Transport layer protocols

Lecture 15:

Operating Systems and Networks

Behzad Bordbar

recap

- interprocess communication across machines?
- application makes data > Application Layer (HTTP, FTP, SMTP, DNS, VoIP) > become message > Transport Layer (TCP, UDP) > become TCP/UDP datagrams> Internet layer (IP, ICMP) > become IP datagram > Network layer (Ethernet and X.25) > becomes Frame
- Frames are through physical medium
- reverse order at the destination!
- We looked at the IP layer
- IP Address, mask , subnet, IPv6

Contents

- Transport layer protocols
- TCP, UDP
- Format of headers
- comparison

Transport layer protocols: UDP

- IP is between machines using IP address.
- TCP and UDP between processes. How?
- Port (16 bits address): delivery of messages within a particular computer. IP delivers to computer, TCP and UDP software delivers to process.
- UDP (basic, used for some IP functions)
- encapsulated inside IP packet (IP addresses of computers)
- Header of UDP datagram has : source port number 16 bits, destination port number 16 bits, total length 16 bits, Checksum 16 bits
- no guarantee of delivery, optional checksum
- messages up to 64KB

Transport layer protocols: TCP

- TCP (reliable, stream transport, port-to-port protocol)
- stream means connection-oriented: connection must be established before data is transferred.
- virtual circuit is created between sender and receiver which remains active (may be routed different ways)
- TCP alert receiver the data is coming.
  - data stream abstraction, reliable delivery of all data
  - messages divided into segments, sequence numbers
  - sliding window, acknowledgement+retransmission
  - buffering (with timeout for interactive applications)
  - checksum (if no match segment dropped)

Transport layer protocols: TCP

- Header of TCP segment
- source/destination port number each 16 bits
- Sequence number (32 bits): shows position of the segment in the original data stream
- Acknowledgement number (32 bits): if ack bit is on, it has number of next sequence expected (ackn… current one)
- HLEN (4 bits) showing number of 32bits words in TCP header
- reserved (6 bits) for future use
- six one bit control fields: URG, ACK,…

Transport layer protocols: TCP

- Window size (16 bits) size of sliding window
- Checksum (16 bits) error detection
- Urgent pointer (16 bits) if URG turned on, what part is urgent data
- padding

Communication service types

- Connectionless: UDP
  - 'send and pray' unreliable delivery
  - efficient and easy to implement
- Connection-oriented: TCP
  - with basic reliability guarantees
  - less efficient, memory and time overhead for error correction

Connectionless service

- UDP (User Datagram Protocol)
  - messages possibly lost, duplicated, delivered out of order, without telling the user
  - maintains no state information, so cannot detect lost, duplicate or out-of-order messages
  - each message contains source and destination address
  - may discard corrupted messages due to no error correction (simple checksum) or congestion
- Used e.g. for DNS (Domain Name System) or RIP.

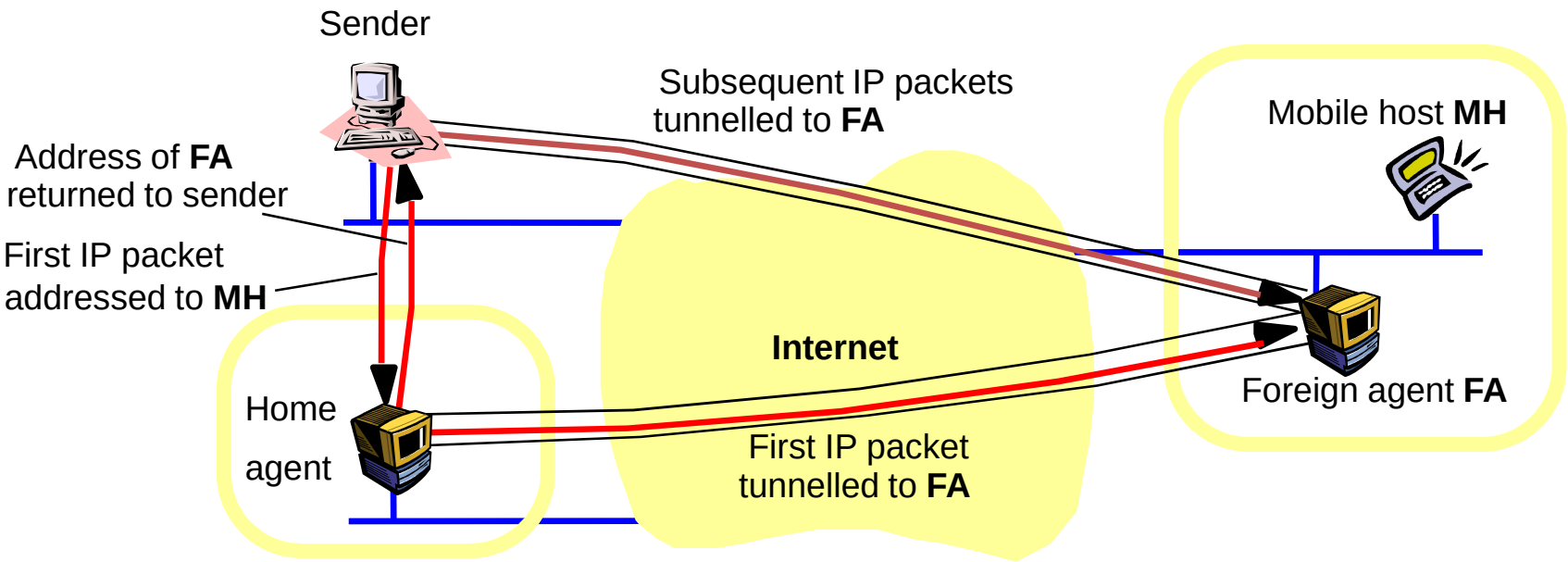
Connection-oriented service

- TCP (Transmission Control Protocol)
  - establishes data stream connection to ensure reliable, in-sequence delivery
  - error checking and reporting to both ends
  - attempts to match speeds (timeouts, buffering)
  - sliding window: state information includes
    - unacknowledged messages
    - message sequence numbers
    - flow control information (matching the speeds)
- Used e.g. for HTTP, FTP, SMTP on Internet.

MobileIP

- At home normal, when elsewhere mobile host:
  - notifies HA before leaving
  - informs FA, who allocates temporary care-of IP address & tells HA
- Packets for mobile host:
  - first packet routed to HA, encapsulated in MobileIP packet and sent to FA (tunnelling)
  - FA unpacks MobileIP packet and sends to mobile host
  - sender notified of the care-of address for future communications which can be direct via FA
- Problems
  - efficiency low, need to notify HA

MobileIP routing

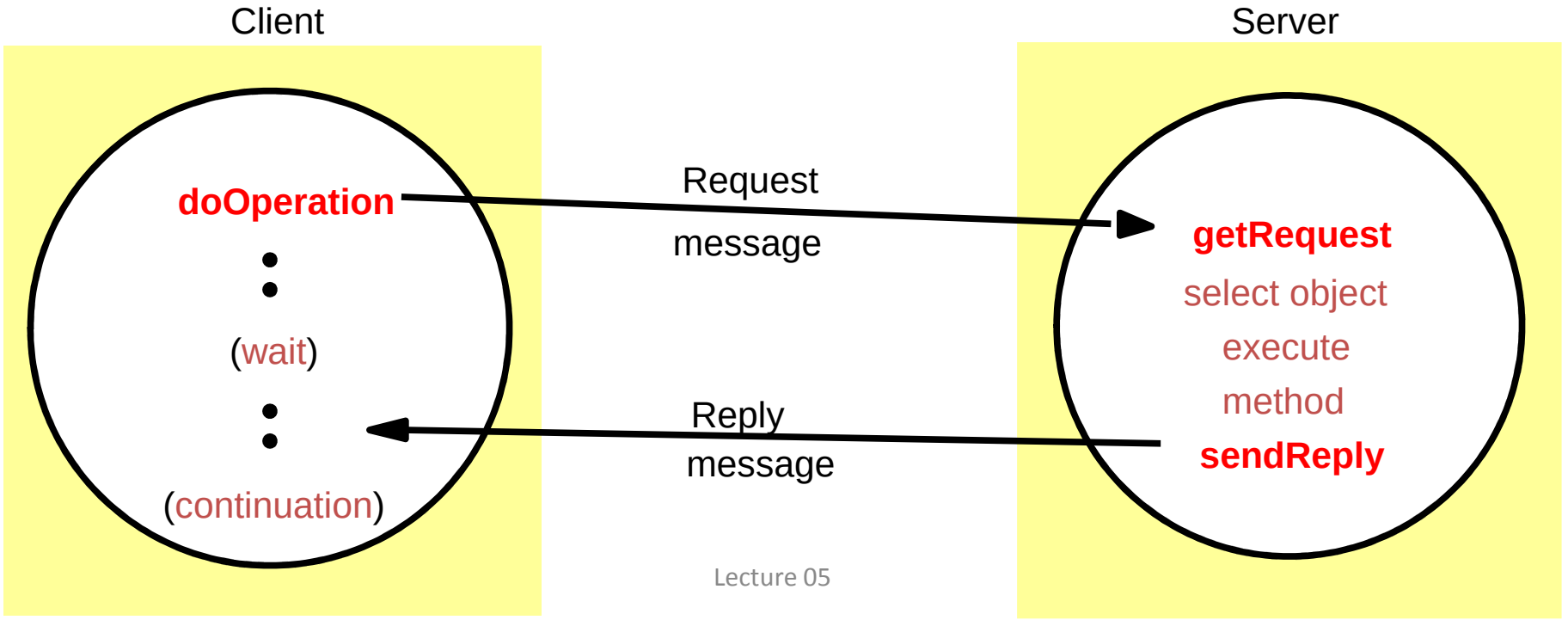


Wireless LAN (802.11)

- Radio broadcast (fading strength, obstruction)
- Collision avoidance by
  - slot reservation mechanism by Request to Send (RTS) and Clear to Send (CTS)
  - stations in range pick up RTS/CTS and avoid transmission at the reserved times
  - collisions less likely than Ethernet since RTS/CTS short
  - random back off period
- Problems
  - security (eavesdropping), use shared-key authentication

Interprocess communication

- Synchronous and asynchronous comm.
- Message destination
- Reliability
- Ordering



Asynchronous vs. Synchronous

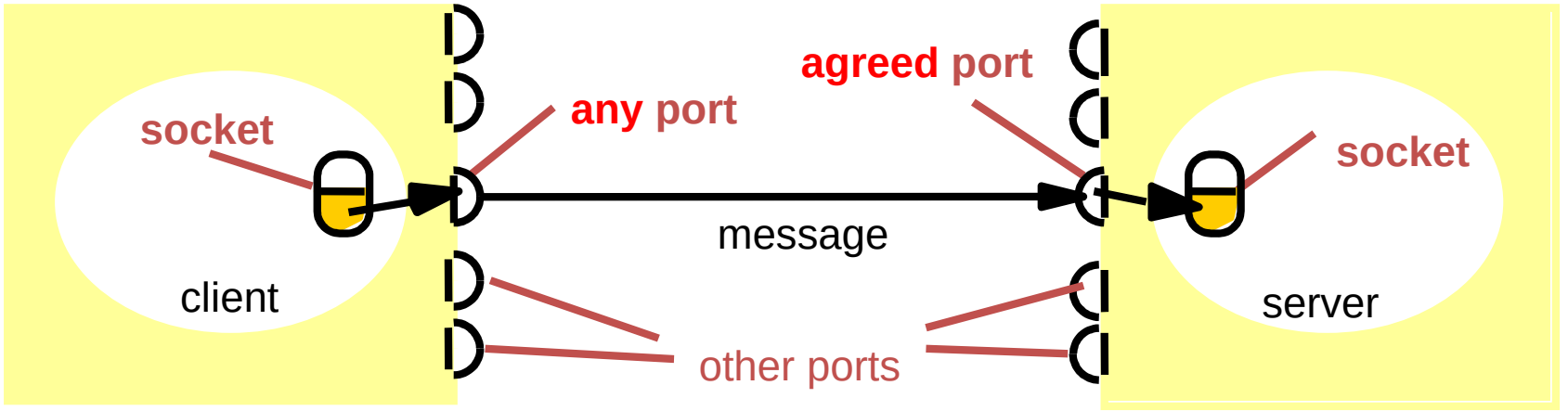
- Synchronize communication: sender and receiver
- synchronise on every message, i.e. *blocking* operations
- Asynchronous

❑ send is non-blocking

❑ receive can be blocking/non-blocking

Which one is better?

Message destination: Socket +Port



Socket = Internet address + port number.

Only one receiver but multiple senders per port.

Sockets

- Characteristics:
  - endpoint for inter-process communication
  - message transmission between sockets
  - socket associated with either UDP or TCP
  - processes bound to sockets, can use multiple ports
- Implementations
  - originally BSD Unix, but available in Linux, Windows,…
  - here Java API for Internet programming

Operations of Request-Reply

- *public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)*
  - sends a request message to the remote object and returns the reply.
  - the arguments specify the remote object, the method to be invoked and the arguments of that method.
- *public byte[] getRequest ();*
  - acquires a client request via the server port.
- *public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);*
  - sends the reply message reply to the client at its Internet address and port.

Java API for Internet addresses

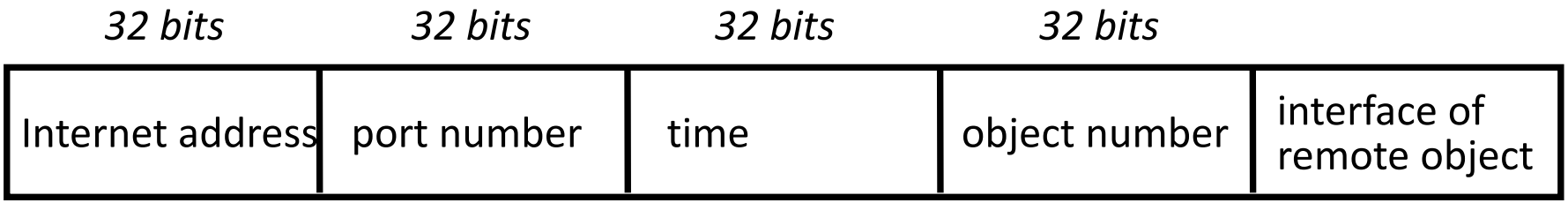
- Class *InetAddress*
  - uses DNS (Domain Name System)

```
InetAddress aC =
   InetAddress.getByName("gromit.cs.bham.ac.uk");
```

  - throws *UnknownHostException*
  - encapsulates detail of IP address (4 bytes for IPv4 and 16 bytes for IPv6)

Remote Object Reference

- An identifier for an object that is valid throughout the distributed system
  - must be unique
  - may be passed as argument, hence need external representation



Reliability

- Reliable communication:
- messages are guaranteed to be delivered despite a
- ‘reasonable’ number of packets being dropped or lost

Unreliable communication:

- messages are not guaranteed to be
- delivered in the face of even a single packet dropped or lost
- >>> Failure

Failure in point 2 point comm.

- DSs expected to continue if failure has occurred:
  - message failed to arrive
  - process stopped (and others may detect this)
  - process crashed (and others cannot detect this)
- Types of failures
  - benign
    - omission, stopping, timing/performance
  - arbitrary (called Byzantine)
    - corrupt message, wrong method called, wrong result

Omission and arbitrary failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

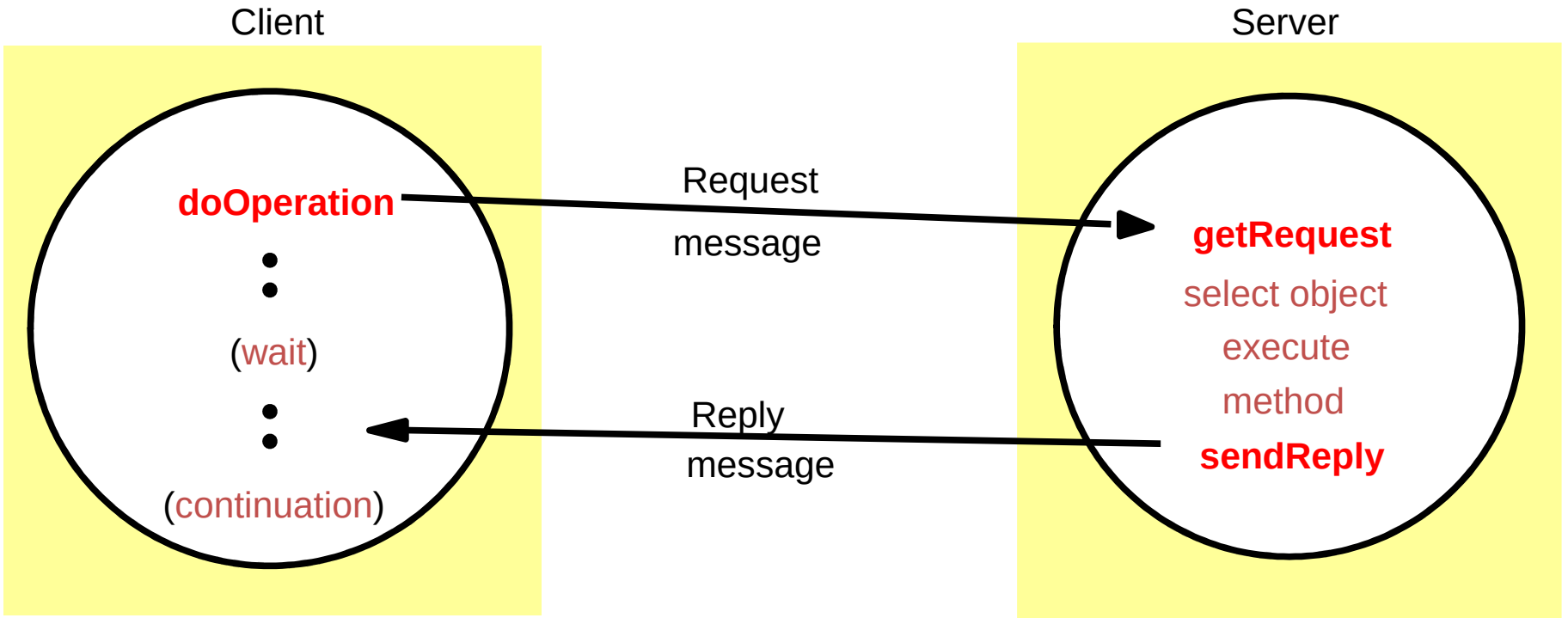
Timing

- **failure can be caused because of timing:**
- No global time
- Computer clocks
  - may have varying drift rate
  - rely on GPS radio signals (not always reliable), or synchronise via clock synchronisation algorithms
- Event ordering (message sending, arrival)
  - carry timestamps
  - may arrive in wrong order due to transmission delays (cf email)

Types of interaction

- Synchronous interaction model:
  - known upper/lower bounds on execution speeds, message transmission delays and clock drift rates
  - more difficult to build, conceptually simpler model
- Asynchronous interaction model (more common, cf Internet, more general):
  - arbitrary process execution speeds, message transmission delays and clock drift rates
  - some problems impossible to solve (e.g. agreement)
  - if solution valid for asynchronous then also valid for synchronous.

Request-Reply Communication



Java API for Datagram Comms

- Simple send/receive, with messages possibly lost/out of order
- Class *DatagramPacket*

message (=array of bytes)	message length	Internet addr	port no

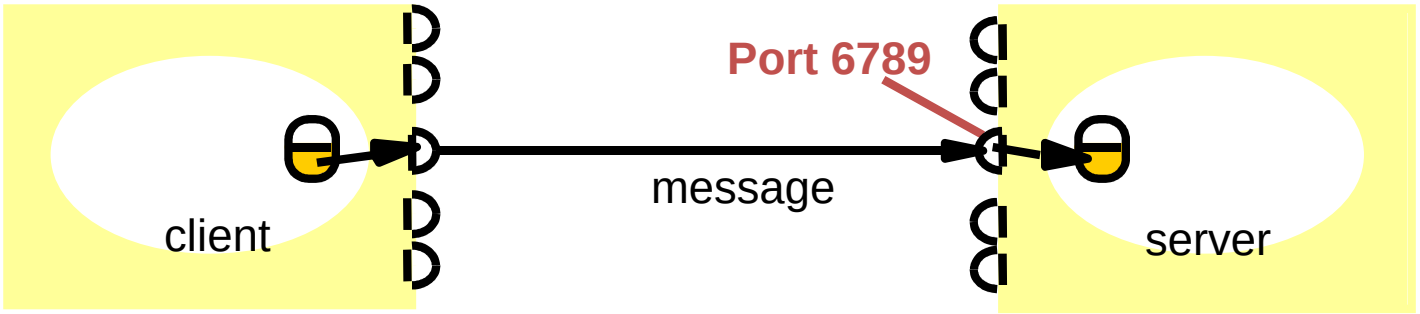
  - packets may be transmitted between sockets
  - packets truncated if too long
  - provides *getData*, *getPort*, *getAddress*

Java API for Datagram Comms

- Class *DatagramSocket*
  - *socket constructor* (returns free port if no arg)
  - *send* a *DatagramPacket*, non-blocking
  - *receive* *DatagramPacket*, blocking
  - *setSoTimeout* (receive blocks for time T and throws *InterruptedIOException*)
  - *close* DatagramSocket
  - throws *SocketException* if port unknown or in use
- See textbook site cdk3.net/ipc for complete code.

In the example...

- UDP Client
  - sends a message and gets a reply
- UDP Server
  - repeatedly receives a request and sends it back to the client



See textbook website for Java code

UDP client example

```
public class UDPClient{
public static void main(String args[]){
// args give message contents and server hostname
 DatagramSocket aSocket = null;
   try {      aSocket = new DatagramSocket();
              byte [] m = args[0].getBytes();
               InetAddress aHost = InetAddress.getByName(args[1]);
               int serverPort = 6789;
               DatagramPacket request = new
                          DatagramPacket(m,args[0].length(),aHost,serverPort);
               aSocket.send(request);
               byte[] buffer = new byte[1000];
               DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
               aSocket.receive(reply);
   }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
  }catch (IOException e){System.out.println("IO: " + e.getMessage());}
} finally {if(aSocket != null)  aSocket.close(); }
}}
```

UDP server example

For further example for TCP Client/Server and explanation of stream communication,

check course book

```
public class UDPServer{
  public static void main(String args[]){
   DatagramSocket aSocket = null;
    try{
      aSocket = new DatagramSocket(6789);
      byte[] buffer = new byte[1000];
      while(true) {
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(request);
            DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort());
            aSocket.send(reply);
                  }
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
       }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
   }finally {if(aSocket != null) aSocket.close();}
}
```

Group Communication:

- Multicast: an operation that sends a single message from one process to each of the members of a
- group of processes
- Fault tolerance based on replicated services
- Finding the discovery servers in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

IP multicast

- multicast group is specified by a class D Internet address
- membership is dynamic, to join make a socket
- programs using multicast use UDP and send datagrams to mutlicast addresses and (ordinary) port

(For example of Java code see book)