CS 360

Networking with Android and Java

Chadd Williams

chadd@pacificu.edu
202 Strain

Office Hours:
Tues 2:30-3:30pm
Wed 11:00-noon
Thur 1:30-2:30pm
Fri 11:00-noon

http://zeus.cs.pacificu.edu/chadd/cs360s12/

Expectations

- Java / Android / Eclipse / Subversion / Linux
  - synchronization / Threads
  - Streams
- I know very little about Android UI programming

- Read the book

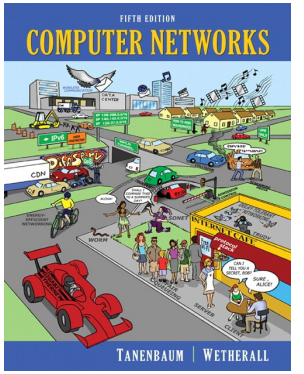
- Ask questions

Assignments

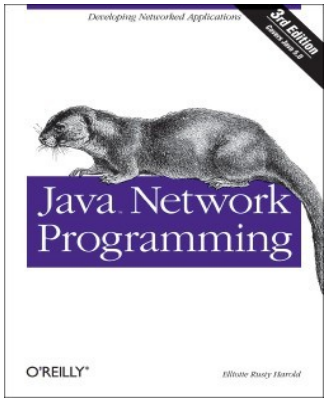
- Read the assignments
  - specifications must be followed very carefully
- Start projects early
  - imagine how hard it will be to debug two pieces of simultaneously that are talking back and forth across a network.
  - Many projects build on previous projects
- Ask questions.

Syllabus

- Computer Networks, 5th Edition, Tanenbaum
  - required
  - top-down vs bottom-up



- Java Network Programming, 3rd Edition, Harold
  - recommended (Java 1.5 or 5.0)



- http://zeus.cs.pacificu.edu/chadd/cs360s12/syllabus.html

Tentative Project Schedule

- Programming Projects
  - simple UDP client/server
  - Instant Message client library
    - will be used in clients
  - Command line Instant Message Client
  - Android Instant Message Client
    - send an image/sound/etc
  - Instant Message Server
    - multithreaded
  - CISCO Router Project

Group Project?

Labs

- In Class Labs
  - 1-2 day lab projects
  - some teams, some individual

- Possibilities
  - MyFirstUDP Client
  - Write code to query the DNS on Turing
  - Inspect packets on the network
    - Wireshark
  - Write code to send/receive HTTP traffic
  - ???

Programming

- Eclipse
- Android
- Java
- Linux
- Subversion

See attached sheet

Coding

- Network socket libraries
- Java/Android Networking API
- Java/Android Threading
- Write Java code to interact with existing C servers
  - how do you do bit twiddling in Java?

Overview

- How do we send data from here to there?
  - Internet
  - non-Internet networks
- Protocols
- Network Models
  - OSI
  - TCP/IP
  - Layers!

Client / Server Model

Terms

- Read Chapter 1
  - 1.4 is especially important
- Network Application

Examples?

- Network Protocol
  - high level?
  - low level?

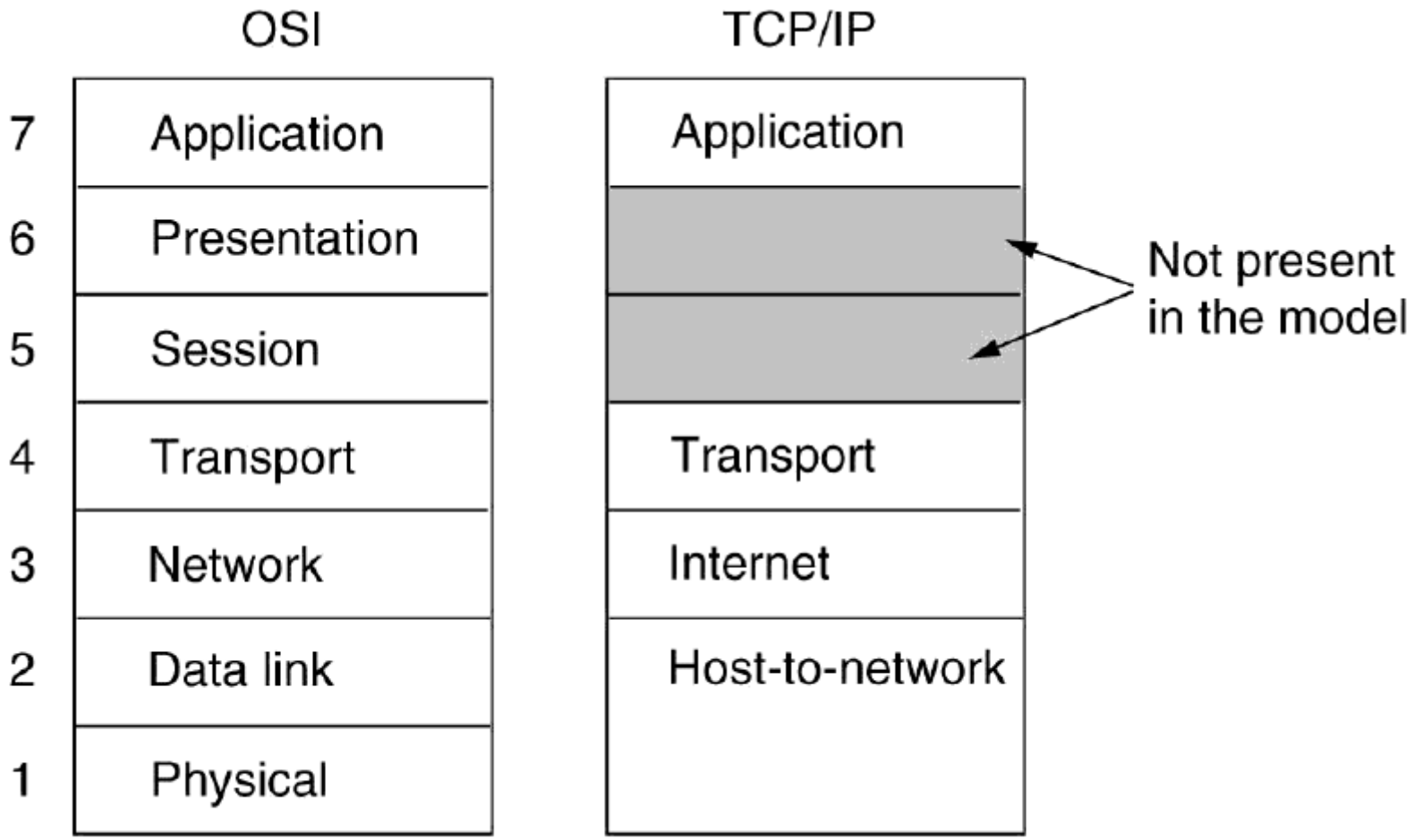
Terms

- Read Chapter 1
  - 1.4 is especially important
- Network Application

Examples?

- Packet

Network Models



Computer Networks, 4th edition, Tanenbaum, page 43.
similar image on page 46 of the 5th edition.

Why Layers?

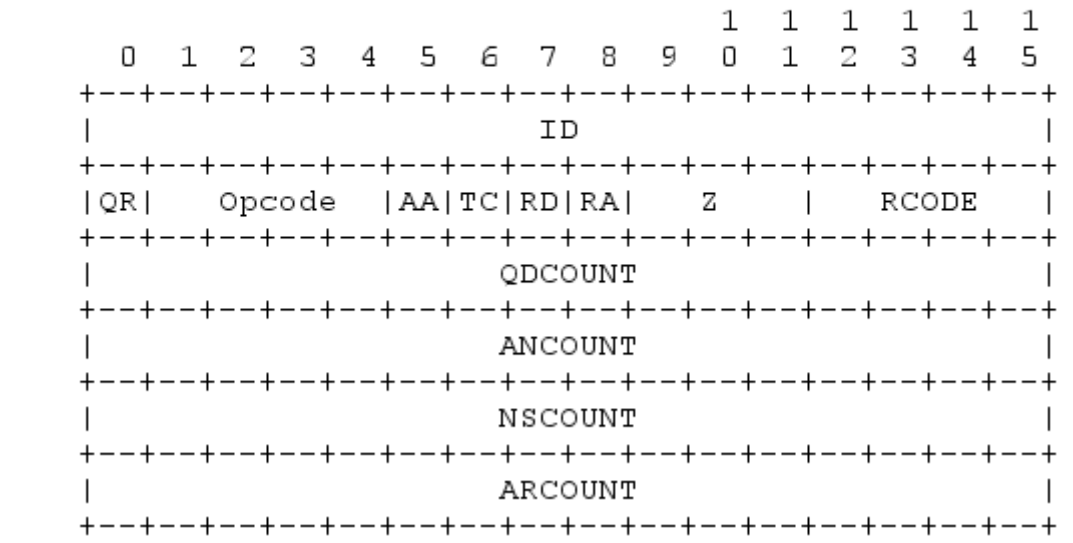
- Separation of functions
- Information Hiding
- Insulation from changes in other layers (mostly)

Protocols

- Define:
- Protocol Stack
- Open/Standardized
- Specificity

4.1.1. Header section format

The header contains the following fields:



partial DNS Packet Specification in RFC 1035

Protocols

- Proprietary Protocols
- Reverse Engineer
- Stateful/Stateless

Protocol Example: HTTP

- What network applications are involved?
  - Client
  - Server
- What needs to happen?

http://web-sniffer.net/

Each Layer

- Application
- Presentation
- Session

Cont.

- Transport
  - TCP
  - UDP
  - connection/connectionless
- Network
  - IP
- Data Link
- Physical

Networks

- Circuit vs Packet switched

- Internet Addresses
  - IPv4: X.X.X.X X is 0-255
  - IPv6: 128 bit address
    - 64 bit network prefix
    - 64 bit host address
    - 2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - Port

RFC 5952
http://tools.ietf.org/html/rfc5952

Interoperability

- Network Byte Order
  - Endianness

- Java & C
  - **`ntohs/ntohl`**
  - **`htons/htonl`**
  - s: 16 bit
  - l: 32 bit

http://www.cs.umass.edu/~verts/cs32/endian.html

Java/Android Network API

- java.net
- java.io

- java.nio
- java.nio.channels

- Socket

API Usage: Call Sequences

- UDP over IP (connectionless):

```
dSock = new DatagramSocket()
dPack = new DatagramPacket()

dSock.bind( )
dSock.send(dPack) // receive()
dSock.close()
```

server

```
dSock = new DatagramSocket()
dPack = new DatagramPacket()

dSock.send(dPack) // receive()
dSock.close()
```

client

Very similar to C

API Usage: Call Sequences

- TCP over IP (connection-based, streams):

Very different from C

```
sSock = new ServerSocket()
sSock.bind()

newSock = sSock.accept()
stream = newSock.getInputStream() / getOutputStream()
stream.read() / write()

newSock.close()
sSock.close()
```

server

```
sock = new Socket()
sock.connect()
stream = sock.getInputStream() / getOutputStream()
stream.read() / write()
sock.close()
```

client

Packets: Java vs C

- C

```
struct MathPacket{
  unsigned char PackType;
  unsigned char Version;
  unsigned short pad;
  int operandOne;
  int operandTwo;
};
```

```
|<-------------32 bits ----------->|
|--------|--------|----------------|
|PackType|Version |xxxxxxxxxxxxxxxx|
|   Operand One                    |
|   Operand Two                    |
|----------------------------------|
```

What does this struct look like in memory?

```
bytesSent = sendto(writeSocket, (char*) &packet, sizeof(struct MathPacket),
0, (struct sockaddr*) &address, addrLen);
```

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);
```

Packets: Java vs C

- Java (java.net)

```
public class MathPacket {

    private char mPackType;
    private char mVersion;
    private short mPad;

    private int mOperandOne;
    private int mOperandTwo;
```

```
|<-------------32 bits ----------->|
|--------|--------|----------------|
|PackType|Version |xxxxxxxxxxxxxxxx|
|   Operand One                    |
|   Operand Two                    |
|----------------------------------|
```

What does this class look like in memory?

DatagramPacket(byte[] data, int length, InetAddress host, int port)

Constructs a new `DatagramPacket` object to send data to the port `aPort` of the address `host`.

http://developer.android.com/reference/java/net/DatagramPacket.html

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Java: ByteBuffer

- http://developer.android.com/reference/java/nio/ByteBuffer.html
  - manage access to a byte array

- contains a series of put/get method
  - putInt()
  - getInt()
  - putDouble()
  - getDouble()
  - getChar() // TWO bytes, why?

- Can get the entire byte array when you are done
  - or **wrap** an existing byte array

java.net

- InetAddress

  IP Address

- InetSocketAddress

  IP Address: Port

Simple UDP Client

In the style of
Socket Programming

Chapter 13, *Java Network Programming*

```
InetAddress address  = InetAddress.getByName("name");

DatagramSocket clientSocket = new DatagramSocket();


sendPacket = new DatagramPacket( <see below> );


clientSocket.send(sendPacket);

clientSocket.close();
```

DatagramPacket(byte[] data, int length, InetAddress host, int port)

Constructs a new `DatagramPacket` object to send data to the port `aPort` of the address `host`.

http://developer.android.com/reference/java/net/DatagramPacket.html

Simple UDP Server

```
DatagramSocket serverSocket = new DatagramSocket();

byte[] receiveData = new byte[MAX_PACKET_SIZE];

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

serverSocket.bind(PORT);

// serverSocket.setSoTimeout(10); // only wait for 10 milliseconds for receive()
serverSocket.receive(receivePacket); // BLOCKING


// parse receiveData to get UDP Payload

System.err.println("PACKET FROM: " + receivePacket.getAddress() +" : "  +
                    receivePacket.getAddress().getCanonicalHostName()
                    + " PORT: " + receivePacket.getPort());

serverSocket.close();
```

Wireshark

- http://www.wireshark.org/
  - network protocol analyzer
  - packet sniffer

Available on Lab machines (Linux)

Sniffs Android Emulator

Not available on Zeus or Ada.

- Read/parse all the packets on a particular network interface
  - Linux: eth0, eth1, wlan0, lo

    /sbin/ifconfig To list your network interfaces
  - Windows:

    ipconfig /all To list your network interfaces

Protocols

- RFC?
- http://www.ietf.org/rfc.html
- http://datatracker.ietf.org/doc/search/

Regular Expressions

RFC 2396

```
The following line is the regular expression for breaking-down a URI
reference into its components.

   ^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
    12            3  4          5       6  7        8 9

The numbers in the second line above are only to assist readability;
they indicate the reference points for each subexpression (i.e., each
paired parenthesis).  We refer to the value matched for subexpression
<n> as $<n>.  For example, matching the above expression to

   http://www.ics.uci.edu/pub/ietf/uri/#Related

results in the following subexpression matches:

   $1 = http:
   $2 = http
   $3 = //www.ics.uci.edu
   $4 = www.ics.uci.edu
   $5 = /pub/ietf/uri/
   $6 = <undefined>
   $7 = <undefined>
   $8 = #Related
   $9 = Related
```

http://en.wikipedia.org/wiki/Regular_expression#POSIX_Basic_Regular_Expressions

For fun, look up a reg ex to validate email addresses.

Grammars

RFC 1034

```
<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [ [ <ldh-str> ] <let-dig> ]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in
upper case and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9
```

Less Simple UDP Server

Using Java New IO Channels

- Non-blocking IO

```
DatagramChannel dChannel = DatagramChannel.open(); // public static
DatagramSocket dSocket = dChannel.socket();

dChannel.configureBlocking(false);
dSocket.bind(new InetSocketAddress(PORT));

ByteBuffer buffer = ByteBuffer.allocateDirect(MAX_SIZE); // public static

SocketAddress client = channel.receive(buffer);  // returns NULL if no data
                                                 // and non-blocking
                                                 // if the packet has more than
                                                 // MAX_SIZE bytes the data is
                                                 // truncated with no
                                                 // notification!
buffer.flip();

String theData = new String(buffer.getArray());

System.out.println(theData);

buffer.clear();

dSocket.close();   // Closes this UDP datagram socket
                   // and all possibly associated channels.
```

Beware:
ByteBuffer.getChar()
ByteBuffer.get()

Socket Libraries (Unix/Linux)

```
#include <sys/types.h>  // data types
#include <sys/socket.h> // socket interface
#include <netinet/in.h> // Internet interface
```

- The *socket* is the common Unix interface to the network
  - a socket represents an end point for network communication
  - Berkeley Software Distribution socket API
    - 4.2 BSD Unix
    - most OSes now provide a BSD socket interface for networking
      - Microsoft Windows almost provides it
    - *defacto* standard
  - a socket is represented by an **`int`**

API Usage: Call Sequences

- UDP over IP (connectionless):

```
socket()
bind()
recvfrom()
close()
```

server

```
socket()
sendto()
close()
```

client

TCP over IP (connection-based):

```
socket()
bind()
listen()
accept()
recv()/send()
close()
```

server

```
socket()
connect()
send()/recv()
close()
```

client

Socket library functions

- UDP over IP
  - domain (protocol family): PF_INET
  - type: SOCK_DGRAM
  - protocol: 0 (IP)
    - see /etc/protocols for a list
  - address family: AF_INET

```
int socket(int domain, int type, int protocol)
```

```
int bind(int sockfd, const struct sockaddr  *my_addr,
   socklen_t addrlen)
```

  - actually use **`struct sockaddr_in`** for IP networking

Socket library functions

`ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)`

- again, use **`struct sockaddr_in`** for IP connections
  - this struct is filled with address information specifying the source of the data
  - this can be used to send a message back to the source.
- flags tells the function how to behave
  - OR together zero or more options:
  - MSG_WAITALL – wait until the full request is satisfied
  - MSG_PEEK – retrieve data but don't remove it from the receive queue
    - subsequent calls to recvfrom return the same data

`sendto()` is the complement of **`recvfrom`**

- **`man sendto`**

`close(int sockfd)`

- just like closing a file