan metode yang disediakan dalam obyek *Address* untuk menangani masing-masing elemen dari *data string* secara terpisah.

2.3.5 Komunikasi dengan Obyek DirectPlay

Microsoft DirectPlay secara esensial terdiri atas kumpulan dari obyek COM. Masing-masing obyek terdiri dari satu atau lebih *interface* yang mengijinkan kita untuk melakukan kontrol pada berbagai aspek yang bervariasi dalam DirectPlay. Misal, *DirectPlay peer object (CLSID_DirectPlay8Peer)* digunakan untuk mengatur *peer-to-peer games.*

Kita berkomunikasi dengan suatu obyek DirectPlay dengan memanggil metode yang telah disediakan. Misal, untuk mengirimkan beberapa data pada *user* yang lain dalam sebuah *peer-to-peer game*, kita akan mengirimkan sebuah *message* dengan memanggil method `DirectPlay8Peer.SendTo`. Sehingga DirectPlay akan mengatur bagaimana cara mendapatkan *message* untuk *target.*

DirectPlay berkomunikasi dengan aplikasi melalui satu atau lebih *message handler*. Sebuah *message handler* adalah suatu obyek yang dipanggil oleh DirectPlay untuk memperingatkan aplikasi terhadap even yang bervariasi. Dokumentasi ini mendeskripsikan method yang digunakan oleh obyek ini, namun kita harus mengimplementasikan seluruh metode obyek pada aplikasi. Kemudian registrasikan obyek tersebut pada saat *startup*, dan DirectPlay akan memanggil metode-metode untuk memperingatkan ketika sebuah *event* terjadi. Informasi

tambahan tentang sebuah event disampaikan melalui parameter yang tersedia dalam metode tersebut.

2.3.6 Komunikasi DirectPlay Voice

Tren mendatang *multiplayer games* berbasis tim membuat komunikasi *player-to-player* sebagai bagian yang esensial dari suatu permainan *game*. Meskipun sesuai untuk proses yang lebih lambat, *turn-based games*, *text-based communication* adalah yang terbaik meskipun tidak menyenangkan untuk *real-time games*. Tidak hanya menampilkan efek dari pengetikan yang lambat sebagai kekurangan pada saat permainan *game* namun juga merupakan suatu *significant break* dalam realitas dimana *game* dibuat untuk pemain. Jelas sekali solusi dari permasalahan tersebut adalah penggunaan pembicaraan untuk komunikasi. Hal ini tidak diperlukan training dan *increases the immersion* dari *game* itu sendiri.

Microsoft Windows *platform* menyediakan seluruh *tools* yang diperlukan untuk menyediakan *real-time voice conferencing* untuk pengembangan *video game*, tapi diperlukan sejumlah usaha dalam bagian pengembangan *game*. Hal ini, merupakan gabungan antar biaya dan tingkat kesulitan yang harus ditempuh dalam mencapai *compression technology* yang tepat dan mampu meng-*handle* kondisi *extremely low bandwidth.*

2.4 DirectPlay Voice

Microsoft DirectPlay Voice menggunakan suatu DirectPlay *session* sebagai media transport jaringan dan pengelolaan *player*. DirectPlay Voice API tidak melakukan duplikasi pada fitur-fitur *session control* dari *DirectPlay*. Sebuah DirectPlay *network transport session* juga harus dibuat sebelum DirectPlay Voice

dapat mentransmisikan dan menerima komunikasi *voice*. DirectPlay Voice dapat menggunakan salah satu dari obyek IDirectPlay4 atau obyek IDirectPlay8 untuk transportasi jaringannya.

Catatan bahwa jika *DirectPlay Voice* sedang digunakan dalam proses dengan suatu *multiplayer game*, *game* akan juga menggunakan *transport session* untuk bertukar data *game-specific*. Hal ini membuatnya mungkin untuk mengoptimalkan penggunaan resource jaringan antara *game* dan *voice data*.

Hal itu juga dapat diterima untuk membuat dan menggunakan suatu *transport session* secara spesifik untuk *voice session*, begitu halnya untuk kasus aplikasi *standalone voice conferencing*. Microsoft DirectPlay Voice adalah sebuah *full-voice communications* API yang terintegrasi dengan DirectPlay untuk mengelola *network session* dan *network transport*.

DirectPlay Voice juga diintegrasikan dengan DirectPlay Sound untuk perekaman suara dan *playback*, dan semua fitur audio DirectPlay Sound adalah *inherit*, termasuk kemampuannya pada *target voice data* pada playback buffers yang berbeda dan penggunaan dari *special audio effects* semacam *three-dimensional sound positioning*.

2.4.1 DirectPlay Voice Topologies

Microsoft DirectPlay Voice *session* membutuhkan suatu DirectPlay *network session* untuk komunikasi suara. Sekali *network session* dibuat, sebuah obyek DirectPlay Voice dapat dibuat dengan menggunakan satu dari tiga topologi.

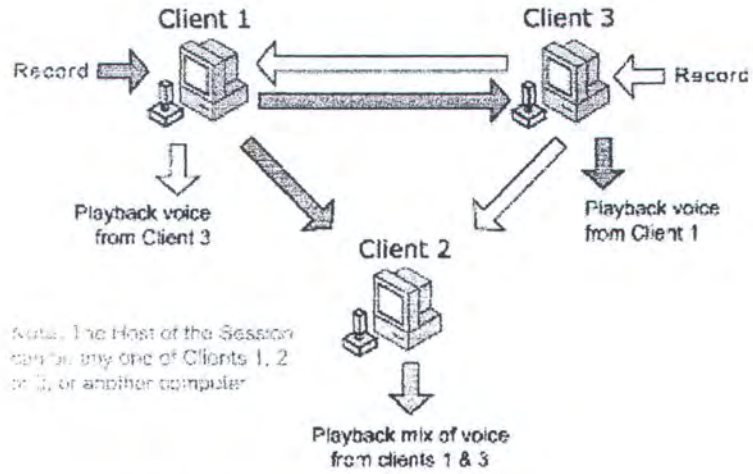
- Peer-to-Peer Voice Topology

- Forwarding Server Voice Topology
- Mixing Server Sessions Topology

Catatan bahwa tidak semua *voice topologi* dapat dikirimkan melalui semua tipe dari *DirectPlay networking sessions.*

2.4.1.1 Peer-to-Peer Voice Topology

Dalam suatu Microsoft DirectPlay Voice *session* menggunakan topologi *peer-to-peer*, masing-masing *voice-session client streams* data *voice audio* secara langsung pada masing-masing *voice-session client* yang lain. Masing-masing *client* menerima semua *individual incoming voice audio streams*, menggabungkan *received streams*, dan memainkan hasil pencampuran sinyal pada client's computer.



Gambar 2.17. Topologi Peer-to-peer Voice Session

Manfaat dari penggunaan suatu topologi *peer-to-peer* adalah tidak ada komputer dalam *voice session* yang membutuhkan *high bandwidth* atau *processor power.* Namun, penggunaan *bandwidth* dan *processor* pada masing-masing komputer klien bervariasi tergantung pada jumlah *incoming* dan *outgoing audio*

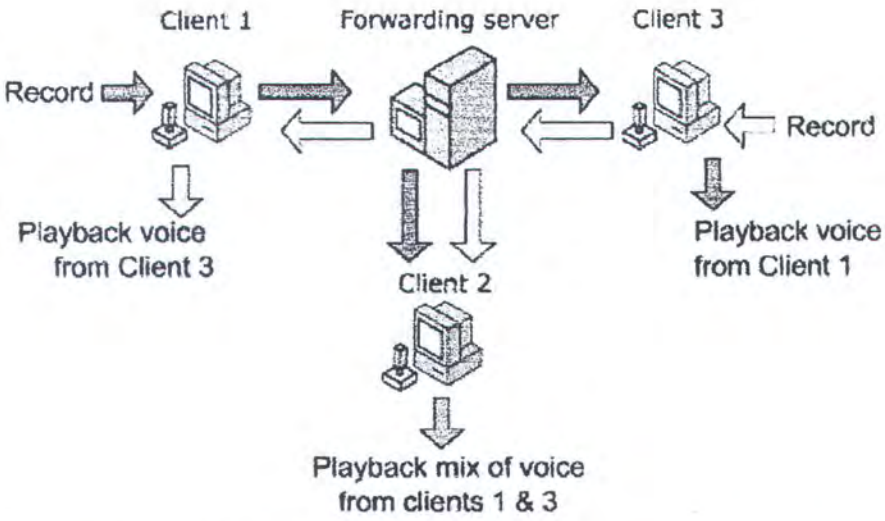
streams. Jumlah dari *outgoing voice audio streams* adalah sama dengan jumlah *target* yang berpartisipasi dalam *voice session*, kecuali jika *network provider* mampu dalam melakukan *multicasting*, seperti penjelasan dibawah ini. Jumlah *voice audio streams* yang masuk tergantung pada berapa banyak *voice-session client* yang menjadi target untuk *client in question* dan berapa banyak klien lain yang sedang berbicara.

Sebagai pertimbangan dalam mendesain *game*, tidak berguna bagi *voice-session client* menjadi *target* dari lebih dari sekitar enam hingga delapan klien yang lain. Jika semua enam hingga delapan klien berbicara secara bersamaan, maka pembicaraan akan menjadi membingungkan, dan komunikasi antar klien dapat menyulitkan.

Jika *DirectPlay network session* mendukung *true multicasting*, jumlah dari *outgoing voice audio streams* dapat dikurangi secara bertahap. Jika semua klien adalah bagian dari suatu *multicast network* dan *target* dari *voice stream* adalah suatu *DirectPlay group*, maka hanya ada satu *outgoing stream*.

2.4.1.2 Forwarding Server Voice Topology

Dalam topologi *forwarding server*, satu komputer dalam *session* bertindak sebagai sebuah *forwarding server*. Masing-masing klien dalam *voice session* melakukan *streaming voice data* kepada *forwarding server*, kemudian *forwarding server* men-*forward voice data* pada semua klien lain dalam *session*. Masing-masing klien menerima semua *incoming audio streams* yang di-*forward* dari *forwarding server*. Masing-masing komputer klien kemudian mencampur *incoming streams* dan memainkan mereka kembali.



Gambar 2.18. Topologi Forwarding Server Voice Session

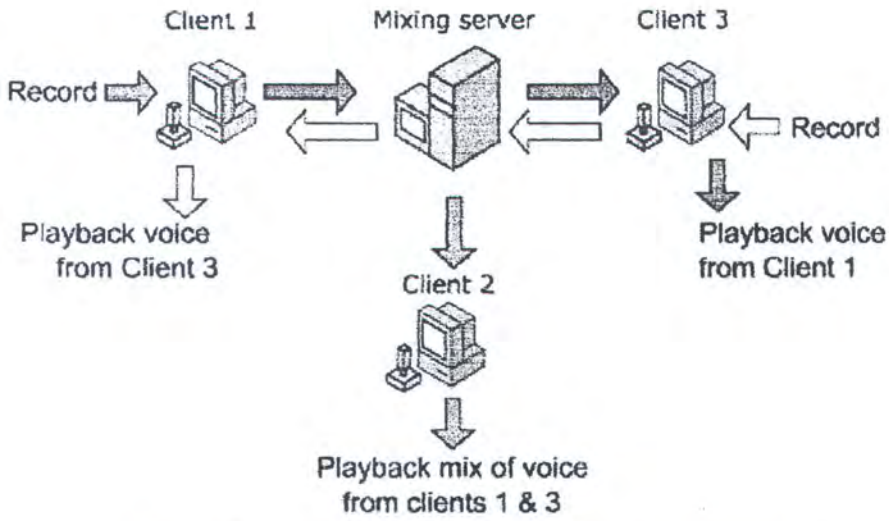
Outgoing bandwidth yang dibutuhkan pada masing-masing klien dalam sebuah *voice session* menggunakan sebuah *forwarding server topology* adalah konstan karena hanya ada satu *outgoing voice audio stream. Incoming bandwidth* dan *processor requirements* identik pada *requirement* dari suatu *voice session* menggunakan topologi *peer-to-peer*, tetapi ia sangat tergantung pada jumlah dari *incoming voice audio streams.*

Server memiliki banyak *bandwidth requirements* yang lebih tinggi dibandingkan *individual clients* dalam sebuah *forwarding server DirectPlay voice session.* Namun, *processor requirements* tidak tinggi karena tidak ada kompresi atau dekompresi dari *voice data* yang terjadi dalam server. Hal ini mengurangi *load* dalam prosesor komputer yang juga berarti bahwa suatu individu komputer klien dengan sebuah koneksi *high bandwidth* dapat menjadi *host* bagi *forwarding server* tanpa menunjukkan efek dari kinerja individu komputer klien atau kinerja dari suatu *game server* dan/atau *client program* yang berjalan pada komputer yang sama.

Catatan bahwa dalam sebuah *voice session* yang menggunakan topologi *peer-to-peer*, *outgoing bandwidth* yang diperlukan pada individu klien pada umumnya lebih tinggi dari pada yang diperlukan pada *incoming bandwidth*. Karena itu, mengurangi kebutuhan *outgoing bandwidth* pada suatu *single stream* dari audio dapat menghasilkan pengurangan yang signifikan dalam total penggunaan *bandwidth*. Contoh, jika sebuah klien mengambil bagian pada sebuah *eight-person voice session* dimana semua *clients* dapat mendengar satu sama lain, klien memiliki tujuh *outgoing voice streams* setiap kali *voice data* di-*captured* dan ditransmisikan pada komputer mereka. Namun, hal tersebut jarang sekali terjadi, dimana semua klien berbicara secara bersamaan, sehingga umumnya lebih sedikit dari dua atau tiga *incoming voice streams* yang terjadi pada satu waktu tertentu.

2.4.1.3 Mixing Server Sessions

Dalam *mixing server session*, satu komputer dalam suatu *session* bertindak sebagai *mixing server*. Masing-masing klien melakukan *stream* pada *voice data* ke *mixing server*. *Mixing server* menganalisa target dari masing-masing *voice stream*, melakukan *decompression*, *mixing*, dan *recompression* sehingga dianggap layak untuk melakukan *generate* sebuah *mixed stream* dari suatu *audio data* untuk masing-masing klien. Masing-masing klien menerima *single stream* dari *pre-mixed audio data* dan memutarnya kembali.



Gambar 2.19. Topologi Mixing Server Voice Session

Outgoing bandwidth, *incoming bandwidth*, dan CPU *requirement* pada klien dalam sebuah *mixing server session* adalah mudah untuk diprediksi karena masing-masing klien hanya memiliki satu *outgoing stream* berupa audio untuk di-*compress* dan dikirimkan, dan satu *incoming stream* berupa audio untuk di-*decompress* dan dimainkan kembali.

Mixing server memiliki lebih banyak *bandwidth* dan *CPU requirements* dari pada klien. Umumnya, *mixing server* adalah bukan sebuah *dedicated computer*, atau merupakan komputer dengan sebuah *dedicated game server*. *Mixing server voice session* tidak mendukung *3-D spatialization* dari *voice data* melalui metode `DirectPlayVoiceClient8.Create3DSoundBuffer`. Kita dapat menjalankan *mixing server voice session* menggunakan *peer-to-peer* atau sebuah *client/server transport session*.

2.4.2 Voice Host Migration

Dalam sebuah *peer-to-peer* Microsoft DirectPlay *network session*, suatu klien dari *networking session* bertindak sebagai *host*. Jika *host* tersebut keluar dari

session atau berhenti merespon untuk beberapa alasan, klien yang lain dalam *session* dipilih sebagai *host*.

Dalam sebuah DirectPlay *voice session*, sebuah proses yang mirip dengan *host migration* terjadi dalam *peer-to-peer voice sessions*, selain dari itu *voice host* bermigrasi secara independent dalam DirectPlay *network session*. *Voice host* bermigrasi ketika *server* memanggil `DirectPlayVoiceServer8.StopSession` atau jika *voice host* berhenti memberikan respon.

Ketika *voice host* bermigrasi, metode `DirectPlayVoiceEvent8.HostMigrated` memanggil masing-masing klien. Jika *local client* telah berubah menjadi *voice session host* yang baru, maka parameter *NewServer* akan mengarah pada obyek DirectPlayVoiceServer8 yang baru saja dibuat yang dapat digunakan oleh *local client* untuk menyedikan *host service*. Jika *local client* bukan *host* yang baru, maka parameter *NewServer* akan menjadi NULL.

2.4.3 Audio Device Testing

Microphone setup disupport oleh Microsoft DirectX Voice Test Wizard. Wizard ini mengkonfirmasikan bahwa sistem telah disupport dengan benar oleh operasi *full duplex* dan memastikan *microphone* dan *playback setting* sudah benar. Wizard hanya perlu dijalankan sekali untuk masing-masing kombinasi dari *playback* dan *capture device* yang dipilih.

Untuk menjalankan *Voice Test Wizard* terdapat dua cara:

- Menjalankan *wizard* melalui program yang dibuat dengan cara membuat sebuah obyek DirectPlayVoiceTest8 dan memanggil metode `DirectPlayVoiceTest8.CheckAudioSetup`.

- Dengan Microsoft Windows Millennium Edition (Windows Me) atau Microsoft Windows XP, *wizard* dapat dibuka melalui *Voice tab* dari *Sound* dan aplikasi *Audio Devices control panel.*

Memanggil metode `DirectPlayVoiceTest8.CheckAudioSetup` menimbulkan reaksi pada *Voice Test Wizard*, untuk menjalankan test dan memberikan konfirmasi bahwa sistem di-support oleh operasi full duplex dengan baik, dan memastikan bahwa *microphone* dan *playback settings* adalah tepat. Kita perlu menjalankan *wizard* hanya sekali untuk masing-masing kombinasi dari *playback* dan meng-*capture device* yang dipilih. Setiap kali aplikasi dimulai, kita harus melakukan test konfigurasi dengan cara memanggil `DirectPlayVoiceTest8.CheckAudioSetup` dengan parameter *lFlags* di-*set* pada `DVFLAGS_QUERYONLY`. Hal ini mengijinkan kita untuk melakukan test dengan cepat meskipun konfigurasi perlengkapan telah berubah sejak terakhir kali perlengkapan di-*test*. Jika perlengkapan belum di test, kita harus menjalankan `DirectPlayVoiceTest8.CheckAudioSetup` lagi untuk memunculkan *wizard*. Jika konfigurasi telah berubah sejak test terakhir dan kita tidak menjalankan *wizard* lagi, `DirectPlayVoiceClient8.Connect` akan mengembalikan `DVERR_RUNSETUP` dan kita tidak akan bisa menginisialisasi Microsoft DirectPlay Voice.

Banyak sistem komputer yang lebih tua yang masih digunakan tidak memiliki sebuah *full duplex sound card*. Tanpa *full duplex*, sebuah *sound card* dapat menerima tapi tidak dapat mengirimkan komunikasi *voice*. Karena *game* umumnya melakukan *hold* pada *sound card* dalam kondisi *playback mode*, DirectX 8.1 mencegah permasalahan tersebut dengan tidak mengijinkan *dynamic*

switching antara *playback* dan *capture*. *Voice Test Wizard* menyediakan *user* informasi tentang *duplexing abilities* dari sistem mereka.

2.4.4 Voice Codecs

Algoritma *Compression/decompression (codec)* yang disediakan dalam *Microsoft DirectPlay* adalah optimal untuk *low-bandwidth voice compression* dan *decompression*. Codec secara keseluruhan beroperasi pada *8 kHz, 16-bit mono-format based data*. Namun, *DirectPlay Voice* menangani semua detail dari mengkonversi *voice data* kedalam dan dari format *intermediate*. *Third-party codecs* tidak di-support, dan kita tidak dapat menulis *proprietary codecs* untuk digunakan dengan DirectPlay Voice.

Sangat penting untuk dicatat bahwa ketika *bandwidth requirements drop*, kualitas audio dari *voice data* juga drop. Tabel berikut me-*list* codec yang di-support, *bandwidth* dalam *kilobits per second* (Kbps), dan *compression* GUID digunakan untuk memilih mereka. *Compression* GUIDs didefinisikan dalam dvoice.h.

Tabel 2.3. Codec

[illegible]	[illegible]	[illegible]
Voxware VR12	variable (1.2 Kbps, avg.)	DPVCTGUID_VR12
Voxware SC03	3.2 Kbps	DPVCTGUID_SC03
Voxware SC06	6.4 Kbps	DPVCTGUID_SC06
TrueSpeech	8 Kbps	DPVCTGUID_TRUESPEECH
Microsoft® GSM	13 Kbps	DPVCTGUID_GSM
Microsoft® ADPCM	32 Kbps	DPVCTGUID_ADPCM
Microsoft® PCM	64 Kbps	DPVCTGUID_NONE

Tiga pertama *codec* menyediakan sebuah *high level* dari suatu *compression* dan secara *approximately* membutuhkan resource yang sama. Pada sebuah

komputer kelas 500 MHz Pentium III, codec ini menggunakan *approximately 1.5%* dari kapasitas CPU. *VR12 codec sounds tinny* dan *robotic*, tapi *SC03* dan *SC06 codecs* menyediakan *fidelity* yang masuk akal. *PCM codec* menyediakan kualiatas suara yang tinggi.

Catatan GSM, ADPCM, and PCM *codec* terdapat dalam instalasi Microsoft Windows tapi kebanyakan tidak di-*install* oleh *user*. Kita dapat men-determinasi *codec* yang mana yang ada pada sistem anda dengan cara memanggil `DirectPlayVoiceServer8.GetCompressionType`. jika sebuah codec tidak muncul, maka ACM *codec* yang terkait tidak di-install.

Dengan *game setup parameters* yang lain, *host* melakukan kontrol pada codec mana yang digunakan untuk *voice session*. Semua *anggota* dari *voice session* harus menggunakan codec yang sama. Ingat bahwa dalam sebuah *peer-to-peer voice session, voice-session host* tidak harus dengan segera memiliki *game-data host* yang sama. Host memilih *codec* ketika ia memanggil `DirectPlayVoiceServer8.StartSession`. Set *guidCT* dari tipe `DVSESSIONDESC` dalam *compression GUID* dari *codec* yang ingin digunakan. Seorang klien dapat melakukan *retrieve* pada struktur ini dengan cara memanggil `DirectPlayVoiceClient8.GetSessionDesc`.

Codec yang sama kemungkinan dapat berakibat tidak ideal bagi seluruh durasi dari game. Sebagai contoh, kita mungkin ingin menggunakan satu *codec* untuk *lobby chat feature* dimana pemain menggunakannya untuk men-*set up game*, dan yang lain untuk menangani komunikasi *voice* setelah *game*

diluncurkan. Kita tidak dapat secara dinamis mengubah *codec* ketika sedang terjadi *voice session*. Untuk melakukan *switch* dengan codec yang lain, kita harus melakukan terminasi pada *voice session* saat ini dan membuat sebuah *voice session* baru dengan *codec* yang baru. Namun, kita dapat berhenti dan memulai sebuah *voice session* tanpa melakukan terminasi pada DirectPlay *session*.

Dalam format komunikasi jaringan manapun, sangat penting untuk menganalisa biaya komunikasi *voice* untuk memastikan bahwa *adequate bandwidth* tersedia untuk mendukung komunikasi dari *game data* dan *voice data*. Menganalisa kebutuhan *voice bandwidth* adalah *straightforward*: mengestimasi jumlah dari *simultaneous voice streams* yang diantisipasi dan melipat gandakan jumlah tersebut dengan cara menjumlah kebutuhan *bandwidth* dari *codec* dan *protocol overhead*. *CPU consumption* adalah faktor lain yang perlu dipertimbangkan ketika memilih sebuah *codec*. Seperti halnya *network bandwidth*, *CPU resource consumption* akan meningkat setiap *stream*.

2.4.5 Automatic Gain Control

Microsoft DirectPlay Voice menawarkan kegunaan untuk menyesuaikan volume *hardware input* dalam *sound card* secara otomatis dan menyediakan *recording input* terbaik pada level yang memungkinkan. Untuk mengijinkan *Gain Control* secara otomatis, set *flag* `DVCLIENTCONFIG_AUTORECORDVOLUME` pada member *lFlags* dari struktur `DVCLIENTCONFIG` ketika kita mengatur konfigurasi klien. Secara otomatis *Gain Control* dapat diaktifkan atau dinonaktifkan setiap waktu ketika terjadi *voice session*.

Kebanyakan aplikasi *game* harus menggunakan *gain control* secara otomatis karena ia membutuhkan suatu *negligible amount* dari *game resources* dan mencegah kebutuhan dari sebuah *in-game volume recording control*. User tidak perlu men-set *level* sendiri, sehingga mereka mengalami kualitas yang lebih tinggi dari *voice transmission* dan *reception possible*.

2.4.6 Transmission Control

Untuk menjaga performansi yang dibutuhkan dari Microsoft DirectPlay Voice *low*, transmisikan *voice data* hanya ketika *user* sedang berbicara. Kita dapat mengontrol transmisi *voice data* melalui tiga cara.

- Push to Talk
- Manual Voice Activation
- Automatic Voice Activation

Kita memilih *transmission control method* yang mana yang akan digunakan ketika kita memanggil `DirectPlayVoiceClient8.Connect` untuk melakukan koneksi ke sebuah *voice session*. Untuk menspesifikasikan tipe *transmission control* yang ingin digunakan, set flag *dwFlags* dalam `DVCLIENTCONFIG`. Kita dapat menggantinya ketika terjadi *session* dengan memanggil `DirectPlayVoiceClient8.SetClientConfig` dan mengganti *flag setting*.

- ***Push to Talk***, *Push-to-Talk*, adalah secara analog ketika ditekan tombol Talk dalam sebuah *two-way radio*. Hal itu menambahkan *reality* pada *game genres* tertentu semacam *first-person shooters*. *Push-to-talk* membutuhkan *user* untuk secara aktif memilih ketika mereka ingin mentransmisikan *voice data*. Tidak

ada bahaya apapun selain *voice data* akan mengaktifkan transmisi. Sebagai tambahan, membutuhkan *user* untuk secara katif memilih ketika mereka ingin berbicara mengurangi *user* yang berbicara secara bersamaan.

Push-to-talk transmission control membutuhkan desain dan pengembangan lebih dari pada *voice activation.* umumnya, kita harus memiliki beberapa cara dalam mendeteksi ketika *user* telah memilih untuk mulai melakukan transmisi, umumnya dengan menekan sebuah tombol *controller* atau sebuah tombol pada keyboard. Ketika kita mendeteksi bahwa *user* ingin memulai pembicaraan, kita harus memulai transmisi dengan memanggil `DirectPlayVoiceClient8.SetTransmitTargets`, dan menyediakan sebuah array dari target *target IDs* yang akan menerima transmisi. Ketika kita mendeteksi bahwa *user* telah selesai, hentikan transmisi dengan memanggil `DirectPlayVoiceClient8.SetTransmit Targets` lagi, dengan *target array* di-set sebagai NULL .

Push-to-talk adalah *default transmission control method.* Ia mengijinkan kecuali jika anda secara ekplisit memilih mengaktifkan *voice* dengan men-set *flag* `DVCLIENTCONFIG_AUTOVOICEACTIVATED` atau `DVCLIENTCONFIG_ MANUALVOICEACTIVATED` dalam `DVCLIENTCONFIG`.

- ***Voice Activation***, dengan *voice activated transmission control, microphone input* secara konstan menganalisa untuk mendeterminasi apakah user sedang berbicara. ketika *input exceeds* sebuah *threshhold level*, *voice activation* di-trigger, dan user memulai transmisi. Secara ideal, transmisi dimulai ketika *user* mulai berbicara, dan berhenti ketika user selesai.

Voice activation adalah lebih sederhana bagi *user* dari pada *push-to-talk*, karena *user* hanya harus berbicara pada microphone. Hal itu juga lebih mudah pengkodingannya karena kita tidak perlu mendeteksi ketika user ingin memulai atau berhenti melakukan transmisi. Kita menspesifikasikan *voice activation* ketika *connect*, dan *transmission control* ditangani oleh sistem melalui titik tersebut. Namun, satu *drawback* dari *voice activation* adalah ketika ia menghasilkan transmisi yang tidak diinginkan. Sebagai tambahan dalam pembicaraan, transmisi dapat di-*trigger* dengan suara semacam *user* mengambil napas secara langsung pada microphone. *Low-quality microphones* dapat meningkatkan probabilitas dari transmisi yang tidak diinginkan.

Voice activation dapat dilakukan baik secara otomatis maupun secara manual. Untuk menspesifikasikan satu dari mode ini, *set* flag `DVCLIENTCONFIG_AUTOVOICEACTIVATED` atau `DVCLIENTCONFIG_MANUAL VOICEACTIVATED`, ketika terkoneksi pada *voice session.*

Automatic voice activation adalah metode *transmission control* yang lebih disarankan untuk kebanyakan aplikasi. Dalam mode ini, *threshold* untuk transmisi adalah dideterminasikan secara otomatis oleh sistem. *theshhold level* adalah *adaptive*, penyesuaian diri dilakukan secara otomatis pada sinyal *input*.

Dengan *manual voice activation*, kita harus secara eksplisit men-set sebuah *threshhold* ketika melakukan koneksi pada *voice session* dengan meng-*assign* sebuah *value* pada *dwThreshhold* dari `DVCLIENTCONFIG`. Sistem ini tidak akan mengubah *value* tersebut untuk anda. Jika kondisi

tersebut berubah, dan *threshhold value* adalah bukan *adequate*, kita harus memanggil `DirectPlayVoiceClient8.SetClientConfig` dan menspesifikasikan sebuah value baru untuk *dwThreshold*.

2.4.7 Jitter Buffers

Fitur Microsoft DirectPlay Voice adalah *jitter buffer*, yaitu sebuah algoritma *adaptive buffering* yang menyediakan kualitas suara optimal dengan jumlah *latency* yang minimum. Pada jaringan yang sibuk, *individual packets* dari informasi *voice data* kemungkinan sampai dalam urutan yang berbeda dari tempat dimana mereka di-*encode* pada *host computer*. Karena *voice data* adalah *sequential in nature*, paket-paket yang datang ini harus diurutkan dalam periode waktu sehingga paket yang tertunda memiliki peluang untuk tiba dan dimainkan kembali dalam urutan yang benar.

Jika *jitter buffer* di-*set* untuk memaksimalkan kualitas dari komunikasi suara, hal itu membutuhkan waktu yang lebih lama untuk sejumlah *voice packet* untuk tiba dan diurutkan dan kemudian dimainkan. Hasilnya adalah *voice latency*, dan efeknya adalah komunikasi *voice* tidak dapat didengar dalam *real time*. Namun, *voice data* kemungkinan dapat didengar dari mana pun dari sebuah *fraction* dari satu detik ke beberapa detik setelah ia direkam. Hal ini dapat menimbulkan masalah ketika permainan *game* karena *event* dapat terjadi dalam *game* tetapi pemain tidak akan mampu mengkomunikasikan informasi berdasarkan pada event tersebut pada *real time*. Misal, jika seorang pemain dalam sebuah *first-person shooter* bermaksud untuk menyerang dari belakang dan

seorang rekan se-tim bermaksud memperingatkan pemain, komunikasi *voice* dapat didengar hingga setelah pemain diserang.

Jika *jitter buffer* di-set untuk mengurangi *latency*, jumlah dari paket yang dibutuhkan untuk mengisi *queue* akan berkurang. Namun, kemungkinan tidak semua *sequential packets* akan tiba tepat pada waktunya dan, sebagai hasilnya, *voice data* akan hilang dari *buffer* ketika ia dimainkan. Komunikasi *voice* akan terdengar lebih dekat dengan waktu sebenarnya ketika ia direkam.

DirectPlay *jitter buffer* menggunakan dua metode untuk mendeterminasi bagaimana menyediakan kualitas terbaik dari komunikasi *voice* dengan *latency* minimum. Pertama, kondisi jaringan dimonitor untuk mendeterminasi jumlah dari *lag* atau *network congestion*. Ukuran dari *jitter buffer*, atau *queue*, adalah dinamis untuk menjaga *latency* serendah mungkin ketika menyediakan jumlah minimum dari *voice break up*.

Tingkah laku umum dari DirectPlay Voice *jitter buffer* adalah secara otomatis menyesuaikan kondisi jaringan. Kita dapat secara manual menyesuaikan seberapa dekat algoritma *tracks network conditions* menggunakan `lBufferAggressiveness` dan `lBufferQuality` dari tipe `DVCLIENTCONFIG`. Level lebih tinggi dari `lBufferAggressiveness`, adalah algoritma yang lebih mendekati *monitors network conditions*. Secara umum, semakin tinggi nilai `lBufferQuality`, maka semakin tinggi pula kualitas *voice*, namun juga semakin tinggi *latency*. Semakin rendah kualitas *value*, semakin rendah *latency* tapi juga semakin rendah kualitas dari *voice*.

2.4.8 Working Set Guidelines

Mendeterminasikan konfigurasi terbaik dari *transport topology*, *voice topology*, *transmission control*, dan *codec* sangat terganting pada tipe atau *genre* dari game yang dibuat, jumlah pemain yang akan berpartisipasi dalam sebuah *single game session*, dan tipe koneksi sebagai targetnya.

Sangat penting untuk dicatat bahwa jumlah pemain yang berpartisipasi dalam *voice session* tidak sepenting jumlah pemain yang secara actual berpartisipasi dalam *game session*. misal, jika game yang dibuat adalah sebuah *first-person shooter*, komunikasi *voice* dapat direpresentasikan dalam *game* sebagai sebuah *radio* atau *communicator* yang ditawarkan sebagai sebuah *time-limited powerup*. juga, *radio metaphor* dapat digunakan untuk membatasi komunikasi melalui radio baik mesin atau stationary.

Contoh kedua sebagai pertimbangan adalah sebuah *online bridge game*, yang melibatkan empat pemain dalam satu waktu. Karena ini merupakan sebuah *small working set*, maka ia sangat tepat untuk memilih menggunakan topologi *peer-to-peer voice* yang ditransportasikan melalui sebuah *peer-to-peer network topology*. *Small working set* ini juga mengijinkan penggunaan *voice activation* sebagai mode dari *transmission control*. Topologi *Peer-to-peer voice* adalah mudah untuk diimplementasikan dan tidak membutuhkan pemain manapun untuk bertindak sebagai sebuah server. Jika keempat pemain menggunakan *Voxware SC6 codec*, menghasilkan *bandwidth* maksimum sebesar *4.2 Kbps per speech stream*, termasuk *codec protocol overhead*. Lebih jauh asumsikan bahwa *game data* membutuhkan *negligible bandwidth*, *Outgoing maximum bandwidth* yang

dibutuhkan untuk sebuah *individual speaker* adalah tiga stream independen untuk tiga pemain yang lain, atau *12.6 Kbps*. Range incoming stream untuk sembarang pemain adalah 0 juka tidak ada pemain lain yang sedang berbicara, hingga 12.6 Kbps, jika tiga pemain yang lain berbicara scara simultan. Kebutuhan CPU adalah 8 persen untuk *encoding* dan 0 hingga 12 persen untuk *decoding*. Hal ini menghasilkan sebuah kebutuhan *worst-case* dari *25.2 Kbps*. Karena itu masing-masing pemain harus memiliki minimum sebesar 14,400-*baud modem*.

Contoh lain adalah sebuah *squad combat game* yang dapat melibatkan hingga 32 pemain yang dibagi dalam 2 team. Asumsikan bahwa *game data* membutuhkan 28,800 *baud modem*. Dalam contoh ini, terdapat pemain lebih besar dan sangat tepat untuk memilih topologi *forwarding server voice*. Kemudian, jika semua pemain menggunakan *Voxware SC6 codec*, *bandwidth* yang dibutuhkan adalah sama dengan *bridge game* diatas: *4.2 Kbps*. Pada contoh ini kita melihat bahwa terdapat *4.2 Kbps outgoing* ketika berbicara, dan maximum *12.6 Kbps incoming* dari *squad*. Maximum kebutuhan CPU adalah 8 persen dari sebuah *Pentium 200* untuk *encode* dan 12 persen *receiving*. Karena itu, masing-masing pemain membutuhkan 28.8 Kbps untuk *game data*, dan *incoming bandwidth* yang lebih besar dari 12.6 Kbps dibutuhkan sebuah minimum 41,400 *baud rate* dari masing-masing *player's modem*.

Skenario *worst-case* untuk *forwarding server* sendiri adalah jika 32 pemain berbicara secara bersamaan, membutuhkan 134.4 Kbps. Penggunaan server CPU adalah minimal karena server tidak melakukan *encoding* atau *decoding* pada

streams. Ia lebih memilih melakukan *redirect* pada mereka. Umumnya, kemungkinan 16 pemain berbicara secara simultan dalam 67.2 Kbps.

Untuk mengilustrasikan perbedaan antara menggunakan sebuah topologi *mixing server voice* dan sebuah topologi *forwarding server voice*, pertimbangkan *32-player squad combat game* yang sama yang didiskusikan diatas. Jika sebuah topologi *mixing server voice* digunakan, masing-masing klien membutuhkan 4.2 Kbps untuk mengirim dan 4.2 Kbps untuk menerima. *Worst-case bandwidth* yang dibutuhkan menurun hingga 8.4 Kbps dan 12 persen dari prosesor Pentium 200 MHz. Hal ini mengurangi *modem requirement* pada sebuah 33,600 Kbps *baud rate* untuk klien.

Untuk server, *CPU burden* berubah. Server sekarang melakukan *decoding* dan *re-encoding* semua *incoming streams* dan juga melakukan *mixing* pada *streams* sebagaimana kebutuhan. *CPU burden* melakukan *mix* pada *stream* adalah secara relatif rendah dan diperkirakan negligible. Worst case-nya adalah *decoding* dan *encoding* dari 32 *simultaneous streams*. Hal ini menghasilkan kebutuhan setidaknya pada prosessor Pentium II 400 MHz untuk *voice service alone*.

2.5 Dual Tone Multiple Frequency

DTMF (*dual tone multi frequency*) adalah sinyal ke perusahaan telpon yang anda *generate* ketika anda menekan sebuah tombol telepon. Di Amerika dan kemungkinan dimanapun, hal itu dikenal sebagai *Touchtone* telepon (sebelumnya terdaftar sebagai merk dagang dari AT&T). DTMF umumnya mengganti *loop disconnect (pulse) dialling*. dengan DTMF, masing-masing tombol yang ditekan pada telpon akan menghasilkan dua *tone* pada frekuensi yang

spesifik. Sehingga sebuah voice dapat membuat tiruan dari *tone*, suatu *tone* dihasilkan dari sebuah *high-frequency group* dari *tone* dan yang lain dari sebuah *low frequency group*. Berikut ini adalah sinyal yang kirimkan ketika anda menekan tombol telepon:

Tabel 2.4. Frekuensi Sinyal DTMF

Digit	Low frequency	High frequency
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
0	941	1336
*	941	1209
#	941	1477

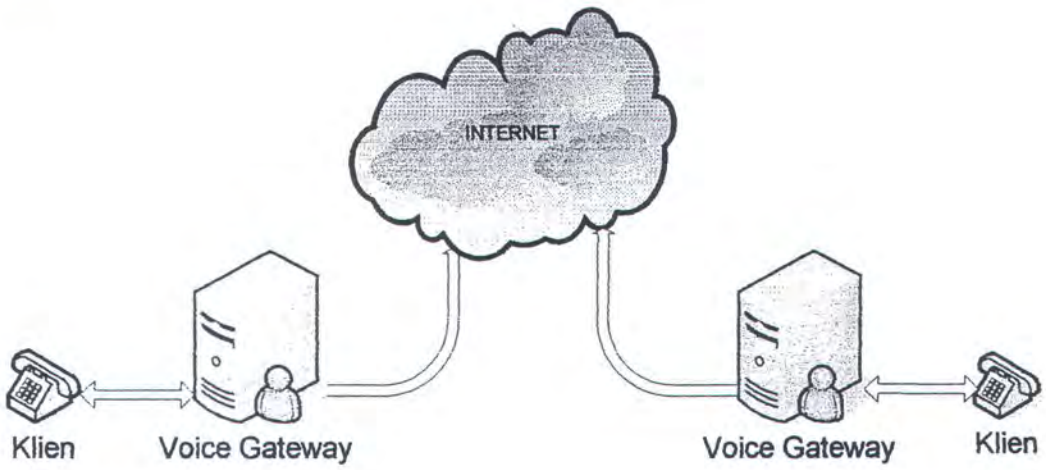
Sejumlah perusahaan membuat *microchips* yang mengirim dan menerima sinyal DTMF. Dalam tugas akhir ini, penulis menggunakan rangkaian elektronika dengan ic CM8870PI untuk mengenali sinyal DTMF yang diterima. Lebih jelasnya tentang pengenalan DTMF bisa dilihat pada Sub bab 3.1.1.2 tentang Tone Decoder.

BAB III

PERANCANGAN

3.1 Arsitektur Voice Gateway

Konfigurasi sistem yang dibuat oleh penulis memanfaatkan fasilitas PSTN pada kedua sub sistem terminalnya. Konfigurasi ini juga membutuhkan gateway sebagai antar muka yang menghubungkan antara PSTN dan jaringan internet. Di dalam gateway sendiri dibutuhkan sebuah sistem tambahan yang berfungsi untuk memetakan nomor panggilan ke kode-kode IP, atau biasa dikenal sebagai *Call Manager*. Berikut adalah ilustrasi dari konfigurasi sistem yang dibuat oleh penulis.



Gambar 3.1. Konfigurasi Sistem Voice Gateway

Komponen-komponen yang dibutuhkan untuk menyusun suatu sistem Voice Gateway adalah sebagai berikut :

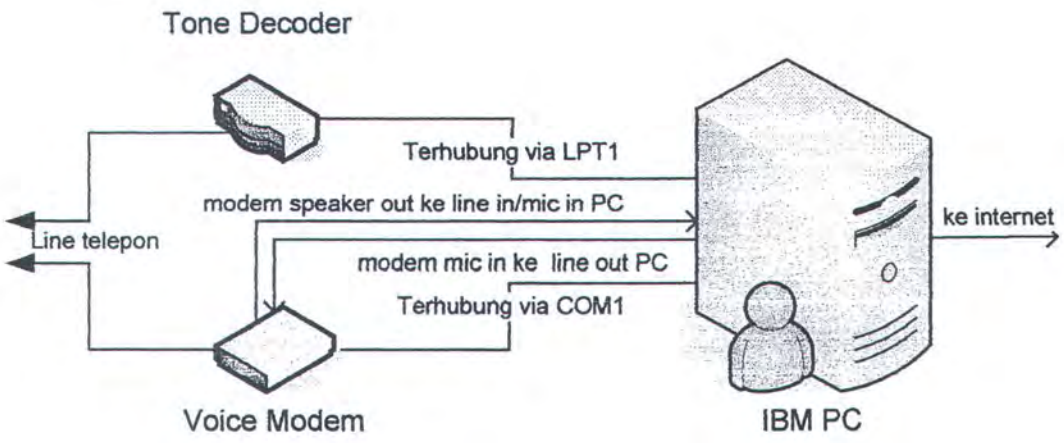
- PSTN, adalah jaringan telepon konvensional yang telah ada terlebih dahulu.

- Voice Gateway, adalah sebuah PC IBM Compatible yang diberi kemampuan bertindak sebagai sebuah VoIP Gateway dengan memanfaatkan perangkat keras tambahan berupa voice modem, soundcard, ethernet card dan tone decoder serta sebuah perangkat lunak.
- Jaringan berbasis protokol internet.

3.1.1 Arsitektur Perangkat Keras Voice Gateway

Perangkat keras yang digunakan dalam membangun suatu sistem Voice Gateway meliputi :

- Sebuah PC yang dilengkapi dengan sound card dan ethernet card
- Voice Modem.
- Tone Decoder.

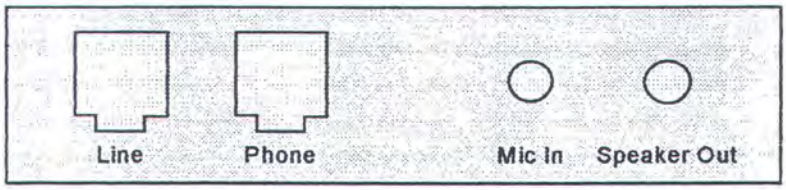


Gambar 3.2. Konfigurasi perangkat keras Voice Gateway

3.1.1.1 Voice Modem

Voice modem adalah modem yang menyediakan masukan dan keluaran berupa suara. Masukan atau keluaran ini dimanfaatkan oleh penulis sebagai alat

masukan/keluaran ke PC Gateway dengan menghubungkannya ke speaker out/line in yang terdapat didalam soundcard di PC Gateway. untuk menghubungkan antara soundcard PC Gateway dan voice modem, penulis menggunakan kabel audio.



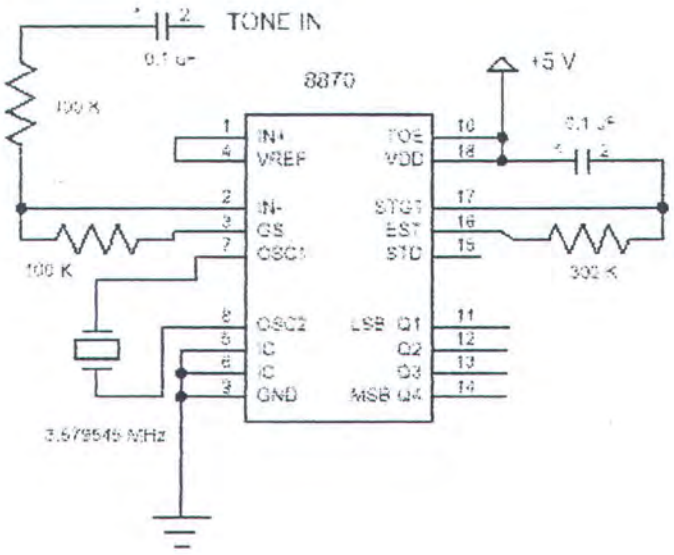
Gambar 3.3. Konfigurasi Voice Modem

Untuk mengendalikan voice modem, penulis menggunakan perintah-perintah AT yang berlaku untuk sebagian besar modem yang ada. Perintah AT adalah sekumpulan karakter yang dikirimkan oleh DTE (Data Terminal Equipment) ke modem. Sebuah perintah AT memiliki prefiks, body dan terminator. Prefiks berisi karakter-karakter ASCII **AT** atau **at**. Body terdiri dari sekumpulan kalimat-kalimat perintah yang terbatas pada karakter-karakter ASCII yang dapat dicetak saja. Sedangkan untuk terminator, yang biasa digunakan adalah karakter **<CR>** (carriage return).

3.1.1.2 Tone Decoder

Merupakan perangkat keras yang berfungsi untuk mengubah sinyal DTMF menjadi data digital 4 bit. Untuk mengubah sinyal DTMF menjadi data digital, penulis menggunakan IC CM8870PI yang mendapat masukan dari saluran telepon dan menghasilkan keluaran 4-bit. Keluaran ini nantinya akan dihubungkan ke port paralel DB25 agar dapat dibaca oleh komputer.

Berikut adalah gambar skema rangkaian tone decoder :



Gambar 3.4. Rangkaian Tone Decoder

Berikut adalah daftar komponen yang dibutuhkan untuk membuat tone decoder:

Tabel 3.1. Daftar komponen rangkaian Tone Decoder

No.	Nama Komponen	Jumlah
1.	IC CM8870	1 buah
2.	Kapasitor 0.1 μF	2 buah
3.	Resistor 100 K	2 buah
4.	Resistor 300 K	1 buah
5.	Crystal 3.579545 MHz	1 buah
6.	Catu Daya 5 Volt	1 buah

Berikut ini adalah tabel keluaran yang dihasilkan oleh rangkaian tone decoder.

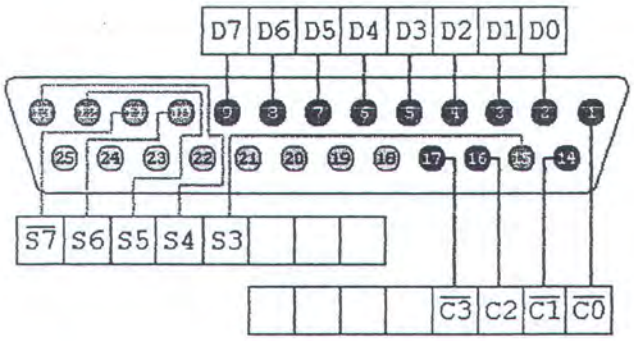
Tabel 3.2. Daftar keluaran rangkaian Tone Decoder

F_{LOW}	F_{HIGH}	KEY	TOW	Q4	Q3	Q2	Q1
697	1209	1	H	0	0	0	1
697	1336	2	H	0	0	1	0
697	1477	3	H	0	0	1	1
770	1209	4	H	0	1	0	0
770	1336	5	H	0	1	0	1
770	1477	6	H	0	1	1	0
852	1209	7	H	0	1	1	1
852	1336	8	H	1	0	0	0
852	1477	9	H	1	0	0	1

941	1209	0	H	1	0	1	0
941	1477	*	H	1	0	1	1
941	1336	#	H	1	1	0	0
697	1633	A	H	1	1	0	1
770	1633	B	H	1	1	1	0
852	1633	C	H	1	1	1	1
941	1633	D	H	0	0	0	0
-	-	ANY	L	Z	Z	Z	Z
L= Logic Low, H=Logic High, Z=High Impedance							

3.1.1.3 Port Paralel

Berikut adalah skema pin dari socket DB25 :



Gambar 3.5. Skema Pin dari paralel port

Secara garis besar lines dalam DB25 konektor dapat dibagi menjadi 3 kelompok, yaitu :

1. Data lines (Pin 2, 3, 4, 5, 6, 7 , 8, 9)
2. Control lines (Pin 10, 11, 12, 13, 15)
3. Status lines (Pin 1, 14, 16, 17)

Sebagaimana namanya, data dikirim melalui data lines, control lines digunakan untuk mengendalikan perangkat keras lainnya dan perangkat tersebut akan mengembalikan sinyal status kembali ke komputer melalui status line. Lebih detail tentang sinyal port paralel dapat dilihat pada tabel 3.3 berikut ini

Tabel 3.3. Fungsi pin-pin didalam port paralel

Pin No (DB25)	Signal name	Direction	Register - bit	Inverted
1	nStrobe	Out	Control-0	Yes
2	Data0	In/Out	Data-0	No
3	Data1	In/Out	Data-1	No
4	Data2	In/Out	Data-2	No
5	Data3	In/Out	Data-3	No
6	Data4	In/Out	Data-4	No
7	Data5	In/Out	Data-5	No
8	Data6	In/Out	Data-6	No
9	Data7	In/Out	Data-7	No
10	nAck	In	Status-6	No
11	Busy	In	Status-7	Yes
12	Paper-Out	In	Status-5	No
13	Select	In	Status-4	No
14	Linefeed	Out	Control-1	Yes
15	nError	In	Status-3	No
16	nInitialize	Out	Control-2	No
17	nSelect-Printer	Out	Control-3	Yes
18-25	Ground	-	-	

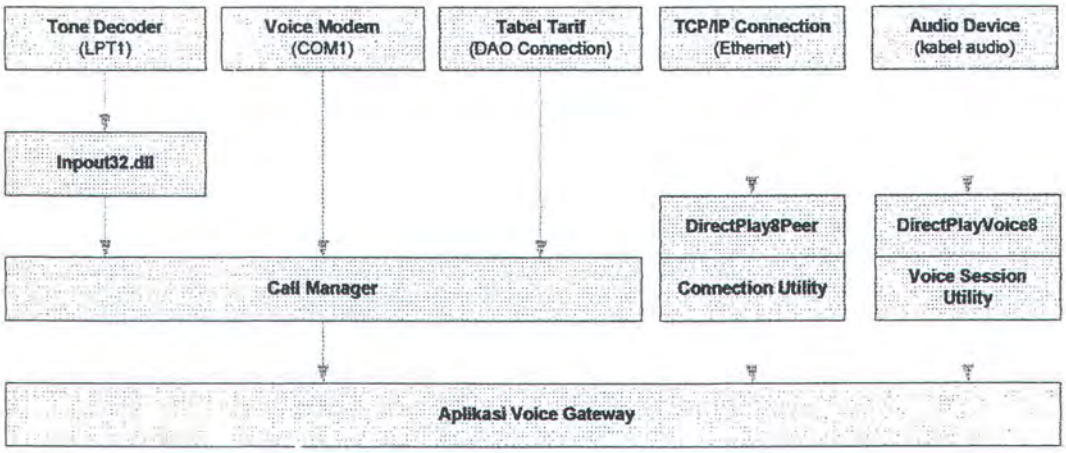
Sebagaimana diketahui, Data, Control dan Status lines terhubung kedalam register yang bersesuaian dikomputer. Sehingga dengan memanipulasi register-register ini didalam program, kita bisa secara mudah membaca atau menulis ke dalam port parallel dengan bahasa pemrograman. Sehubungan Visual Basic 6.0 tidak menyediakan akses langsung ke port paralel maka program yang dibuat untuk mengakses register menggunakan Visual C++ yang dikompilasi menjadi library. Setelah menjadi library maka Visual Basic akan dengan mudah memanfaatkan fungsi-fungsi didalam library tersebut.

Register yang ada dalam standar port paralel adalah data register, status register dan control register. Didalam IBM-PC, register-register ini dipetakan dan memiliki alamat yang unik. Untuk PC standard, alamat dasar untuk LPT1 adalah 0x378 dan untuk LPT2 adalah 0x278. Register untuk Data berada dialamat dasar

ini, status register berada pada baseaddress + 1 dan control register berada pada baseaddress + 2.

3.1.2 Arsitektur Perangkat Lunak

Dari gambar di bawah ini dapat dilihat skema sederhana perancangan perangkat lunak dan perangkat keras sistem Voice Gateway. Dari gambar ini pula dapat disusun apa saja yang harus terpasang, khususnya perangkat lunak untuk membentuk sistem Voice Gateway.



Gambar 3.6. Arsitektur Perangkat Lunak Sistem Voice Gateway

Perangkat lunak yang dibuat untuk sistem Voice Gateway nantinya adalah :

1. Call Manager, yang berfungsi untuk :
    - Membaca hasil output dari tone decoder
    - Mencari kode area yang diterima dan mengembalikan nilai alamat IP Voice Gateway tujuan.
    - Menjawab panggilan telepon yang masuk ke Voice Gateway.
    - Memutus panggilan yang sedang terjadi.
2. Connection Utility, yang berisi fungsi-fungsi untuk :

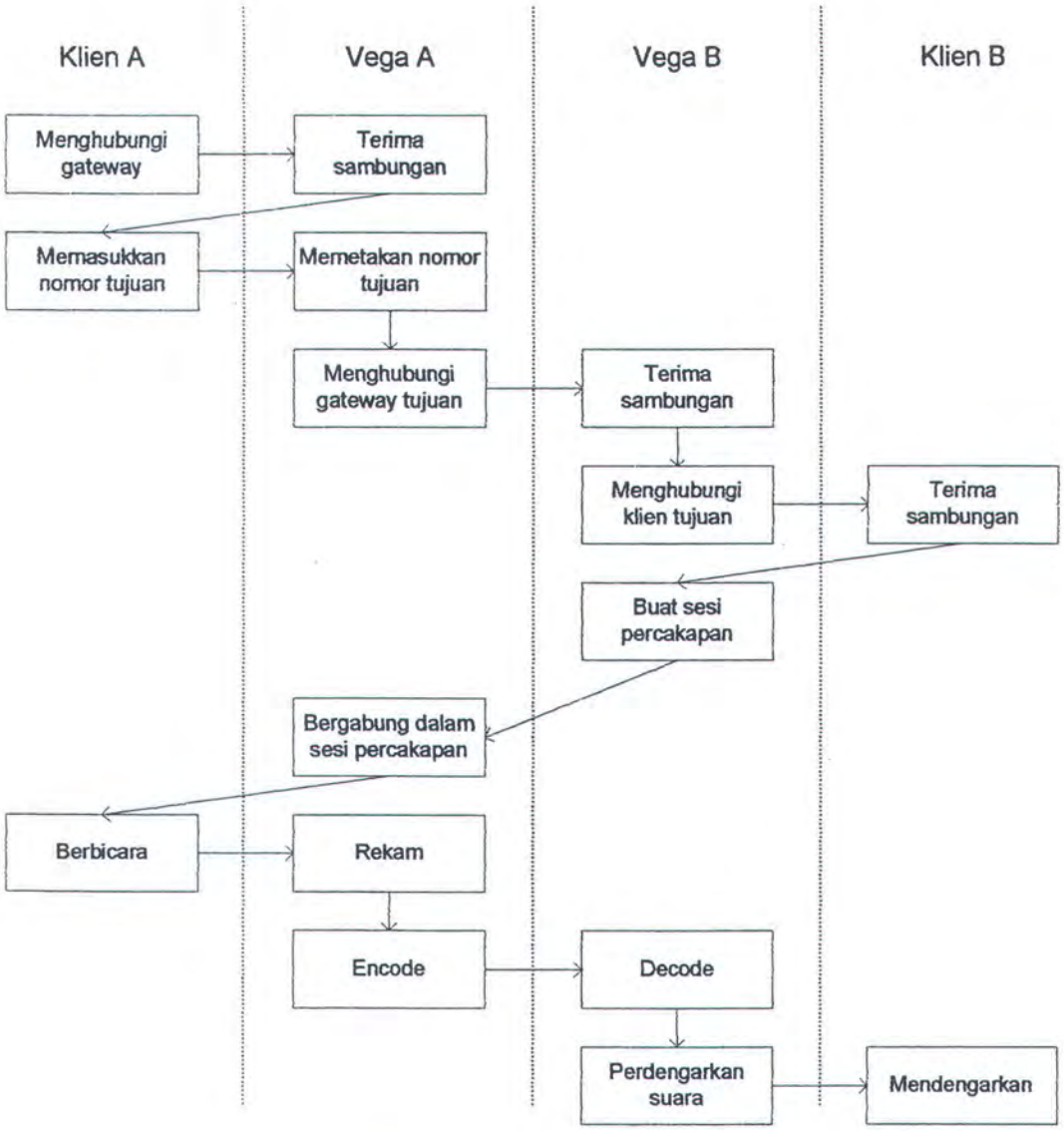
- Melakukan listening ke port tertentu untuk menerima koneksi yang masuk.
- Melakukan permintaan koneksi.
- Memberi respon terhadap setiap permintaan koneksi yang masuk.
- Memutus koneksi.
- Melakukan pertukaran data-data kondisi masing-masing gateway.

3. Voice Session Utility, yang berisi fungsi-fungsi untuk :
   - Membuat sesi percakapan dengan segala kemungkinan konfigurasi yang ada.
   - Bergabung dalam suatu sesi percakapan.
   - Memberi respon terhadap permintaan bergabung dalam suatu sesi percakapan.
   - Keluar dari suatu sesi percakapan.

3.2 Mekanisme

3.2.1 Alur Proses

Gambar 3.7 di bawah ini menerangkan alur kerja Sistem Voice Gateway secara garis besar. Untuk lebih jelasnya , lihat keterangan alur kerja Sistem Voice Gateway setelah gambar 3.7.



Gambar 3.7. Diagram Alir Cara Kerja Sistem Voice Gateway

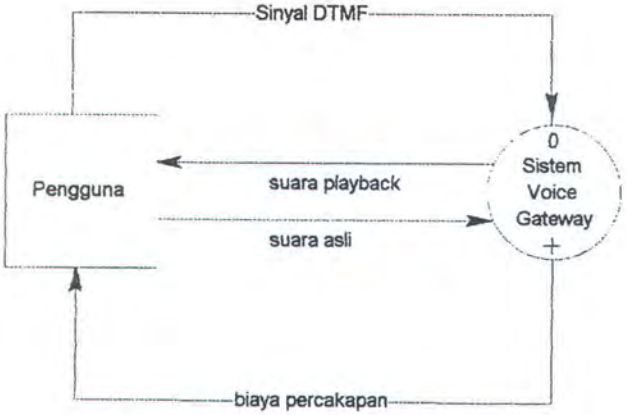
Keterangan :

1. Seorang pengguna (Klien A) yang ingin melakukan sambungan ke rekannya yang berada dikota lain harus terlebih dahulu menghubungi gateway lokal (Vega A) melalui telepon.
2. Gateway lokal akan menerima sambungan dari klien A kemudian menantikan masukan nomor tujuan (Klien B) yang akan dihubungi.

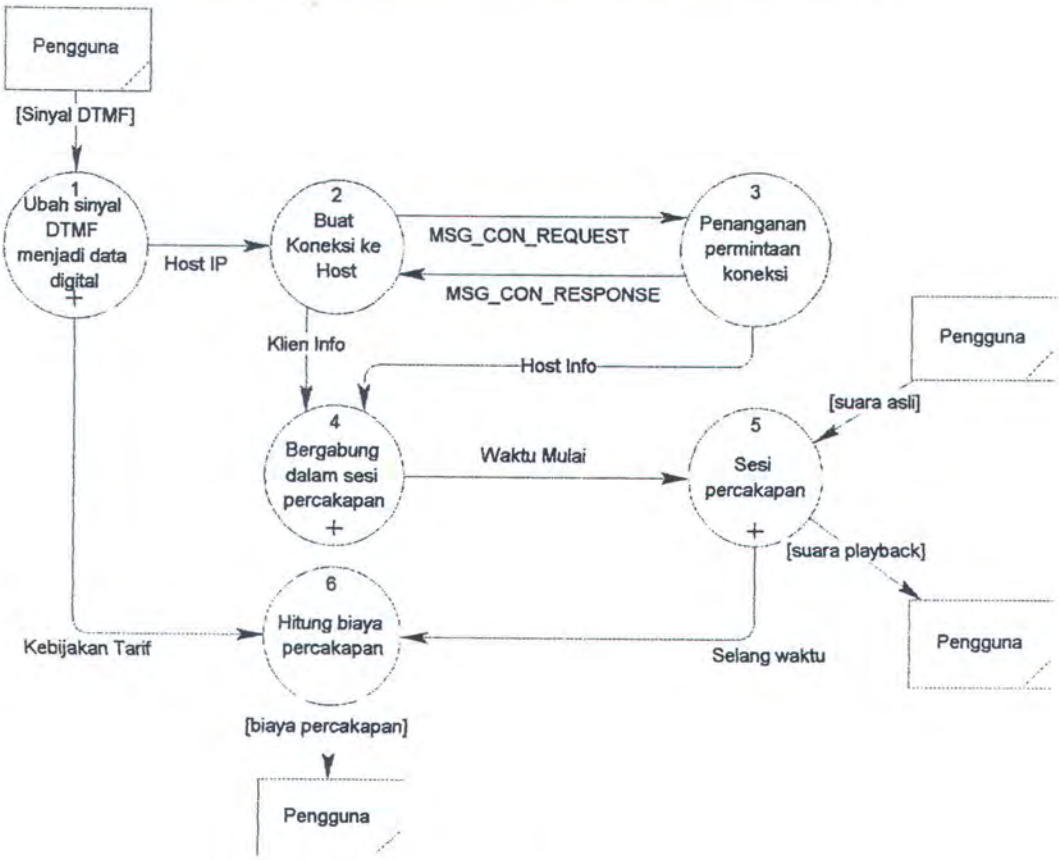
3. Klien A memasukkan nomor tujuan dengan cara menekan tombol telepon dengan format XXX*YYYYYYY# dimana XXX diisi dengan kode kota tujuan dan YYYYYYY diisi dengan nomor telepon klien B yang kemudian diakhiri dengan tanda pagar "#".
4. Setelah menerima masukan dari klien A, kode area akan dipetakan menjadi kode IP gateway yang bersesuaian dan kemudian gateway lokal akan melakukan permintaan sambungan ke gateway tujuan sembari mengirimkan nomor telepon tujuan.
5. Setelah menerima permintaan sambungan dari Vega A, gateway akan menghubungi nomor yang dikirimkan dari Vega A. Jika sukses maka Vega B akan membuat sebuah sesi percakapan dan memberitahukan ke Vega A agar Vega A bergabung dalam sesi percakapan yang dibuat.
6. Setelah Vega A bergabung didalam sesi percakapan maka dialog antar pengguna bisa dilakukan dimana pada saat klien A berbicara maka Vega A akan merekamnya, melakukan encode kemudian mengirimkannya ke Vega B. Setelah tiba di Vega B maka data tadi akan di decode dan kemudian dimainkan untuk diperdengarkan kepada klien B. Begitu pula sebaliknya.

3.2.2 Alur Data

Dengan memperhatikan gambar di bawah dapat terlihat bahwa entitas luar yang terlibat hanyalah pengguna saja, karena sistem Voice Gateway yang dibuat disini bekerja secara otomatis seperti halnya sebuah gateway.



Gambar 3.8. Context Diagram Sistem Voice Gateway



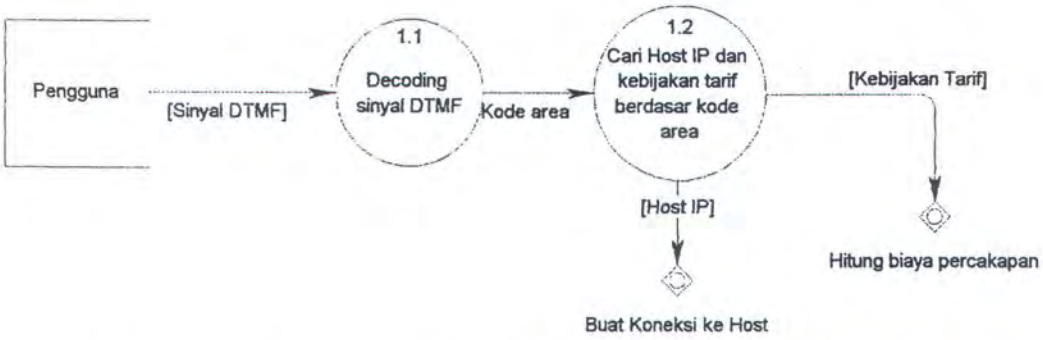
Gambar 3.9. Sistem Voice Gateway

Gambar 3.9 menggambarkan diagram Sistem Voice Gateway level 1, dimana di dalam diagram tersebut ada beberapa proses yang terjadi, diantaranya adalah :

1. *Proses Ubah sinyal DTMF menjadi data digital*, proses ini bertujuan untuk merubah sinyal DTMF yang masuk dari line telepon yang kemudian diteruskan oleh tone decoder dan kemudian dikirim ke komputer melalui port paralel. Setelah diterima oleh komputer, sinyal ini akan dipecah menjadi dua, yaitu kode area dan nomor tujuan yang hendak dituju. Kode area digunakan sebagai kunci untuk mencari alamat IP dari host yang hendak dituju dan tarif dasar yang berlaku untuk percakapan ke host tersebut.
2. *Proses Buat Koneksi ke Host*, adalah proses untuk membuat sambungan ke Gateway tujuan. Pihak penghubung disebut sebagai *Guest* sedangkan pihak yang dituju disebut sebagai *Host.*
3. *Proses Penanganan Permintaan Koneksi*, adalah proses yang terjadi di Host ketika ia menerima permintaan untuk melakukan sambungan percakapan. Host berhak untuk menentukan apakah permintaan yang masuk diterima atau ditolak.
4. *Proses Bergabung dalam Sesi Percakapan*, merupakan proses terjadinya sesi percakapan antara host dan guest mulai dari pembuatan sesi, bergabung dalam sesi dan mengakhiri sesi percakapan.
5. *Proses Sesi Percakapan*, pada dasarnya proses ini menunjukkan urutan pengiriman suara mulai dari klien berbicara, melakukan encoding, mengirim data hasil encoding, melakukan decoding dan kemudian diplayback sehingga bisa didengar kembali oleh klien yang dituju.

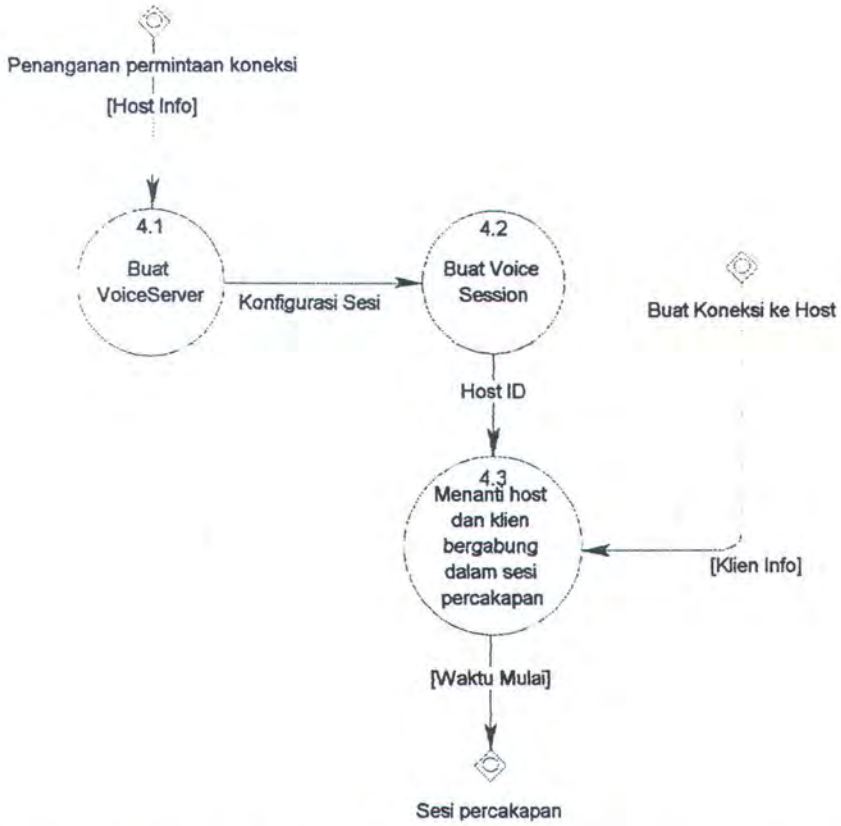
6. *Proses Hitung Biaya Percakapan*, merupakan proses akhir yang terjadi setelah dilakukan percakapan antara kedua belah pihak. Hasilnya biaya akan ditagihkan pihak yang melakukan sambungan pertama kali.

Berikut ini adalah hasil decompose dari proses ubah Sinyal DTMF menjadi data digital.



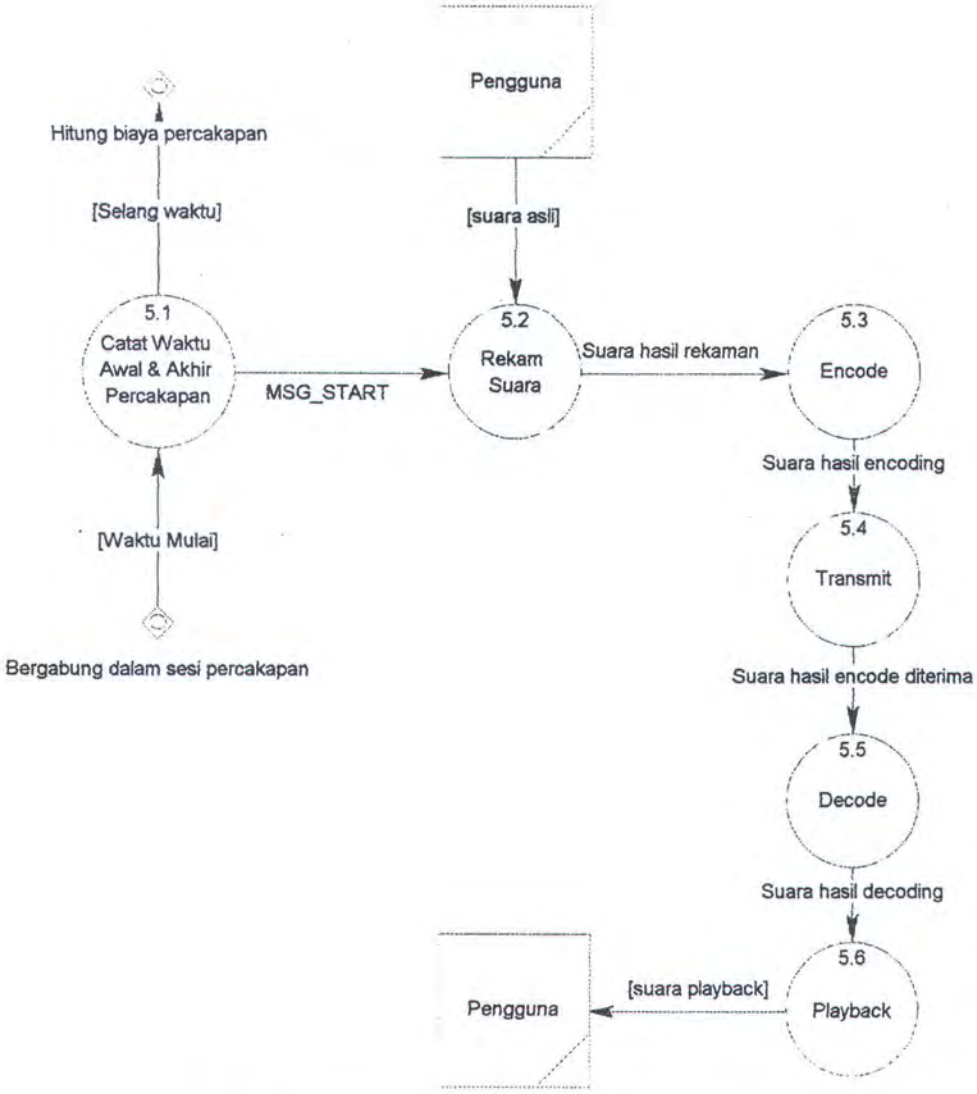
Gambar 3.10. Sub Proses Ubah Sinyal DTMF menjadi data digital

Gambar 3.11 berikut ini menjelaskan lebih detail bagaimana proses terjadinya sesi percakapan antara dua pengguna layanan Voice Gateway ini. Untuk membangun sesi percakapan antara dua pengguna, dibutuhkan minimal 2 buah gateway dimana salah satu gateway berfungsi sebagai *host* dan gateway yang lain akan bertindak sebagai *guest*. Host merupakan gateway yang dituju sedangkan guest adalah gateway asal tempat dimana pengguna melakukan request sambungan pembicaraan.



Gambar 3.11. Sub Proses Bergabung dalam sesi percakapan

Gambar 3.12 berikut ini menjelaskan tentang proses-proses yang terjadi pada sesi percakapan. Proses-proses ini berlaku baik dari host ke guest maupun sebaliknya. Sesi percakapan adalah suatu sesi yang mengijinkan terjadinya dialog antara pengguna satu dan yang lainnya melalui voice gateway. Transmisi paket suara hanya dilakukan bila level input suara minimal telah tercapai. Atau dengan kata lain, proses rekam hanya terjadi pada saat pengguna mengeluarkan suara dengan volume yang mencukupi sehingga memenuhi level suara minimum yang diijinkan oleh aplikasi Voice Gateway. Hal ini bertujuan untuk mengurangi paket-paket data yang tidak berguna agar tidak memenuhi bandwith yang digunakan.



Gambar 3.12. Sub Proses Sesi Percakapan

BAB IV

IMPLEMENTASI

BAB IV

IMPLEMENTASI

4.1 Pustaka DirectX 8.1

Sebelum dapat menggunakan referensi kepustaka DirectX8 terlebih dahulu harus dipastikan bahwa Microsoft DirectX 8.1 telah terinstall didalam sistem. Berikut langkah-langkah untuk menambahkan referensi ke pustaka DirectX 8.

- Melalui jendela Visual Basic 6.0 pilih menu **Project** kemudian pilih menu **References**. Setelah itu akan muncul sebuah jendela References seperti gambar dibawah ini :
- Setelah muncul jendela references, dari daftar references yang ada, cari references **DirectX 8 for Visual Basic Type Library**, kemudian beri tanda cawang dikotak checkbox yang bersesuaian kemudian tekan tombol **OK.**

4.1.1 Class DirectX8

Class DirectX8 merupakan penyedia awal komponen-komponen teknologi Microsoft DirectX®. Class ini terdiri dari metode-metode yang digunakan untuk membuat obyek-obyek utama berbagai macam komponen DirectX. Komponen ini biasanya merupakan obyek pertama yang dibuat dalam aplikasi yang menggunakan teknologi DirectX.

Metode-metode dalam class DirectX8 secara garis besar dapat dibagi menjadi beberapa kelompok sebagai berikut :

Tabel 4.1 Daftar metode untuk obyek DirectX8

Kelompok	Metode
ObjectCreation	Direct3Dcreate
	DirectInputCreate
	DirectMusicComposerCreate
	DirectMusicLoaderCreate
	DirectMusicPerformanceCreate
	DirectPlayAddressCreate
	DirectPlayClientCreate
	DirectPlayLobbiedApplicationCreate
	DirectPlayLobbyClientCreate
	DirectPlayPeerCreate
	DirectPlayServerCreate
	DirectPlayVoiceClientCreate
	DirectPlayVoiceServerCreate
	DirectPlayVoiceSetupCreate
	DirectSoundCaptureCreate
	DirectSoundCreate
	DirectXFileCreate
Enumeration	GetDSCaptureEnum
	GetDSEnum
Event Handling	CreateEvent
	DestroyEvent
	SetEvent
GUID Creation	CreateNewGuid

Untuk membuat obyek DirectX8, deklarasikan obyek tersebut sebagai obyek baru, misal:

```
Public objDX as New DirectX8
```

Gambar 4.1. Deklarasi Obyek DirectX8

Sedangkan untuk kebanyakan komponen-komponen DirectX tidak bisa dibuat langsung sepertihalnya obyek DirectX, tetapi harus dibuat dengan memanggil metode yang terdapat didalam obyek DirectX8. Sebagai contoh, untuk membuat obyek `DirectSound8`, buat sebuah variabel dan *assign* nilainya sebagai hasil dari `DirectX8.DirectSoundCreate`, misal :

```
Public objDS as DirectSound8
Set objDS  = objDX.DirectSoundCreate(vbNullString)
```

Gambar 4.2. Membuat Obyek Anggota DirectX8

Beberapa metode yang digunakan penulis dalam membuat aplikasi antara lain adalah :

- DirectPlayAddressCreate

  Sintak : `object.DirectPlayAddressCreate()`

  `As DirectPlay8Address`

  Metode ini digunakan untuk membuat obyek DirectPlay8Address, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectPlay8Address jika sukses, bila gagal maka akan terjadi kesalahan dan muncul sebuah nomor kesalahan.

- DirectPlayPeerCreate

  Sintak : `object.DirectPlayPeerCreate()`

  `As DirectPlay8Peer`

  Metode ini digunakan untuk membuat obyek DirectPlay8Peer, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectPlay8Peer jika sukses, bila gagal maka akan terjadi kesalahan dan muncul sebuah nomor kesalahan.

- DirectPlayVoiceServerCreate

  Sintak : `object.DirectPlayVoiceServerCreate()`

  `As DirectPlayVoiceServer8`

Metode ini digunakan untuk membuat obyek DirectPlayVoiceServer8, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectPlayVoiceServer8 jika sukses, bila gagal maka akan terjadi kesalahan dan muncul sebuah nomor kesalahan.

- DirectPlayVoiceClientCreate

  Sintak : `object.DirectPlayVoiceClientCreate()`

  `As DirectPlayVoiceClient8`

  Metode ini digunakan untuk membuat obyek DirectPlayVoiceClient8, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectPlayVoiceClient8 jika sukses, bila gagal maka akan terjadi kesalahan dan muncul sebuah nomor kesalahan.

- DirectPlayVoiceSetupCreate

  Sintak : `object.DirectPlayVoiceServerCreate()`

  `As DirectPlayVoiceTest8`

  Metode ini digunakan untuk membuat obyek DirectPlayVoiceTest8, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectPlayVoiceTest8 jika sukses, bila gagal maka akan terjadi kesalahan dan muncul sebuah nomor kesalahan.

- GetDSCaptureEnum

  Sintak : `object.GetDSCaptureEnum()`

  `As DirectSoundEnum8`

  Metode ini digunakan untuk membuat obyek DirectSoundEnum8, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectSoundEnum8 jika sukses, bila gagal maka akan

terjadi kesalahan dan muncul nomor kesalahan DSERR_OUTOFMEMORY. Obyek DirectSoundEnum8 yang dihasilkan dari metode ini digunakan untuk meng-enumerasi obyek DirectSoundCapture8 obyek yang terinstall pada sistem.

- GetDSEnum

  Sintak : `object.GetDSEnum()`

  `As DirectSoundEnum8`

  Metode ini digunakan untuk membuat obyek DirectSoundEnum8, bagian *object* mengacu pada obyek DirectX8. Return value-nya adalah berupa obyek DirectSoundEnum8 jika sukses, bila gagal maka akan terjadi kesalahan dan muncul nomor kesalahan DSERR_OUTOFMEMORY. Obyek DirectSoundEnum8 yang dihasilkan dari metode ini digunakan untuk meng-enumerasi driver-driver Microsoft® DirectSound® yang terinstall pada sistem.

4.1.2 Class DirectPlay8Peer

Berikut adalah beberapa metode yang dimiliki oleh class DirectPlay8Peer :

Tabel 4.2 Daftar Metode untuk Obyek DirectPlay8Peer

Kelompok	Metode
Session Management	Close
	Connect
	EnumHosts
	GetApplicationDesc
	GetCaps
	GetConnectionInfo
	GetCountServiceProviders
	GetEnumHostResponseAddress
	GetServiceProvider
	GetSPCaps
	Host

	SetApplicationDesc
	SetCaps
	SetSPCaps
	TerminateSession
Message Management	GetSendQueueInfo
	RegisterMessageHandler
	SendTo
	UnregisterMessageHandler
Player Management	DestroyPeer
	GetPlayerOrGroup
	GetCountClientsAndGroups
	GetPeerAddress
	GetPeerInfo
	RemovePlayerFromGroup
	SetPeerInfo
Group Management	AddPlayerToGroup
	CreateGroup
	DestroyGroup
	GetCountGroupMembers
	GetGroupInfo
	GetGroupMember
	SetGroupInfo
Miscellaneous	CancelAsyncOperation
	RegisterLobby

Beberapa metode yang digunakan penulis dalam membuat aplikasi antara lain adalah :

- CancelAsyncOperation

  Sintak : `CancelAsyncOperation(lAsyncHandle As Long, _`
  `lFlags As CONST_DPNCANCELFLAGS)`

  Digunakan untuk membatalkan *asynchronous request*. Sejumlah metode class DirectPlay8Peer secara default berjalan secara asynchronous. Berdasarkan situasi, suatu saat kita ingin membatalkan request sebelum mereka diproses secara keseluruhan. Semua metode class ini yang dapat berjalan secara asynchronous mengembalikan paramater lA*syncHandle.*

Untuk membatalkan semua operasi asynchronous yang dipending, *passing* nilai 0 kedalam parameter lAsyncHandle dan konstanta DPNCANCEL_ALL_OPERATIONS dalam parameter lFlags.

- Close()

  Sintak : `Close()`

  Digunakan untuk menutup koneksi pada session.

- Connect()

  Sintak :
```
Connect(AppDesc As DPN_APPLICATION_DESC, _
Address As DirectPlay8Address, _
DeviceInfo As DirectPlay8Address, _
LFlags As Long, _
UserData As Any, _
UserDataSize As Long) As Long
```

  Digunakan untuk membangun koneksi ke server. Setelah koneksi berhasil dibuat, saluran komunikasi didalam obyek akan terbuka dan aktiv sehingga aplikasi mendapatkan pesan yang tiba seketika. Tidak ada pesan yang dapat dikirimkan melalui metode `DirectPlay8Peer.SendTo` hingga koneksi selesai. Pada saat ada permintaan koneksi ke host, method `DirectPlay8Event.IndicateConnect` akan dipanggil didalam *message handler* komputer Host. Host bisa saja menerima ataupun menolak koneksi yang dilakukan. Setelah host bertindak maka *message handler* disisi klien akan memanggil metode `DirectPlay8Event.ConnectComplete` untuk menanggapi response yang diberikan oleh host.

- GetPeerInfo

Sintak : `GetPeerInfo(idPeer As Long) _`

`As DPN_PLAYER_INFO`

Digunakan untuk mengambil informasi player yang terhubung pada peer yang bersesuaian.

- GetPeerAddress

Sintak : `GetPeerAddress(idPlayer As Long) _`

`As DirectPlay8Address`

Digunakan untuk mengambil alamat dari remote player yang terhubung didalam suatu session.

- Host

Sintak : `Host(AppDesc As DPN_APPLICATION_DESC, _`

`Address As DirectPlay8Address)`

Digunakan untuk membuat host pada session peer-to-peer.

- SetPeerInfo

Sintak : `SetPeerInfo(PlayerInfo As DPN_PLAYER_INFO) _`

`As Long`

Men-set nilai-nilai statis sebuah peer dengan aplikasi. Metode ini dipanggil sebelum melakukan koneksi untuk menyampaikan informasi dasar kepada aplikasi. Setelah peer berhasil terkoneksi, informasi yang disampaikan ini bisa diambil oleh player lainnya dengan memanggil metode DirectPlay8Peer.GetPeerInfo.

- SendTo()

Sintak : `SendTo(idSend As Long, _`

`Buffer() As Byte, lPriority As Long, _`

```
lTimeOut as Long, lFlags As CONST_DPNSENDFLAGS)_
As Long
```

Digunakan untuk mengirimkan data ke player yang ditentukan. Pesan dapat dikirim baik secara synchronous maupun asynchronous. Untuk mengirimkan pesan ke seluruh player dalam suatu session set nilai *lFlags* menjadi `DPNID_ALL_PLAYERS_GROUP`.

- TerminateSession

  Sintak : `TerminateSession(lFlags As Long, _`
  `UserData As Any, _`
  `UserDataSize As Long)`

  Digunakan untuk memutus session Microsoft DirectPlay saat ini.

- RegisterMessageHandler

  Sintak : `RegisterMessageHandler(event As DirectPlay8Event)`

  Mendaftarkan sebuah point masukan ke dalam kode klien yang menerima pesan dari class DirectPlay8Peer. Metode ini dipanggil sebelum memanggil metode lainnya.

- UnregisterRegisterMessageHandler()

  Sintak : `UnRegisterMessageHandler()`

  Melakukan unregister terhadap *message handler* yang aktif.

4.1.3 Class DirectPlay8Address

Berikut adalah daftar metode untuk kelas DirectPlay8Address :

Tabel 4.3 Daftar Metode untuk Class DirectPlay8Address

AddComponentLong	AddComponentString
BuildFromURL	Clear
Duplicate	GetComponentLong

GetComponentString	GetDevice
GetNumComponents	GetSP
GetURL	GetUserData
SetDevice	SetEqual
SetSP	SetUserData

Dari sekian metode yang terdapat pada class DirectPlay8Address, berikut metode-metode yang digunakan oleh penulis :

- AddComponentLong

  Sintak : `AddComponentLong(sComponent As String, _`
  `lValue As Long)`

  Menambahkan sebuah komponen bertipe long kedalam obyek Address. Jika komponen tersebut merupakan bagian dari Address maka nilainya akan digantikan dengan nilai baru yang dibawa pada saat pemanggilan method ini.

- AddComponentString

  Sintak : `AddComponentString(sComponent As String, _`
  `sValue As String)`

  Menambahkan sebuah komponen bertipe string ke dalam obyek *Address*. Jika komponen tersebut merupakan bagian dari obyek *Address* maka nilainya akan digantikan dengan nilai baru.

- SetSP

  Sintak : `SetSP(guidSP As String)`

  Mengatur GUID service provider obyek Address. Jika sebuah service provider telah ditentukan sebelumnya maka akan diganti dengan nilai baru pada saat pemanggilan.

4.1.4 Class DirectPlayVoiceServer8

Aplikasi ini menggunakan class DirectPlayVoiceServer8 untuk mengelola host yang terlibat dalam sesi percakapan. Secara garis besar metode-metode didalam obyek DirectPlayVoiceServer8 dapat dikelompokan menjadi beberapa kelompok berikut :

Tabel 4.4 Daftar Metode untuk Class DirectPlayVoiceServer8

Kelompok	Metode
Miscellaneous	GetCaps
	GetCompressionType
	GetCompressionTypeCount
Session Management	GetSessionDesc
	GetTransmitTargets
	Initialize
	SetSessionDesc
	SetTransmitTargets
	StartServerNotification
	StartSession
	StopSession
	UnregisterMessageHandler

4.1.5 Class DirectPlayVoiceClient8

Tabel 4.5. Daftar Metode untuk Class DirectPlayVoiceClient

Kelompok	Metode
Buffer Management	Create3DSoundBuffer
	Delete3DSoundBuffer
Miscellaneous	GetCaps
	GetCompressionType
	GetCompressionTypeCount
	GetSoundDeviceConfig
	GetSoundDevices
	StartClientNotification
Session Management	Connect
	Disconnect
	GetClientConfig
	GetSessionDesc
	GetTransmitTargets
	Initialize

	SetClientConfig
	SetCurrentSoundDevices
	SetTransmitTargets
	UnregisterMessageHandler

4.1.6 Fungsi-fungsi

Bagian ini membahas tentang fungsi-fungsi yang terdapat dalam referensi Microsoft DirectPlay. Fungsi-fungsi ini didesain untuk menyederhanakan proses penanganan *byte arrays*.

- AddDataToBuffer

  Digunakan untuk menambahkan data non string kedalam byte array. Jika jumlah Byte Array yang dipassing ke dalam Buffer tidak cukup besar untuk menampung data, ukuran array akan diperbesar tanpa menghilangkan data.

  Sintak : `AddDataToBuffer(`

  ```
  Buffer() As Byte, lData As Any, _
  lSize As Long, lOffset As Long)
  ```

  Parameter-parameter :

  1. *Buffer*, adalah byte array tempat data ditambahkan.
  2. *lData*, data yang akan ditambahkan ke dalam *Buffer*.
  3. *lSize*, ukuran data didalam *lData* dalam byte. Cara mudah untuk menentukan nilai ini adalah menggunakan fungsi *built in* Microsoft Visual Basic, yaitu fungsi ***LenB***. Selain itu juga dapat digunakan salah satu nilai dari enumerasi CONST_DPLAYBUFSIZE.
  4. *lOffset*, suatu nilai berisi offset dalam bytes, yang menunjukkan tempat didalam byte array dimana data akan ditambahkan. Ketika fungsi ini

mengembalikan nilai parameter ini akan diset pada offset dibyte pertama diikuti dengan data yang ditambahkan.

- AddStringToBuffer

  Menambah sebuah string ke dalam byte array.

  Sintak : `AddStringToBuffer( _`
  `Buffer() As Byte, StringData As String, _`
  `lOffset As Long)`

  Parameter-parameter :

  - *Buffer*, byte array tempat data ditambahkan.
  - *StringData*, nilai string yang akan dimasukkan ke dalam *Buffer*.
  - *lOffset*, nilai long berisi offset dalam byte, tempat dalam byte array dimana data akan ditambahkan. Ketika fungsi selesai dijalankan, parameter ini akan berisi offset pada byte pertama diikuti dengan data yang baru saja ditambahkan.

- GetDataFromBuffer

  Digunakan untuk mengambil data selain string dari sekumpulan byte array.

  Sintak : `GetDataFromBuffer( Buffer() As Byte,`
  `lData As Any, lSize As Long,`
  `lOffset As Long)`

  Parameter-parameter :

  - *Buffer*, byte array tempat data akan diambil.
  - *lData*, variabel yang menampung data dari pengambilan ke buffer.
  - *lSize*, ukuran data yang ada dalam *lData*, dalam byte.

- GetStringFromBuffer

Mengambil data string dari dalam byte array. Nilai kembalinya berupa string.

Sintak : `GetStringFromBuffer(`

```
Buffer() As Byte,
lOffset As Long) As String
```

Parameter-parameter :

- *Buffer*, byte array tempat asal data diambil.
- *lOffset*, nilai long berisi offset, dalam byte, menunjukan lokasi dalam byte array dimana data diambil.

- NewBuffer

Membuat sebuah byte array. Mengembalikan nilai long berisi offset untuk mengawali sebuah array. Nilai ini selalu bernilai 0 (zero).

Sintak : `NewBuffer(Buffer() As Byte) As Long`

Parameter-parameter :

- *Buffer*, byte array baru yang akan diinisialisasi dengan jumlah anggota standard (jumlah array standard adalah 20).

4.2 Class Modul clsHost.cls

Class modul clsHost digunakan untuk menjalankan fungsi-fungsi sebagai host. Fungsi-fungsi tersebut antara lain adalah :

4.2.1 Fungsi InisialisasiHost

Potongan program dibawah ini, merupakan suatu fungsi untuk menjalankan aplikasi VoiceGateway sebagai Host. Fungsi ini memiliki sebuah parameter oEvent yang bertipe DirectPlay8Event.

```
InisialisasiHost(oEvent)
1  objDPPeer.RegisterMessageHandler(oEvent)
2  SetPeerInfo
3  SetAddress
4  SetAppDesc
5  objDPPeer.Host(pAppDesk, objAddress, 0)
```

Gambar 4.3 Pseudocode fungsi InisialisasiHost

Terlihat diatas terdapat beberapa prosedur yang dipanggil didalam fungsi InisialisasiHost. Berikut adalah pseudocode untuk prosedur-prosedur tersebut :

4.2.2 Prosedur SetPeerInfo

```
SetPeerInfo
1  pInfo.Name ← "Host"
2  pInfo.lInfoFlags ← DPNINFO_NAME
3  objDPPeer.SetPeerInfo(pInfo, 0)
```

Gambar 4.4 Pseudocode prosedur SetPeerInfo

4.2.3 Prosedur SetAddress

```
SetAddress
1  objAddress ← objDirectX.DirectPlayAddressCreate
2  objAddress.SetSP(DP8SP_TCPIP)
3  objAddress.AddComponentLong(DPN_KEY_PORT, cPort)
```

Gambar 4.5 Pseudocode prosedur SetAddress

Terlihat diatas terdapat sebuah constanta `cPort` yang berisi nomor port yang akan digunakan oleh host untuk melakukan hosting.

4.2.4 Prosedur SetAppDesk

```
SetAppDesk
1  pAppDesk.guidApplication ← cGuid
2  pAppDesk.lMaxPlayers ← 2
3  pAppDesk.SessionName ← "VeGa Pro"
4  pAppDesk.lFlags ← DPNSESSION_MIGRATE_HOST
```

Gambar 4.6 Pseudocode prosedur SetAppDesk

cGuid disini adalah sebuah konstanta string yang menunjukkan nomor guid aplikasi yang akan digunakan sebagai identitas aplikasi.

4.2.5 Fungsi CreateVoiceServer

```
CreateVoiceServer()
1  Set objVceServer ← objDirectX.DirectPlayVoiceServerCreate
2  objVceServer.Initialize objDPPeer, 0
3  pSesDesk.guidCT ← pGuidCT
4  pSesDesk.lBufferAggressiveness ← 0
5  pSesDesk.lBufferQuality ← 0
6  pSesDesk.lSessionType ← DVSESSIONTYPE_PEER
7  objVceServer.StartSession(pSesDesk, 0)
```

Gambar 4.7 Pseudocode fungsi CreateVoiceServer

4.3 Class Modul clsGuest.cls

Class modul clsGuest berisi fungsi-fungsi / prosedur-prosedur yang digunakan untuk membangun koneksi ke gateway klien dan bergabung dalam sesi percakapan. Fungsi/prosedur tersebut antara lain adalah :

4.3.1 Fungsi ConnectToHost

```
ConnectToHost(HostIP)
1  AppDesk.guidApplication ← cGuid
2  AppDesk.SessionName ← "Vega Pro"
3  pInfo.lInfoFlags ← DPNINFO_NAME
4  pInfo.Name ← "Guest"
```

```
5  objDPPeer.SetPeerInfo(pInfo, DPNOP_SYNC)
6  objHostAddr.SetSP(DP8SP_TCPIP)
7  objHostAddr.AddComponentLong(DPN_KEY_PORT, cPort)
8  objHostAddr.AddComponentString(DPN_KEY_HOSTNAME, HostIP)
9  objGuestAddr.SetSP(DP8SP_TCPIP)
10 objDPPeer.Connect(AppDesk, objHostAddr, objGuestAddr,
   DPNHOST_OKTOQUERYFORADDRESSING, ByVal 0&, 0)
```

Gambar 4.8 Pseudocode fungsi ConnectToHost

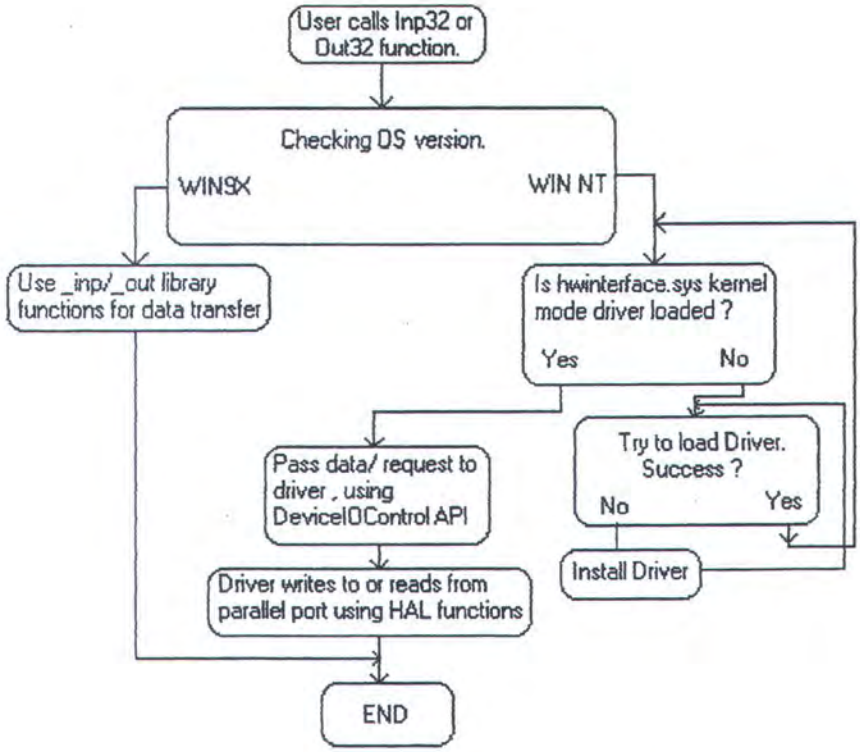
4.3.2 Fungsi CreateVoiceClient

```
CreateVoiceClient
1  objVceClient.Initialize(objDPPeer, 0)
2  objVceClient.StartClientNotification(oEvent)
3  oClient.lBufferAggressiveness ← DVBUFFERAGGRESSIVENESS_DEFAULT
4  oClient.lBufferQuality ← DVBUFFERQUALITY_DEFAULT or
                            DVCLIENTCONFIG_AUTOVOICEACTIVATED
5  oClient.lNotifyPeriod ← 0
6  oClient.lThreshold ← DVTHRESHOLD_UNUSED
7  oSound.guidCaptureDevice ← vbNullString
8  oSound.guidPlaybackDevice ← vbNullString
9  oSound.hwndAppWindow ← hwnd
10 objVceClient.Connect(oSound, oClient, 0)
11 lTarget(0) ← DVID_ALLPLAYERS
12 objVceClient.SetTransmitTargets(lTarget, 0)
```

Gambar 4.9 Pseudocode fungsi CreateVoiceClient

4.4 Pustaka Inpout32.dll

Merupakan referensi yang digunakan penulis untuk mengakses port paralel untuk melakukan pengecekan tombol telepon yang dilakukan pengguna pada saat memasukkan nomor telepon tujuan yang ingin dihubunginya. Pustaka ini penulis download dari *http://www.logix4u.net/inpout32_source_and_bins.zip*. Pustaka ini dibuat dengan bahasa Visual C++ mengingat keterbatasan fungsi-fungsi Visual Basic dalam mengakses hardware secara langung. Berikut adalah alur kerja dari inpout32.dll :



Gambar 4.10. Alur proses didalam inpout32.dll

2 blok penting dalam program ini adalah :

- Sebuah device driver mode kernel yang tertanam didalam DLL binary form
- DLL itu sendiri

Tiga fungsi yang diimplementasikan dalam driver mode kernel Hwinterface.sys ini adalah :

1. *DriverEntry*, dipanggil pada saat driver diload. Membuat obyek device dan symbolic link.
2. hwinterfaceUnload, dipanggil pada saat driver di unload, digunakan untuk menghapus semua obyek.

3. *hwinterfaceDeviceControl*, menangani pemanggilan fungsi yang dibuat melalui DeviceIOControl API. Melakukan proses baca tulis ke paralel port untuk menyampaikan kode kontrol.

Beberapa fungsi yang disediakan dalam library ini adalah :

1. *inp32*, membaca data dari register port paralel yang ditentukan.
2. out32, menulis data ke dalam register port paralel .
3. *dllMain*, dipanggil pada saat DLL di load atau di unload. Pada saat DLL di load fungsi ini akan memeriksa versi sistem operasi yang digunakan dan me-load hwinterface.sys jika dibutuhkan.
4. *CloseDriver*, menutup handle driver yang terbuka, dipanggil sesaat sebelum melakukan unload driver.
5. *OpenDriver*, membuka sebuah *handle* ke *driver hwinterface.*
6. *inst*, melakukan ekstraksi *hwinterface.sys* dari dalam resource binary ke direktori 'systemroot\drivers' dan membuat sebuah service. Fungsi ini dipanggil saat fungsi OpenDriver gagal membuka sebuah handle valid ke service hwinterface.
7. *start*, memulai service *hwinterface* menggunakan Service Control Manager API.
8. *SystemVersion*, memeriksa versi OS dan menjalankan kode yang bersesuaian.

BAB V

U J I C O B A

5.1. Lingkungan Ujicoba

Lingkungan ujicoba yang dibuat oleh penulis untuk menguji system Voice Gateway yang dibuat adalah sebagai berikut :

Tabel 5.1. Spesifikasi lingkungan ujicoba sistem Voice Gateway

Item	Voice Gateway A	Voice Gateway B
CPU	MITAC 6513WU	MITAC 6513WU
Prosesor	Pentium III-800 Mhz	Pentium III-800 Mhz
Soundcard	ESS Alegro (Onboard)	ESS Alegro (Onboard)
Modem	Prolink Internal Modem (chipset Motorola)	Prolink Internal Modem (chipset Motorola)
IP Address	192.168.30.10	192.168.40.25
Netmask	255.255.255.224	255.255.255.224
Gateway	192.168.30.30	192.168.40.30
Primary DNS	192.168.100.30	192.168.100.30

Kedua PC Gateway terhubung melalui jaringan *local area network* (LAN) dimana diantara keduanya terdapat 3 buah switch hub 10/100 mbps.

5.2. Tone Decoder

Ujicoba tone decoder adalah menguji inputan yang dilakukan pengguna melalui penekanan tombol telepon pada saat terjadi pembicaraan. Penekanan tombol dilakukan dengan menggunakan berbagai macam interval dan kombinasi tombol. Hasil yang diharapkan adalah tombol apapun yang ditekan oleh pengguna dengan segala variasi dapat dikenali dan dibaca oleh aplikasi yang dibuat. Berikut ini adalah tabel hasil ujicoba yang telah dilakukan penulis :

Tabel 5.2. Hasil Ujicoba Tone Decoder

No	Tombol yg ditekan	Hasil Pembacaan	Tingkat kesalahan
1	679043579#	679043579#	0%
2	85412479*45684	85412479*45684	0%
3	3467752*3456	3467752*3456	0%
4	*#6808025386	*#6808025386	0%
5	000001111122222	000001111122222	0%
6	777777754433367	777777754433367	0%
7	111	111	0%
8	5675686797809	5675686797809	0%
9	*******########	*******########	0%
10	1234567890*#	1234567890*#	0%

Dari hasil 10 kali ujicoba dapat diketahui bahwa hasil penekanan tombol dapat terbaca keseluruhan sesuai dengan apa yang ditekan oleh pengguna.

5.3. Delay

Ujicoba Delay adalah ujicoba yang dilakukan untuk mengetahui seberapa besar delay pengiriman suara antar kedua Voice Gateway. Metode yang dilakukan adalah pada saat proses perekaman dimulai penulis mencatat waktu awal data suara direkam dan dikirim (t_{kirim}). Kemudian pada saat Voice Gateway penerima menerima data suara dan memainkannya untuk pertama kalinya maka Voice Gateway tersebut akan mengirimkan suatu pesan kepada Voice Gateway pengirim yang menyatakan bahwa data suara tersebut telah diterima. Selanjutnya Voice Gateway akan mencatat waktu saat menerima pesan konfirmasi tersebut ($t_{konfirmasi}$). Dalam kasus ini penulis mengasumsikan bahwa waktu yang dibutuhkan untuk mengirimkan suara dari Voice Gateway asal ke Voice Gateway tujuan adalah sama dengan waktu yang dibutuhkan untuk mengirimkan pesan konfirmasi dari Voice Gateway tujuan ke Voice Gateway asal sehingga akan

diperoleh waktu delay (t_{delay}) merupakan selisih antara waktu konfirmasi dengan waktu kirim dibagi 2. Ujicoba ini dilakukan dengan menggunakan beberapa metode kompresi/dekompresi untuk mengetahui apakah penggunaan codec yang berbeda akan mempengaruhi delay yang terjadi. Berikut ini adalah hasil ujicoba yang telah dilakukan :

Tabel 5.3. Hasil Ujicoba delay untuk Codec Voxware VR12 1.4 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}
1	240	120
2	160	80
3	170	85
4	80	40
5	80	40
6	100	50
7	211	105,5
8	120	60
9	191	95,5
10	301	150,5
Rata-rata delay		**82,65**

Tabel 5.4. Hasil Ujicoba delay untuk Codec Voxware SC06 6.4 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}
1	90	45
2	20	10
3	190	95
4	10	5
5	40	20
6	50	25
7	40	20
8	60	30
9	70	35
10	40	20
Rata-rata delay		**30,5**

Tabel 5.5. Hasil Ujicoba delay untuk Codec Voxware SC03 3.2 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}

1	40	20
2	10	5
3	60	30
4	60	30
5	60	30
6	60	30
7	61	30,5
8	50	25
9	60	30
10	50	25
Rata-rata delay		**25,55**

Tabel 5.6. Hasil Ujicoba delay untuk Codec MS-PCM 64 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}
1	50	25
2	10	5
3	50	25
4	50	25
5	60	30
6	40	20
7	50	25
8	51	25,5
9	10	5
10	20	10
Rata-rata delay		**19,55**

Tabel 5.7. Hasil Ujicoba delay untuk Codec MS-ADPCM 32.8 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}
1	20	10
2	100	50
3	40	20
4	60	30
5	31	15,5
6	10	5
7	10	5
8	50	25
9	50	25
10	60	30
Rata-rata delay		**21,55**

Tabel 5.8. Hasil Ujicoba delay untuk Codec True Speech 8.6 kbit/s

No	$t_{konfirmasi} - t_{kirim}$	t_{delay}
1	40	20
2	40	20
3	40	20
4	40	20
5	50	25
6	40	20
7	40	20
8	40	20
9	90	45
10	90	45
Rata-rata delay		**25,5**

5.4. Jitter

Ujicoba Jitter adalah ujicoba yang digunakan untuk menghitung selisih waktu antara waktu perekaman suara dari awal hingga terakhir dengan waktu data suara mulai diterima dan dimainkan hingga akhir di Voice Gateway tujuan. Ujicoba ini dilakukan dengan menggunakan beberapa kombinasi metode Codec untuk membuktikan apakah penggunaan metode Codec yang berbeda akan mempengaruhi *jitter* yang terjadi.

Tabel 5.9. Hasil Ujicoba Jitter dengan Codec Voxware VR12 1.4 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	3605	4046	441
2	2503	2964	461
3	6930	7371	441
4	4316	4777	461
5	4136	4576	440
6	4777	5297	520
7	4157	5057	900
8	6640	7191	551
9	3525	3966	441
10	4046	4486	440
Rata-rata			**509,6**

Tabel 5.10. Hasil Ujicoba Jitter dengan Codec Voxware SC06 6.4 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	8102	8502	400
2	7501	7891	390
3	7301	7701	400
4	7591	8022	431
5	8802	9294	492
6	3905	4407	502
7	10294	10896	602
8	3925	4396	471
9	4326	4797	471
10	2693	3195	502
Rata-rata			**466,1**

Tabel 5.11. Hasil Ujicoba Jitter dengan Codec Voxware SC03 3.2 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	6199	6680	481
2	5999	6429	430
3	5227	5608	381
4	721	1201	480
5	591	1001	410
6	4106	4496	390
7	4206	4597	391
8	4397	4797	400
9	1302	1693	391
10	4696	6198	1502
Rata-rata			**525,6**

Tabel 5.12. Hasil Ujicoba Jitter dengan Codec MS-PCM 64 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	3785	4287	502
2	7010	7491	481
3	490	1032	542
4	6690	7200	510
5	5077	5558	481
6	5898	6360	462
7	6509	6990	481
8	3906	4386	480
9	3205	3696	491
10	2563	3335	772
Rata-rata			**520,2**

Tabel 5.13. Hasil Ujicoba Jitter dengan Codec MS-ADPCM 32.8 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	5748	6269	521
2	7060	7561	501
3	6189	6649	460
4	7381	7831	450
5	1422	2434	1012
6	4377	4817	440
7	6479	6960	481
8	6510	6960	450
9	3675	4196	521
10	2193	2684	491
Rata-rata			**532,7**

Tabel 5.14. Hasil Ujicoba Jitter dengan Codec True Speech 8.6 kbit/s

No	t_{record}	$t_{playback}$	Selisih
1	10054	10986	932
2	4306	4857	551
3	4156	4687	531
4	2233	2794	561
5	3956	4767	811
6	4295	4857	562
7	641	1101	460
8	4246	4686	440
9	610	1092	482
10	6380	6840	460
Rata-rata			**579**

Berikut adalah hasil rangkuman dari keseluruhan ujicoba di atas:

Tabel 5.15. Rangkuman ujicoba delay dan jitter

Audio Format	Voxware VR12	Voxware SC06	Voxware SC03	MS-PCM	MS-AD PCM	True Speech
Data Rate	Rata-rata 1.4 kbps	6.4 kbps	3.2 kbps	64 kbps	32 kbps	8.6 kbps
Delay	82.6 ms	30.5 ms	25.55 ms	19.5 ms	21.5 ms	25.5 ms
Jitter	509.6 ms	466.1 ms	525.6 ms	520.2 ,s	532.7 ms	579 ms
MOS	2.3	3.7	3	4.4	4.1	3.4

Tiga codec pertama menyediakan tingkat kompresi yang tinggi dan membutuhkan kebutuhan resource yang sama. Pada kelas komputer Pentium III

500 codec ini menggunakan 1.5 persen dari kapasitas CPU. Codec Voxware VR12 terdengar lebih pelan dan seperti robot, sedangkan SC03 dan SC06 menyediakan kualitas suara yang lebih memadai. Codec PCM menyediakan kualitas suara yang paling tinggi dalam format audio 8khz 16 bit mono.

5.5. Ujicoba Keseluruhan

Ujicoba ini dilakukan untuk menguji sistem Voice Gateway secara keseluruhan, dengan tahapan-tahapan sebagai berikut :

- Membangun koneksi ke Voice Gateway melalui *line* telepon, pada tahapan ini secara garis besar Voice Gateway sudah dapat menangani respon yang terjadi pada *voice modem*, dimana pada saat telepon berdering, secara otomatis Voice Gateway akan menanggapi deringan tersebut.
- Menerima masukan tombol telepon melalui *tone decoder*, pada tahapan ini semua jenis tombol telepon yang diwakilkan oleh sinyal DTMF telah dapat dikenali dengan baik. Proses pemecahan kode area dan nomor tujuan juga dapat dilakukan dengan baik.
- Membangun koneksi antara Voice Gateway satu dengan Voice Gateway yang lainnya. Untuk tahapan ini, penulis masih menemukan beberapa kendala dimana walaupun koneksi antara keduanya sudah terjadi namun transmisi suara terkadang tidak dapat dilakukan. Menurut pendapat penulis hal ini terjadi karena baik *host* maupun *guest* melakukan akses ke perangkat suara yang sama sehingga pada saat *guest* sudah melakukan

koneksi seharusnya perangkat suara ditangani oleh *guest* namun tertahan karena *host* masih menggunakan perangkat suara tersebut.

- Memutus koneksi, dalam tahapan ini juga masih terdapat masalah dimana beberapa kali penulis menemui PC Gateway yang digunakan tidak memberikan respon pemutusan koneksi. Menurut penulis hal ini terjadi pada saat kelas **clsGuest** menjalankan metode *destroy*, proses tersebut tidak berjalan mulus karena tertahan oleh akses ke perangkat suara. Namun secara garis besar pemutusan koneksi sudah dapat dlakukan.

BAB VI

KESIMPULAN DAN SARAN

BAB VI

KESIMPULAN DAN SARAN

Dalam bab ini dijelaskan mengenai kesimpulan dan kemungkinan pengembangan lebih lanjut dari perangkat lunak yang telah dibuat.

6.1. Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari tugas akhir ini adalah sebagai berikut :

1. Tone Decoder mampu digunakan untuk mengenali masukan yang dilakukan pengguna melalui penekanan tombol telepon dan mengirimkan hasilnya kepada aplikasi Voice Gateway.
2. Aplikasi mampu mengirimkan data suara dua arah antara kedua pengguna dengan berbagai macam metode kompresi yang dapat disesuaikan dengan kondisi jaringan masing-masing penyelenggara jasa ITSP.
3. Berdasarkan ujicoba *delay* yang telah dilakukan maka metode kompresi/dekompresi yang paling baik untuk digunakan adalah codec **MS PCM 64 kbit/s** dengan rata-rata *delay* **19.55 ms**.
4. Berdasarkan ujicoba *jitter* yang telah dilakukan maka metode kompresi/dekompresi yang baik untuk digunakan adalah **Voxware SC06 6.4 kbit/s** dengan rata-rata *jitter* sebesar **466,1 ms**.
5. Berdasarkan hasil ujicoba keseluruhan dengan melihat besar delay, jitter dan kualitas suara yang dihasilkan maka codec yang paling baik untuk digunakan adalah **MS PCM 64 kbit/s.**

6.2. Saran

Berdasarkan evaluasi yang dilakukan terhadap sistem , ada beberapa saran yang perlu dipertimbangkan dalam pengembangan aplikasi ini, yaitu :

1. Perlu mencoba sistem ini dengan menggunakan beberapa jenis modem dan *sound card* sehingga mampu meningkatkan kompatibilitas dengan perangkat keras dengan merek dan jenis yang berbeda.
2. Mengembangkan aplikasi ini sehingga peserta sesi percakapan tidak terbatas hanya pada dua orang saja sehingga bisa membentuk suatu kelompok diskusi.
3. Perlu mengembangkan aplikasi ini untuk tidak hanya menangani transmisi suara saja namun juga bisa dimanfaatkan untuk data faximili maupun *video conference.*

DAFTAR PUSTAKA