UNIT- IV

UDP Datagrams and Socket

* It is a userdatagram protocol.
* It is an alternative protocol for sending data over IP that is very quick but not reliable
* When you send UDP data you have no way of knowing if the data is arrived
* And we are also not aware if the different pieces of data arrived in the order in which you sent them
* The pieces that arrive generally arrive quickly

The UDP Protocol:

* UDP is located between the Application layer and the IP layer
* It serves as an intermediary between the application programs and the network operations.

Position of UDP in the TCP/I protocol suite

Application Layer : SMTP, FTP, TFTP, DNS, SNMP, DHCP

Transport Layer: SCTP TCP UDP

Network Layer: IGMP, ICMP, IP, ARP

Data Link Layer: Underlying LAN or

Physical Layer: WAN Technology

User datagram

UDP packets called user datagrams have fixed size header of 8 bytes

Header : 8 bytes

Header and data: 8 to 65535

Header format: source port 0-16 destination port 16 – 31

Source port : gives the total length

Destination port: gives check sum

* Udp header add only 8 bytes to the IP header .
* Udp includes source and destination port number
* Port numbers are given as 2 byte unsigned integers 65,536-8bytes for the header this is redundant with the data gram length field of the IP header
* The exact number depends on the size of the header
* Check sum field is an optional and not used in or accessible from application laye
* If the checksum for data fails the native software silently discards the datagram
* Neither the sender or the receiver are notified
* UDP is an unreliable protocol

Structure of UDP datagram:

Version header length Types of service datagram length

Identification flags fragment

--------------------------------------------------------------------------------------------------------------------------------------Time to live (TTL) Protocol Header check sum

--------------------------------------------------------------------------------------------------------------------------------------

Source Address

Destination Address

Option

Souce port (0-65,535) destination port (0-65,535)

Combined length of data and UDP header(8-65535) destination port (0-65,535)

In java UDP datagram is represented as public final class DatagramPacket extends object

Datagram Socket Class:

* They are bound to local port
* They listen for incoming data and which they place in the header of outgoing datagram.
* If you are writing a server clients you should know which port the server is listening for incoming datagrams
* When server constructs a datagram socket it specifies the local port on which it will listen

The constructorsL

* The datagram socket constructors are used in different situation

1. 1st constructor opens a datagram on an anonymous local port
2. Opens a datagram on all local network interface
3. Opens a datagram socket on a well known local port on specific network interface

Program for look for local ports:

Import java.et.\*;

Public class UDPPort Scanner{

Public static void main(String [] args)

{

For(int port = 1024;port<=65535;port++)

Try

{

//the next line will fail and fall into the catch block if there is already a server running on port i

DatagramSocket server=new DatageramSocket(port);

Server.Close();

}

Catch(SocketException ex) {

System.out.println(“there is a server on port” +port +”.”);

}

}

}

Write a program to construct a Datagram packet to receive data

Import java.net.\*;

Public class datagram {

Public static void main(String[] args)

{

String s=”this is a test”;

Byte[] data=s.getBytes();

Try

{

InetAddress ia = InetAddress.getByName(“[www.google.com](http://www.google.com)”);

Int port=7;

DatagramPacket dp=new DatagramPacket(data,data.Length,ia,port);

System.out.println(“this packet is addressed to “ +dp.getAddress() +”on port” + dp.getPort());

System.out.println(“There are” +dp.getLength() + “bytes of data in the packet”);

System.out.println(new String(dp.getData(), dp.getOffset(), dp.getLength()));

}

Catch(UnknownHostException e)

{

System.err.println€;

}

}

}

Some useful Applications:

* In this section several internet servers and clients use the DatagramPacket and DatagramSocket.
* Many internet protocols have both TCP and UDP implementation
* When the IP packet is received by a host , the host determines whether the packet is a TCP packet or a UDP Datagram by inspecting the header
* TCP and UDP servers can share the same port number without problems

Simple UDP Client:

* Several internet services need to know only the clients address and port
* They ignore any data the client sends in the datagram
* Daytime, quote of the day,time and char gen are four such protocol
* Each of these responds the same way, regardless of the data contained in the datagram
* Clients for these protocol simply send a UDP datagrams to the server and read the response that come back

Sample program:

Import java.net.\*;

Import java.io.\*;

Import java.lang.\*;

Public class EC

{

Public static void main(String args[]) throws IOException

{

Byte[] buff = new byte[1024];

DatagramSocket soc = new DatagramSocket(9999);

String S = “ From Client hello server”;

Buff=s.getBytes();

InetAddress a= InetAddress.getByName(“Gfi-335”);

DatagramPacket pac=new DatagramPacket(buff,buff.length,a,8888);

Soc.send(pac);

System.Out.println(“end of sending”);

Byte[] buff1=new byte[1024];

Buff1=s.getBytes();

Pac=new DatagramPacket(buff1,buff.Length);

Soc.receive(pac);

String msg=new String(pac,getDate());

System.out.println(msg);

} System.out.println(“eof”)l

}

}

Import java.net.\*;

Import java.io.\*;

Import java.lang.\*;

Public class ES

{

Public static void main(String args[]) throws IOException

{

Byte[] buff=new byte[512];

DatagramSocket soc = new DatagramSocket(8888);

DatagramPacket pac=new DatagramPacket(buff, buff.length);

System.out.printl n(“serverver started”);

Soc.receive(pac);

String msg=new String(pac.getData());

System.out.println(msg);

System.out.println(“end of reception”);

String s=”From Server hello client”;

Byte[] buff=new byte[512];

Buff1=s.getBytes();

InetAddress a = pac.getAddress();

Int port=pac.getPort();

Pac=new DatagramPacket(buff, buff1.length,a,Port);

Soc.send(pac);

System.out.println(“end of sending”);

}

}

Datagram Channel:

\*datagram channel is a near complete alternate abstraction for UDP I/O

\*Datagram channel is a subclass of selectable that can be registered with a selector

\*datagram channel is used to bind a channel to a port.

Opening Socket:

\*The java.nio.channels.DatagramChannel class does not have any public constructor

\* You create a new Datagram Channel object using static open() method

Public static DatagramChannel Open() throws IOException

Datagram Channel=DatagramChannel.Open();

* Datagram channel is initially not bound to any port.
* To bind it you need to access the channels peer DatagramSocket object using the Socket method.
* Public abstract DatagramSocket Socket()

For e.g. this binds channel to port 3141.

SocketAddress address = new InetSocketAddress(3141);

DatagramSocket socket=channel.Socket();

Socket.bind(address);

Connecting:

Datagram Channel can be connected and it can be configured to only receive datagrams from and send datagrams to one host.

\*this is accomplished with connect method.

Public abstract DatagramChannel Connect(SocketAddress remote) throws IOException

TCP\_NODELAY() : it will makemultiple buffers as Individual packet

SO\_TIMEOUT(): java.net.socket.time out blocks from reader enable disable

SO\_LINGER(): socket is shut down

SO\_RCVBUF(): it is the size of the buffer kernel allocates to hold the data arriving into the given socket and then read

SO-SNDBUF: only in TCP communicates to kernel and stops sending data and accumulates in local buffer

Receiving:

* The receive() method reads one datagram packet from the channel into a bytebuffer

Public abstract SocketAddress receive(ByteBuffer dst) throws IOException

Send() :

* The send() method writes one datagram packet into the channel from a bytebuffer to the address specified as the second argument
* Public abstract int Send\*(ByteBuffer src, SocketAddress target) throws IOException
* The send() method returns the number of bytes written
* This will either be the number of bytes remaining in the output buffer or zero

Reading()

Three readmethods:

1. Public abstract int read(Byte Buffer for dst) throws IOException
2. Public final lonread(Byte Buffer[] dsts) throws IOException
3. Public final long read(Byte Buffer[] dsts,int offset,int length) throws IOException

* As with the receive() method if the datagram packet has more data then the ByteBuffer(s) can hold extra data

Writing:

* This can be used instead of send() method
* i. public abstract int write(ByteBuffer src) throws IOException
* ii. Public final long write(Byte Buffer[]dsts , int offset, int length) throws IOException

Writing Data to server:

* Sometimes you may need to write data to a URL connection

For e.g when you submit a form using web server using POST or upload a file using PUT the getOutPutStream() returns an OutputStream on which you can write data

For e.g.

Try

{

URL u = new URL(<http://google.com/cgi-bin/acgi>);

//open connect and prepare it to POST

URLConnection uc=U.OpenConnection();

Uc.Set DoOutput(true);

OutputStream raw = uc.getOutputStream();

Uc.SetDoOutput(true);

OutputStream raw=uc.getOutputStream();

OutputStream buffered=new BufferedOutputStream(raw);

OutputStream Writer out=new OutputStreamWriter(buffered,”8859\_1”);

Out.Write(“first=ram&middle=&last=kumar& work=String +Quartet\r\n”);

Out.flush();

Out.close();

}

Catch(IOException ex) {

System.err.println(ex);

}

Sending data with POST is almost as easy as with GET

Content Handler:

* Content handler deals with only actual data
* It takes raw input after all headers and converts it into right kind of object for java to deal with.
* For e.g ImageStream or an ImageProducer

Getting Content:

* The get content() method of URL connection use a content handler to turn the raw data of a connection into a java object

Public object getContent() throws IOException

It works only for protocolsj like HTTP, which has a clear understanding of MIME content type.

Public object get Content(class[] classes) throws IOException

* It tells you choose what class you would like
* It attempts to return the content in the form of one of the classes in the classes array
* The order of preference is the order of array

For e.g if you prefer an HTML file to be returned as a string, second choice as a reader and third choice as a InputStream you would write it as

URL u=new URL (“http:/www.the hungersite.com”);

URLConnection uc=u.OpenConnection();

Class[]type={String.class,Reader.class,InputStream.class};

ContentHandlerFactory: getCotent() of URL ConnectionInterested

* Java looks in the Hashtable to find the rightj content handler for the current URL as indicated by the URLContent type.
* It telle the program where it can find a content handler for a text/htmlfile an image file, giffile, or someother kind of file.

Public static void set conferenceHandlerFactory(ContentHandlerFactory)throws securityException, error

* Since this class is an abstract class can create directly instances of HttpURLConnection
* URL u=new URL(<http://www.amnesty.org>);
* HttpURLConnection http =(HttpURLConnection)u.openConnection();
* Or
* URL u=new URL(<http://www.amnesty.org/>);
* HttpURLConnection http=(HttpURLConnection) u.OpenConnection();
* The method argument should be one of these seven case sensitive strings
* GET
* POST
* HEAD
* PUT
* OPTIONS
* DELETE
* TRACE
* HEAD: it behaves much like GET it tells server only to return the HLTTP header. Not to actually send the file.
* The most common use of this method is to see if the file has been modified since the last time it was cached.
* E.g

Import java.net.\*;

Import java.io.\*;

Import java.util.\*;

Public class lastmodified {

Public static void main (String args[])

{

For(int i=0;i<args.length;i++) {

Try

{

URL u=new URL(args[i]);

Http URLConnection http=(HttpURLConnection) u.openConnection();

http.SetRequestMethod(“HEAD”);

System.out.println(u +”was last modified at” +new Date(http.getLastModified());

}

Catch(MalformedURLException ex){

System.err,println(args[i] + “is no a URLI understand”);

}

Catch(IOException ex) {

System.err.println(ex);

}

System.out.println();

}

}

}

UDP Echo client:

Step1: Start the main method in which we first assign a port variable to be 8000 and also set an address variable to IPAddress 127.-.0%

datagramSocket = new DatagramSocket();

Step2: Next we need start the thread to start a thread to handle incoming data (packets)

Byte [] buffer=new byte[1024];

Step3: then we need to create a datagram packet(i.e. incoming) passing it in buffer and its length size to the constructor

DatagramPacket incoming=new DatagramPacket(buffer, buffer.length);

Step4: after this while loop is used to call the receive method of our datagramSocket object, which will block (wait) until it receives a packet from a remote source.

datagramSocket.receive(incoming);

step 5: once we have data (apacket) from the server, we need to construct a string from the byte data from our packet by calling getData and getLength Methods

receivedString=new String(incoming.getData(), 0,incoming.getLength());

step 6: once we have a data as a string we then simply output it to the console.

System.out.println(receivedString);

Step 7: after this we need to catch the possible IOException and then end our while loop

Step8: again go back to the main code where we left it (after calling the start method) now simply call the readLine method of the keyboard reader object within an infinite loop

Input=keyboardReader.readLine();

Step 9: create a new DatagramPacket object called datagramPacket, to pan the Length of yhe string, the address to send Packets and finally the port at that address.

Step 10: once the packet is ready we send it using the send method passing datagramPacket as parameter

datagramSocket.send(datagramPacket);

UDP Echo Server/client:

Step1: first in the main method define a port for the server to be created

For e.g int port=8000;

Step2: create the server – UDP has no connection so no streams are associated, so first we have to create a datagram socket on a specified port.

For e.g .

serverDatagramSocket=new DatagramSocket(port);

step3: we need to pass the port to the constructor of the DatagramSocket class we have to enter an infinite while loop we can then create a DatagramPacket object which we store the incoming data

byte buffer[]=new byte[1024],

DatagramPacket datagramPacket=new DatagramPacket(buffer, buffer.length);

Step3: we need to pass the port to the constructor of the DatagramSocket class we need to have to enter an infinite while loop. We can then create a DatagramPacket object which we store the incoming data

Byte buffer[]=new byte[1024],

DatagramPacket datagramPacket=new DatagramPacket(buffer, buffer.length);

Step 4: we then call the receive method of our Datagram Socket object, ServerDatagramSocket.

The method block waits until the Packet is sent to the server and then store it in Packet. We then retrieve the byte data within the packet to create a string object which is our network message.

serverDatagramSocket .receive(datagram Packet)

input=new String(datagramPacket.getData(),0,datagramPacket.getLength());

step5: once we have received the message in the String object input,we then output the string to the console window and finally we send the packet to the sender by passing the packet we received to the send method of the server DatagramSocket object

serverDatagramSocket.send(datagramPacket);

The object methods:

* The URL connection class overrides only one method from java.lang.object to String();
* Public String toString()
* Even so there is little reason to print a URLConnection object or to convert to string
* If you are debugging toString() is called the same way as every other Tostring() method

Security considerations for URL Connections:

* URL connection objects are subject to all the usual security restrictions about making network connections reading or writing files and so forth.
* For e.g. a jar URL that points into the applets ownjar file should be fine however a file URL that points to local hard drive should not be.
* Before attempting to connect a URL you may want to know whether the connection is allowed for this purpose URL connection class has a get Permission method
* Public permission getPermission method() throws IOException
* This returns java.security.permission object that specifies what permission is needed to connect to th URL
* It returns null if no permission is needed e.g
* Sub classes of URL Connection return different sub classesof java.security.permission.
* For.eg.
* [www.gwbush.com.getPermission()](http://www.gwbush.com.getPermission()) returns a java.net.SocketPermission for the host [www.gwbush.com](http://www.gwbush.com) with the connect and resolve actions

Guessing MIME content type:

* Every protocol and every server would use MIME types to specify the kind of file being transferred
* Not only do we have to deal with older protocols such as FTP that predicate MIME
* Public static String guessContentType fromName(String name)
* The guesses are determined by the content type properties file
* E.g.
* No extension, or unrecognised Extension
* .odo application oda
* .pdf application/pdf
* .eps,oi,ps- application/postscript
* .dvi application/x-dvi

The second MIME type guesser method is URLConnectiontype from Stream()

Public static String guessContentTypeFromStream(InputStream in)

* The method tries to guess the content type by looking at the first fe w bytes of data in the stream
* For this method to work, the InputStream must support marking the beginning of the stream after the first bytes have been read.

Http URL Connection:

* It is an abstract sub class
* It provides some additional methods that are helpful when working with http

httpURL:

public abstract class HttpURL Connection extends URLConnection extend URLConnection

* It contains methods to get and set the request method
* It decides whether to follow redirects, get the response code and message. It identifies if proxy server is being used.
* It also consists of various mnemonic constants matching the various HTTP response code

Public int getContentlength() : it shows how many bytes are there in the content.

If there is no getContent -length returns -1

Public String getContentEncoding()

* It returns a string that tells you , how the content is encoded
* If the content is send unencoded this method returns null

Public long getDate()

* It returns a long that tells you when the document was sent in milliseconds since midnight

Public long getExpiration()

* It is similar to get date differs only in how the return value is interpreted
* It returns long indicating the number of milliseconds after 12.00 am @ which point the document expires if HTTP header not included the method returns zero and the document will remain in cahe indefinitely.

Public long getLastModified()

* Returns the date on which the document was last modified
* If the HTTP header does not include a last modified field this method returns zero,

Configuring the connection:

The URL connection has server protected instance fields that define exactly how the client makes request to the server.

Protected URL url;

Protected Boolean doInput=true;

Protected Boolean doOutput=false;

Protected allowUserInteraction = defaultAllowUserInteraction

Protected Boolean usecaches = defaultUseCaches

Protected long ifModifiedsince = 0;

Protected Boolean connected = false;

URL connections:

1. It is an abstract class
2. It has 2 different but related purposes

* First it provides more control over the instruction with a server (HTTP server) rather than URL class with URL connection you can set the header fields used in the client request
* Second URL connection class is part of javas protocol handler mechanism . which includes the URL streamHandler class

Opening URL Connections:

A program that uses the URL Connection class follows the sequence of steps:

1. Construct a URL object
2. Invoke the URL object’s open connection () method to retrieve a URL connection object for that object
3. Configure the URL connection
4. Read the header file
5. Get an inputStream and ReadData
6. Get OutputStream and WriteData
7. Close the connection

e.g.

try

{

URL u=new URL(<http://www.greenpeas.org>);

URL Connection uc=u.OpenConnection();

}

Catch(MalFormedURLException ex) {

System.err.println(ex);

}

Catch(IOException ex){

System.err.println(ex);

}

Reading data from a server:

This is a minimal set of steps needed to retrieve data from URL using a URLConnection objects

1. Construct a URL object
2. Invoke the URL object s OpenConenction() method to retrieve a URLConnection object for that URL
3. Invoke jthe URL Connection’s getInputStream() method
4. Read from the inputStream using the usual Stream API

Download a web page with URL connection:

Import java.net.\*;

Import java.io.\*;

Public class source viewer2{

Public static void main(String[] args){

If(args.length>0) {

Try {

URL u= new URL(args[0]);

URLConnection uc=u.OpenConnection();

InputStream raw=uc.getInputStream(raw);

Readermr=new InputStreamReader(buffer);

Int c;

While((c=r.read())!=-1) {

System.out.print((Char) c);

}

}

Catch(MalformedURLException ex) {

System.err.println(args[0] +”is not parseableURL”);

}

Catch(IOException ex) {

System.err.println(ex);

}

}

}

Retrieving specific header fields:

The first six methods request specific particularly common fields from the header

These are:

\*Content –type

\* Content – length

\*Content-encoding

\*Date

\*Last-modified

\*Expires

Public string getContentType() -> this method returns MIME content type of data

* It relies on web server to send a valild content type
* E.g. Content-type:text/html;char-set UTF-8

Caches:

* Web browsers have been caching pages and images for years
* If logo is repeated on every page of a site the browser normally loads it only once and stores it in cache and reloads it from cache whenever it is needed
* If the cache returns the desired content the protocol handler need not connect to the remote server
* Two abstract methods in response cache class store and retrieve data from the systems single cache
* Public abstract CacheResponse get(URI uri, String request method, Map<String,List<String>> request(Headers) throws IOException
* Public abstract CacheRequest Put(URI uri,URLConnection Connection) throws IOException

JarURLConnection:

* This class is a specialized URL Connection that represents a connection to a Jar
* Jar URL is a compound URL that includes the URL of a JAR archieve and , optionally a reference to a file or directory within the Jar archieve
* Applet often store their class files in a JAR archieve which bundles all the classes in one package
* One package that still maintains the directory hierarchy needed to resolve fully qualified class names like com.mac.fac.net.QueryString
* Since the entire archieve is compressed and can be downloaded in a single HTTP connection it requires much less time to download.jar file

For e.g.

Try{

//open the URL connection for reading

URL u=new URL(“jar:http://www.cafeaulait.org/course/week1.jar!)week1/0.5.html”);

URL connection uc = u.open Connection();

InputStream in= uc.getInputStream();

//chain the InputStream to a reader

Reader r=new InputStreamReader(in);

Int c;

While((c=r.read())!=-1){

System.out.print((char) c);

}

}

Catch(IOException ex) {

System.err.println(ex);

}

Besides used methods of the URL connection class JarURLConnection in herits this class add eight new methods

Public URL getJarFileURL()

Public String getEntryName()

Public JarEntry() throws IOException

Public Manifest get Manifest() throws IOException

Public attributes get attribute() throws IOException

Public certificate[] get certificate() throws IOException

Public abstractJarFile getJarFile() throws IOException

getJarFileURL() –> it returns the URL of the jar file being used by this connection.

e.g.

A Jar Entry:

Jar:http://www.foo.com/bar/baz.Jar!/COM/foo/Que.class

A Jar file:

Jar:http://www.foo.com/bar/baz.jar!/com/foo/!/

Is referred to as the separator

getAttribute() -> returns the attributes object for this connection if the URL for it points to a Jar file entry, null otherwise

getCertificates():

returns the certificate object for this connection if the URL for it points to a Jar file entry, null otherwise

getEntryName():

returns the entryname for this connection

getJarFile():

returns the jarfile for this connection

getjarFileURL():

returns the URLfor the jar file for this connection

getMainAttributes():

returns the main attributes for the jar file for this connection.

getManifest():

returns the manifest for this connection or null if none.

getJarFile

public abstract JarFile get JarFile() throws IOException

>