BAB 2
LANDASAN TEORI

2.1 Dasar Teori

Pada bab ini akan dibahas mengenai uraian dasar teori dan konsep yang akan digunakan dalam perancangan aplikasi yang akan dibangun. Ada beberapa konsep yang akan dibahas pada bab ini, diantaranya adalah konsep dasar pemrograman *socket* pada protokol TCP (*Transport Control Protocol*) dan UDP (*User Datagram Protocol*), konsep dasar arsitektur aplikasi; *client-server*, *peer-to-peer*, dan *hybrid peer-to-peer*. Namun sebelumnya akan dibahas tentang *instant messaging*, perkembangan *instant messaging*, keuntungan *instant messaging*, dan cara kerja *instant messaging* serta pembahasan tentang *voice* dan *video conferencing*. Selain itu juga dibahas tentang *tool* dan teknologi yang digunakan dalam pengembangan aplikasi yang akan dibangun, yaitu teknologi *.NET Framework*, *Microsoft Studio .Net* dan *DirectShow*.

2.2 *Instant Messaging*

Instant messaging (pesan instan) adalah sebuah teknologi internet yang mengizinkan para pengguna dalam jaringan internet untuk mengirimkan pesan-pesan singkat secara langsung pada saat yang bersamaan (*real time*) menggunakan teks kepada pengguna lainnya yang sedang terhubung ke jaringan yang sama.

Konsep yang digunakan oleh teknologi ini muncul pada awal-awal pengembangan sistem operasi *UNIX* dan

jaringan internet; para pengguna yang sudah *login* ke jaringan dapat mengirimkan perintah berupa <*code*>*talk*</*code*>, *write*, dan *finger* untuk melihat siapa saja yang sudah *login* dan akhirnya mengirimkan pesan singkat kepada mereka.

Istilah *instant messaging* saat ini pada umumnya mengacu kepada sebuah teknologi yang dipopulerkan oleh *America Online (*AOL*)*, yang kemudian diikuti oleh *Yahoo* (*Yahoo Messenger*), *Google*, dan *Microsoft* (*Windows Live Messenger*)dan perusahaan-perusahaan lainnya.

2.2.1 Perkembangan *Instant Messaging*

Walaupun *instant messaging* tampak seakan-akan sebuah teknologi baru, sebenarnya *instant messaging* telah berusia hampir duapuluh tahun. Beberapa literatur menyatakan bahwa layanan *instant messaging* yang pertama adalah layanan *instant messaging* yang diperkenalkan oleh AOL *(America Online)* pada awal tahun 1990-an. Layanan ini merupakan fasilitas yang disediakan bagi para pengguna penyedia jasa internet terbesar di Amerika Serikat ini. Layanan *instant messaging* AOL memungkinkan para penggunanya untuk saling berkomunikasi dan bertukar pesan secara *online*. Namun, penggunanya baru terbatas hanya pelanggan AOL saja. Di kemudian hari, AOL memperkenalkan *AOL Instant Messenger* (AIM) yang tidak dibatasi hanya untuk pelanggan AOL saja, sehingga memungkinkan setiap orang untuk menggunakan AIM.

Beberapa literatur lain menyatakan bahwa *instant messaging* telah ada dalam bentuk yang beragam jauh

sebelum AIM dikembangkan. Sistem *instant messaging* awal telah diimplementasikan pada jaringan komputer privat seperti misalnya *Term-Talk* pada sistem *PLATO* pada awal 1970-an. Sistem yang serupa juga telah diimplementasikan pada komputer *DEC PDP-11* sebagai program *talk*. Sampai sekitar tahun 1990-an, program *talk* masih lazim digunakan di kalangan akademik untuk saling berkomunikasi dengan menggunakan sistem operasi *UNIX/Linux*. Kemudian, pada tahun 1987, MIT mengembangkan *instant messaging client* pertama yang berbasis grafis pada sistem *Zephyr*.

Pada tahun 1988, Jarkki Oikarinen mengembangkan sistem *Internet Relay Chat* (IRC). IRC memungkinkan pengguna untuk saling berkomunikasi dalam "ruang percakapan" (*chat room*) dan saling berkirim file. IRC menjadi sangat terkenal pada tahun 1990-an. Namun, tidak seperti sistem *instant messaging* modern, IRC tidak memiliki fasilitas untuk mengetahui apakah seorang kontak atau rekan sedang *online* atau tidak. Selain itu, pada IRC tidak ada keharusan untuk melakukan registrasi pengguna maupun batasan untuk menggunakan nama tertentu.

Perkembangan *instant messaging* yang sangat pesat dimulai sejak tahun 1996 ketika *Mirabilis*, sebuah perusahaan Israel, memperkenalkan ICQ (biasa dibaca *I Seek You* dalam bahasa Inggris) ke internet. Sejak saat itu, penggunaan istilah "*instant messaging*" pun menjadi populer.

Hampir bersamaan dengan ICQ, AOL pun memperkenalkan AIM yang dapat digunakan tidak hanya oleh pelanggan AOL saja. Kedua layanan ini pun kemudian

bersaing ketat. Namun, pada akhirnya AOL mengakuisisi Mirabilis dan layanan ICQ pada tahun 1998. Sehingga AIM & ICQ pun menjadi layanan *instant messaging* yang paling dominan saat itu.

Setelah itu, banyak layanan *instant messaging* bermunculan dengan fasilitas khasnya masing-masing. *Yahoo* sebagai portal internet terbesar saat itu memperkenalkan layanan terbarunya, yaitu *Yahoo Messenger*. Layanan ini terintegrasi dengan layanan Yahoo yang lain, seperti *email* dan kalender. *Microsoft* pun merilis layanan *instant messaging* yang terintegrasi dengan *port*al *MSN* dan layanan *Hotmail*, yaitu *MSN Messenger*. Layanan *instant messaging* yang paling terakhir dirilis adalah *Google Talk*, yang diperkenalkan oleh *Google* sebagai *search engine* terbesar saat ini. *Google Talk* terintegrasi dengan layanan *Google* lainnya, terutama layanan *Gmail*.

Selain layanan tersebut, terdapat beberapa layanan *instant messaging* lain yang ada. Misalnya *QQ*, *Gadu-Gadu*, dan *Sametime*. Bahkan penyedia jasa *VoIP* (*Voice over Internet Protocol*) terkenal, *Skype*, juga memiliki fasilitas *instant messaging* dalam sistemnya.

Masing-masing layanan tersebut menggunakan protokolnya sendiri. Kebanyakan protokol tersebut bersifat tertutup (*proprietary*). Selain itu, pengguna layanan yang satu tidak dapat saling berkomunikasi dengan pengguna layanan yang lain. Oleh karena itu, ada usaha-usaha untuk mengembangkan protokol *instant messaging* yang terbuka dan dapat saling berkomunikasi. Saat ini, telah dikembangkan beberapa protokol *instant*

messaging yang bersifat terbuka, seperti *XMPP/Jabber* dan *SIP/SIMPLE*.

Selain itu, para penyedia layanan *instant messaging* besar juga mulai mengembangkan cara agar para pengguna layanan mereka dapat berkomunikasi dengan pengguna layanan lain. Misalnya *Yahoo* dan *Microsoft* yang telah sepakat untuk menyediakan fasilitas interkoneksi, sehingga pengguna kedua layanan dapat saling berkomunikasi. Selain itu, *Google* dan AOL juga mengembangkan fasilitas serupa.

Di kalangan perusahaan, berkembang pula sistem *instant messaging* khusus untuk internal perusahaan. Perusahaan dapat memasang perangkat *instant messaging* privat yang tidak terhubung dengan layanan *instant messaging* publik lain untuk digunakan oleh karyawannya. Dengan adanya sistem ini, diharapkan perusahaan dapat meningkatkan produktivitas dan efisiensi dalam berkomunikasi.

Saat ini, *instant messaging* merupakan media komunikasi yang paling cepat pertumbuhannya. Pada akhir tahun 2006, diperkirakan terdapat 390 juta pengguna, baik individual maupun *enterprise*. Dilaporkan terdapat 1 milyar pesan yang dikirimkan setiap harinya melalui layanan *instant messaging* besar seperti AIM, *MSN Messenger*, dan *Yahoo Messenger*. Diperkirakan pula, trafik *instant messaging* pada jaringan internet akan melebihi trafik *email* pada akhir tahun 2006.

2.2.2 Keuntungan *Instant Messaging*

Instant messaging memungkinkan efisiensi komunikasi dan memudahkan dalam berkolaborasi. Dibandingkan dengan *email* dan telepon, pengguna *instant messaging* dapat mengetahui jika seorang kontak atau rekannya dapat dihubungi (*available*). Kebanyakan sistem *instant messaging* juga memungkinkan pengguna untuk mengeset status *online* mereka sehingga teman-temannya dapat mengetahui apakah pengguna tersebut sedang *available*, sibuk (*busy*), atau pergi dari depan komputer (*away*). Di sisi lain, pengguna tidak harus menjawab setiap pesan yang masuk sesegera mungkin. Oleh karena itu, penggunaan *instant messaging* dianggap lebih sopan dan tidak 'memaksa' seperti layaknya telepon.

Selain itu, penggunaan *instant messaging* juga cocok untuk saling bertukar informasi singkat seperti alamat URL atau potongan teks dokumen. Hal-hal ini sulit dilakukan jika menggunakan media telepon.

Para penyedia layanan *instant messaging* pun berlomba-lomba untuk menambah fasilitas pada layanan mereka. Beberapa fasilitas yang lazim ditemukan pada *instant messaging* meliputi:

a. *Chat*

   Melakukan percakapan dengan kontak.

b. *Buddy list & presence information*

   Menyimpan daftar rekan yang biasa dihubungi serta mengetahui rekan manasaja yang sedang *online*.

c. *Conference*

   Melakukan percakapan dengan banyak orang sekaligus dalam satu sesi.

d. *File transfer*

Memungkinkan untuk berkirim *file* dengan pengguna lain.

e. *History*

Menyimpan daftar percakapan yang telah dilakukan.

Selain itu, beberapa layanan *instant messaging* juga memberikan nilai tambah berupa integrasi dengan layanan mereka yang lain, seperti:

a. Notifikasi *email*

Memberi informasi jika ada *email* baru yang masuk ke *mailbox* pengguna.

b. *Content*

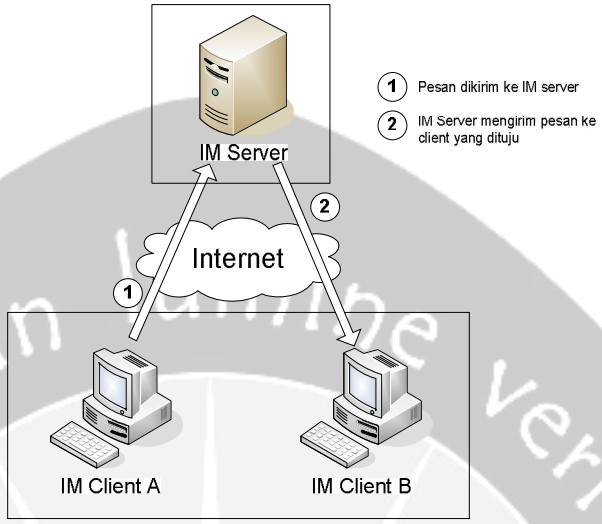
Menampilkan informasi seperti berita terbaru, cuaca, harga saham, maupun jadwal dalam kalender.

c. *Voice* & video *chat*

Bercakap-cakap dengan menggunakan media suara dan *video*, sehingga pengguna dapat melihat dan mendengar rekan yang sedang diajak berkomunikasi.

2.2.3 Cara Kerja *Instant Messaging*

Pada umumnya, sistem *instant messaging* bekerja berdasarkan arsitektur *client-server*. Pengguna menjalankan *instant messaging client* pada komputernya dan kemudian program *client* tersebut akan berkomunikasi dengan *server instant messaging* yang telah disediakan oleh pengelola layanan untuk bertukar pesan atau informasi tentang pengguna dengan pengguna lain.



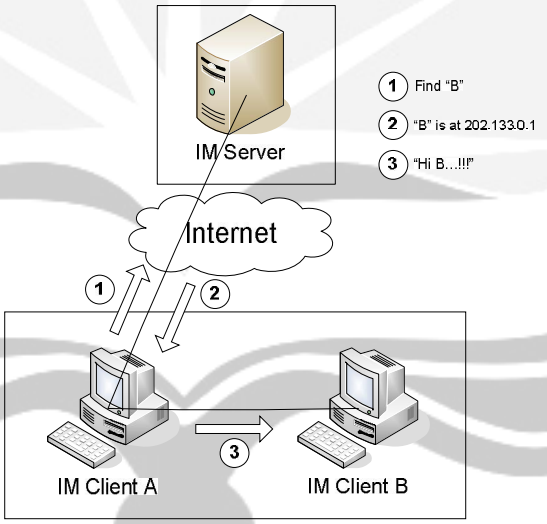
Gambar 2.1 *Instant messaging* dengan arsitektur *client-sever*

Pada kebanyakan sistem *instant messaging* berarsitektur *client-server*, pesan yang dipertukarkan di antara pengguna berupa *plaintext*. Pesan ini sangat rentan terhadap penyadapan di sepanjang jalur komunikasi dari *client* ke *server*. Hanya beberapa sistem *instant messaging* saja yang mengenkripsi pesan yang dikirimkan dari *client* ke *server* dan sebaliknya.

Dapat dilihat pada Gambar 2.1 bahwa walaupun kedua komputer berdekatan atau berada pada jaringan yang sama, namun trafik yang dikirimkan di antara kedua komputer tersebut harus melalui jaringan internet global. Dengan penggunaan *tools* yang tepat, seseorang dapat menyadap komunikasi yang terjadi antara kedua komputer tersebut walaupun penyadap berada di luar jaringan lokal.

Beberapa sistem *instant messaging* juga memiliki mekanisme koneksi dengan arsitektur *peer-to-peer*. Pada sistem dengan arsitektur *peer-to-peer*, *client* menghubungi *server* untuk mengetahui lokasi pihak lain yang akan dihubungi. Setelah mengetahui lokasinya, *client* akan menghubungi pihak lain secara langsung.

Semua informasi yang dibutuhkan agar program *client instant messaging* dapat menghubungi *server* telah diprekonfigurasi. Misalnya daftar alamat *server* yang harus dihubungi ketika akan memulai sesi percakapan. Setelah terhubung, *client instant messaging* dapat bertukar pesan dengan *client* lainnya.



Gambar 2.2 *Instant messaging* dengan arsitektur *peer-to-peer*

Banyak perusahaan dan organisasi yang memasang *firewall* pada jaringan internalnya. Tujuannya untuk membatasi akses hanya pada layanan tertentu yang

dianggap penting, seperti layanan *email*, *web*, atau DNS. Oleh karena itu, penyedia layanan *instant messaging* biasanya mendesain agar program *client* layanan tersebut dapat melewati *firewall* dengan berbagai teknik, misalnya dengan membuat tunnel di atas layanan yang diizinkan oleh *firewall*.

Pada kebanyakan program *client instant messaging*, jika koneksi dengan *server* layanan gagal, maka *client* akan mencoba beberapa langkah lain agar dapat menghubungi *server*. Sebagai contoh, *client* akan mencoba mengontak *server* pada *port* yang umum seperti *port* 80 (HTTP). Jika *firewall* di-*set* untuk mengizinkan paket melalui *port* ini, maka koneksi akan berhasil.

2.3 *Voice* dan *Video Conference*

Pada saat dua orang atau lebih berkumpul untuk bertukar informasi, berkolaborasi, bertukar pikiran, berdebat, atau berdiskusi, bisa dikatakan mereka melakukan konferensi (*conference*).

Conference biasanya dilakukan dengan bertatap muka secara langsung. Tetapi di jaman globalisasi sekarang ini, dimana kebutuhan untuk bertukar informasi secara cepat, murah dan efisien sangat dibutuhkan, bertatap muka secara langsung tidak mudah dilakukan.

Kemajuan di bidang teknologi komunikasi telah membuat *conference* dapat dilakukan dari jarak jauh. *Conference* yang dilakukan dari jarah jauh disebut *teleconference*.

Jika anda pernah melakukan hubungan telepon tiga arah, anda telah melakukan *teleconference*. Ketika berbicara tentang *teleconferencing*, kita berbicara tentang *conference call*, dimana sekelompok kecil atau besar orang yang berpartisipasi didalam percakapan tesebut.

Teknologi *teleconference* yang hanya berupa *conference call*, sejalan dengan perkembangan internet, kemudian berkembang kearah *web conference* dan *web seminar*. Konsep *web conference* dan *web seminar* sebenarnya sama, yang membedakannya ada jumlah orang yang terlibat didalamnya. *Web conference* biasanya terdiri tidak lebih dari 10 orang, sedangkan *web seminar* terdiri dari banyak orang, ratusan atau bahkan sampai ribuan orang, walaupun hanya beberapa orang saja yang berhak untuk berbicara, sedangkan yang lainnya hanya mengamati.

Web conference dan *web seminar* pada awalnya digunakan untuk kepentingan usaha dan pendidikan saja dengan menggunakan insfrastruktur pribadi atau layanan penyedia web conference dengan biaya yang cukup malah. Dengan semakin banyaknya pengguna internet kebutuhan *web conference* untuk kepentingan personal dengan biaya yang terjangkau semakin meningkat. Untuk itu dikembangkanlah bentuk sederhana dari *web conference*.

Unsur penting dari *web conference* adalah *voice* dan *video conferencing*. Aplikasi untuk melakukan *voice* dan *video conference* untuk kepentingan pribadi kemudian banyak dibuat oleh pengembang perangkat lunak. *Voice* dan *video conference* bisa dilakukan dengan *headset*

(*earphone* dan *microphone*), *webcam* atau kamera *video* digital yang terpasang pada komputer yang terhubung pada jaringan LAN atau internet.

Berikut cara kerja *voice* dan *video conferece*:

1. *Video* diambil dari kamera *video* dital atau *webcam.*
2. Suara diambil dari *microphone*.
3. *Video* dan suara tersebut kemudian bisa dikompres berdasarkan *codec* tertentu (*MPEG, Widows Media, Quictime*).
4. Kemudian data *video* dan suara yang telah terkompres dikirimkan melalui jaringan *IP* (*IP network*).
5. Setelah sampai di tujuannya, kompresi data *video* dan suara dibuka di-*decode*. *Video* akan tampil dilayar monitor dan suara dapat didengarkan melalui *earphone*.

Aplikasi *voice* dan *video conferencing* saat ini berkembang pesat. Trend terakhir adalah menggabungkan fasilitas *voice* dan *video conferencing* pada aplikasi *instant messenger*. Fasilitas ini menjadi sangat populer karena sebagian besar aplikasi *instant messenger* gratis dan tidak memungut biaya tambahan untuk fasilitas *voice* dan *video conference* tersebut.

2.4 Pemrograman *Socket*

Sistem operasi *Unix* telah merevolusi dunia pemrograman. Salah satunya adalah *file descriptor*. *File descriptor* menyediakaan *interface* pemrograman untuk *file object*. Karena hampir semua obyek di sistem *Unix* didefinisikan sebagai *file*. Maka *file descriptor* dapat digunakan untuk mengirim dan menerima data antar sistem *Unix*. Ini mempermudah pemrograman pada sistem *Unix*. Model pemrograman yang sama dapat diimplementasikan tanpa menghiraukan jenis perangkat atau *file* yang akan diakses.

Mulai pada sistem *Unix* versi *4.2BSD*, akses jaringan juga didefinisikan menggunakan *file descriptor*. Sebuah *network file descriptor* disebut *socket*. Program jaringan *Unix* dan *Windows* keduanya menggunakan *socket* untuk semua komunikasi jaringan.

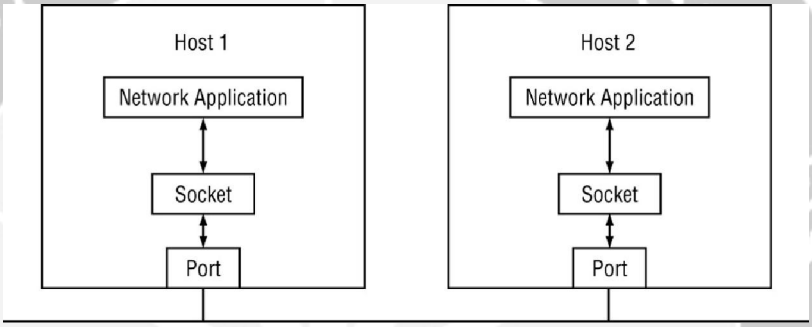
Pada pemrograman jaringan yang berdasarkan *socket*, pengiriman dan penerimaan paket data melalui jaringan tidak langsung melalui *network interface device*. Sistem Unix mengatasi pemilihan *network interface device* yang akan digunakan dalam pengiriman data dan bagaimana pengiriman dilakukan.

File descriptor yang khusus digunakan sebagai referensi koneksi jaringan disebut *sokcets*. *Sokect* mendefinisikan hal-hal berikut:

1. Domain komunikasi, koneksi jaringan atau *Unix Interprocess Communication* (IPC).
2. Tipe komunikasi, *stream* atau *datagram*.

3. Protokol, TCP atau UDP.

Setelah *socket* dibuat, *socket* harus diikat (*bind/bound*) pada *network address* dan *port* pada sistem *local* atau *network address* dan *port* pada sistem *remote*. Setelah *socket* diikat, maka bisa digunakan untuk mengirim dan menerima data.



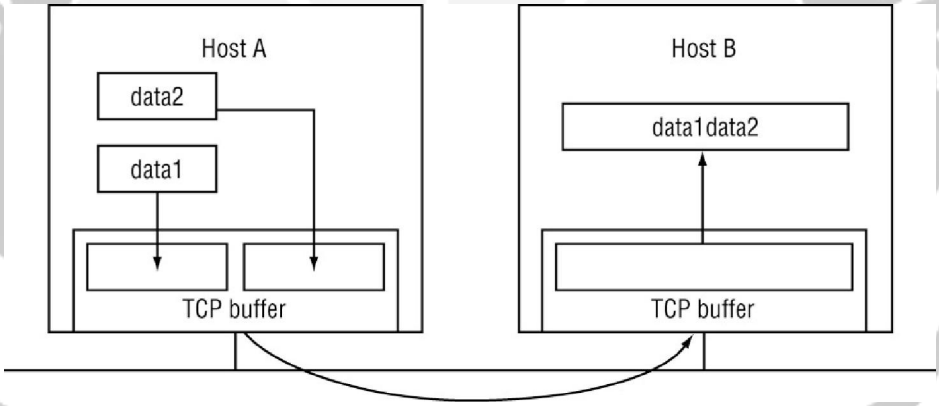
Gambar 2.3 *Socket Interface*

2.4.1 TCP (*Transmission Control Protocol*)

2.4.1.1 Konsep Dasar TCP

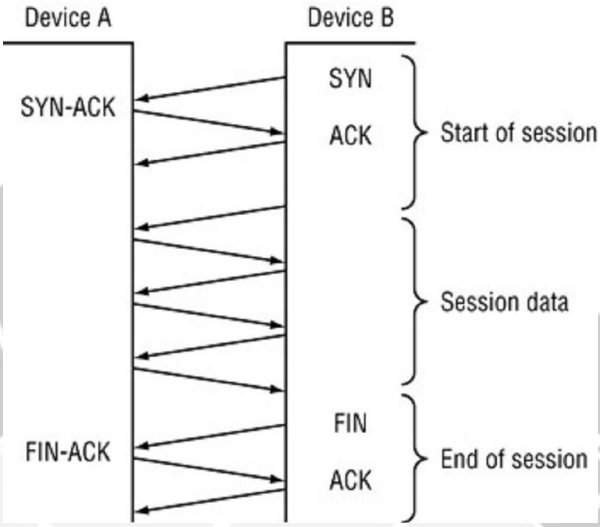
Hal yang paling penting untuk diingat pada penggunaan protokol TCP adalah bahwa TCP merupakan *connection-oriented protocol*. Sekali koneksi terjadi antara dua *device*, aliran data yang *reliable* antara keduanya tercipta, untuk memastikan bahwa data yang dipindahkan antara satu *device* ke *device* lainnya dilakukan secara akurat. Walaupun dengan TCP aplikasi anda tidak perlu mencemaskan data yang hilang atau data yang tidak berurutan, ada satu pertimbangan tertentu jika anda menggunakan pemrograman dengan TCP: *buffer*.

Karena TCP harus memastikan integritas data, maka data yang dikirimkan disimpan di *buffer* lokal sampai ada *acknowlagement* dari penerima data. Begitu juga sebaliknya, ketika menerima data dari jaringan, TCP harus menyimpan data yang diterima di *buffer* lokal untuk memastikan semua potongan data diterima dengan benar sebelum data tersebut disampaikan ke aplikasi. Dengan demikian pemrosesan data di applikasi di *device* lokal tidak selalu sama dengan aplikasi di *remote decive*. Gambar 2.4 berikut menunjukan cara kerjanya:



Gambar 2.4 Transfer data TCP

Pengiriman paket data yang membutuhkan *acknowlagement* pada setiap sesi pengiriman paket data pada protokol TCP disebut sebagai "*3-way handshake*".



Gambar 2.5 *"3-Way Handshake"* Pada Protokol TCP

Subsistem TCP di sistem operasi *Windows* bertanggung jawab atas penerimaan data dari *remote device*. Data yang diterima akan ditempatkan di *buffer* untuk waktu tertentu baru kemudian dikirimkan ke jaringan. Sementara itu aplikasi anda mungkin mengirimkan data lagi ke *remote host*, yang akan menyebabkan bertambahnya data yang disimpan di *buffer* (pada gambar diatas: *data1* dan *data2*). Ketika subsistem TCP sudah siap mengirim data ke *remote device*, akan dicoba untuk mengirimkan semua data dalam satu paket, bukan potongan-potongan data, sehingga baik *buffer data1* maupun *data2* dikirimkan dalam satu paket. *Host* penerima menerima data dalam satu paket yang berisi *data1* dan *data2*. Kemudian aplikasi akan menentukan apakah *data1* dan *data2* merupakan data yang terpisah atau suatu kesatuan data.

Karena TCP mengirimkan data sebagai satu kesatuan aliran (*stream*) data maka batasan-batasan data tidak dapat dipertahankan. Oleh sebab itu maka perlu ditambahkan mekanisme pembacaan data tersendiri di tingkat aplikasi. Ada dua cara untuk mengatasi hal tersebut:

1. Membuat protokol yang memerlukan responsi *one-for-one* untuk setiap data yang dikirim dari *host*.
2. Merancang penanda pesan data untuk menentukan batasan-batasan diantara aliran data.

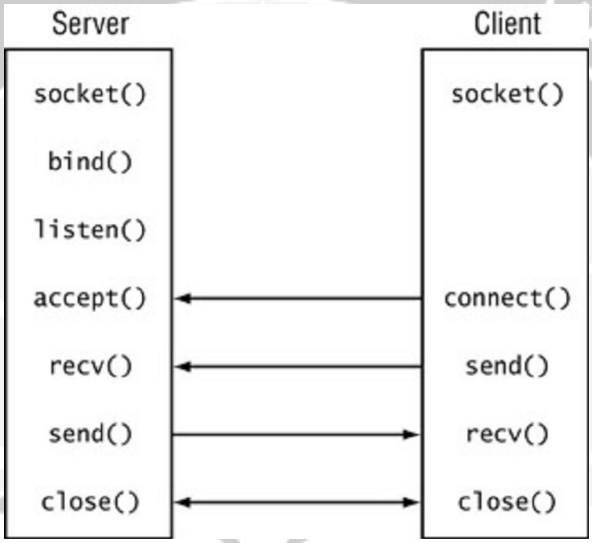
Dua cara diatas membutuhkan pertimbangan dan perencanaan tertentu sebelum pembuatan aplikasi dimulai. Kebanyakan protokol internet yang menggunakan cara yang pertama untuk memilah-milah data. FTP dan protokol yang serupa mengimplementasikan sistem *command/response* dimana *client* mengirim satu pesan *command* terlebih dahulu dan menunggu respon untuk setiap *command* dari *romote host*.

2.4.1.2 Pemrograman *Socket* di *TCP*

Untuk menciptakan *socket* yang *connection-oriented* pada protokol TCP diperlukan fungsi-fungsi berikut:

1. **Socket()** - *Server/client* menciptakan *socket*.
2. **Bind()** - *Server* memberi nama *socket*.
3. **Listen()** - *Server* menunggu hubungan dari *client*.

4. **Connect()** - *Client* menghubungi *server*.
5. **Send()** - *Server/client* mengirim data.
6. **Recv()** - *Server/client* menerima data.
7. **Close()** - *Server/client* memutuskan hubungan.



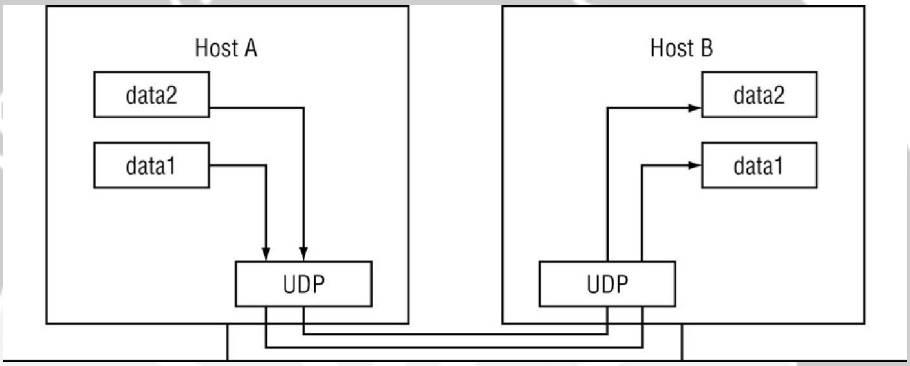
Gambar 2.6 Fungsi *socket* pada TCP

2.4.2 UDP (*User Datagram Protocol*)

2.4.2.1 Konsep Dasar UDP

UDP dibuat untuk memecahkan permasalahan pada TCP. UDP menjaga batasan-batasan dari semua data yang dikirim dari suatu aplikasi ke jaringan. Karena UDP secara khusus dirancang untuk tidak menghiraukan keandalan (*unreliable*) data yang dikirim, maka tidak diperlukan *buffer* untuk penyimpanan data sementara. Sebagai gantinya, setiap dikirimkan sebagai satu paket

seperti yang diterima dari aplikasi. Dan masing-masing pesan yang diterima dari jaringan dikirimkan ke applikasi sebagai satu pesan data. UDP dapat menjaga pesan dalam satu batasan di paket jaringan seperti digambarkan pada Gambar 2.7 berikut ini:



Gambar 2.7 Transfer data UDP

Terlepas dari kemampuan UDP untuk menjaga batasan-batasan pesan, ada satu masalah lain yang muncul. Karena UDP tidak menjamin pengiriman data, aplikasi harus menjalankan fungsi untuk memastikan data yang dikirimkan benar-benar sampai pada tujuannya. Hanya karena satu *device* mengirim paket data UDP, bukan berarti *device* yang dituju pasti menerima pesan yang dikirimkan. Harus dipastikan bahwa aplikasi data menangani paket-paket data yang hilang.

1. Kirim data ke *remote device*.
2. Menentukan batasan waktu (*timer*) penerimaan jawaban.

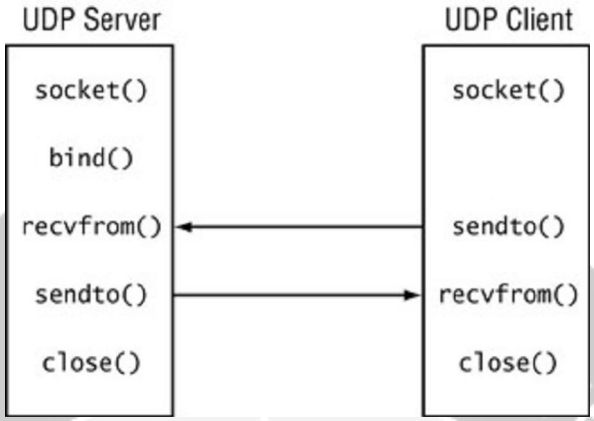
3. Menunggu jawaban dari *remote device*. Saat menerima jawaban *timer* dihentikan.
4. Jika sampai batasan waktu habis masih belum menerima jawaban, kembali ke langah 1. Tentukan jumlah repetisi untuk langkah 3 untuk menentukan asumsi bahwa *remote host* tidak bisa dihubungi.

Meskipun pengiriman data UDP dianggap lebih mudah dilihat dari segi batasan data, sebenarnya UDP lebih rumit dari pada TCP karena mekanisme pemeriksaan data yang hilang harus dibuat sendiri.

2.4.2.2 Pemrograman *Socket* di UDP

Untuk menciptakan *socket* protokol UDP diperlukan fungsi-fungsi berikut:

1. **Socket()** – *Server/client* menciptakan *socket*.
2. **Bind()** – *Server* memberi nama *socket*.
3. **Sendto()** / **recvform()** – *Server/client* mengirim dan menerima data.
4. **Close()** – *Server/client* memutuskan hubungan.



Gambar 2.8 Fungsi Socket pada UDP

2.5 Arsitektur Aplikasi

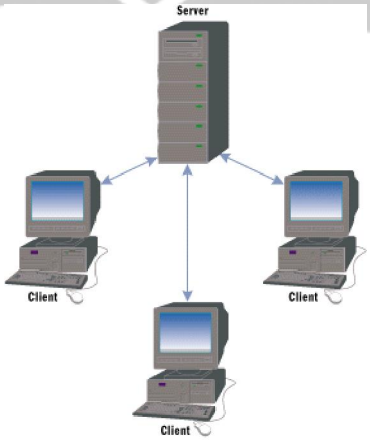
2.5.1 *Client-Server*

Client-server merupakan sebuah paradigma dalam teknologi informasi yang merujuk kepada cara untuk mendistribusikan aplikasi ke dalam dua pihak: pihak *client* dan pihak *server*.

Dalam model *client-server*, sebuah aplikasi dibagi menjadi dua bagian yang terpisah, tapi masih merupakan sebuah kesatuan yakni komponen *client* dan komponen *server*. Komponen *client* juga sering disebut sebagai *front-end*, sementara komponen *server* disebut sebagai *back-end*. Komponen *client* dari aplikasi tersebut dijalankan dalam sebuah *workstation* dan menerima masukan data dari pengguna. Komponen *client* tersebut akan menyiapkan data yang dimasukkan oleh pengguna dengan menggunakan teknologi pemrosesan tertentu dan mengirimkannya kepada komponen *server* yang dijalankan di atas mesin *server*, umumnya dalam bentuk *request*

terhadap beberapa layanan yang dimiliki oleh *server*. Komponen *server* akan menerima *request* dari *client*, dan langsung memprosesnya dan mengembalikan hasil pemrosesan tersebut kepada *client*. *Client* pun menerima informasi hasil pemrosesan data yang dilakukan *server* dan menampilkannya kepada pengguna, dengan menggunakan aplikasi yang berinteraksi dengan pengguna.

Sebuah contoh dari aplikasi *client-server* sederhana adalah aplikasi *web* yang didesain dengan menggunakan *Active Server Pages* (*ASP*) atau *PHP*. Skrip *PHP* atau *ASP* akan dijalankan di dalam *web server* (*Apache* atau *Internet Information Services*), sementara skrip yang berjalan di pihak *client* akan dijalankan oleh *web browser* pada komputer *client*. *Client-server* merupakan penyelesaian masalah pada *software* yang menggunakan *database* sehingga setiap komputer tidak perlu di-*install database*, dengan metode *client-server database* dapat di-*install* pada suatu komputer sebagai *server* dan aplikasinya di-*install* pada *client*.



Gambar 2.9 Arsitektur *Client-Server*

2.5.2 *Peer-to-Peer*

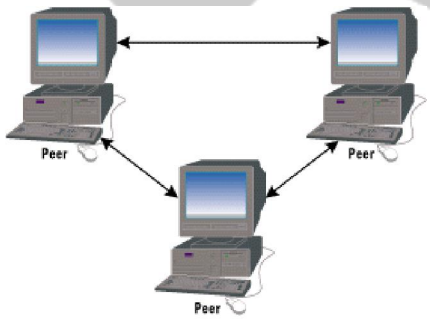
Peer-to-Peer (*P2P*) merupakan sebuah jaringan yang terdiri atas dua komputer atau lebih yang menggunakan program yang sama atau menggunakan jenis program yang sama untuk saling berkomunikasi dan berbagi data. Setiap komputer, yang dalam arsitektur *peer-to-peer* disebut dengan "*Peer*", dianggap sama dan setiap *peer* juga bertindak sebagai *server* bagi *peer* lainnya di dalam jaringan tersebut, yang bertindak sebagai *client*-nya. Tidak seperti arsitektur *client-server* yang mendedikasikan sebuah komputer agar menjadi *server*, dalam arsitektur *peer-to-peer*, *server* terdedikasi tidak dibutuhkan. Meski menghemat biaya pembelian *server* (yang umumnya berharga lebih mahal dibandingkan komputer *desktop* biasa), kinerja yang ditunjukkan metode ini umumnya tidak sebagus apa yang dilakukan oleh *client-server*, khususnya dalam beban yang berat.

Ide mengenai konsep ini muncul kira-kira pada akhir dekade 1980-an, ketika jaringan komputer telah mulai masuk ke dalam salah satu barang wajib dalam perusahaan, baik itu perusahaan kecil maupun besar. Tetapi, arsitektur ini berkembang dalam jaringan yang terlalu kecil untuk memiliki sebuah *server* yang terdedikasi, sehingga setiap komputer *client* pun menyediakan layanan untuk berbagi data untuk melakukan kolaborasi antara pengguna.

Jaringan *peer-to-peer* pun mulai banyak digemari ketika *Microsoft* merilis sistem operasi *Windows for Workgroups*, meski sebelumnya sistem operasi *MS-DOS* (*IBM PC-DOS*) dengan perangkat *MS-NET* (*PC-NET*) juga dapat

digunakan untuk tujuan ini. Karakteristik kunci jaringan tersebut adalah dalam jaringan ini tidak terdapat sebuah *server* pusat yang mengatur *client-client*, karena memang setiap komputer bertindak sebagai *server* untuk komputer *client* lainnya. Sistem keamanan yang ditawarkan oleh metode ini terbilang lebih rendah dibandingkan dengan metode *client-server* dan manajemen terhadapnya pun menjadi relatif lebih rumit.

Konsep ini pun kemudian berevolusi pada beberapa tahun terakhir, khususnya ketika jaringan internet menjadi jaringan yang sangat besar. Hal ini mulai muncul kira-kira pada akhir dekade 1990-an, di saat banyak pengguna internet mengunduh banyak berkas musik *mp3* dengan menggunakan metode *peer-to-peer* dengan menggunakan program *Napster* yang menuai kritik pedas dari industri musik. Selanjutnya beberapa aplikasi juga dibuat dengan menggunakan konsep ini: *eDonkey*, *Kazaa*, *BitTorrent*, dan masih banyak lainnya. Meski banyak aplikasi *peer-to-peer* ini digunakan oleh pengguna rumahan, ternyata sistem ini juga diminati oleh banyak perusahaan juga.



Gambar 2.10 Arsitektur *Peer-to-Peer*

2.5.3 *Hybrid Peer-to-Peer*

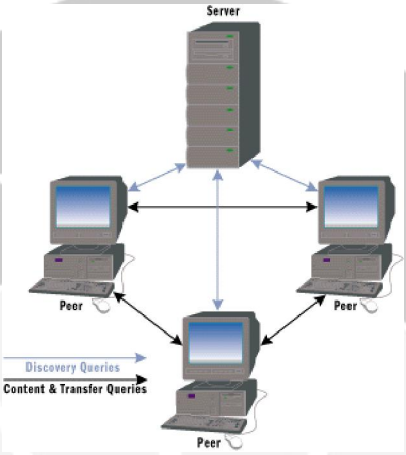
Hybrid peer-to-peer merupakan arsitektur *peer-to-peer* yang tidak sepenuhnya berdiri sendiri tanpa bantuan dari *server*. Pada arsitektur ini dibutuhkan satu *server* untuk menghubungkan *peer-peer*.

Server pada arsitektur ini memiliki fungsi yang lebih ringan dibandingkan *server* pada arsitektur *client-server*. Fungsi utama *server* di arsitektur ini adalah sebagai *"simple discovery server"* yang menyediakan informasi *peer / client* yang sedang *online*. Sehingga setiap saat aplikasi *client* dijalankan, harus terhubung ke *server* untuk meminta daftar *peer* yang *online* dan kemudian komunikasi *peer-to-peer* dapat dilakukan tanpa perantara *server*.

Selain sebagai *"simple discovery server"*, *server* pada arsitektur ini juga bisa berfungsi sebagai *"lookup server"*. *Lookup server* berfungsi untuk menyimpan informasi tentang *content* yang dimiliki oleh masing-masing *peer*. Dengan demikian pada saat satu *peer* hendak mencari *content* tertentu, tidak perlu mencari ke semua *peer* tetapi cukup meminta informasi dari *server*, *peer* mana yang memiliki content tersebut.

Contoh aplikasi dengan arsitektur *hybrid peer-to-peer* yang banyak digunakan adalah aplikasi *sharing file Bittorent* dan aplikasi *instant messenger* seperti *Yahoo Messenger* dan *MSN Messenger*. Aplikasi tersebut sebenarnya merupakan aplikasi *peer-to-peer*, tetapi bergantung kepada satu *server* sebagai sentral penyimpanan data atau informasi *peer* yang *online* agar

peer-peer dapat dengan mudah dan cepat terhubung dengan *peer* yang *online*.



Gambar 2.11 Arsitektur *Hybrid Peer-to-Peer*

2.6 *Tools* dan Teknologi yang Digunakan

2.6.1 *Microsoft .Net Framework*

Microsoft .NET Framework (dibaca *Microsoft Dot Net Framework*) adalah sebuah komponen yang dapat ditambahkan ke sistem operasi *Microsoft Windows* atau telah terintegrasi ke dalam *Windows* (mulai dari *Windows Server* 2003 dan versi-versi *Windows* terbaru). *Framework* ini menyediakan sejumlah besar solusi-solusi program untuk memenuhi kebutuhan-kebutuhan umum suatu program baru, dan mengatur eksekusi program-program yang ditulis secara khusus untuk *framework* ini. *.NET Framework* adalah kunci penawaran utama dari *Microsoft*, dan dimaksudkan untuk digunakan oleh sebagian besar aplikasi-aplikasi baru yang dibuat untuk *platform Windows*.

Pada dasarnya, *.NET Framework* memiliki 2 komponen utama: CLR dan *.NET Framework Class Library*.

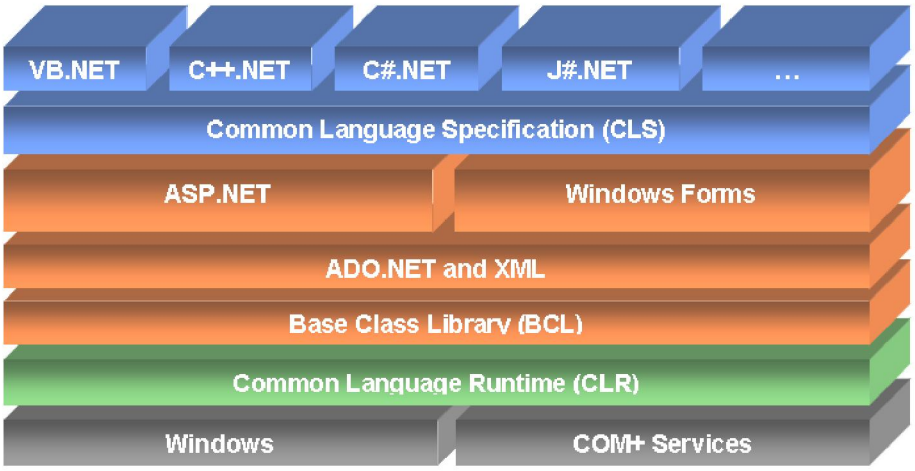
Program - program yang ditulis untuk *.NET Framework* dijalankan pada suatu lingkungan *software* yang mengatur persyaratan-persyaratan *runtime program*. *Runtime environment* ini, yang juga merupakan suatu bagian dari *.NET Framework*, dikenal sebagai *Common Language Runtime* (CLR). CLR menyediakan penampilan dari *application virtual machine*, sehingga para *programmer* tidak perlu mengetahui kemampuan *CPU* tertentu yang akan menjalankan program. CLR juga menyediakan layanan-layanan penting lainnya seperti jaminan keamanan, pengaturan memori, *garbage collection* dan *exception handling* / penanganan kesalahan pada saat *runtime*. *Class library* dan CLR ini merupakan komponen inti dari *.NET Framework*. Kerangka kerja itu pun dibuat sedemikian rupa agar para *programmer* dapat mengembangkan program komputer dengan jauh lebih mudah, dan juga untuk mengurangi kerawanan aplikasi dan juga komputer dari beberapa ancaman keamanan.

CLR adalah turunan dari CLI (*Common Language Infrastructure*) yang saat ini merupakan standar *ECMA*.

Solusi-solusi program pembentuk *class library* dari *.NET Framework* melingkupi area yang luas dari kebutuhan program pada bidang *user interface*, pengaksesan data, koneksi basis data, kriptografi, pembuatan aplikasi berbasis *web*, algoritma numerik, dan komunikasi jaringan. Fungsi-fungsi yang ada dalam *class library* dapat digabungkan oleh *programmer* dengan kodenya sendiri untuk membuat suatu program aplikasi baru.

Tujuan dari *.NET Framework* adalah :

1. Menyediakan lingkungan pemrograman berorientasi objek, apakah kode objek disimpan dan dijalankan secara lokal, dijalankan secara lokal tetapi disebarkan melalui internet atau dijalankan secara *remote* (dijalankan dari suatu tempat).
2. Menyediakan lingkungan untuk menjalankan suatu kode yang menjamin keamanan saat kode dijalankan.
3. Menyediakan lingkungan untuk menjalankan suatu kode yang dapat mengeliminasi masalah performa dari lingkungan *scripted* dan *interpreted*.
4. Menyediakan lingkungan untuk menjalankan suatu kode yang meminimalkan konflik pada *deployment* dan *versioning* perangkat lunak.
5. Menyatukan model-model pemrograman dengan didukung oleh banyak bahasa dan membuat berbagai tipe aplikasi.



Gambar 2.12 Arsitekur *.NET Framework*

2.6.2 *Microsoft Studio .NET*

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi *Windows*, ataupun aplikasi *Web*. *Visual Studio* mencakup kompiler, *SDK*, *Integrated Development Environment* (*IDE*), dan dokumentasi (umumnya berupa *MSDN Library*). Kompiler yang dimasukkan ke dalam paket *Visual Studio* antara lain *Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro,* dan *Visual SourceSafe*.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas *Windows*) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas *.NET Framework*). Selain itu, *Visual Studio* juga dapat digunakan untuk mengembangkan aplikasi *Windows Mobile* (yang berjalan di atas *.NET Compact Framework*).

Visual Studio kini telah menginjak versi *Visual Studio* 9.0.21022.08, atau dikenal dengan sebutan *Microsoft Visual Studio 2008* yang diluncurkan pada 19 November 2007, yang ditujukan untuk platform *Microsoft .NET Framework 3.5*. Versi sebelumnya, *Visual Studio 2005* ditujukan untuk platform *.NET Framework* 2.0 dan 3.0. *Visual Studio 2003* ditujukan untuk *.NET Framework* 1.1, dan *Visual Studio 2002* ditujukan untuk *.NET Framework* 1.0. Versi-versi tersebut di atas kini dikenal dengan sebutan *Visual Studio .NET*, karena

memang membutuhkan *Microsoft .NET Framework*. Sementara itu, sebelum muncul *Visual Studio .NET*, terdapat *Microsoft Visual Studio* 6.0 (VS1998).

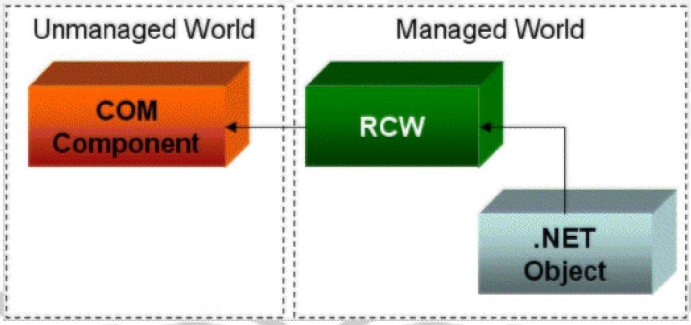
2.6.3 *.NET Interoperability*

.NET adalah suatu *platform* yang efisien untuk mengembangkan sebuah aplikasi, kode program yang penulisannya cukup membutuhkan banyak usaha, kini dapat ditulis dengan lebih sederhana. Tetapi investasi yang cukup besar telah diberikan pada teknologi lainnya seperti *Component Object Model* (*COM*) yang tidak dapat ditinggalkan begitu saja. Dan *.NET* belum tentu cocok untuk semua jenis aplikasi. Masih ada beberapa aplikasi yang lebih baik dilakukan oleh bahasa kelas rendah (*low-lavel languages*).

Agar *.NET* dapat diterima dengan mudah oleh para pengembang, maka pada saat pengembangan *.NET*, sudah dipikirkan mekanisme untuk mendukung teknologi sebelumnya. Mekanisme tersebut disebut *.NET interoperability*. *Interoperability* di *.NET* ada tiga jenis:

1. *Interoperability* .NET dengan komponen *COM* (disebut *COM interop*).
2. *Interoperability* komponen COM dengan *.NET* (disebut *.Net interop*).
3. *Interoperability .Net* dengan *WIN32 DDL* (disebut *Platform Invoke*)

Satu hal yang perlu dipahami adalah bahwa komponen yang ditulis dengan *.NET* adalah komponen yang termanajemen (*managed*), sedangkan komponen yang ditulis dengan COM tidak termanajemen (*unmanaged*). Masing-masing model memiliki cara yang berbeda untuk menangani pengalokasian memori, siklus suatu obyek atau pertukaran parameter antar fungsi, yang menyebabkan dua model ini membutuhkan penengah yang dapat menjebatani perbedaan tersebut. Dengan adanya penengah, tidak diperlukan penambahan kode untuk setiap komponen COM. Pengembang *.NET* menyadari pentingnya penengah tersebut, sehingga dibuatlah *Runtime Callable Wrapper* (*RCW*). *RCW* dapat menjembatani komunikasi antar *.NET* dan komponen *COM*.



Gambar 2.13 *.NET – RCW - COM*

2.6.4 *Microsoft Windows Multimedia API*

Microsoft Windows Multimedia API merupakan bagian dari *Win32 API* yang menyediakan fungsi-fungsi generik yang berhubungan dengan *audio* dan *video* untuk hampir semua jenis perangkat keras *multimedia*. Fungsi-fungsi *Windows Multimedia* dirangkum dalam *library winmm.lib*.

Microsoft Windows Multimedia API terdiri dari:

1. *Media Control Interface* (*MCI*), menyediakan perintah-perintah untuk memutar dan merekam berkas *multimedia*.
2. *Windows Audio*, menyediakan fungsi yang memungkinkan aplikasi untuk menggunakan layanan *audio*.
3. *Multimedia Input*, menyediakan fungsi yang memungkinkan aplikasi untuk menerima data *multimedia*.
4. *Video for Windows*, menyediakan fungsi yang memungkinkan aplikasi untuk mengolah data *video*.

2.6.5 *Microsoft DirectShow*

Microsoft DirectShow merupakan arsitektur untuk media *steaming* di sistem operasi *Windows*. *DirectShow* menyediakan penangkapan (*capture*) dan pemutaran (*playback*) data *multimedia* berkualitas tinggi. *DirectShow* mendukung banyak format, termasuk *Advanced System Format* (*ASF*), *Motion Picture Experts Groups* (*MPEG*), *Audio-Video Interleaved* (*AVI*), *MPEG Audio Layer-3* (*MP3*), dan *WAV*. *DirectShow* mendukung penangkapan (*capture*) menggunakan *Windows Driver Model* (*WDM*) atau perangkat *Video for Windows* (*VfW*). *DirectShow* terintegrasi dengan teknologi *DirectX* lainnya. *DirectShow* secara otomatis mendeteksi dan

menggunakan akselerasi perangkat keras *video* dan *audio* yang tersedia.

DirectShow dikembangkan menggunakan *Component Object Model* (*COM*). Untuk menulis suatu aplikasi *DirectShow* atau komponennya, anda harus memahami pemrograman *COM*. Untuk kebanyakan aplikasi, anda tidak perlu untuk membuat object *COM* sendiri. *DirectShow* telah menyediakan komponen yang anda perlukan. Tetapi jika anda ingin memperluas DirectShow dengan komponen tambahan, komponen tersebut harus diterapkan sebagai object *COM*. Applikasi *DirectShow* harus ditulis dalam bahasa *C++*.

Bekerja dengan multimedia dihadapkan pada beberapa tantangan:

- Data *multimedia* berisi sejumlah data yang besar, yang harus diproses dengan sangat cepat.
- *Audio* dan *video* harus disinkronisasi sehingga dapat dimulai dan berhenti bersamaan dengan *rate* yang sama.
- Data dapat berasal dari banyak sumber, termasuk *file* lokal, jaringan komputer, acara siaran televisi, dan kamera *video*.
- Data yang masuk bisa dalam berbagai format, seperti *Advanced System Format* (*ASF*), *Motion Picture Experts Groups* (*MPEG*), *Audio-Video Interleaved* (*AVI*), *MPEG Audio Layer-3* (*MP3*), dan *WAV* dan *Video Digital* (*DV*).

- Programmer itu tidak mengetahui perangkat keras yang tersedia di sistem *end-user*.

DirectShow dirancang untuk mengatasi tantangan-tantangan diatas. Tujuan utamanya adalah untuk menyederhanakannya dengan membuat aplikasi media digital yang bekerja di sistem operasi *Windows*, dengan menghilangkan kompleksitas pengolahan data, perbedaan perangkat keras dan sinkronisasi.

Untuk mencapai keluaran yang diperlukan untuk mengolah data *audio* dan *video*, *DirectShow* menggunakan *DirectDraw* dan *DirectSound*. Teknologi tersebut mengolah data secara efisien ke kartu suara (*sound card*) dan kartu grafik (*graphic card*). *DirectShow* mensinkronkan pemutaran (*playback*) media dengan mengenkapsulasi data media yang telah diberi tanda waktu (*tipe-stamped*). Untuk menangani berbagai sumber, format, dan perangkat keras yang mungkin digunakan, *DirectShow* menggunakan arsitetur modular, dengan memadu-padankan berbagai jenis komponen perangakat lunak yang disebut *filter*.

DirectShow menyediakan *filter-filter* yang mendukung penangkapan (*capture*) dan penyetelan (*tuning*) perangkat berdasarkan *Windows Driver Model* (*WDM*), dan juga *filter* yang mendukung perangkat *Video for Windows* (*VfW*), dan *codec* (*Compression Decompression*) yang ditulis untuk antarmuka (*interfaces*) *Audio Conpression Manager (ACM)* dan *Video Compression Manager (VCM).*