Network Programming

[CACS355]

BCA 6th Sem

Er. Sital Prasad Mandal

(Email : info.sitalmandal@gmail.com)

Bhadrapur, Jhapa, Nepal

https://networkprogam-mmc.blogspot.com/

https://networkprogam-mmc.blogspot.com/

Unit-10

UDP

1. The UDP Protocol
2. UDP Clients
3. UDP Servers
4. The DatagramPacket Class
   i. The Constructors
   ii. The get Methods
   iii. The setter Methods
5. The DatagramSocket Class
   i. The Constructors
   ii. Sending and Receiving Datagrams
   iii. Managing Connections
6. Socket Options
   i. SO_TIMEOUT
   ii. SO_RCVBUF
   iii. SO_SNDBUF
   iv. SO_REUSEADDR
   v. SO_BROADCAST
   vi. IP_TOS
7. Some Useful Applications
   i. Simple UDP Clients
   ii. UDPServer
   iii. A UDP Echo Client
8. DatagramChannel
   i. Using DatagramChannel

https://networkprogam-mmc.blogspot.com/

Unit-10

User Datagram Protocol

- Unreliable Datagram Protocol
- Packet Oriented, not stream oriented like TCP/IP
- Much faster but no error correction
- NFS, TFTP, and FSP use UDP/IP
- Must fit data into packets of about 8K or less

The UDP Classes:

- Java's support for UDP is contained in two classes:
  1. java.net.DatagramSocket
  2. java.net.DatagramPacket
- A datagram socket is used to send and receive datagram packets.

1. NFS is a **network file sharing protocol that defines the way files are stored and retrieved from storage devices across networks**.
2. **Trivial File Transfer Protocol** (TFTP) is a simple lockstep File Transfer Protocol which allows a client to get a file from or put a file onto a remote host.
3. **FSP** stands for File Service **Protocol**. It is a very lightweight UDP based **protocol** for transferring files.

Unit-10

java.net.DatagramPacket

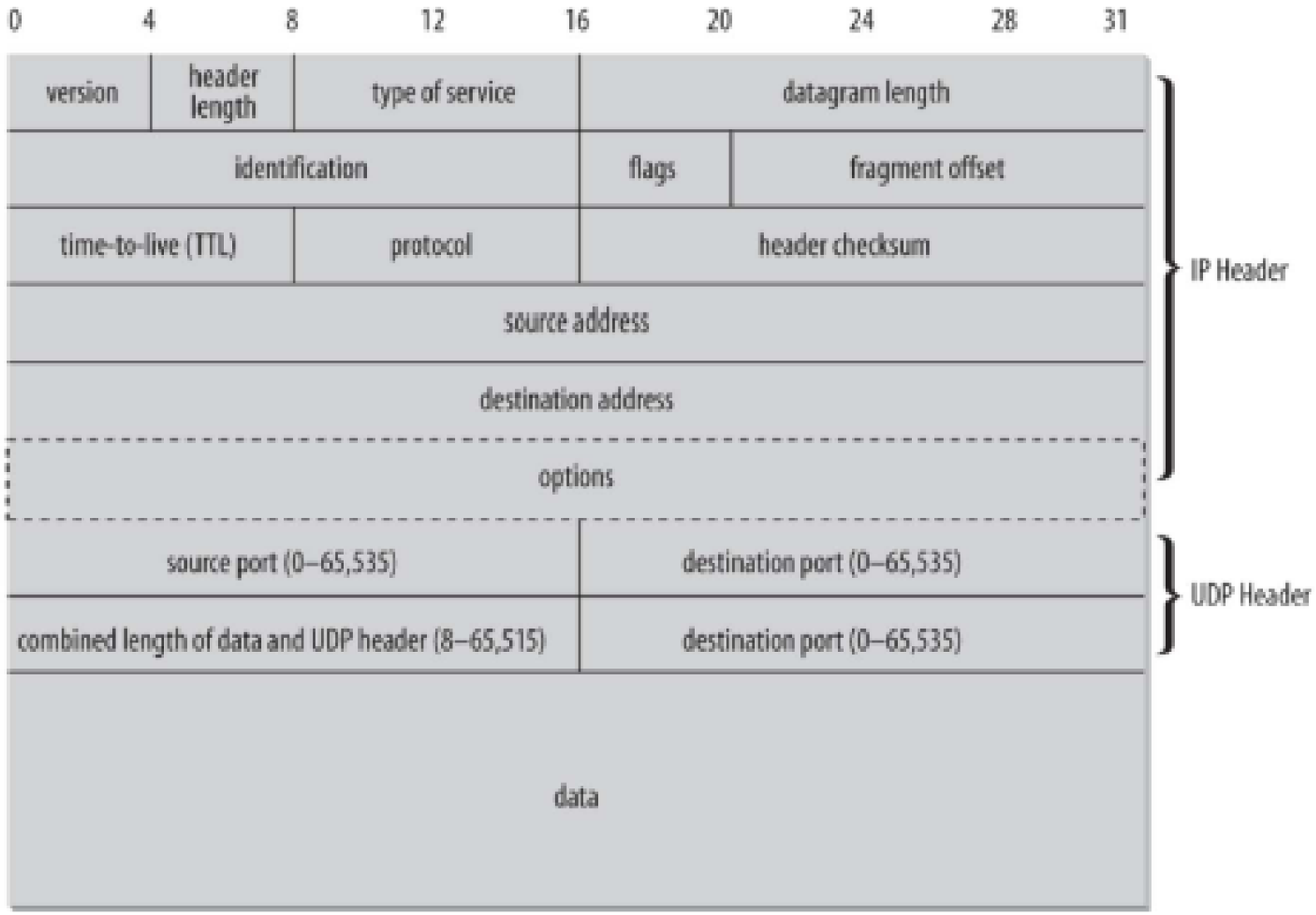
- A wrapper for an array of bytes from which data will be sent or into which data will be received.
- Also contains the address and port to which the packet will be sent.

- Java's implementation of UDP: split into two classes
  - DatagramPacket: stuffs bytes of data into UDP packets called datagrams
  - DatagramSocket: sends/receives datagrams
    - To send: put the data in a datagramPacket and send it using a DatagramSocket
    - To receive: take a DatagramPacket object from a DatagramSocket
    - Have no notion of a unique connection between two hosts
    - Always work with individual datagram packets

Unit-10

The DatagramPacket Class

- In Java, a UDP datagram is represented by an instance of the DatagramPacket class



Unit-10

DatagramPacket Constructors

- 6 Constructors
  - Two constructors for receiving data

```
public DatagramPacket(byte[] buffer, int length)
public DatagramPacket(byte[] buffer, int offset, int length)
```

    - Example: receiving a datagram of up to 8,192 bytes

```
byte[] buffer = new byte[8192];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
```

  - Four constructors for sending data: assigning destination host and port

```
public DatagramPacket(byte[] data, int length,
    InetAddress destination, int port)
public DatagramPacket(byte[] data, int offset, int length,
    InetAddress destination, int port)
public DatagramPacket(byte[] data, int length,
    SocketAddress destination)
public DatagramPacket(byte[] data, int offset, int length,
    SocketAddress destination)
```

    - Example: using InetAddress object

```
String s = "This is a test";
byte[] data = s.getBytes("UTF-8");

try {
  InetAddress ia = InetAddress.getByName("www.ibiblio.org");
  int port = 7;
  DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
  // send the packet...
} catch (IOException ex)
}
```

Unit-10

The DatagramSocket Class

- DatagramSocket: all datagram sockets bind to a local port, on
  - which they listen for incoming data and
  - which they place in the header of outgoing datagrams
- The constructors
  - DatagramSocket(): Constructs a datagram socket and binds it to any available port on the local host machine

```
try {
  DatagramSocket client = new DatagramSocket();
  // send packets...
} catch (SocketException ex) {
  System.err.println(ex);
}
```

  - DatagramSocket(DatagramSocketImpl impl): Creates an unbound datagram socket with the specified DatagramSocketImpl
  - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine
  - DatagramSocket(int port, InetAddress laddr): Creates a datagram socket, bound to the specified *local* address
  - DatagramSocket(SocketAddress bindaddr): Creates a datagram socket, bound to the specified *local* socket address

```
SocketAddress address = new InetSocketAddress("127.0.0.1", 9999);
DatagramSocket socket = new DatagramSocket(address);
```

Unit-10

Look for Local UDP Ports

Local port scanner that looks at ports from 1024 and up

```
import java.net.*;

public class UDPPortScanner {

  public static void main(String[] args) {
    for (int port = 1024; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on port i
        DatagramSocket server = new DatagramSocket(port);
        server.close();
      } catch (SocketException ex) {
        System.out.println("There is a server on port " + port + ".");
      }
    }
  }
}
```

```
% java UDPPortScanner
There is a server on port 2049.
There is a server on port 32768.
There is a server on port 32770.
There is a server on port 32771.
```

Unit-10

Socket Options

- SO_TIMEOUT: receive() waits for an incoming datagram (or InterruptedIOException) (in ms)

```
public void setSoTimeout(int timeout) throws SocketException
public int getSoTimeout() throws IOException
```

- SO_RCVBUF: the max size of datagram packets can be received
  - Platform dependent, ie. 64K for Linux

```
public void setReceiveBufferSize(int size) throws SocketException
public int getReceiveBufferSize() throws SocketException
```

- SO_SNDBUF: suggested send buffer size used for network output

```
public void setSendBufferSize(int size)  throws SocketException
public int getSendBufferSize() throws SocketException
```

- SO_REUSEADDR: controls whether multiple datagram sockets can bind to the same port and address at the same time
  - If multiple sockets are bound to the same port, received packets will be copied to all bound sockets

```
public void setReuseAddress(boolean on) throws SocketException
public boolean getReuseAddress() throws SocketException
```

  - setReuseAddress() must be called before the new socket binds to the port
- SO_BROADCAST: controls whether a socket is allowed to send packets to and receive packets from broadcast addresses

```
public void setBroadcast(boolean on) throws SocketException
public boolean getBroadcast() throws SocketException
```

- IP_TOS: set or inspect TOS field (class of service) of IP header for a socket

```
public int getTrafficClass() throws SocketException
public void setTrafficClass(int trafficClass) throws SocketException
```

Unit-10

DatagramChannel

- DatagramChannel class: support nonblocking UDP applications
  - Methods return quickly if network isn't immediately ready to receive or send data
    - Since UDP is more asynchronous than TCP, the net effect is smaller
  - Can be registered with a Selector
    - Monitors multiple sockets
- Using DatagramChannel
  - Java 6-: need to use the DatagramSocket class to bind a channel to a port
  - Java 7+: need not to use DatagramSocket, nor DatagramPacket
    - Read and write byte buffers as do with a SocketChannel

Unit-10

Using DatagramChannel

- open(): open a socket
  - Not initially bound to any port. Bind it with the channel's socket object

Java 6-

```
DatagramChannel channel = DatagramChannel.open();
SocketAddress address = new InetSocketAddress(3141);
DatagramSocket socket = channel.socket();
socket.bind(address);
```

Java 7+

```
DatagramChannel channel = DatagramChannel.open();
SocketAddress address = new InetSocketAddress(3141);
channel.bind(address);
```

- receive(): read one datagram packet from the channel into a ByteBuffer
  - If the packet has more data than the buffer can hold, the extra data is thrown away with no notification of the problem

```
public SocketAddress receive(ByteBuffer dst) throws IOException
```

- send(): write one datagram packet into the channel from a ByteBuffer

```
public int send(ByteBuffer src, SocketAddress target) throws IOException
```

  - Return the number of bytes written; the source ByteBuffer can be reused
    - 0 if the channel is in nonblocking mode and the data can't be sent immediately
- close(): close the socket
  - isOpen(): return false if the channel is closed

Java 6-

```
DatagramChannel channel = null;
try {
  channel = DatagramChannel.open();
  // Use the channel...
} catch (IOException ex) {
  // handle exceptions...
} finally {
  if (channel != null) {
    try {
      channel.close();
    } catch (IOException ex){// ignore
} } }
```

Java 7+ (AutoCloseable)

```
try (DatagramChannel channel = DatagramChannel.open()) {
  // Use the channel...
} catch (IOException ex) {
  // handle exceptions...
}
```

Unit-10

Reading and Writing on Connected Channels

- connect(): enforce to connect to a particular remote address
  - Does not block in any meaningful sense

```
SocketAddress remote = new InetSocketAddress("time.nist.gov", 37);
channel.connect(remote);
```

  - The channel will only send data to or receive data from this address
  - isConnected(): tell if the DatagramChannel is limited to one host

```
public boolean isConnected()
```

  - Disconnect(): break the connection

```
public DatagramChannel disconnect() throws IOException
```

- read(): like receive(), but only used on connected channels

```
public int read(ByteBuffer dst) throws IOException
public long read(ByteBuffer[] dsts) throws IOException
public long read(ByteBuffer[] dsts, int offset, int length)throws IOException
```

  - Only read a single datagram packet from the network
    - If the packet has more data than the ByteBuffer can hold, the extra data is thrown away
  - Return the number of bytes, -1 if the channel has been closed, or 0 if
    - The channel is nonblocking and no packet was ready
    - A datagram packet contained no data
    - The buffer is full
- write(): like send(), but also only used on connected channels

```
public int write(ByteBuffer src) throws IOException
public long write(ByteBuffer[] dsts) throws IOException
public long write(ByteBuffer[] dsts, int offset, int length)throws IOException
```

  - Not guaranteed to write the complete contents of the buffer
    - But can call again and again until the buffer is fully drained and sent

```
while (buffer.hasRemaining() && channel.write(buffer) != -1) ;
```

https://networkprogam-mmc.blogspot.com/

Unit-10

UDP Client

- Common steps to write UDP clients
  1. Open a socket on port 0

```
DatagramSocket socket = new DatagramSocket(0);
```

  2. (Optional, highly recommended) setSoTimeout(): set a timeout on the connection

```
socket.setSoTimeout(10000);
```

  3. Set up the packets: two packets, one to send (request) and one to receive
     - Specify the destination host and port in the sent/request packet

```
InetAddress host = InetAddress.getByName("time.nist.gov");
DatagramPacket request = new DatagramPacket(new byte[1], 1, host, 13);
byte[] data = new byte[1024];
DatagramPacket response = new DatagramPacket(data, data.length);
```

  4. Send the request packet and then receive the response

```
socket.send(request);
socket.receive(response);
```

  5. Extract the bytes from the response

```
String daytime = new String(response.getData(), 0, response.getLength(),
    "US-ASCII");
System.out.println(daytime);
```

  6. Close: Java 7+ Autocloseable or Java 6-

```
try (DatagramSocket socket = new DatagramSocket(0)) {
  // connect to the server...
} catch (IOException ex) {
  System.err.println("Could not connect to time.nist.gov");
}
```

```
DatagramSocket socket = null;
try {
  socket = new DatagramSocket(0);
  // connect to the server...
} catch (IOException ex) {
  System.err.println(ex);
} finally {
  if (socket != null) {
    try {
      socket.close();
    } catch (IOException ex) {
      // ignore
} } }
```

Unit-10

UDP Server

- Steps to create a UDP server
  1. Open a datagram socket on a well-known port
     - No separate datagram server socket

```
DatagramSocket socket = new DatagramSocket(13);
```

  2. Create a packet into which to receive a request

```
DatagramPacket request = new DatagramPacket(new byte[1024], 0, 1024);
socket.receive(request);
```

  3. Create a response packet

```
String daytime = new Date().toString() + "\r\n";
byte[] data = daytime.getBytes("US-ASCII");
InetAddress host = request.getAddress();
int port = request.getPort();
DatagramPacket response = new DatagramPacket(data, data.length, host, port);
```

  4. Send the response packet back over the same socket that received it

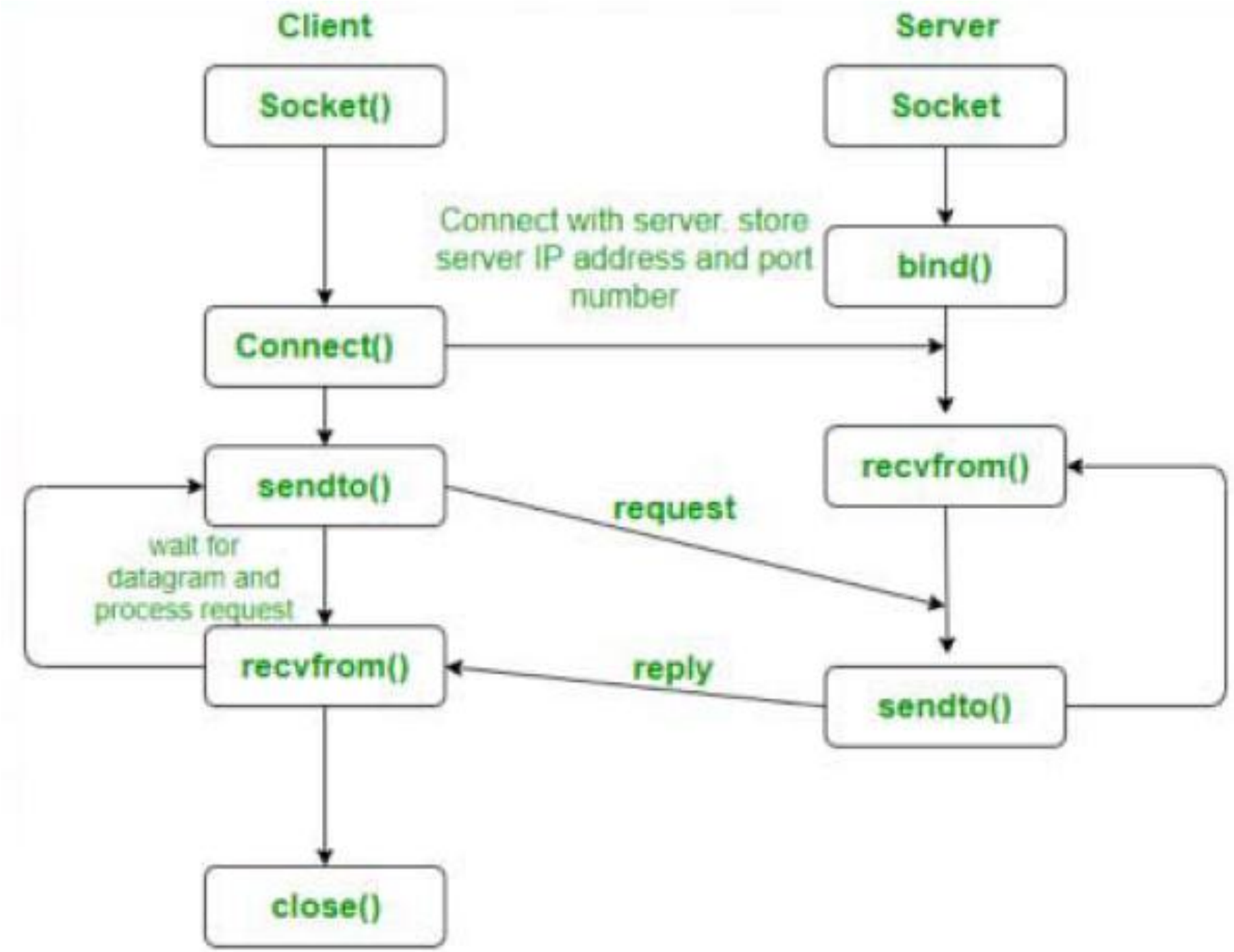
```
socket.send(response);
```

- UDP servers tend not to be as multithreaded as TCP servers
  – They usually don't do a lot of work for any one client

https://networkprogam-mmc.blogspot.com/

Unit-10

UDP Socket



Unit-10 Write a program for UDP Server example.

```
import java.net.*;

public class UDPServer {
    public static void main ( String[] args ) throws Exception{
        DatagramSocket ss = new DatagramSocket(9876); //Socket() and Bind()
        InetAddress ia = InetAddress.getByName("localhost");
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while (true){
            DatagramPacket revPkt = new DatagramPacket(receiveData,receiveData.length);
            ss.receive(revPkt);
            String sentence = new String(revPkt.getData());
            System.out.println("Recived: " + sentence);
            InetAddress ip = revPkt.getAddress();
            int port = revPkt.getPort();
            sendData = sentence.getBytes();
            DatagramPacket sndPkt = new DatagramPacket(sendData,sendData.length,ia,port);
            ss.send(sndPkt); // SendTo()
        }
    }
}
```

Unit-10 Write a program for UDP Client example.

```
public class UDPClient {
    public static void main ( String[] args ) throws  Exception{
        System.out.println("Enter any Text");
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket cs = new DatagramSocket(); //Socket()
        InetAddress ia = InetAddress.getByName("localhost");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = br.readLine();
        sendData = sentence.getBytes();

        DatagramPacket sndPkt = new DatagramPacket(sendData,sendData.length,ia,9876);
        cs.send(sndPkt); //SendTo()

        DatagramPacket revPkt = new DatagramPacket(receiveData,receiveData.length);
        cs.receive(revPkt); //ReceiveFrom()

        String modifiedSentence = new String(revPkt.getData());
        System.out.println("From Server:" + modifiedSentence);
        cs.close();
    }
}
```

Unit-10

Summary

12.1 The UDP Protocol
12.2 UDP Clients
- DaytimeUDPClient (Example 12-1)

12.3 UDP Servers
- DaytimeUDPServer (Example 12-2)

12.4 The DatagramPacket Class
- DatagramExample (Example 12-3)

12.5 The DatagramSocket Class
- UDPPortScanner (Example 12-4)
- UDPDiscardClient (Example 12-5), UDPDiscardServer (Example 12-6)

12.6 Socket Options
12.7 Some Useful Applications
- UDPPoke (Example 12-7), UDPTimeClient (Example 12-8)
- UDPServer (Example 12-9), FastUDPDiscardServer (12-10), UDPEchoServer (12-11)
- A Multi-threaded UDP Echo Client
  UDPEchoClient (Example 12-12), SenderThread (12-13), ReceiverThread (12-14)

12.8 DatagramChannel
- UDPDiscardServerWithChannels (Example 12-15)
- UDPEchoServerWithChannels (Example 12-16)
- UDPEchoClientWithChannels (Example 12-17)
- DefaultSocketOptionValues (Example 12-18)