Modul 9
Protokol Transport (TCP/UDP)

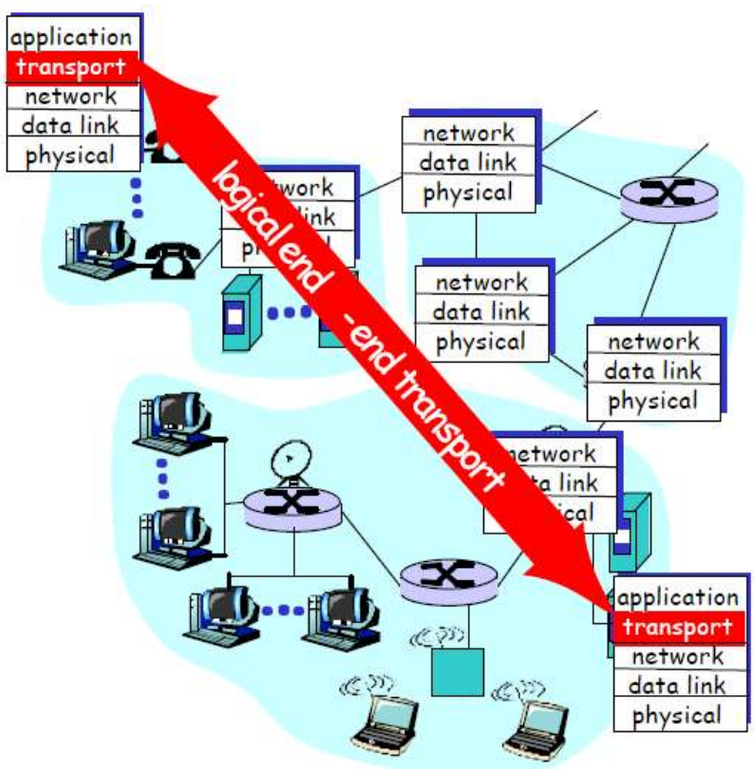
Tipe Transfer Data

komunikasi logika pada lapisan Transport dapat berbentuk :

- **connectionless** atau **connection-oriented**.
- **Reliable** atau **unreliable** : Reliable berarti data ditransfer ke tujuannya dalam suatu urutan seperti ketika dikirim. Pengiriman data Unreliable sangat menggantungkan diri pada lapisan jaringan di bawahnya, sehingga tidakdapat menyakinkan apaah segment data dapat dikirimkan sampai ditujuannya atau tidak.
- **Stateful** atau **stateless**. Pengiriman data stateful berarti informasi yang dimasukkan pada satu *request,* yang dikirimkan dari pengirim ke penerima, dapat dimodifikasi untuk *request* berikutnya. Sedangkan pengiriman stateless berarti informasi dalam satu request tidak dapat dikaitkan dengan *request* lainnya, sehingga tidak dapat digunakan untuk *request* lainnya.

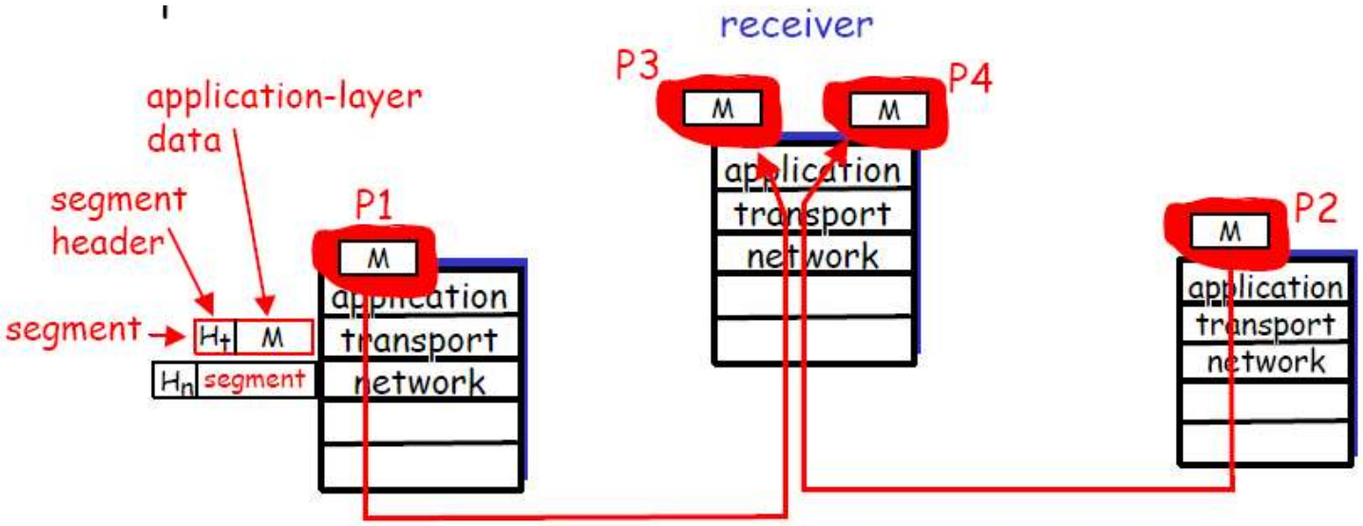
Fungsi

- menyediakan komunikasi logika antar proses aplikasi yang berjalan pada host yang berbeda
- protokol transport berjalan pada **end systems**
- Perbedaan dengan Lapisan Network :
  - lapisan *network* : transfer data antar end-system
  - lapisan *transport* : transfer data antar proses

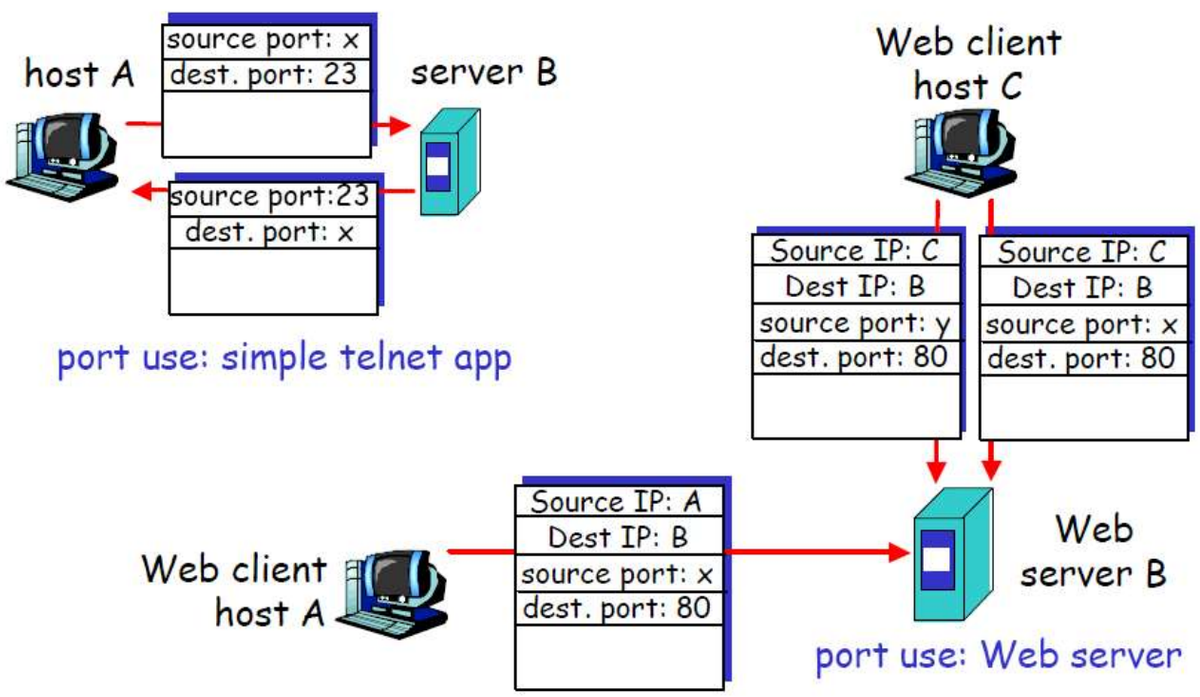


- Layanan transport pada Internet :
  - **reliable**, pengiriman dalam suatu urutan dengan model *unicast*. Contoh : TCP
  - **unreliable**, pengiriman tidak dalam suatu urutan dengan model *unicast* atau *multicast*. Contoh : UDP
- Pada layanan transport, satuan data yang dipertukarkan disebut sebagai ***segment*** *(TPDU = Transprot Protocol Data Unit)*

- Layanan transport menyediakan **demultiplexing** untuk dapat mengirimkan segment ke proses lapisan aplikasi yang sesuai berdasar alamat dan port proses tersebut.



- Selain itu juga melakukan multiplexing, yang akan mengambil data dari beberapa proses aplikasi, dan membungkus data dengan header.
- Multiplexing dan Demultiplexing di dasarkan pada pengirim dan nomor port serta alamat IP penerima.
  - Terdapat nomor port sumber dan tujuan pada tiap segment.



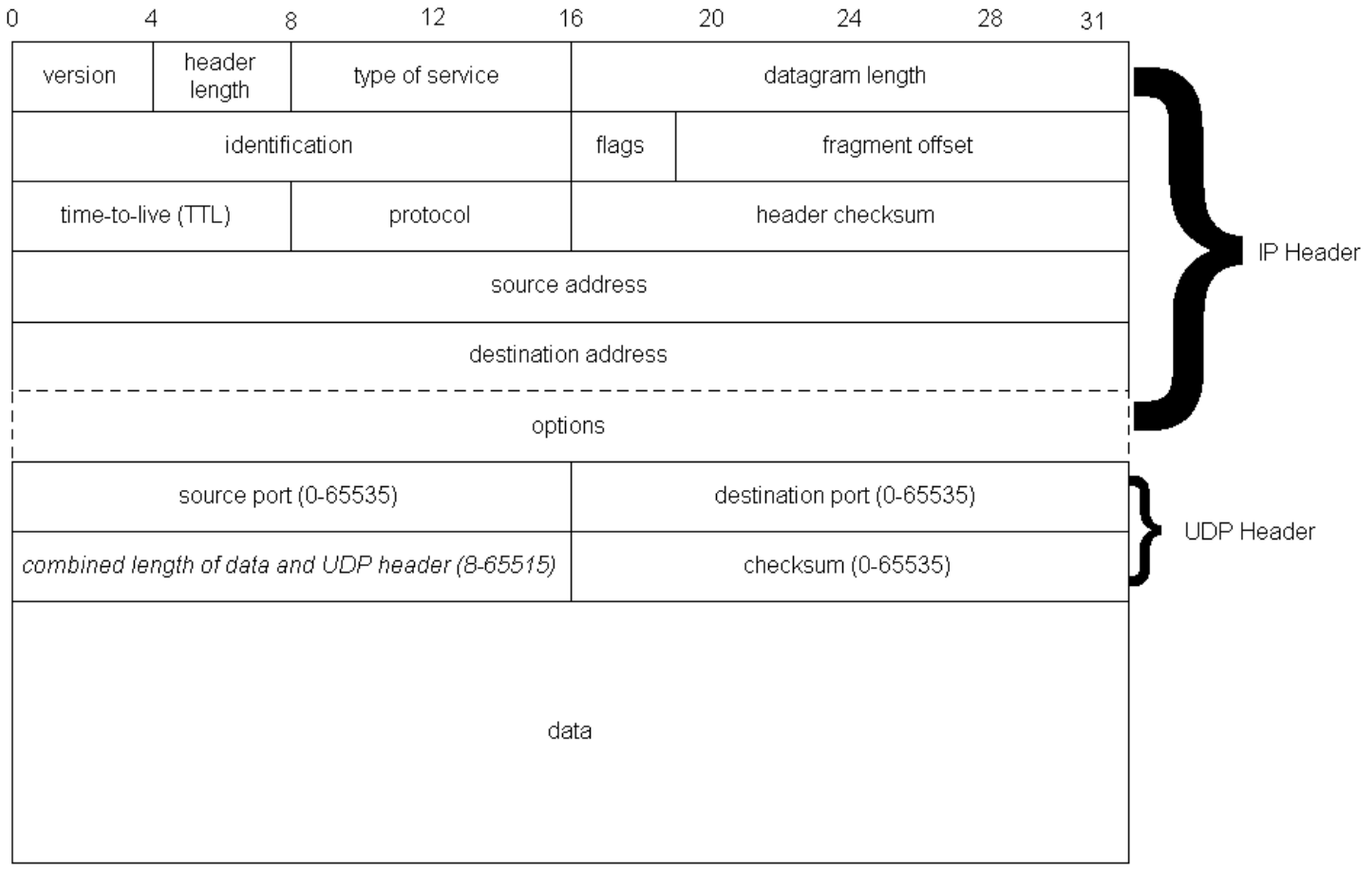
Untuk alamat proses, yaitu port, berdasar standarisasi dari IANA, dapat dikelompokkan menjadi 3, yaitu

- *Well-known* port, yaitu 0 - 1023
- *Registered* port, yaitu 1024 - 49151
- *Dynamic/Private* port, 49152 - 65535

Silahkan Anda mencari contoh untuk masing-masing kategori tersebut!

UDP (User Datagram Protocol)

- **Mengapa ada UDP?**
  - ✔ Tidak perlu adanya setup koneksi terlebih dahulu (hal ini dapat menyebabkan tambahan delay)
  - ✔ sederhana, artinya antara penerima dan pengirim tidak perlu menjaga session atau status koneksi
  - ✔ ukuran header segment sederhana
  - ✔ tidak perlu kontrol kemacetan koneksi, artinya UDP dapat mengirimkan per segment tanpa dipengaruhi oleh kesibukan jaringan



- Didefinisikan dengan RFC 768
- Menerapkan layanan connectionless :
  - tidak ada handshaking antara pengirim UDP dan penerimanya
  - setiap segment UDP ditangani secara independen dengan segment UDP lainnya
- Kelemahan :
  - segment UDP mungkin akan hilang
  - paket diterima mungkin dalam keadaan yang tidak urut. Jika paket yang diterima tidak urut, paket akan dibuang.
- tidak ada kontrol kemacetan koneksi (*congestion control*), artinya UDP dapat mengirimkan per segment tanpa dipengaruhi oleh kesibukan jaringan. Secara prinsip lapisan transport harus memberikan *congestion*

control ini selama transmisi terjadi. Congestion dapat terjadi karena terlalu banyak paket data pada jaringan sehingga membuat unjuk kerja jaringan menjadi menurun. Hal ini dapat disebabkan, misalnya karena adanya router terlalu penuh buffernya sehingga memperlambat.

Checksum UDP

- Untuk membantu pengecekan kondisi paket segment UDP yang diterima, pada header UDP terdapat field checksum (16 bit). Checksum dihitung dari Pseudo-header yang di dalamnya terdapat alamat IP sumber dan tujuan, plus field protocol dari header IP. (untuk calculator checksum Ipv4 silahkan berkunjung ke http://byerley.cs.waikato.ac.nz/~tonym/hec.html)
- Silahkan kunjungi : http://www.netfor2.com/udpsum.htm
- Berikut adalah skema untuk pseudo-header :

offset	+0	+1	+2	+3
+0	src ip address			
+4	dst ip address			
+8	0:zero	protocol	UDP data length	

- Pseudocode untuk penghitungan checksum UDP dari Pseudo-header :

```
typedef usingend short u16;
typedef unsigned long u32;

u16 short udp_sum_calc(
              u16 len_udp,
              u16 src_addr[],u16 dest_addr[],
              BOOL padding, u16 buff[])
{
  u16 prot_udp=17;
  u16 padd=0;
  u16 word16;
  u32 sum;

  // Find out if the length of data is even or odd number.
  // If odd,
  // add a padding byte = 0 at the end of packet
  if (padding&1==1){
    padd=1;
    buff[len_udp]=0;
  }

  //initialize sum to zero
  sum=0;
```

```
  // make 16 bit words out of every two adjacent 8 bit words
  // and calculate the sum of all 16 vit words
  for (i=0;i<len_udp+padd;i=i+2){
    word16 =((buff[i]<<8)&0xFF00)+(buff[i+1]&0xFF);
    sum = sum + (unsigned long)word16;
  }

  // add the UDP pseudo header which contains
  //the IP source and destinationn addresses
  for (i=0;i<4;i=i+2){
    word16 =((src_addr[i]<<8)&0xFF00)+
            (src_addr[i+1]&0xFF);
    sum=sum+word16;
  }
  for (i=0;i<4;i=i+2){
    word16 =((dest_addr[i]<<8)&0xFF00)+
             (dest_addr[i+1]&0xFF);
    sum=sum+word16;
  }

  // the protocol number and the length of the UDP packet
  sum = sum + prot_udp + len_udp;

  // keep only the last 16 bits of the 32 bit calculated
  //sum and add the carries
  while (sum>>16)
    sum = (sum & 0xFFFF)+(sum >> 16);

  // Take the one's complement of sum
  sum = ~sum;

  return ((u16) sum);
}
```

Parameter untuk fungsi di atas :

u16 buff[]	array header dan data UDP
u16 len_udp	panjang dari header+data UDP
BOOL	1 jika memiliki bilangan okta genap dan 0 jika ganjil
u16 src_addr[4]	
u16 dest_addr[4]	alamat IP sumber dan tujuan

- Sebagai contoh :
  - A mengirim segment UDP dengan jumlah 16 bit pseudo-header adalah 1100101011001010.

Maka checksumnya adalah
0011010100110101

- B menerima data dari A, menghitung jumlah 16-bit dan menambahkannya dengan nilai field checksum. Jika jumlahnya adalah 1111111111111111, maka tidak ada error.

Aplikasi UDP

- Digunakan untuk multimedia streaming, yang sangat memberikan toleransi kehilangan segment cukup baik dan yang sangat tidak sensitif terhadap kerusakan atau kehilangan segment
- Contoh protokol aplikasi yang menggunakan UDP :
  - DNS (Domain Name System) 53
  - SNMP, (Simple Network Management Protocol) 161, 162
  - TFTP (Trivial File Transfer Protocol) 69
  - SunRPC port 111.
  - dsb. (tolong sebutkan contoh lain! Wajib untuk diketahui!)
- Untuk membuat pengiriman yang reliable dengan protokol UDP, maka pada lapisan protokol aplikasinya harus menyediakan penanganan kesalahan tersendiri.

Berikut adalah contoh aplikasi client/server dengan menggunakan protokol UDP :

UDPClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class UDPClient {
  public static void main (String args[ ]) {
    try{
      // buat socket UDP untuk port 2000
      DatagramSocket socket = new DatagramSocket(2000);

      // buat paket UDP yang berisi buffer 256 byte
      DatagramPacket packet =
              new DatagramPacket( new byte[256], 256 );

      // terima paket - ini adalah operasi terblok
      socket.receive(packet);

      // tampilkan informasi paket
      InetAddress remote_addr = packet.getAddress();
```

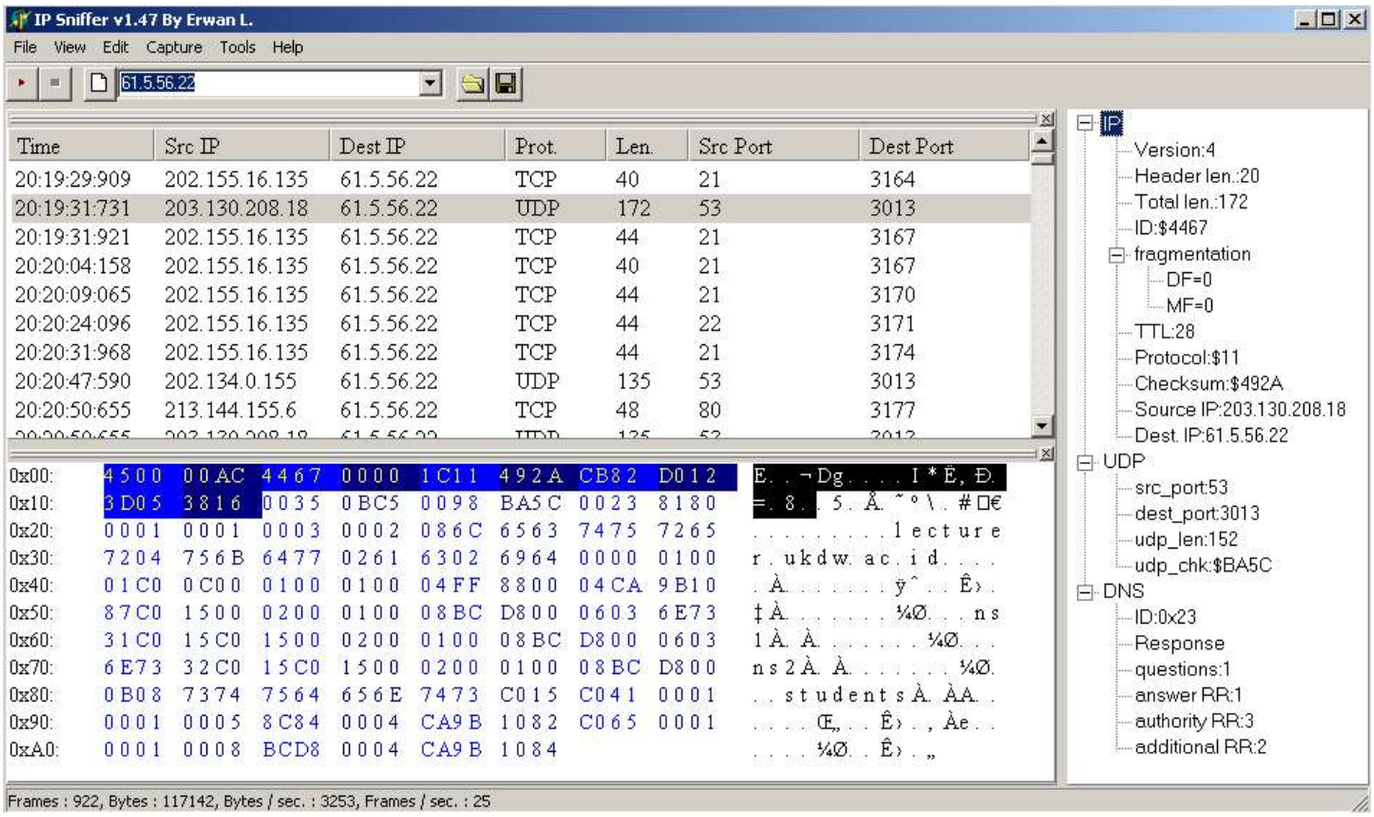
```
      System.out.println ("Pengirim : " +
                          remote_addr.getHostAddress( ) );
      System.out.println ("dari Port: " + packet.getPort());

      // tampilkan isi paket
      ByteArrayInputStream bin =
           new ByteArrayInputStream(packet.getData());

      for (int i=0; i < packet.getLength(); i++) {
        int data = bin.read();
        if (data == -1)
          break;
        else
          System.out.print ( (char) data) ;
      }
      socket.close( );
    }
    catch (IOException e) {
      System.out.println ("Error - " + e);
    }
  } //akhir dari main()
} //akhir dari class
```

UDPServer.java

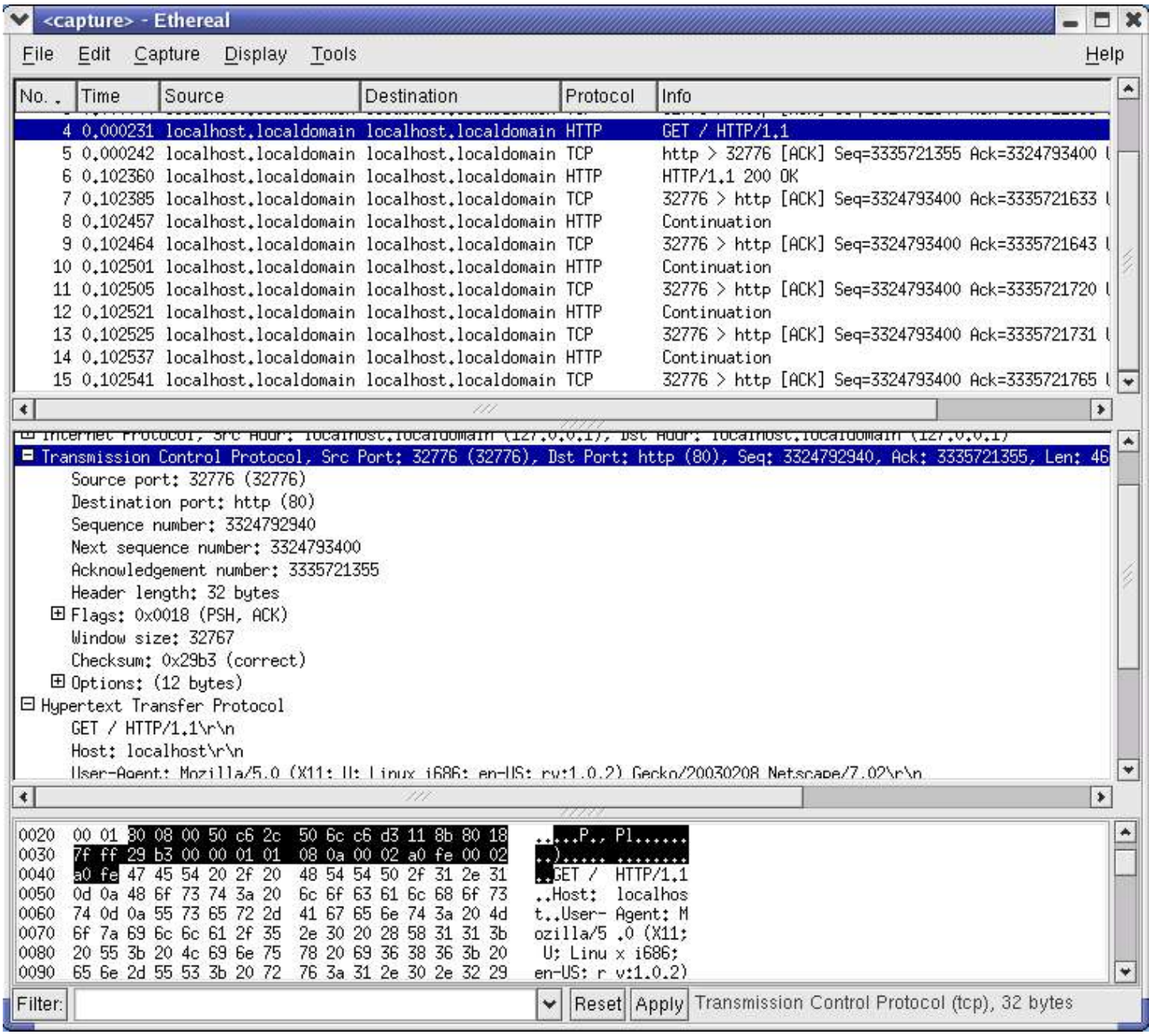
```
import java.net.*;
import java.io.*;
import java.util.*;

public class UDPServer {
  public static void main (String args[ ]) {
    String hostname="localhost";
    String message = "Hallo UDP!";
    try {
      // buat socket UDP, dan cari port yang tersedia
      DatagramSocket socket = new DatagramSocket();
      System.out.println ("Local port: " +
                          socket.getLocalPort());
      ByteArrayOutputStream bOut = new ByteArrayOutputStream();

      //konversi printstream ke array byte
      byte [ ] bArray = bOut.toByteArray();

      // buat paket datagram dengan ukuran buffer 256 byte
      DatagramPacket packet =
           new DatagramPacket( bArray, bArray.length );

      //buat objek dari class InetAddress
      InetAddress remote_addr = InetAddress.getByName(hostname);
```

```
      //cek alamat IP dari hostname
      System.out.println("Hostname has IP address = " +
                         remote_addr.getHostAddress());

      //konfigurasi DataGramPacket
      packet.setAddress(remote_addr);
      packet.setPort(2000);

      //kirim paket UDP
      socket.send(packet);

      // tampilkan informasi paket
      System.out.println ("Dikirim oleh : " +
                          remote_addr.getHostAddress() );
      System.out.println ("Dikirim dari : " + packet.getPort());
    }
    catch (UnknownHostException ue){
      System.out.println("Unknown host "+hostname);
    }
    catch (IOException e) {
      System.out.println ("Error - " + e);
    }
  }//akhir dari main
}//akhir dari class
```

Anda dapat menggunakan berbagai macam program IP Sniffer atau sejenisnya untuk menangkap lalu lintas paket data yang melalui interface jaringan Anda, seperti yang saya gunakan di bawah ini, yaitu menggunakan program IP Sniffer versi 1.47 yang dibuat oleh Erwan L (erwal.l@free.fr).

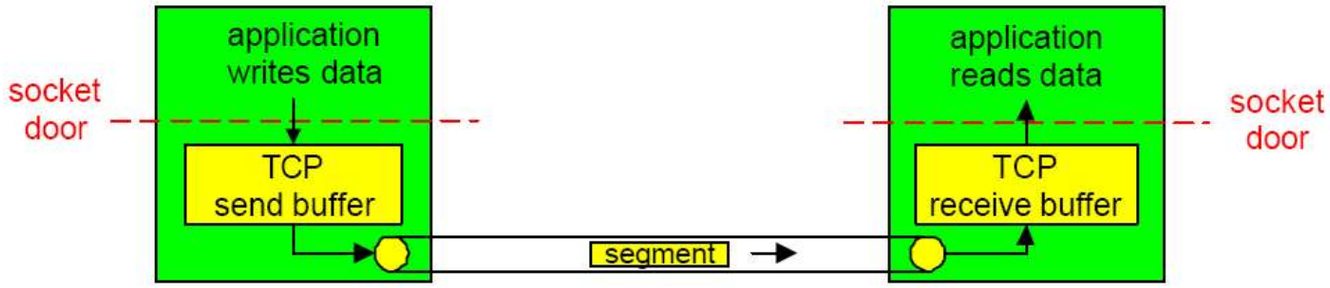


Contoh aplikasi lain yang sudah lama menjadi tool baku di lingkungan Unix/Linux adalah TCPdump yang kemudian digunakan pada program Ethereal yang merupakan User Interface dari TCPdump tersebut. Berikut contoh snapshoot nya :

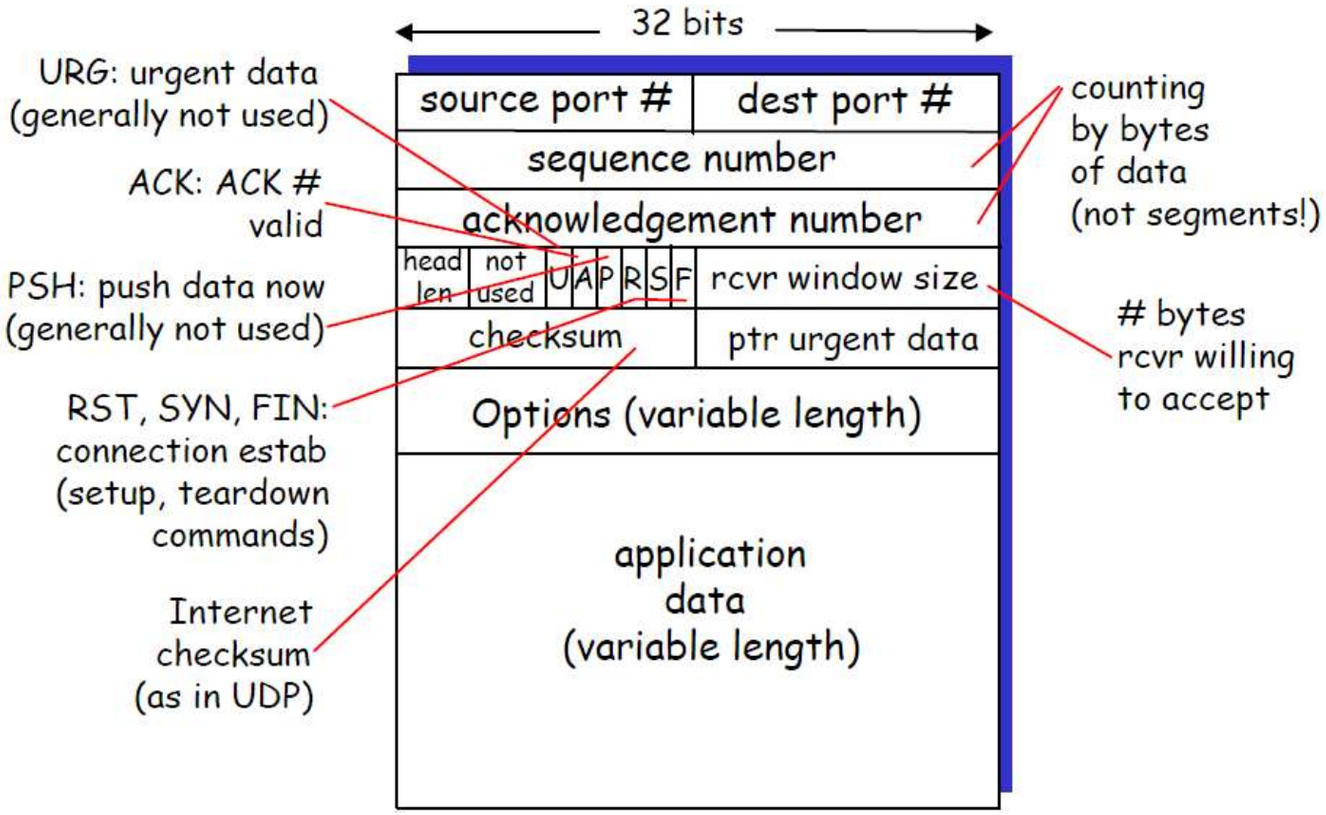


TCP (Transmission Control Protocol)

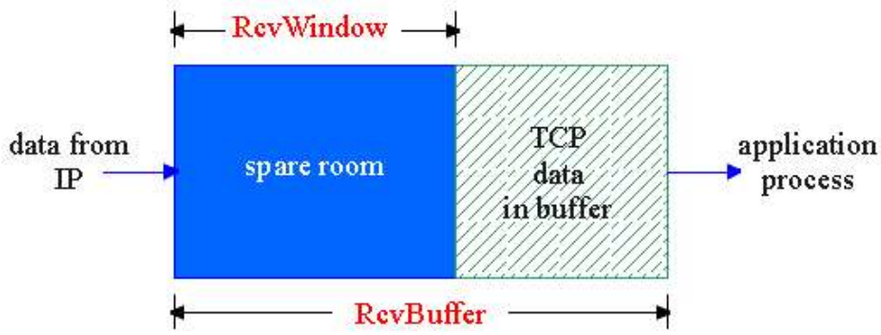
- Didefinisikan dengan RFC: 793, 1122, 1323, 2018, 2581
- Berikut beberapa karakteristik TCP :
  - point-to-point
  - reliable dan stateful
  - pipeline, menerapkan flow control window sliding untuk mengontrol kemacetan dan aliran datanya
  - terdapat buffer pengirim dan penerima
  - full-duplex
  - connection-oriented
  - aliran data yang dikontrol



Struktur paket segment TCP lebih komplek daripada UDP :

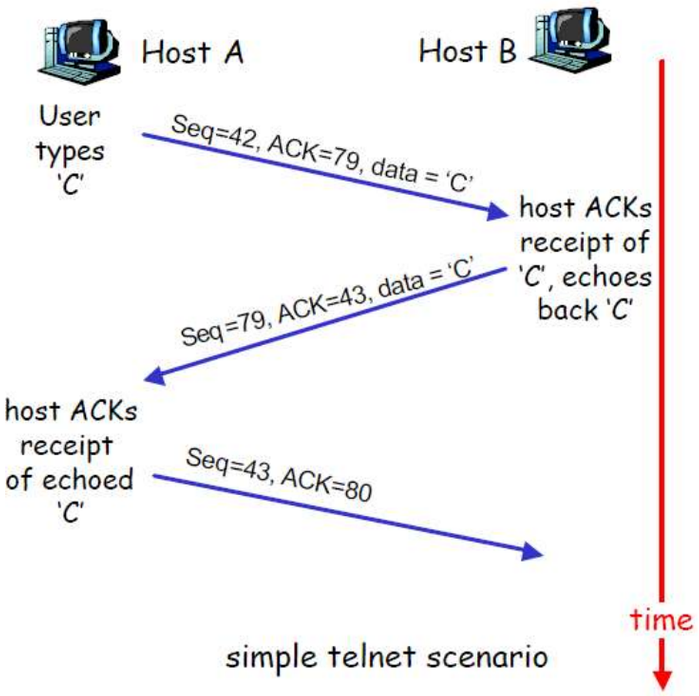


Pada header TCP tersebut terlihat bahwa TCP menerapkan flow control sliding window, yaitu pada *rcvr windows size*, yang berisi jumlah buffer (byte) penerima. Berdasar informasi field tersebut, pengirim akan selalu mempertahankan untuk tidak mengirim data dengan ukuran yang melebihi ukuran yang tersebut pada field *rcvr windows size* tersebut.



Oleh karena TCP menerapkan model koneksi connection oriented, maka ketika setup data sudah terbentuk, antara pengirim dan penerima dapat saling mempertukarkan data secara full duplex dengan tetap menjaga session koneksi diantara penerima dan pengirim (*stateful*). Untuk dapat melaksanakan mekanisme tersebut, di samping ini adalah contoh,

bagaimana mekanisme *three-way handshake* terjadi pada TCP dengan menggunakan field *sequence number* dan *acknowledge number* :



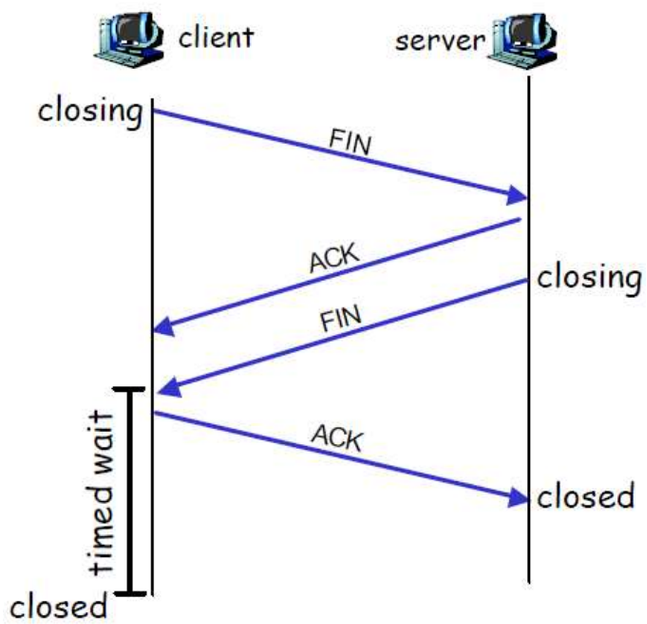
Manajemen Koneksi TCP

Pada saat **Setup Koneksi**

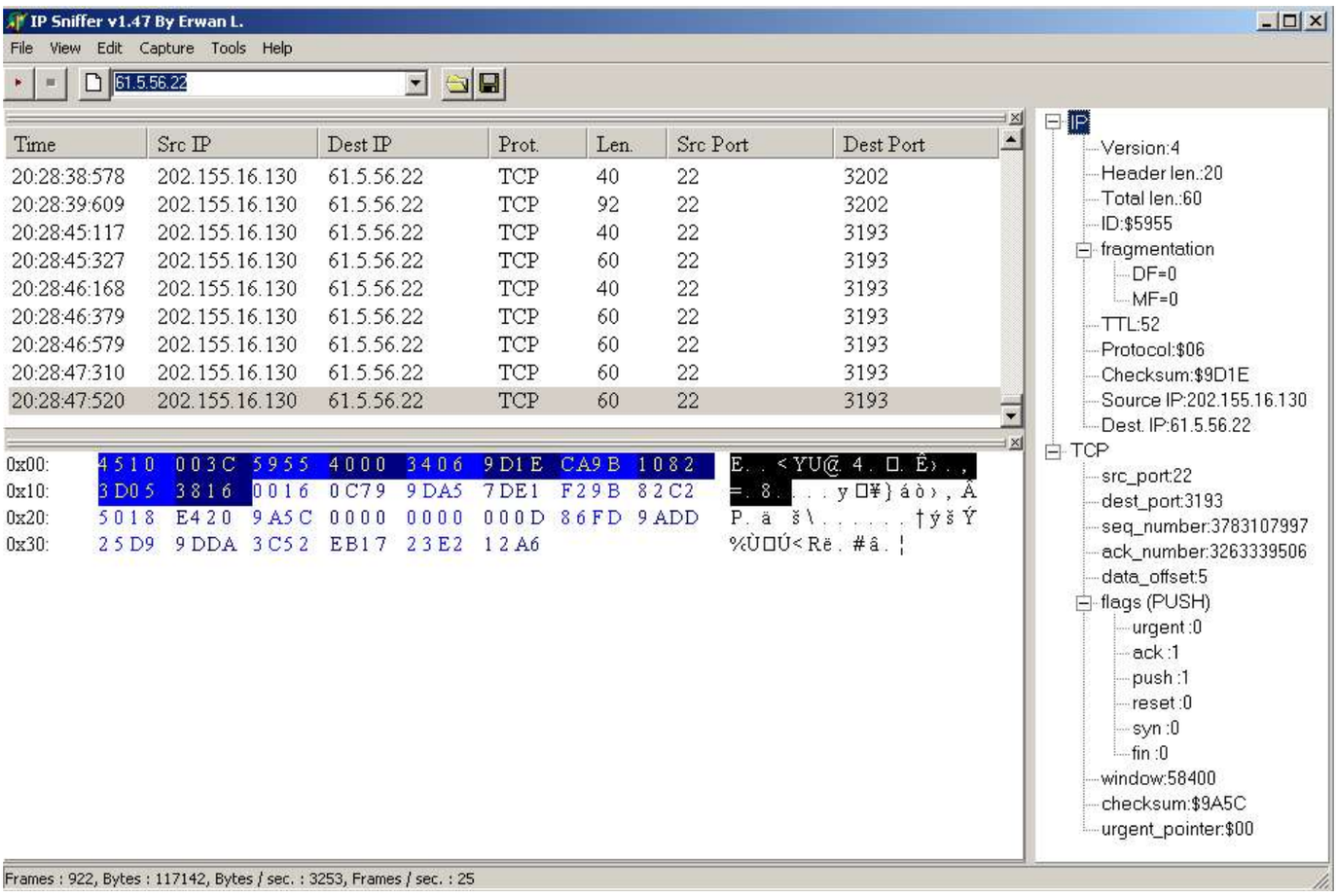
1. client mengirimkan kontrol TCP SYN ke server, dengan memberikan *sequence number* inisial
2. server menerima TCP SYN, dan membalasnya dengan kontrol SYNACK
   - ACK yang menyatakan telah menerima SYN
   - mengalokasikan buffer
   - menghasilkan sequence number untuk ke client

Pada saat **Menutup Koneksi**

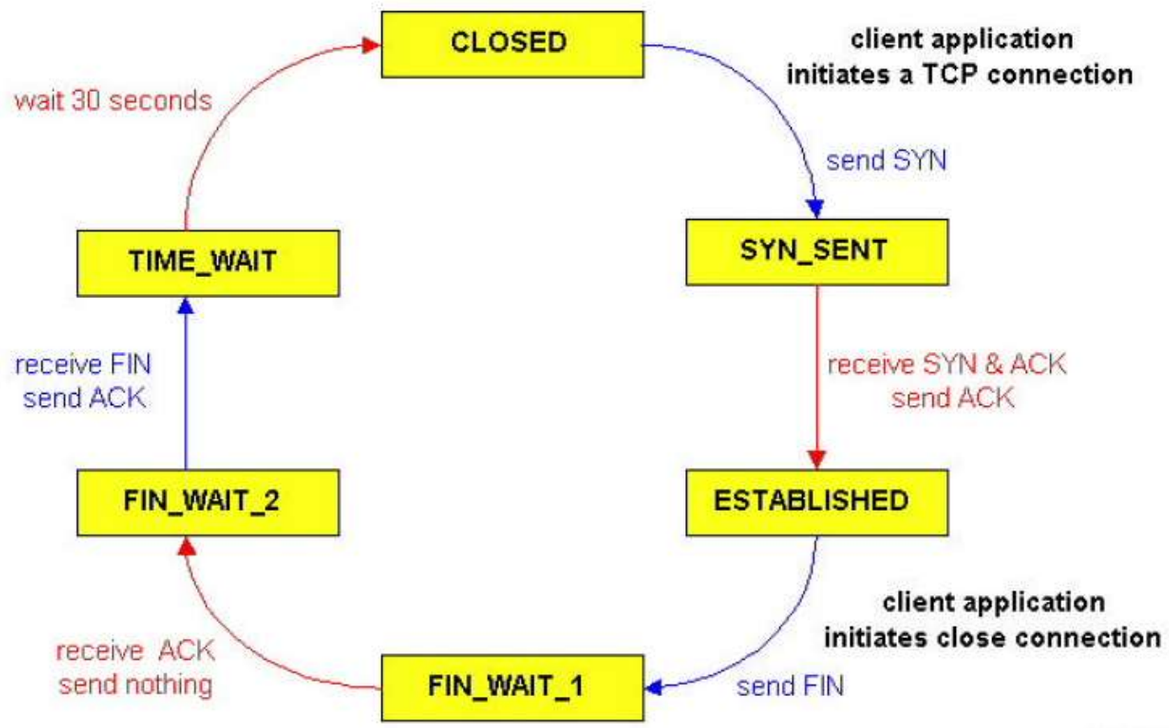
1. client mengirim kontrol TCP FIN ke server
2. server menerima FIN, dan membalas dengan ACK. Menutup koneksi dan mengirimkan FIN ke client.
3. Client menerima FIN dan membalas ACK
   - masuk pada masa menunggu balasan ACK terhadap dari server
4. Server menerima ACK dan koneksi tertutup.



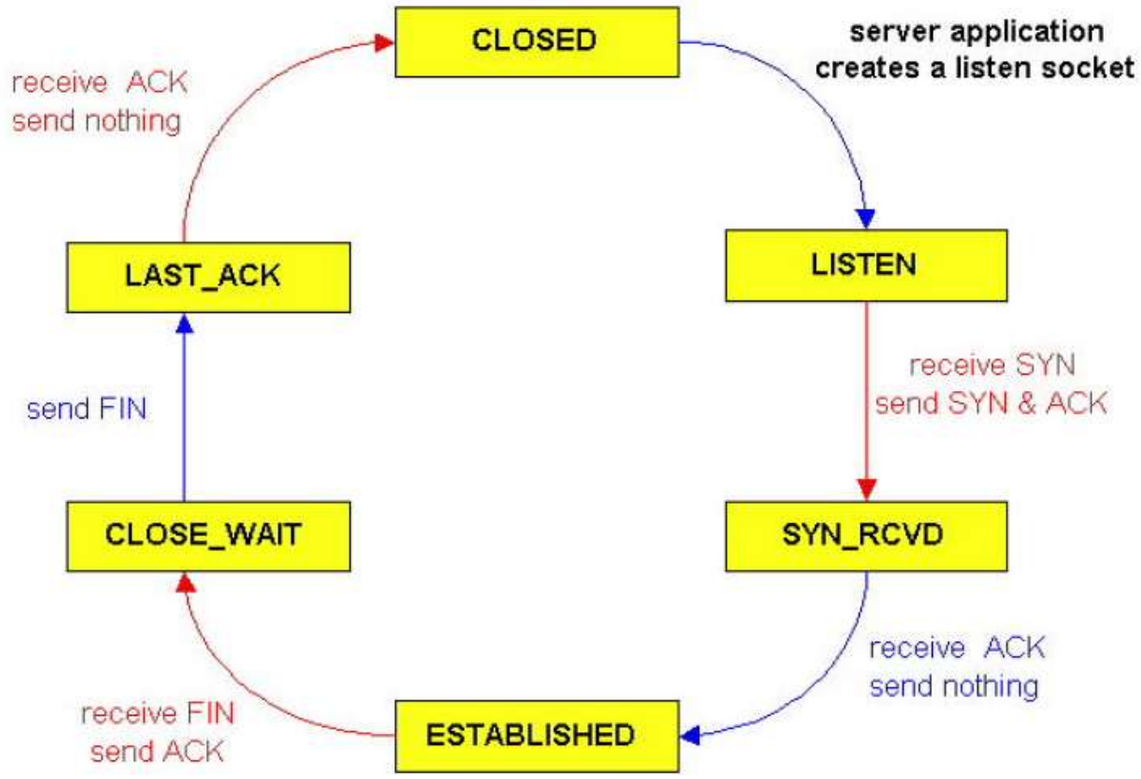
Berikut contoh sniffing paket TCP dengan menggunakan program IP Sniffer yang sama untuk menangkap paket UDP :



TCP Client Life Cycle



TCP Server Lifecycle



Berikut adalah contoh aplikasi client/server dengan memanfaatkan protokol TCP :

TCPServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class TCPServer {
  private final int INFO_PORT=50000;
  private String datafromClient;

  public TCPServer() {
    BufferedReader inFromClient;
    DataOutputStream outToClient;
    Socket serverSocket;

    try {
      ServerSocket infoServer = new ServerSocket(INFO_PORT);


      while (true) {
```

```
          serverSocket = infoServer.accept();
          System.out.println("Ada client yang terkoneksi!");

          inFromClient = new BufferedReader(
            new InputStreamReader(serverSocket.getInputStream()));

          outToClient = new DataOutputStream(
                    serverSocket.getOutputStream());

        boolean isQUIT = false;
        while (!isQUIT) {
          datafromClient = inFromClient.readLine();

          if (datafromClient.startsWith("QUIT"))
             isQUIT = true;
          else
             outToClient.writeBytes(
                datafromClient.replaceAll(" ", ":") + "\n");
         }
        outToClient.close();
        inFromClient.close();
        serverSocket.close();

        System.out.println("Koneksi client tertutup..");
       }
     }
     catch (IOException ioe){System.out.print("error: " + ioe);}
     catch (Exception e) {System.out.print("error: " + e);}
  }
  /* program utama */
  public static void main(String[] args) {
     new TCPServer();
  }
}
```

TCPClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class TCPClient {
```

```
  private final int INFO_PORT=50000;
  private final String TargetHost = "localhost";

  public TCPClient() {
    try {
      BufferedReader inFromUser = new BufferedReader(
                      new InputStreamReader(System.in));

      Socket clientSocket = new Socket(TargetHost, INFO_PORT);

      DataOutputStream outToServer = new DataOutputStream(
             clientSocket.getOutputStream());

      BufferedReader inFromServer = new BufferedReader(
         new InputStreamReader( clientSocket.getInputStream()));

      boolean isQuit = false;
      while (!isQuit) {
        System.out.print("Perintah Anda : ");
        String cmd = inFromUser.readLine();

        cmd = cmd.toUpperCase();
        if (cmd.equals("QUIT")) isQuit = true;

        outToServer.writeBytes(cmd + "\n");
        String result = inFromServer.readLine();
        System.out.println("Dari Server: " + result);
      }

      outToServer.close();
      inFromServer.close();
      clientSocket.close();
    }
    catch(IOException ioe){System.out.println("Error:" + ioe);}
    catch (Exception e) {System.out.println("Error:" + e);}
  }
  /* program utama */
  public static void main(String[] args) {
    new TCPClient();
  }
}
```