IMPLEMENTASI PROTOKOL TCP/IP UNTUK PENGENDALIAN KOMPUTER JARAK JAUH

Rudy Adipranata
Fakultas Teknologi Industri, Jurusan Teknik Informatika – Universitas Kristen Petra
e-mail : rudya@petra.ac.id

ABSTRAK: Sekarang ini jaringan komputer sudah menjadi suatu kebutuhan yang sangat penting untuk mempermudah pertukaran data antar komputer. Dan seiring dengan makin berkembangnya jumlah komputer pada suatu jaringan, maka makin bertambah pula tingkat kesulitan untuk mengelola jaringan tersebut. Oleh sebab itu pada penelitian ini akan dibuat aplikasi untuk mengendalikan komputer secara jarak jauh dengan menggunakan protokol TCP/IP.

Aplikasi ini dibuat dengan menggunakan bahasa pemrograman Delphi 5.0 dan dWinsock 2.75 yang merupakan komponen antar muka *Windows Socket API* yang dikhususkan untuk bahasa pemrograman Delphi, dan terdiri dari dua bagian yaitu *server* yang diaktifkan pada komputer yang akan dikendalikan, dan *client* yang diaktifkan pada komputer pengendali. Pada aplikasi ini data yang dikirimkan berbentuk teks dan biner, dan menggunakan dua pasang komponen utama dWinsock yaitu TTextServer dan TTextClient serta TBinaryServer dan TBinaryClient. Fungsi yang diaplikasikan ialah penguncian komputer, *reboot, shutdown,* eksekusi program, pengiriman pesan, melihat tampilan layar dan melihat program yang sedang dijalankan pada komputer *server*.

Pengujian dilakukan pada jaringan komputer internal dan juga pada jaringan internet. Hasil dari pengujian yang dilakukan pada jaringan internal didapat bahwa semua fungsi yang diaplikasikan dapat dijalankan dengan sempurna tanpa mengalami penundaan yang berarti karena *delay* yang terjadi di bawah 1 detik, sedangkan untuk jaringan internet terdapat delay yang besarnya bervariasi tergantung dari tingkat kepadatan *traffic* data pada saat tersebut.

Kata kunci: TCP/IP, jaringan komputer, pengendalian jarak jauh.

ABSTRACT: *This day, computer networking has become important necessity for data exchange between computers. And along with the growing of number of computer in a network, the difficulty for managing that network also increased. Because of that reality, in this paper will be build application for remote controlling computer using TCP/IP protocol.*

This application build using Delphi 5.0 programming language and dWinsock 2.75, component interface for Windows Socket API which is created especially for Delphi language, and separate to two sub applications, namely server application, which run on computer to be controlled and the secondly is client application, run on controller computer. In this application, data text or binary will be send, so two pair components will be used : TTextServer - TTextClient and TBinaryServer–TBinaryClient. The functions which implemented in this applications are : lock and unlock computer, reboot, shutdown, execute remote program, send message, capture screen and view remote programs.

Test is doing on internal network and internet network, and the result of that test for internal network, all of functions can be activated perfectly without significant delay (less than 1 second). But for internet network, there is delay, which the number variable depend on data traffic at that moment.

Keywords: *TCP/IP, computer network, remote control.*

1. PENDAHULUAN

Jaringan komputer sudah menjadi sesuatu kebutuhan yang sangat penting, dimana komputer tidak lagi berdiri sendiri tetapi saling terhubung satu sama lain. Dan seiring dengan makin berkembangnya jumlah komputer yang terdapat pada suatu jaringan, maka makin bertambah pula tingkat kesulitan untuk mengelola jaringan tersebut. Oleh sebab itu pada penelitian ini dibuat aplikasi untuk mengendalikan komputer secara jarak jauh dengan memanfaatkan protokol TCP/IP. Protokol TCP/IP adalah protokol yang paling banyak digunakan masa sekarang ini pada jaringan komputer dan merupakan protokol standard untuk Internet.

Protokol TCP/IP ini tersusun atas lima lapisan yaitu mulai dari lapisan teratas:

Application, *Transport*, *Internet*, *Network Access* dan lapisan terbawah adalah *Physical*. Aplikasi pengendalian komputer ini berada pada lapisan *Application*, dibuat khusus untuk sistem operasi Windows 9x dan disusun dengan bantuan bahasa pemrograman Delphi 5.0. Untuk komunikasi dengan lapisan di bawahnya (*Transport*) dapat menggunakan *library* yang disediakan oleh Windows (Windows API/*Application Programming Interface*), tetapi pada aplikasi ini digunakan komponen dWinsock 2.75. Komponen dWinsock digunakan untuk berkomunikasi dengan lapisan *Transport* karena pada komponen dWinsock telah disediakan fungsi-fungsi yang mudah untuk digunakan yang merupakan enkapsulasi dari Windows API.

Hasil dari aplikasi ini digunakan untuk melakukan pengendalian jarak jauh pada komputer lain yang meliputi : penguncian komputer, *reboot*, *shutdown*, eksekusi program, pengiriman pesan, melihat tampilan layar dan melihat program yang sedang dijalankan.

2. dWINSOCK

dWinsock adalah komponen antar muka untuk *Windows Socket API* yang dikhususkan bagi bahasa pemrograman Borland Delphi. Dengan menggunakan komponen dWinsock tidak lagi diperlukan pengubah untuk mengubah header-header *Windows Socket API* yang menggunakan bahasa pemrograman C++ menjadi Delphi.

Komponen dWinsock terdiri dari beberapa bagian yang mempunyai fungsi umum maupun spesifik. Yang mempunyai fungsi umum terdiri dari TBasicClientSocket dan TBasicServerSocket. Sedangkan yang mempunyai fungsi khusus adalah TBinaryClient dan TBinaryServer, TTextClient dan TtextServer, serta TTimeClient dan TTimeServer.

TBasicClientSocket menyediakan semua fungsi yang diperlukan untuk berhubungan ke *remote server*. Komponen ini mempublikasikan semua *property* yang telah didefinisikan pada komponen TcustomClientSocket. Untuk berhubungan dengan *server* menggunakan *stream socket*, dapat dilakukan dengan mengisi *property Address* menuju ke alamat *IP (Internet Protocol)* tujuan atau ke nama tujuan (misal: 202.43.253.4 atau www.petra.ac.id), dan mengisi *property* Port menuju ke nomor *port* atau nama *service* yang sesuai (misal : 80 atau 'http'). *Property* ini dapat diisi pada saat desain ataupun saat *runtime*, dan dilanjutkan dengan memanggil prosedur Open(TStreamSocket). Untuk mendapatkan obyek *socket* yang digunakan oleh komponen ini, dapat dilakukan pembacaan pada *property* Conn. *Property* ini dapat digunakan untuk mengirim dan menerima data. Selain itu juga dapat diketahui apakah suatu koneksi telah terbentuk atau belum dengan menggunakan *event* OnConnect, serta dapat pula mendeteksi apakah suatu koneksi telah ditutup oleh *socket* yang lain dengan menggunakan *event* OnDisconnect.

Ketika terdapat data yang siap dibaca pada *socket*, *event* OnRead akan dipanggil dan pada *event* tersebut dapat dilakukan pembacaan pada *property* Text atau dilakukan pemanggilan ke Recv. Terdapat *Event* OnRead akan selalu dipanggil setiap kali ada data yang siap dibaca, dan terdapat pula *event* OnWrite yang akan selalu dipanggil pada saat *socket* siap untuk melakukan penulisan data. *Event* OnRead dan OnWrite ini digunakan untuk melakukan pengiriman data yang tidak dapat dikirimkan jika menggunakan fungsi TsocketBase.Send.

Untuk mengirimkan datagram ke *server*, dapat dilakukan dengan mengisi alamat IP tujuan atau nama *host* dan *property* Port kemudian dilakukan pemanggilan Open (TDatagramSocket). Atau jika tidak diinginkan mengisi alamat IP tujuan, dapat dilakukan pemanggilan TDatagramSocket.SentTo.

TBasicServerSocket menyediakan semua fungsi yang dibutuhkan untuk menerima panggilan masuk suatu koneksi dari *remote clients*. Komponen ini mempublikasikan *property* dari TCustomBasicServerSocket. Untuk menerima koneksi dari *client*, harus dilakukan pengisian pada *property* Address dan Port atau nama *service*. Sama seperti TBasicClientSocket, pengisian ini dapat dilakukan pada saat desain atau *runtime*. *Client* yang melakukan koneksi ke *server* dapat dibatasi dengan menggunakan *property* Restriction. Untuk memulai pemo-

nitoran pada *port* dapat dilakukan dengan memanggil fungsi Listen, dan saat terjadi koneksi masuk maka *event* OnAccept akan dipanggil.

Event-event lain yang terdapat pada TBasicServerSocket ini diantaranya ialah *event* OnDisconnect, yang akan dipanggil pada saat *client socket* ditutup, *event* OnRead, yang akan dipanggil pada saat salah satu *client* siap untuk melakukan pembacaan dan *event* OnWrite, yang akan dipanggil pada saat salah satu *client* siap melakukan penulisan.

TBinaryClient adalah komponen yang menyediakan fungsi-fungsi untuk pengiriman data biner. Terdapat *property* RecordSize untuk pengiriman dengan jumlah data yang tidak berubah. Untuk mengakses metode *socket* sehingga dapat melakukan pengiriman data, dapat dilakukan *casting* tipe *property* Conn menjadi TBufferedSocket seperti: *(BinaryClient1. Conn as TLineSocket).WriteToBuffer(mybuf, SizeOf(mybuf));*. Untuk pengiriman *file*, dapat dilakukan dengan metode SendFile dan RecvFile.

TBinaryServer menyediakan fungsi untuk mengatur pengiriman data ke *buffered sockets*. Komponen ini khusus digunakan untuk berfungsi sebagai *server* dimana data yang dikirimkan berbentuk biner. Sama seperti TBinaryClient, untuk mengetahui jumlah data yang dikirimkan dapat dilihat pada *property* RecordSize. Jika memanggil prosedur Listen, akan terbentuk sebuah *socket* dan akan menimbulkan *exception* jika kelas *socket* bukan turunan dari TbufferedSocketBase.

TTextClient menyediakan fungsi untuk mengatur pengiriman data berbentuk teks ke Line Sockets, sehingga komponen ini ideal jika digunakan dalam pengiriman data yang berbasi baris-baris teks. Terdapat *event* OnLineReceived yang akan dipanggil jika *socket* menerima data baris baru. Dan terdapat metode WriteLine yang digunakan untuk menulis baris teks ke *buffer* dan kemudian akan menambahkan karakter Enter (CR) dan LineFeed (LF) yang sesuai pada tiap akhir baris.

Untuk mengakses metode *socket* sehingga dapat melakukan pengiriman data, harus dilakukan *casting* pada tipe *property* Conn menjadi TLineSocket seperti contoh berikut: *(TextClient1.Conn as TLineSocket). WriteLine ('Hello');*. Untuk mengirim *file* dapat digunakan metode SendFile dan untuk menerima *file* dapat digunakan metode RecvFile.

TTextServer adalah komponen yang mempunyai fungsi sebagai *server* khusus pengiriman data berupa baris-baris teks. Sama sepert TTextClient, pada TTextServer ini juga terdapat metode OnLineReceived yang akan dipanggil pada saat *socket* menerima baris teks baru serta metode WriteLine yang berguna untuk menulis baris teks ke *buffer* serta menambahkan karakter CR dan LF yang sesuai pada tiap akhir baris.

TTimeClient adalah komponen sederhana yang mengaplikasikan pengambilan data waktu dari *time server* yang berada pada jaringan. Untuk pengambilan waktu dari *server* ini digunakan *port* 37 dan menggunakan koneksi *oriented socket* (TCP). Komponen ini adalah turunan dari TCustomClientSocket dan menyediakan *event-event* dan metode-metode untuk mengambil data waktu dari *server*. Komponen ini tidak mempublikasikan *event-event* umum yang dimiliki oleh TbasicClientSocket.

TTimeServer.adalah pasangan dari komponen TTimeClient yang berfungsi sebagai *server* yang menyediakan data waktu dan melayani permintaan dari TTimeClient. Sama seperti TTimeClient, komponen ini juga merupakan turunan dari dari TcustomBasicServerSocket dan menyediakan *event-event* dan metode-metode untuk mengirim data waktu dari *server*. Juga pada TTimeServer ini tidak mempublikasikan *event-event* umum yang dimiliki oleh TBasicServerSocket.

3. PERANCANGAN DAN IMPLEMENTASI

Pada aplikasi ini digunakan komponen dWinsock untuk menghubungkan aplikasi pengendalian yang dibuat dengan lapisan transport yang berada di bawahnya. Aplikasi ini terdiri dari dua bagian yaitu *server* dan *client* dimana *server* ini akan dijalankan

pada sebuah komputer yang akan dikendalikan dan *client* akan berada pada komputer pengendali. Di sini istilah *server* ditujukan kepada komputer yang akan dikendalikan, bukan pada komputer pengendali mengingat bahwa sebenarnya komputer yang dikendalikan inilah yang memberikan pelayanan yang diminta oleh *client*. Sedangkan istilah *client* digunakan untuk komputer pengendali karena secara prinsip komputer pengendali ini hanya meminta layanan (*service request*) dari komputer yang dikendalikan.

Aplikasi *server* dapat ditempatkan pada lebih dari satu komputer yang masing-masing mempunyai alamat IP berbeda. Sedangkan aplikasi *client* walaupun juga bisa ditempatkan pada lebih dari satu komputer, tapi akan lebih efektif jika ditempatkan pada satu komputer yang berfungsi sebagai pusat pengendali. Sehingga pada suatu jaringan yang terdiri dari banyak komputer, administrator dapat mengendalikan semua komputer yang berada pada jaringan tersebut hanya dengan menggunakan satu buah komputer.

Pada tahap perancangan, langkah pertama yang dilakukan adalah merencanakan fungsi-fungsi apa saja yang akan dibuat pada aplikasi ini. Aplikasi ini dibuat untuk menjalankan fungsi : pengiriman pesan (*send message*), menjalankan program (*execute program*), mengunci komputer *server* (*lock computer*), membuka komputer *server* yang terkunci (*unlock computer*), mematikan komputer *server* (*shutdown*), me-reboot komputer *server* (*reboot*), menangkap gambar layar (*capture screen*), melihat program (*view program).* Setelah menentukan jenis-jenis fungsi yang akan diimplementasikan pada aplikasi ini, langkah berikutnya adalah menentukan protocol untuk komunikasi data antara aplikasi *server* dan *client*. Pada aplikasi ini, data yang dikirimkan berbentuk data biner dan teks, sehingga digunakan komponen TbinaryServer, TBinaryClient, TTextServer dan TTextClient.

Jenis protokol *transport* yang digunakan adalah TCP, yang dilakukan dengan mengisi *property* Protocol pada masing-masing komponen, sedangkan nomor *port* yang digunakan adalah 264048 yang dilakukan dengan mengisi *property* Port pada masing-masing komponen.

- Perancangan Aplikasi *Server*
  Pada bagian ini akan dibahas semua fungsi-fungsi yang terdapat pada aplikasi *server*. Tugas utama dari *server* ini adalah menerima perintah yang dikirimkan oleh *client* kemudian melaksanakan perintah tersebut dan mengirimkan hasilnya ke *client* jika diperlukan. Aplikasi *server* ini harus berjalan terus-menerus dan idealnya mulai dari saat komputer pertama kali dinyalakan hingga dimatikan kembali. Dan juga aplikasi yang sedang berjalan ini tidak boleh mengganggu pengguna, atau jika dimungkinkan pengguna komputer tersebut tidak mengetahui mengenai aplikasi yang sedang berjalan sehingga tidak dapat dimatikan oleh pengguna.
  Agar aplikasi dapat secara otomatis berjalan pada saat komputer dinyalakan, dilakukan dengan menambahkan baris perintah pada Registry pada bagian HKEY_LOCAL_MACHINE \ Software \ Microsoft\Windows\CurrentVersion\Run yang diisi dengan nama aplikasi ini (StateMgr.Exe). Secara program, dapat dilakukan dengan menggunakan perintah berikut:

```
Reg := TRegistry.Create;
  Reg.RootKey := HKEY_LOCAL_
MACHINE;
  Reg.OpenKey('Software\Microsoft\
Windows\CurrentVersion\Run', True);
if (not Reg.ValueExists('StateMgr')) or
  (Reg.ReadString('StateMgr') <>
'C:\WINDOWS\SYSTEM\RPC\State
Mgr.exe') then
  Reg.WriteString('StateMgr',
'C:\WINDOWS\SYSTEM\RPC\State
Mgr.exe');
  Reg.CloseKey;
```

  Kemudian pada aplikasi *server* ini juga harus terus menerus mendeteksi adanya koneksi dari *client*. Hal ini dilakukan dengan menggunakan sebuah komponen TTimer dimana komponen ini akan menjalankan fungsi di bawah ini dengan interval 1 detik:

```
if State then
  TextClient.Open(TLineSocket)
```

```
    else
      TextClient.Close;
    State := not State;
```

Pada saat aplikasi *server* sudah menerima koneksi dari *client*, langkah selanjutnya yang harus dilakukan oleh *server* adalah mendeteksi perintah yang dikirimkan oleh *client*. Hal ini dapat dilakukan dengan menuliskan program pada event OnLineReceived pada TTextClient sbb.:

```
procedure TMainForm.TextClient-
LineReceived(Socket:
TLineSocketBase;
  const Line: string; Complete:
Boolean);
begin
  if Line = '[SHUTDOWN]' then
    begin
      CheckReg;
      TextClient.Close;
ExitWindowsEx(EWX_SHUTDOWN
or EWX_FORCE or
EWX_POWEROFF, 0);
    End;
end;
```

Setelah membuat program untuk mendeteksi adanya pengiriman perintah dari *client*, berikutnya adalah membuat program untuk menjalankan masing-masing perintah tersebut dan mengirimkan hasilnya ke *client* kembali. Berikut ini adalah cuplikan beberapa program yang menjalankan perintah dari *client*.

Menjalankan perintah mematikan komputer (shutdown):

```
if Line = '[SHUTDOWN]' then
    begin
      CheckReg;
      TextClient.Close;
ExitWindowsEx(EWX_SHUTDOWN
or EWX_FORCE or
EWX_POWEROFF, 0);
    End
```

Menjalankan perintah me-reboot komputer:

```
  if Line = '[REBOOT]' then
    begin
      CheckReg;
      TextClient.Close;
      ExitWindowsEx(EWX_REBOOT
or EWX_FORCE, 0);
    End
```

Menerima pesan dari client :

```
  if Copy(Line, 1, 9) = '[MESSAGE:'
then
   begin
    case Line[10] of
      '0' : MsgIcon := IDI_WINLOGO;
      '1' : MsgIcon :=
IDI_APPLICATION;
      '2' : MsgIcon := IDI_ASTERISK;
      '3' : MsgIcon :=
IDI_EXCLAMATION;
      '4' : MsgIcon := IDI_HAND;
      '5' : MsgIcon :=
IDI_QUESTION;
    end;
    S := Copy(Line, 13,
Length(Line));
    for i := 1 to Length(S) do
      if S[i] = #9 then S[i] := #13;
//Initialize for message display
    Top   := 0;
    Left  := 0;
    Color := clBlack;
    Width := ScrWidth;
    BorderStyle := bsNone;
    CloseState  := False;
    Label1.Font.Color := clWhite;
SetMessage(S);
    Show;
    Repaint;
  End
```

Menjalankan perintah untuk eksekusi sebuah program :

```
   if Copy(Line, 1, 9) = '[EXECUTE]'
  then
     begin
       S := Copy(Line, 11,
  Length(Line));
       C := Copy(S, Pos(#9, S) + 1,
  Length(S));
       S := Copy(S, 1, Pos(#9, S) - 1);
       D := ExtractFileDir(S);
       S := S + ' ' + C;
       try
         SetCurrentDir(D);
         WinExec(@S[1],
  SW_SHOWNORMAL);
       except
         Application.MessageBox('Can''t
  run command.', 'Remote PC',
  MB_OK or MB_ICONWARNING);
       end;
     end
```

Mengunci komputer sehingga tidak biasa digunakan oleh pengguna dan membuka komputer yang telah terkunci:

```
if Line = '[LOCK]' then
  begin
    if LockAvail and (not Locking) then
    begin
      Locking  := True;
      LockForm := TLockForm.Create(Self);
      LockForm.CloseState := False;
      LockForm.Show;
      CtrlAltDel(True);
      xBlockInput(True);

    TLineSocket(TextClient.Conn).WriteLine('[LOCKING]');
    end
    else

    TLineSocket(TextClient.Conn).WriteLine('[LOCKERR]');
  end
 else
 if Line = '[UNLOCK]' then
  begin
    if Locking then
    begin
      LockForm.CloseState := True;
      LockForm.Close;
      LockForm.Free;
      Locking := False;
      CtrlAltDel(False);
      xBlockInput(False);

    TLineSocket(TextClient.Conn).WriteLine('[UNLOCKING]');
    end
    else

    TLineSocket(TextClient.Conn).WriteLine('[UNLOCKERR]');
  end
```

- Perancangan aplikasi *client*

  Pada aplikasi *client*, hal yang terutama adalah membuat deteksi koneksi dari masing-masing *server*, dimana jumlah *server* yang terhubung ke *client* dapat lebih dari satu, maka harus dibuat sebuah sistem yang dapat mendeteksi semua *client* tersebut. Untuk menampilkan semua *client* yang sedang terhubung saat tersebut, digunakan komponen TlistView, dimana isi TListView ini akan selalu disesuaikan dengan adanya koneksi baru ataupun pemutusan koneksi.

  Koneksi baru dapat dideteksi dengan menempatkan program berikut pada *event* OnAccept komponen TTextServer:

```
procedure TMainForm.TextServer1-
Accept(Sender: TObject; Socket:
TSocketBase);
var
  C : string[1];
begin
  with Socket do
  begin

  AddComputer(UpperCase(RemoteHost), RemoteAddress, 'Connected', SocketID);
    if RemoteHost = '' then

  TLineSocket(Socket).WriteLine('[NAME]');
    if ListView1.Items.Count > 1 then C := 's' else C := '';
    IconTrayHint := Format(Application.Title + ' (%d Client%s Connected)', [ListView1.Items.Count, C]);
    Move(IconTrayHint[1], NID.szTip[0], Length(IconTrayHint) + 1);
    Shell_NotifyIcon(NIM_MODIFY, @NID);
  end;
end;
```

Sedangkan untuk mendeteksi pemutusan koneksi, dilakukan dengan menempatkan program berikut pada *event* OnDisconnect:

```
procedure TMainForm.TextServer1-
Disconnect(Sender: TObject;
  Socket: TSocketBase);
var
  C : string[1];
begin
  RemoveComputer(Socket.SocketID);
  if ListView1.Items.Count = 0 then
    IconTrayHint := Application.Title + ' (No Client Connected)'
  else
  begin
    if ListView1.Items.Count > 1 then C := 's' else C := '';
```

```
    IconTrayHint :=
Format(Application.Title + ' (%d
Client%s Connected)',
[ListView1.Items.Count, C]);
  end;
  Move(IconTrayHint[1],
NID.szTip[0], Length(IconTrayHint)
+ 1);
  Shell_NotifyIcon(NIM_MODIFY,
@NID);
        end;
```

Untuk pengiriman perintah ke *server*, dapat dilakukan secara sederhana dengan mengirimkan identitas perintah yang akan dijalankan oleh *server*. Identitas ini ialah: MESSAGE untuk pengiriman pesan, EXECUTE untuk menjalankan program pada *server*, LOCK untuk mengunci komputer *server* sehingga tidak dapat digunakan, UNLOCK untuk membuka kunci, SHUTDOWN untuk mematikan komputer *server*, REBOOT untuk me-reboot computer, CAPTUREINIT untuk mengambil data layar pada komputer *server*, PROCESSLIST untuk mengetahui program yang sedang dijalankan pada *server*. Identitas perintah ini kemudian dikirimkan dengan menggunakan prosedur:

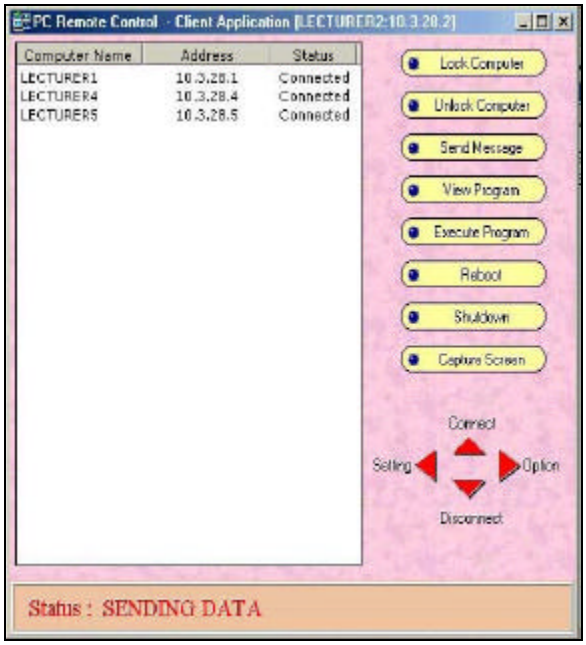
```
procedure TMainForm.SendMsg
(Msg : string);
var
  i   : Integer;
  Pos : Integer;
begin
  with ListView1 do
    for i := 0 to
ListView1.Items.Count - 1 do
      if Items[i].Selected or
Items[i].Focused then
      begin
        Pos :=
FindSocket(Integer(Items[i].Data^));
      if Pos <> -1 then

TlineSocket (TextServer1.Client
[Pos]).WriteLine(Msg);
      end;
end;
```

Pada sisi *server* akan menerima identitas perintah ini dan kemudian menjalankan sesuai dengan yang diminta, seperti yang telah dibahas di bagian perancangan aplikasi *server*.

4. PENGUJIAN

Pengujian aplikasi ini dilakukan dengan menggunakan 4 buah komputer dimana sebuah komputer berfungsi sebagai *client* dan 3 buah lainnya berfungsi sebagai *server*. Komputer ini terletak pada jaringan internal dan mempunyai alamat IP: 10.3.28.1, 10.3.28.2, 10.3.28.4 dan 10.3.28.5. Pengujian dilakukan dengan mencoba semua fungsi yang telah diaplikasikan ke semua *server*.

Berikut ini adalah gambaran tampilan pada *client*:



Gambar 1. Tampilan Aplikasi Client

Hasil pengujian menunjukkan bahwa semua fungsi dapat dijalankan dengan sempurna pada semua *server* tanpa mengalami penundaan yang berarti dimana pada saat *client* mengirimkan perintah, maka *server* akan langsung menjalankan perintah tersebut (kurang dari 1 detik). Hanya saja untuk fungsi yang mengirimkan data kembali ke *client* seperti capture screen, explore terdapat penundaan yang lebih besar dikarenakan terdapat data dalam jumlah cukup besar yang harus dikirimkan kembali ke *client*.

Pengujian berikutnya adalah mencoba untuk mengendalikan komputer *client* yang berada pada jaringan lain (internet). Dari

pengujian didapatkan bahwa semua fungsi juga dapat dijalankan dengan sempurna hanya mengalami penundaan (*delay*) antara saat pengiriman perintah dari *client* sampai *server* menjalankan perintah tsb dibandingkan dengan pengujian yang dilakukan pada jaringan internal. Besar *delay* ini bervariasi mulai kurang dari satu detik hingga lima detik. Diperkirakan besar *delay* yang berbeda ini disebabkan karena kepadatan *traffic data* pada saat pengujian dilakukan. Dimana pada kepadatan tinggi akan menyebabkan *delay* yang semakin besar.

Pengujian juga dilakukan pada komputer yang mula-mula terhubung pada jaringan, kemudian dengan sengaja diputus koneksinya. Hasil dari pengujian ini adalah pada aplikasi *client* mampu mendeteksi komputer yang terputus tersebut setelah beberapa saat dan kemudian menghapus identitas komputer tersebut dari daftar komputer yang terhubung. Tetapi saat koneksi tersebut dikembalikan lagi secara normal, aplikasi *client* tidak dapat mendeteksi komputer yang sudah terhubung kembali tersebut. Agar dapat dikenali kembali, maka program *server* pada komputer yang bersangkutan dimatikan dan diaktifkan kembali.

5. KESIMPULAN

Dengan menggunakan komponen dWinsock ternyata pembuatan aplikasi jaringan komputer dengan menggunakan bahasa pemrograman Delphi menjadi lebih sederhana dan mudah disbanding dengan menggunakan Windows Socket API secara langsung.

Dari pengujian yang telah dilakukan terlihat bahwa aplikasi pengendalian komputer yang telah dibuat dapat memenuhi harapan semula dengan dapat dijalankannya semua fungsi tanpa ada kendala pada jaringan internal, dan hanya mengalami penundaan (*delay*) jika diaplikasikan pada jaringan internet. Besar *delay* ini bervariasi dalam hitungan satu hingga lima detik tergantung dari kepadatan *traffic* data yang sedang terjadi saat itu.

Kekurangan dari aplikasi ini ialah tidak adanya kemampuan untuk mendeteksi komputer / aplikasi *server* yang mula-mula terhubung kemudian tiba-tiba terputus dan terhubung kembali. Untuk mengenali kembali aplikasi *server* yang telah terhubung dilakukan dengan mematikan terlebih dahulu aplikasi *server* tersebut kemudian diaktifkan kembali.

DAFTAR PUSTAKA

1. Lane, Malcolm G. *Data Communication Software Design*. Boston : Boyd & Fraser Publishing Company, 1985.

2. Stallings, William. *Data and Computer Communication 5 th Ed*. New Jersey : Prentice Hall, Inc, 1987.

3. Söderberg, Ulf; Palmer, Marc and Hawes, Keith. *Help for dWinsock Components for Delphi and C++ Builder Version 2.51,* http://www.aait.com/dwinsock/, 1997.

4. Hunt, Craig. *TCP/IP Network Administration*, Sebastopol: O'Reilly & Associates, Inc, 1993.

5. Watson, Blake. *Delphi by Example.* Indianapolis : Que Corporation, 1995.