Network Programming: A Java Perspective

COMP 30220: Distributed Systems

Lecturer: Rem Collier

Email: rem.collier@ucd.ie

TCP vs. UDP Datagram

- TCP is designed for reliable communications and ensures that lost or damaged data is resent
  - if packets arrive out of order TCP puts them back into the correct order
  - if the receiver is being flooded by data arriving too quickly TCP slows down the transmission to prevent packets being lost
- The User Datagram Protocol (UDP) is not reliable
- UDP datagrams add very little to the IP datagrams they sit on top of:
  - The header adds 8 bytes for storing source and destination ports
  - The number of bytes of data following the IP header
  - And an optional checksum…

UDP Datagram

- UDP doesn't provide any of the checks that TCP does
  - These checks are useful but slow down TCP
  - UDP is **lean** and **fast** but **error prone**

- If you require reliable transmission (e.g. FTP) you wouldn't normally use UDP.

- If you want fast transmission and don't care about a lost or swapped packet here or there then you would consider UDP (e.g. streaming real-time audio or video)

- You should also consider how reliable your network is…

Example: UDPServer

```
class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);

        while (true) {
            // Wait for a packet
            DatagramPacket receivePacket = new DatagramPacket(new byte[1024], 1024);
            serverSocket.receive(receivePacket); // Blocking Method

            // Process the received data
            String sentence = new String(receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);

            // Generate response and transmit
            byte[] sendData = sentence.toUpperCase().getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                                        receivePacket.getAddress(), receivePacket.getPort());
            serverSocket.send(sendPacket);
        }
    }
}
```

Example: UDPClient

```
class UDPClient {
    public static void main(String args[]) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();

        // Read User Input (and convert to byte array for transmission)
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        byte[] sendData = inFromUser.readLine().getBytes();

        // Send the packet
        DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, InetAddress.getByName("localhost"), 9876);
        clientSocket.send(sendPacket);

        // Wait for a response…
        DatagramPacket receivePacket = new DatagramPacket(new byte[1024], 1024);
        clientSocket.receive(receivePacket);

        // Display the output
        System.out.println("FROM SERVER:" + new String(receivePacket.getData()));
        clientSocket.close();
    }
}
```

UDP Deconstructed

- Creating a socket:
  - Server:

    ```
    DatagramSocket socket = new DatagramSocket(<port>);
    ```

  - Client:

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

- Sending data:

  ```
  byte[] data = new byte[1024];
  DatagramPacket packet =
      new DatagramPacket(data, data.length, <host>, <port>);
  socket.send(packet);
  ```

- Receiving data:

  ```
  byte[] data = new byte[1024];
  DatagramPacket packet = new DatagramPacket(data, data.length);
  socket.receive(packet);
  ```

UDP Deconstructed

- Closing a socket:

```
socket.close();
```

- UDP Packet Operations:
  - Getting the source address:

    ```
    InetAddress IPAddress = packet.getAddress();
    ```

  - Getting the source port:

    ```
    int port = packet.getPort();
    ```

  - Getting the data:

    ```
    String modifiedSentence = new String(packet.getData());
    ```

Multicast

- Previous two examples were point-to-point (unicast)
- Some tasks require 1-many communications
  - E.g. streaming radio and television
- But broadcast is too indiscriminate
  - Especially in switched networks
- Multicast fills the gap
  - it is broader than point-to-point unicast communications but narrower than broadcast
- Clients express interest in multicast data by joining a particular multicast group

Multicast Sockets

- Multicast is implemented by **multicast sockets**
  - Multicast sockets are similar to UDP sockets with the exception that they express interest in a multicast group rather than a specific port.
  - Multicast addresses are IP addresses in the range 224.0.0.0 to 239.255.255.255.
    - This range was chosen because all addresses have the binary digits 1110 as their first fours bits
    - Collectively they are known as **class D addresses**
  - Like any IP address multicast addresses can have a hostname
    - for example 224.0.1.1 (the network time protocol service) is ntp.mcast.net

- Multicast Applications:
  - Audio and Video
  - Multiplayer (network) games
  - distributed file systems
  - Conferencing
  - database replication
  - Multicast DNS (mDNS / Bonjour)

- Multicast is typically seen as **a LAN-level service** due to bandwidth concerns.

Multicast Addresses

- Network nodes can join together and form a multicast group all represented by a single multicast address.

- Multicast addresses are IP addresses in the range 224.0.0.0 to 239.255.255.255
  - All addresses in this range have the binary digits 1110 as their first fours bits and are collectively known as class D addresses.
  - Like any IP address multicast addresses can have a hostname
    - For example 224.0.1.1 (the Network Time Protocol service) is ntp.mcast.net
  - Some IP addresses are reserved by the Internet Assigned Numbers Authority (IANA)
    - You can create your own transient **multicast group** by picking a random address between 225.0.0.0 and 238.255.255.255
  - There is no multicast address that sends data to all hosts on the Internet!

Sending Multicast Data

- When a host wants to send to a multicast group it puts the data into a UDP Datagram packet
  - This is done for speed over reliability.
  - Connection-orientated TCP would be a bad choice for the multicast protocols because of ACK explosion.
  - Reliable Multicast Services do exist.

- From the programmers perspective it looks pretty much the same as UDP communications
  - However you have to worry about the time-to-live (TTL) value
  - The TTL says how many router hops a packet may take before being discarded
  - The TTL in multicast geographically limits the range of the packets.

Multicast Receiver in Java

```
public class MCastReceiver {
    public static void main(String[] args) {
        byte[] buffer = new byte[1024];
        DatagramPacket gift = new DatagramPacket(buffer, buffer.length);
        try {
            InetAddress address = InetAddress.getByName("225.0.0.1");
            MulticastSocket ms = new MulticastSocket(5000);
            ms.joinGroup(address);
            while (true) {
                ms.receive(gift);
                String present =
                    new String(gift.getData(), 0, gift.getLength());
                System.out.println(present);
            }
        } catch (Exception e) {}
    }
}
```

Multicast Sender in Java

```
public class MCastSender {
    public static void main(String[] args) {
        byte[] giftData = "Santa has got you a new bike".getBytes();
        try {
            // Join a multicast group
            MulticastSocket ms = new MulticastSocket();
            InetAddress address = InetAddress.getByName("225.0.0.1");
            ms.joinGroup(address);
            ms.setTimeToLive(1);

            // Send some data
            ms.send(new DatagramPacket(giftData, giftData.length, address, 5000));

            // Leave the group
            ms.leaveGroup(address);
            ms.close();
        } catch (Exception e) {}
    }
}
```

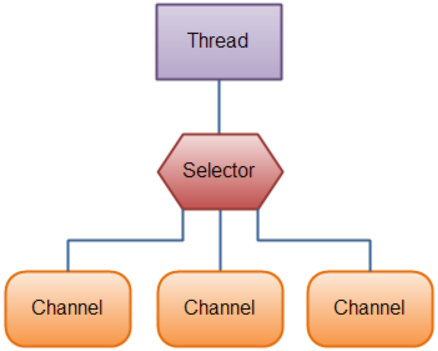
Dynamic Health/Discovery Service

- Each node joins a on a specified group and listens on a predefined port.

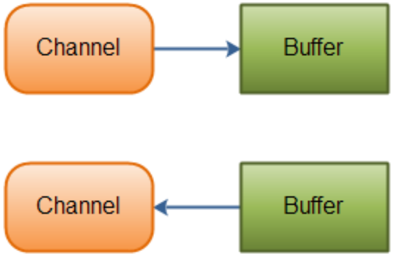
- Nodes periodically broadcast a “HELLO” message to the group.
  - The IP Addresses are used to identify the nodes AND that they are still available.
  - Failure to receive a “HELLO” in a given period of time can indicate that the node is no longer available.

- Variations on this theme can be used to offer enhanced services:
  - The hello message can include point-to-point contact info.
  - Bridges can be used to link multiple multicast networks.

A Third Way: Java NIO



- Part of Java since v1.4 (2002)
- Socket based implementations are slow:
  - Threads are expensive (~2Mb per thread)
  - Most CPU time is spend blocking (waiting to read)
- Java NIO (New IO) is an alternative IO API for Java.
  - Decouples data processing from connection handling
  - Read/Writes occur in parallel with data processing
  - Designed to scale
- Based on Channels, Buffers and Selectors:
  - Data is read from a buffer into a channel
  - Data is written from a channel into a buffer
  - Selectors monitor channels and handle “events”



Java NIO: Channels

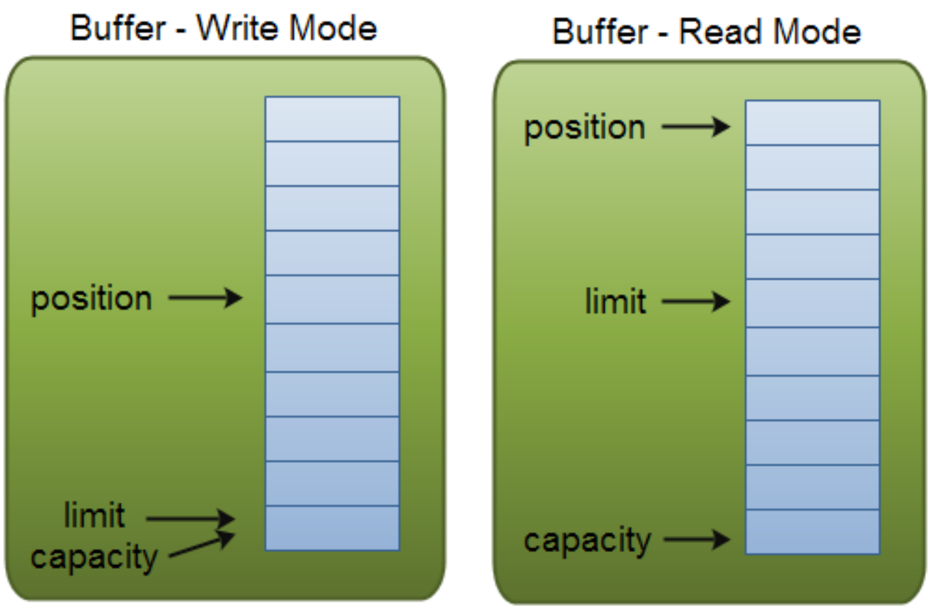
- Channels are like streams:
  - Data can be written to a channel or read from a channel.
  - Some commonly used channels are:
    - `FileChannel`
    - `DatagramChannel`
    - `SocketChannel`
    - `ServerSocketChannel`
- Channels are sources (or destinations) for the data being processed.
  - Like a stream you can read from or write to a channel.
- Channels generate events that can be monitored by a selector:
  - Connect, Accept, Read, Write

JAVA NIO: BUFFER DESIGN

- Buffers can be in either read or write mode.

- Buffers are typed (all Java primitives supported)
  - E.g. `ByteBuffer, LongBuffer`

- Three key pieces of information:
  - **Capacity**: the maximum number of values that can be stored in the buffer.
  - **Position**: the location of the next read / write in the buffer.
  - **Limit**: how much you can actually read/write for the buffer.



Java NIO: Basic Concept

- NIO Buffers have a mode:
  - The can be set to **read** or **write** mode.
  - The program defines when the buffer changes mode.

- Using a **Buffer** involves 5 steps:
  - Create a buffer (in write mode)
  - Write data to a buffer
  - Flip the buffer (to read mode)
  - Read data from the buffer
  - Clear / Compact the buffer and return to write mode

- Other functions of a buffer include:
  - Rewinding, marking and resetting, …

Java NIO: Reading From a File

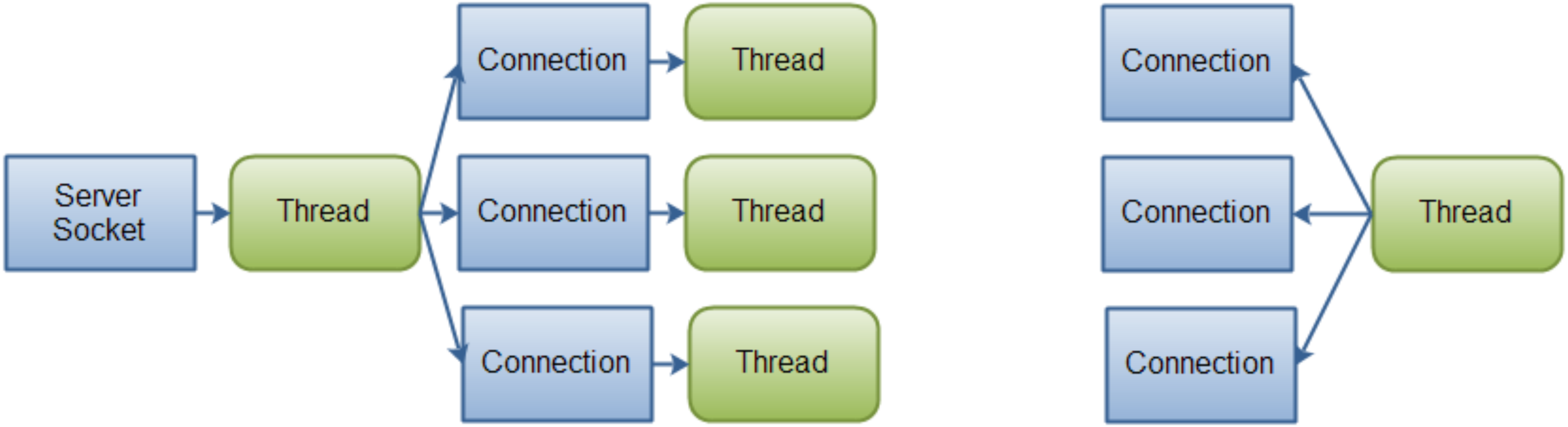
```
RandomAccessFile aFile =
    new RandomAccessFile("data/nio-data.txt", "rw");

FileChannel inChannel = aFile.getChannel();// Create a FileChannel
ByteBuffer buf = ByteBuffer.allocate(48);  // Buffer with 48 bytes (Write mode)

// Read from FileChannel to Buffer
int bytesRead = inChannel.read(buf);

// Check if any bytes were read...
while (bytesRead != -1) {
    System.out.println("Read " + bytesRead);

    // Flip Buffer to Read mode
    buf.flip();

    // Read from Buffer and write to Console (1 byte at a time)
    while(buf.hasRemaining()){
        System.out.print((char) buf.get());
    }

    // Empty Buffer and return to Write mode
    buf.clear();

    // Read more from FileChannel to Buffer
    bytesRead = inChannel.read(buf);
}

aFile.close();
```

Java NIO: Writing To a File

```
String newData = "New String to write to file..." + System.currentTimeMillis();

// Create a buffer
ByteBuffer buf = ByteBuffer.allocate(48);

// Clear it
buf.clear();

// Write data to the buffer
buf.put(newData.getBytes());

// Flip it to read mode
buf.flip();

// read data from the buffer..
while(buf.hasRemaining()) {
    channel.write(buf);
}
```

Java NIO: Selectors

- Component that manages multiple channels:
  - Channels register with a selector identifying which events the selector should handle.
  - The selector polls the registered channels whenever the select() method is invoked.
  - The selector returns a list of the "ready" channels that can then be processed.



Java NIO In Action

- Java NIO Support:
  - Native Java support
  - Basic Library for components
  - http://tutorials.jenkov.com/java-nio/

- Netty:
  - Alternate Java implementation
  - Includes: HTTP, SMTP, FTP support out of the box
  - https://netty.io

Other Technologies

- Socket based communication is low layer and requires conversion between language specific data types and byte representations/strings for transmission.

- There are alternatives that attempt to alleviate the strain of coding bespoke solutions:
  - Common Object Request Broker Architecture (CORBA)
    - Language-neutral middleware
  - Java Remote Method Invocation (RMI)
    - Java-based middleware solution
  - Web Services
    - XML, SOAP-based architecture, JSON, REST
  - Java Messaging Service (JMS)
    - Message Oriented Middleware