Tutorial 1 for COMP4621

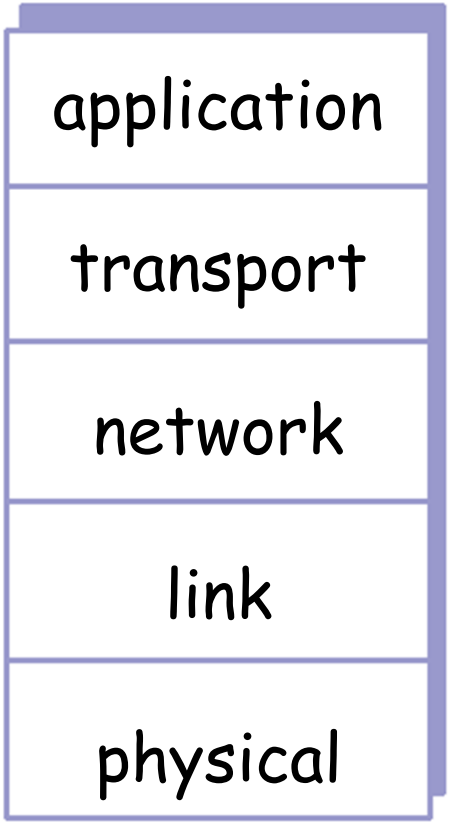
Spring 2015

Outline

- Concepts Review
- Socket Programming
- Practices
- Discussion

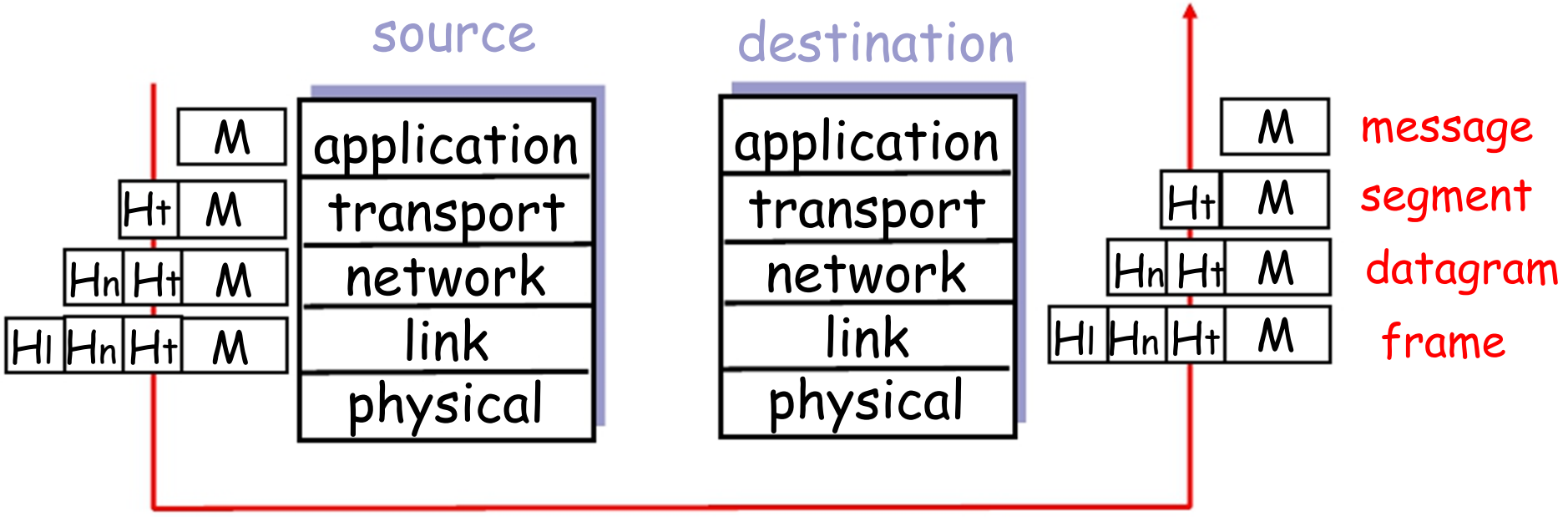
Internet Protocol Stack

- Application: supporting network app.
  - FTP, SMTP, EMAIL
- Transport: process-process
  - TCP, UDP
- Network: routing of datagrams from source to destination, host-host
  - IP, routing protocols
- Link: data transfer between neighboring network elements
  - PPP, Ethernet
- Physical: bits “on the wire”

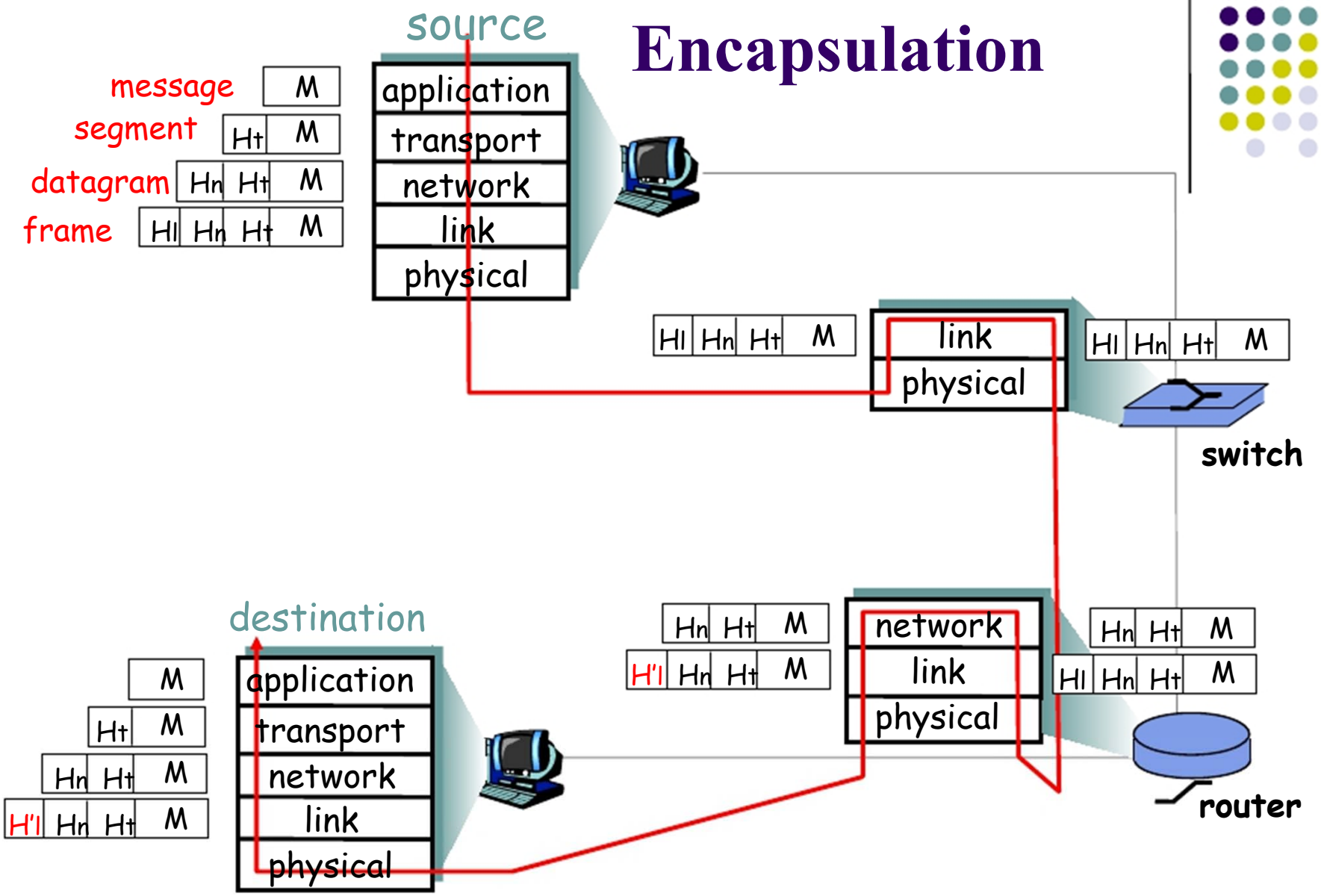


Protocol Layering and Data

- Each layer takes data from above
  - adds header information to create new data unit
  - passes new data unit to layer below



Encapsulation



What is Socket

- Just like in real life we use telephone to connect to others, in internet world, we use socket to connect to others
- We use telephone to send and receive sounds, we use socket to send and receive messages
- To connect to somebody using telephone, you need his telephone number, to connect to somebody using socket, you need his IP address and port number

Socket programming

Goal: learn how to build client/server applications that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - Unreliable datagram: UDP
  - Reliable, byte-stream oriented : TCP

socket

An *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket programming basics

- Server must be running before client can send anything to it.
- Server must have a socket (door) through which it receives and sends segments
- Similarly client needs a socket
- Socket is locally identified with a port number
  - Analogous to the apt # in a building
- Client needs to know server IP address and socket port number.

Socket programming with *UDP*

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each segment
- OS attaches IP address and port of sending socket to each segment
- Server can extract IP address, port of sender from received segment

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Note: the official terminology for a UDP packet is "datagram". In this class, we instead use "UDP segment".

UDP

- Often used for streaming multimedia apps.
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- Reliable transmission over UDP
  - add reliability at app. layer

bits	0 – 7	8 – 15	16 – 23	24 – 31
0	Source address			
32	Destination address			
64	Zeros	Protocol	UDP length	
96	Source Port		Destination Port	
128	Length		Checksum	
160+	Data			

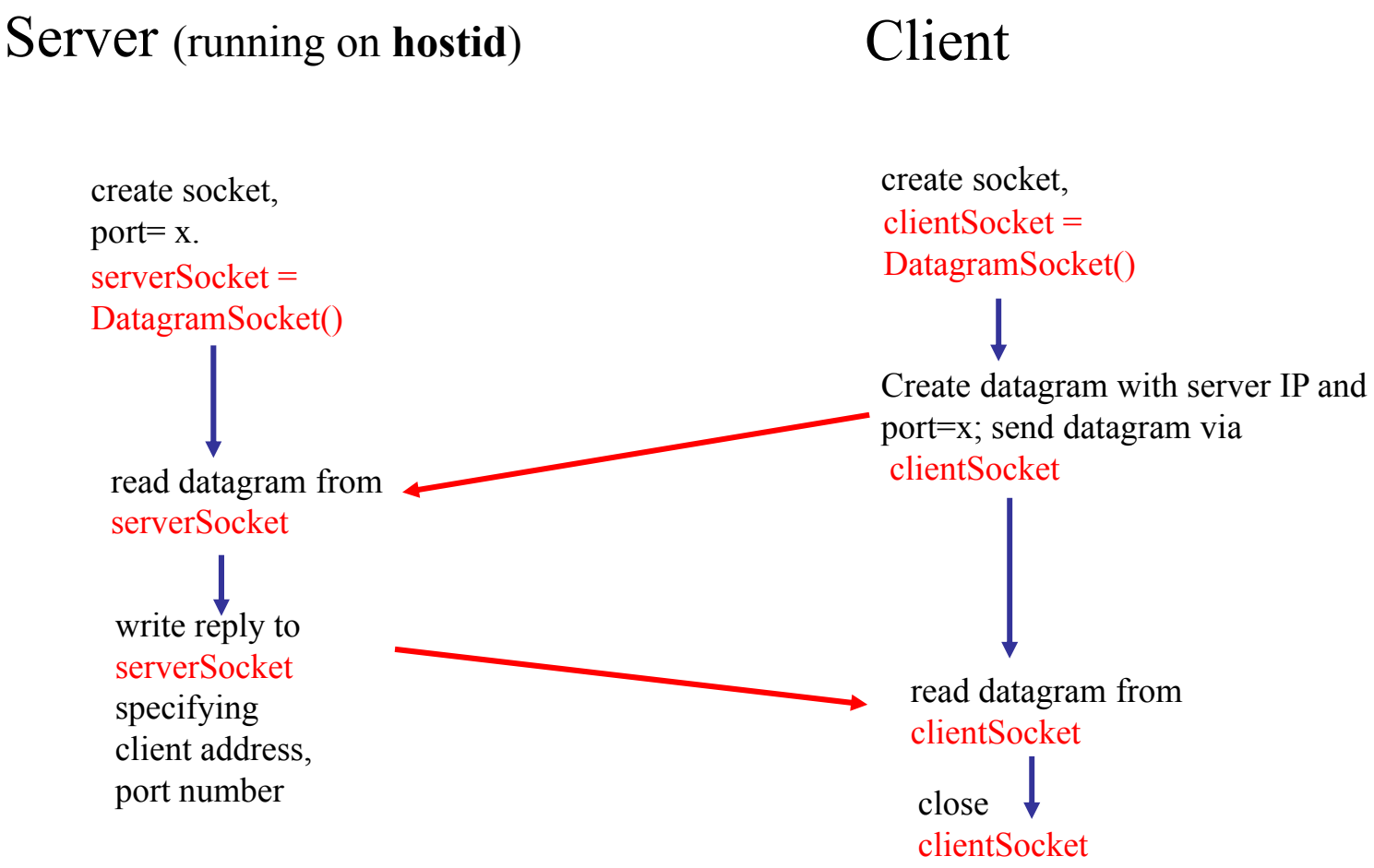
Our First Example in Java

- A simple client-server application to demonstrate socket programming for UDP
  - A client reads a line from its standard input and sends out the line through its socket to the server
  - The server reads a line from its socket
  - The server converts the line to uppercase
  - The server sends out the modified line through its socket to the client
  - The client reads the modified line from socket and prints the line on its standard output

Our First Example in Java

- Client:
  - User types line of text
  - Client program sends line to server
- Server:
  - Server receives line of text
  - Capitalizes all the letters
  - Sends modified line to client
- Client:
  - Receives line of text
  - Displays

Client/server socket interaction: UDP



Implementation: Java server (UDP)

```
import java.io.*;
import java.net.*;

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

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Implementation: Java server (UDP)

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
    }
  }
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

Implementation: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("localhost");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Implementation: Java client (UDP)

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                    new String(receivePacket.getData());

                System.out.println("FROM SERVER:" + modifiedSentence);
                clientSocket.close();
                }

            }
```

Practice 1

- Modify the UDP server. The server converts the strings to upper cases. For each string obtained by the server, the server add sits reception order at the beginning of this string; then returns the modified string back to the client.

- For example, if the server receives “hello”, it will return “1 HELLO” back to the client

Practice 1

- Add

```
...
byte[] sendData = new byte[1024];
int seq = 0;
while(true)
...
String capitalizedSentence = sentence.toUpperCase();
seq++;
capitalizedSentence = seq + " " + capitalizedSentence;
sendData = capitalizedSentence.getBytes();
...
```

Practice 2

- Modify the UDP client. Let it send the numbers from 0 to 50 to the server.
- Hint: use loop

Practice 2

- Add

```
for(int i = 0; i < 50; i++){
    ...
    String sentence = Integer.toString(i);
    ...
}
```

Question

- Can we use UDP to transmit a file?
- How?

File Download:

https://www.dropbox.com/s/9q20jndcrqpb5mw/lab1_2015week5.ppt?dl=0

https://www.dropbox.com/s/viqivyhr02hf38z/UDPClient.java?dl=0

https://www.dropbox.com/s/kwsncjk8l01mvm0/UDPSvr.java?dl=0