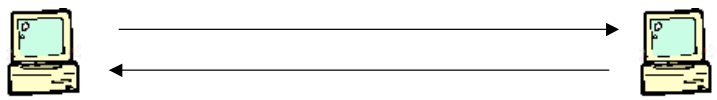
TCP/IP Sockets in Java: Practical Guide for Programmers

Kenneth L. Calvert
Michael J. Donahoo

Computer Chat

- How do we make computers talk?



- How are they interconnected?

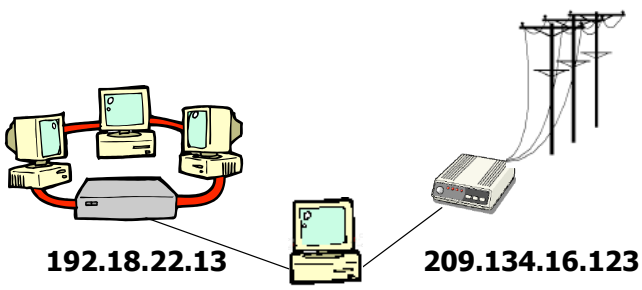
Internet Protocol (IP)

Internet Protocol (IP)

- Datagram (packet) protocol
- Best-effort service
  - Loss
  - Reordering
  - Duplication
  - Delay
- Host-to-host delivery

IP Address

- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com –> 167.208.101.28
- Identifies a host interface (not a host)



Transport Protocols

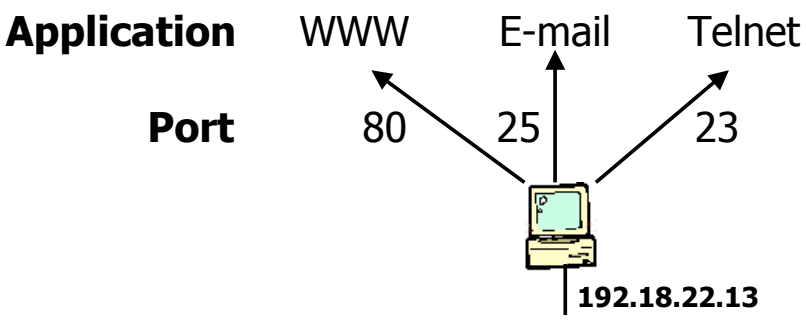
Best-effort not sufficient!

- Add services on top of IP
- User Datagram Protocol (UDP)
  - Data checksum
  - Best-effort
- Transmission Control Protocol (TCP)
  - Data checksum
  - Reliable byte-stream delivery
  - Flow and congestion control

Ports

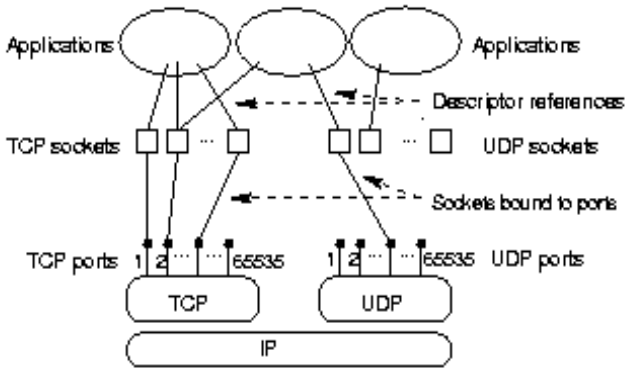
Identifying the ultimate destination

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)



Sockets

- Identified by protocol and local/remote address/port
- Applications may refer to many sockets



Clients and Servers

- Client: Initiates the connection

Client: Bob

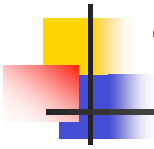
Server: Jane

"Hi. I'm Bob." →

← "Hi, Bob. I'm Jane"

"Nice to meet you, Jane." →

- Server: Passively waits to respond



TCP Client/Server Interaction

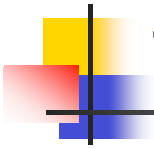
Server starts by getting ready to receive client connections…

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

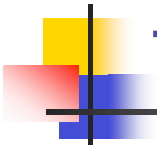
```
ServerSocket servSock = new ServerSocket(servPort);
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

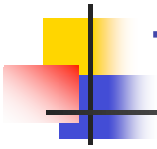
```
for (;;) {
   Socket clntSock =   servSock.accept();
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

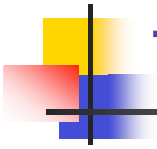
Server is now blocked waiting for connection from a client

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

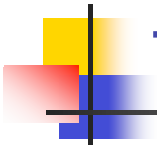
Later, a client decides to talk to the server…

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

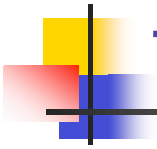
```
Socket socket = new Socket(server, servPort);
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

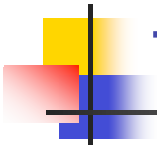
```
OutputStream out = socket.getOutputStream();
out.write(byteBuffer);
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Client/Server Interaction

```
Socket clntSock =   servSock.accept();
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection

TCP Client/Server Interaction

```
InputStream in = clntSock.getInputStream();
recvMsgSize = in.read(byteBuffer);
```

Client

- Create a TCP socket
- Communicate
- Close the connection

Server

- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection

TCP Client/Server Interaction

close(sock);

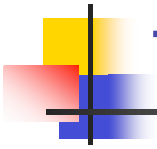
close(clntSocket)

Client

- Create a TCP socket
- Establish connection
- Communicate
- Close the connection

Server

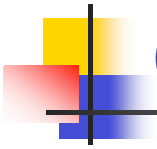
- Create a TCP socket
- Bind socket to a port
- Set socket to listen
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection



TCP Tidbits

- Client knows server address and port
- No correlation between `send()` and `recv()`

Client	Server
out.write("Hello Bob")	
	in.read() -> "Hello "
	in.read() -> "Bob"
	out.write("Hi ")
	out.write("Jane")
in.read() -> "Hi Jane"	



Closing a Connection

- close() used to delimit communication
- Analogous to EOF

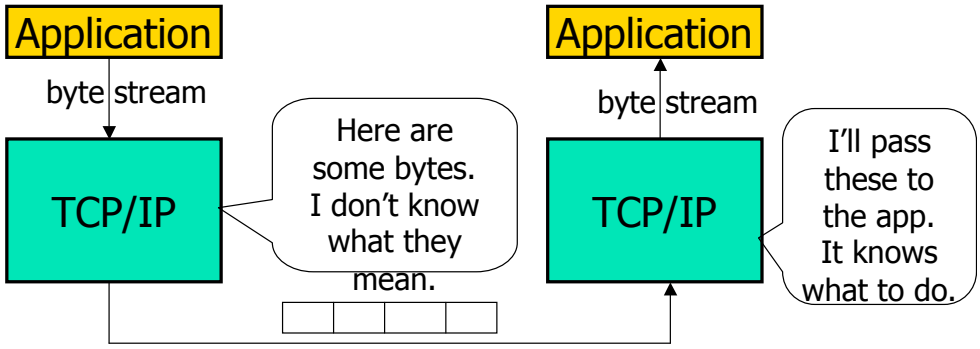
Client	Server
out.write(*string*)	
	in.read(*buffer*)
while (not received entire string)	while(client has not closed connection)
in.read(*buffer*)	out.write(*buffer*)
out.write(*buffer*)	in.read(*buffer*)
close(*socket*)	
	close(*client socket*)

Constructing Messages

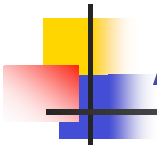
...beyond simple strings

TCP/IP Byte Transport

- TCP/IP protocols transports bytes

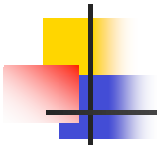


- Application protocol provides semantics



Application Protocol

- Encode information in bytes
- Sender and receiver must agree on semantics
- Data encoding
  - Primitive types: strings, integers, and etc.
  - Composed types: message with fields



Primitive Types

- String
  - Character encoding: ASCII, Unicode, UTF
  - Delimit: length vs. termination character

	0	77	0	111	0	109	0	10
	M		o		m		\n	
3	77		111		109			

Primitive Types

- Integer
  - Strings of character encoded decimal digits

49	55	57	57	56	55	48	10
'1'	'7'	'9'	'9'	'8'	'7'	'0'	\n

  - Advantage: 1. Human readable
    2. Arbitrary size
  - Disadvantage: 1. Inefficient
    2. Arithmetic manipulation

Primitive Types

- Integer
  - Native representation

Little-Endian

0	0	92	246

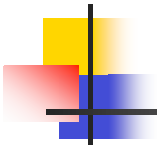
23,798

Big-Endian

246	92	0	0

4-byte two's-complement integer

  - Network byte order (Big-Endian)
    - Use for multi-byte, binary data exchange
    - htonl(), htons(), ntohl(), ntohs()



Message Composition

- Message composed of fields
  - Fixed-length fields

integer	short	short

  - Variable-length fields

M	i	k	e		1	2	\n