Networks and Internet Programming (0907522)

CHAPTER 5

User Datagram Protocol (UDP)

Instructor: Dr. Khalid A. Darabkh

Overview

- The User Datagram Protocol (UDP) is a commonly used transport protocol employed by many types of applications.

- UDP is a connectionless transport protocol, meaning that it doesn't guarantee either packet delivery or that packets arrive in sequential order.

UDP

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

Advantages of UDP

- UDP communication can be more efficient than guaranteed-delivery data streams.
- Unlike TCP streams, which establish a connection, UDP causes fewer overheads.
- Real-time applications that demand up-to-the-second or better performance may be candidates for UDP, as there are fewer delays due to the error checking and flow control of TCP.
- UDP sockets can receive data from more than one host machine.

The UDP Classes

- Java's support for UDP is contained in two classes:

  `java.net.DatagramSocket`

  `java.net.DatagramPacket`

- A datagram socket is used to send and receive datagram packets.

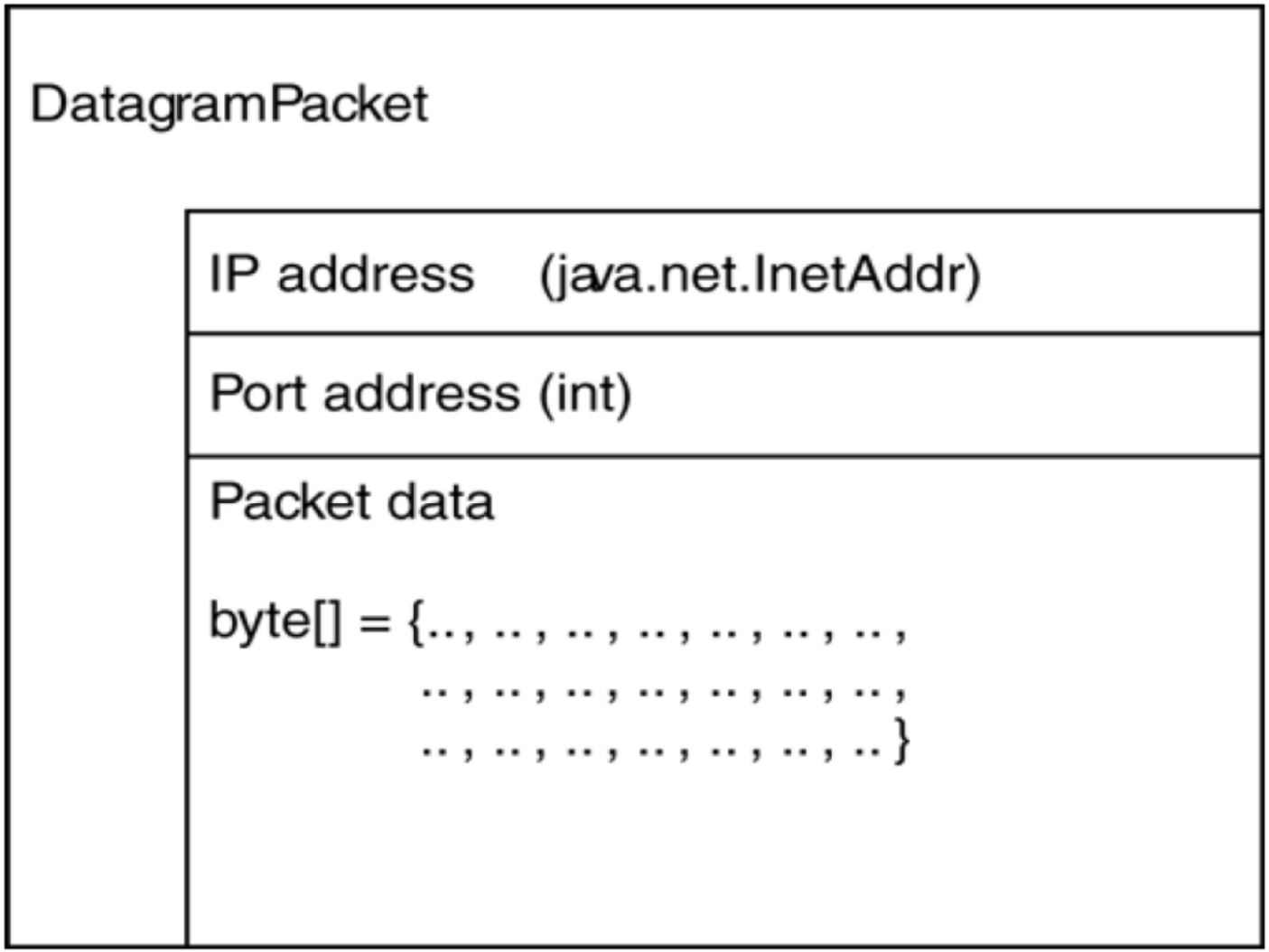
java.net.DatagramPacket

- The DatagramPacket class represents a data packet intended for transmission using the User Datagram Protocol

- Packets are containers for a small sequence of bytes, and include addressing information such as an IP address and a port.

java.net.DatagramPacket Cont.

Figure 5-2. `DatagramPacket` **representation of a UDP packet**



java.net.DatagramPacket Cont.

- The meaning of the data stored in a DatagramPacket is determined by its context.
- When a DatagramPacket has been read from a UDP socket, the IP address of the packet represents the address of the sender (likewise with the port number).
- However, when a DatagramPacket is used to send a UDP packet, the IP address stored in DatagramPacket represents the address of the recipient (likewise with the port number).

Creating a DatagramPacket

- There are two reasons to create a new DatagramPacket:

1. To send data to a remote machine using UDP
2. To receive data sent by a remote machine using UDP

- **Constructors:** The choice of which DatagramPacket constructor to use is determined by its intended purpose. Either constructor requires the specification of a byte array, which will be used to store the UDP packet contents, and the length of the data packet.

Two DatagramPacket Constructors

- Constructors for *receiving* datagram

public **DatagramPacket**(byte[] buffer, int length)

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

- **Example:**

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

- Constructors for *sending* datagram

public **DatagramPacket**(byte[] data, int length, InetAddress remote, int port)

public **DatagramPacket**(byte[] data, int offset, int length, InetAddress remote, int port)

- **Example:**

```
InetAddress addr = InetAddress.getByName("192.168.0.1");
DatagramPacket packet = new DatagramPacket ( new byte[128], 128, addr, 2000);
```

DatagramPacket (Example)

```
String s = “This is a test";
 byte[] data = s.getBytes();
try {
 InetAddress ia = InetAddress.getByName(“www.ju.edu.jo");
  int port = 7;
DatagramPacket dp = new DatagramPacket(data, data.length,
   ia, port);
}
catch (UnknownHostException e) {
  System.err.println(e);
}
```

Using a DatagramPacket

- The DatagramPacket class provides some important methods that allow the remote address, remote port, data (as a byte array), and length of the packet to be retrieved.
- As of JDK1.1, there are also methods to modify these, via a corresponding set method.
- This means that a received packet can be reused. For example, a packet's contents can be replaced and then sent back to the sender.

DatagramPacket: Methods

- **InetAddress getAddress()—** returns the IP address from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination IP address.
- **byte[] getData()—** returns the contents of the DatagramPacket, represented as an array of bytes.
- **int getLength()—** returns the length of the data stored in a DatagramPacket. This can be less than the actual size of the data buffer.

DatagramPacket: Methods Cont.

- **int getPort()—** returns the port number from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination port number.
- **void setAddress(InetAddress addr)—** assigns a new destination address to a DatagramPacket.
- **void setData(byte[] buffer)—** assigns a new data buffer to the DatagramPacket. Remember to make the buffer long enough, to prevent data loss.

DatagramPacket: Methods Cont.

- **void setLength(int length)—** assigns a new length to the DatagramPacket. Remember that the length must be less than or equal to the maximum size of the data buffer, or an IllegalArgumentException will be thrown. When sending a smaller amount of data, you can adjust the length to fit—you do not need to resize the data buffer.
- **void setPort(int port)—** assigns a new destination port to a DatagramPacket.

java.net.DatagramSocket

- The DatagramSocket class provides access to a UDP socket, which allows UDP packets to be sent and received.
- A DatagramPacket is used to represent a UDP packet, and must be created prior to receiving any packets.
- The same DatagramSocket can be used to receive packets as well as to send them.
- However, read operations are blocking, meaning that the application will continue to wait until a packet arrives.

java.net.DatagramSocket Cont.

- Since UDP packets do not guarantee delivery, this can cause an application to stall if the sender does not resubmit packets.
- There is no distinction between a UDP socket and a UDP server socket.
- Also unlike TCP sockets, a **DatagramSocket** can send to multiple, different addresses.
- The address to which data goes is stored in the packet not in the socket.

Creating a DatagramSocket

- A DatagramSocket can be used to both send and receive packets.
- public DatagramSocket() throws SocketException
- public DatagramSocket(int port) throws SocketException
- public DatagramSocket(int port, InetAddress laddr) throws SocketException

- The first is for *client datagram sockets*; that is sockets that send datagrams before receiving any.
- The second two are for *server datagram sockets* since they specify the port and optionally the IP address of the socket

Using a DatagramSocket

- DatagramSocket is used to receive incoming UDP packets and to send outgoing UDP packets.
- It provides methods to send and receive packets, as well as to specify a timeout value when nonblocking I/O is being used, to inspect and modify maximum UDP packet sizes, and to close the socket.

DatagramSocket: Methods

- **void close()—** closes a socket, and unbinds it from the local port.
- **void connect(InetAddress remote_addr int remote_port)—** restricts access to the specified remote address and port. The designation is a misnomer, as UDP doesn't actually create a "connection" between one machine and another. However, if this method is used, it causes exceptions to be thrown if an attempt is made to send packets to, or read packets from, any other host and port than those specified.
- **void disconnect()—** disconnects the DatagramSocket and removes any restrictions imposed on it by an earlier connect operation.

DatagramSocket: Methods Cont.

- **InetAddress getInetAddress()—** returns the remote address to which the socket is connected, or null if no such connection exists.
- **int getPort()—** returns the remote port to which the socket is connected, or –1 if no such connection exists.
- **InetAddress getLocalAddress()—** returns the local address to which the socket is bound.
- **int getLocalPort()—** returns the local port to which the socket is bound.

DatagramSocket: Methods Cont.

- **int getReceiveBufferSize()** throws java.net.SocketException— returns the maximum buffer size used for incoming UDP packets.
- **int getSendBufferSize()** throws java.net.SocketException— returns the maximum buffer size used for outgoing UDP packets.
- **int getSoTimeout()** throws java.net.SocketException— returns the value of the timeout socket option. This value is used to determine the number of milliseconds a read operation will block before throwing a java.io.InterruptedIOException. By default, this value will be zero, indicating that blocking I/O will be used.

DatagramSocket: Methods Cont.

- **void receive(DatagramPacket packet)** throws java.io.IOException—reads a UDP packet and stores the contents in the specified packet. The address and port fields of the packet will be overwritten with the sender address and port fields, and the length field of the packet will contain the length of the original packet, which can be less than the size of the packet's byte-array. If a timeout value hasn't been specified by using DatagramSocket.setSoTimeout(int duration), this method will block indefinitely. If a timeout value has been specified, a java.io.InterruptedIOException will be thrown if the time is exceeded.

DatagramSocket: Methods Cont.

- **void send(DatagramPacket packet)** throws java.io.IOException— sends a UDP packet, represented by the specified packet parameter.
- **void setReceiveBufferSize(int length)** throws java.net.SocketException— sets the maximum buffer size used for incoming UDP packets. Whether the specified length will be adhered to is dependent on the operating system.
- **void setSendBufferSize(int length)** throws java.net.SocketException— sets the maximum buffer size used for outgoing UDP packets. Whether the specified length will be adhered to is dependent on the operating system.

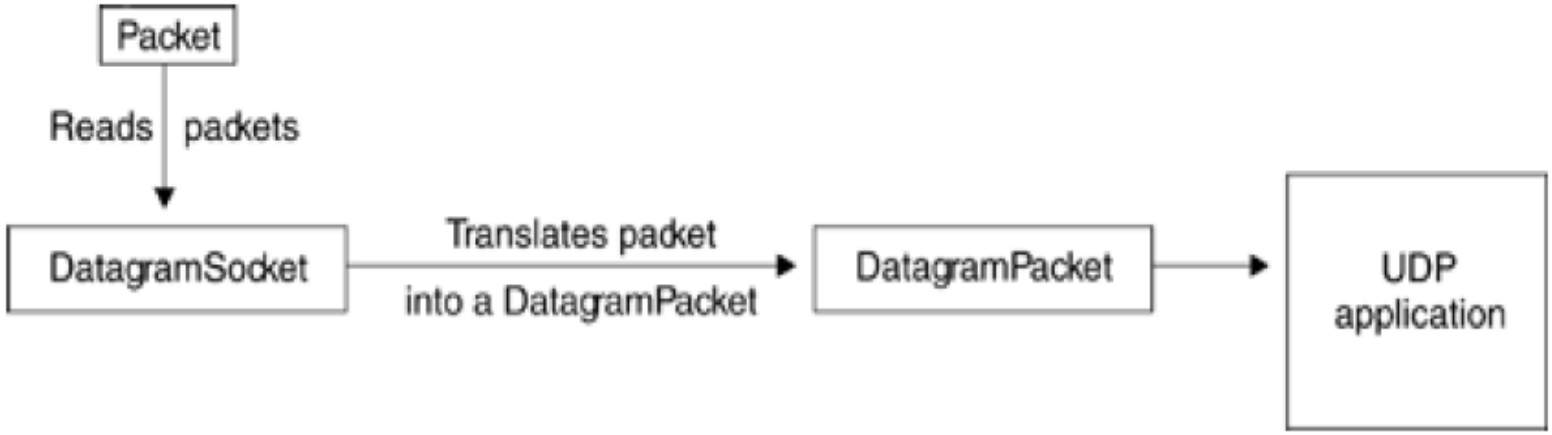
DatagramSocket: Methods Cont.

- **void setSoTimeout(int duration)** throws java.net.SocketException— sets the value of the timeout socket option. This value is the number of milliseconds a read operation will block before throwing a java.io.InterruptedIOException.

Listening for UDP Packets

- Before an application can read UDP packets sent to it by remote machines, it must bind a socket to a local UDP port using DatagramSocket, and create a DatagramPacket that will act as a container for the UDP packet's data.

Figure 5-3. UDP packets are received by a `DatagramSocket` and translated into a `DatagramPacket` object.



Listening for UDP Packets Cont.

- When an application wishes to read UDP packets, it calls the DatagramSocket receive method, which copies a UDP packet into the specified DatagramPacket. The contents of the DatagramPacket are processed, and the process is repeated as needed.

Listening for UDP Packets Cont.

The following code snippet illustrates this process:

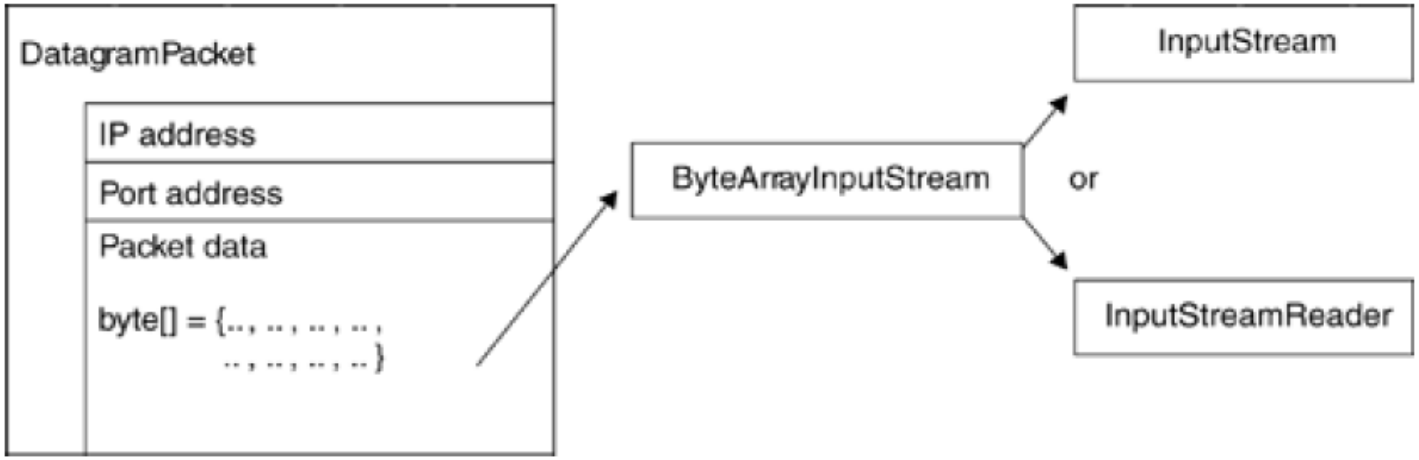
```
DatagramPacket packet = new DatagramPacket (new
byte[256], 256);
DatagramSocket socket = new
DatagramSocket(2000);
boolean finished = false;
while (! finished )
{
  socket.receive (packet);
  // process the packet
}
socket.close();
```

Listening for UDP Packets Cont.

- When processing the packet, the application must work directly with an array of bytes. If, however, your application is better suited to reading text, you can use classes from the Java I/O package to convert between a byte array and another type of stream or reader.
- By hooking a ByteArrayInputStream to the contents of a datagram and then to another type of InputStream or an InputStreamReader, you can access the contents of UDP packets relatively easily

Listening for UDP Packets Cont.

Figure 5-4. Reading from a UDP packet is simplified by applying input streams.



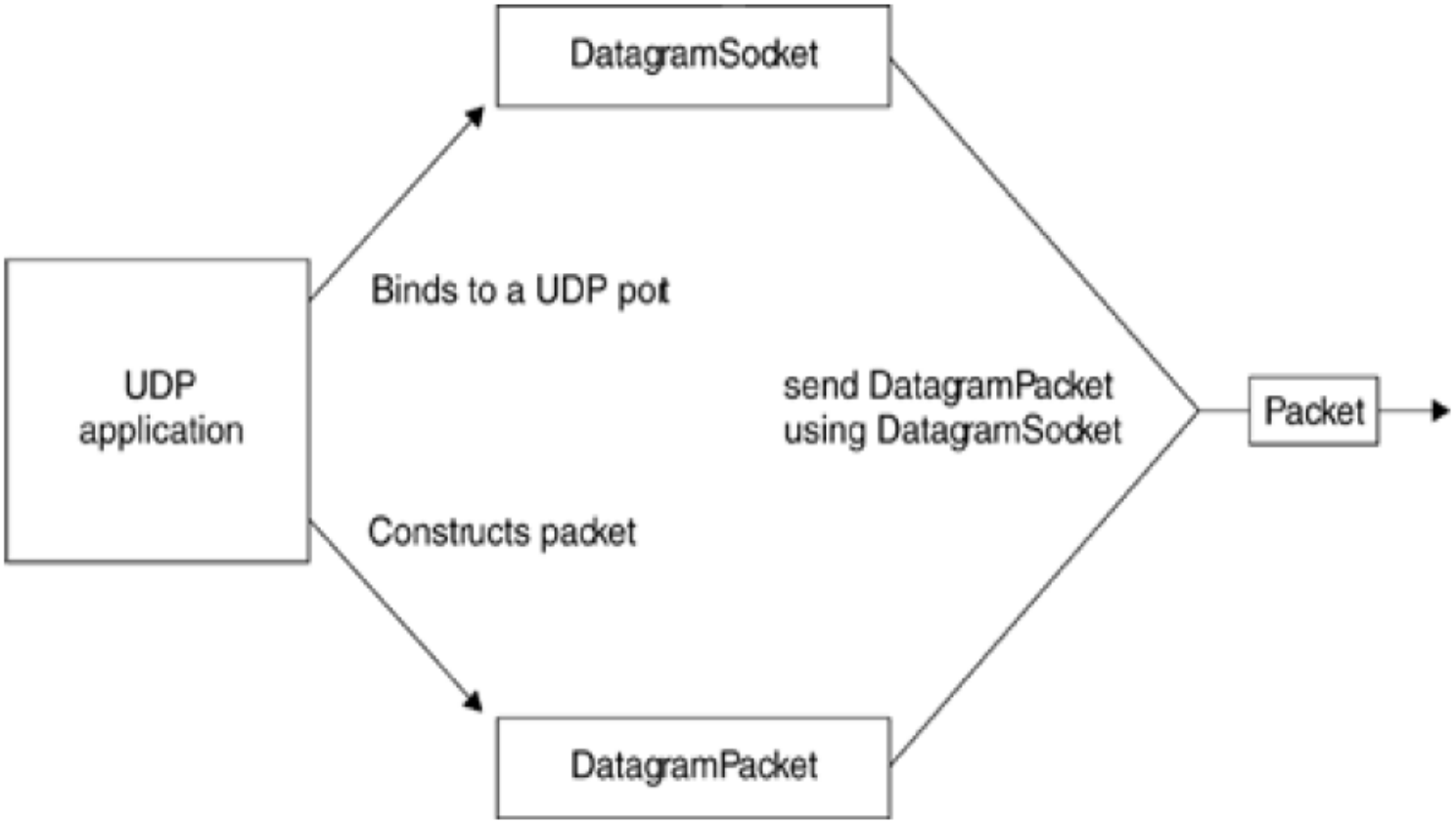
- For example, to hook up a DataInputStream to the contents of a DatagramPacket, the following code can be used:

```
ByteArrayInputStream bin = new
ByteArrayInputStream(packet.getData() );
DataInputStream din = new DataInputStream (bin);
// Read the contents of the UDP packet
.....
```

Sending UDP packets

- The same interface (DatagramSocket) employed to receive UDP packets is also used to send them.
- When sending a packet, the application must create a DatagramPacket, set the address and port information, and write the data intended for transmission to its byte array.
- If replying to a received packet, the address and port information will already be stored, and only the data need be overwritten.
- Once the packet is ready for transmission, the send method of DatagramSocket is invoked, and a UDP packet is sent

Sending UDP packets Cont.



- **Figure 5-5. Packets are sent using a DatagramSocket.**

Sending UDP packets Cont.

- The following code snippet illustrates this process:
- DatagramSocket socket = new DatagramSocket(2000);
- DatagramPacket packet = new DatagramPacket (new byte[256], 256);
- packet.setAddress (InetAddress.getByName (somehost));
- packet.setPort (2000);
- boolean finished = false;
- while !finished)
- {
- // Write data to packet buffer
- .....
- socket.send (packet);
- // Do something else, like read other packets, or check to
- // see if no more packets to send
- .....
- }
- socket.close();

Disadvantages of UDP

- **Lack of Guaranteed Delivery**
- **Lack of Guaranteed Packet Sequencing**
- **Lack of Flow Control**

Chapter Highlights

In this chapter, you have learned:

- How to bind to a local port using DatagramSocket
- How to create a DatagramPacket
- How to read from, and write to, a DatagramPacket using
  ByteArrayInputStream and ByteArrayOutputStream
- How to listen for UDP packets
- How to send UDP packets
- How to create a UDP server and a UDP client