CO324: TCP clients and servers

Ziyan Maraikar

February 12, 2015

Lecture Outline

1. TCP Clients

2. TCP Servers

3. Framing

TCP

TCP is a *stream protocol* in which packet boundaries are invisible to the application. Data is received and transmitted as a sequence of bytes.

It provides

1. Reliability
2. Ordering
3. Flow control

TCP clients

The Java Socket class represents a TCP socket.

```
Socket socket = new Socket();

InputStream sin = socket.getInputStream();
OutputStream sout = socket.getOutputStream();
```

Binary data is read and written to the socket via the associated I/O streams.

Sending and receiving text

```
try (Socket socket = new Socket(address, TCPServer.PORT);
  Scanner sin = new Scanner(socket.getInputStream() );
  PrintStream sout = new PrintStream(
      socket.getOutputStream() ) ) {

    sout.println("client says hello to server "+address);
    System.out.println(sin.nextLine());
}
```

Note that application messages must be properly *framed* e.g. using a delimiter like `\n`.

What do the `Scanner` and `PrintStream` classes do?

Lecture Outline

1. TCP Clients
2. TCP Servers
3. Framing

TCP Servers

Java uses a separate ServerSocket class to bind to a port and accept connections from clients.

```
ServerSocket ss = new ServerSocket(PORT);
Socket socket = ss.accept();
```

accept returns a new socket connected to the client.

Server example

```
try( ServerSocket ss = new ServerSocket(PORT)) {
  while (true)
    try (Socket socket = ss.accept();
      Scanner sin = new Scanner(socket.getInputStream() );
      PrintStream sout = new PrintStream(
          socket.getOutputStream()) )  {

        sout.println("server says world");
        System.out.println(sin.nextLine());
    }
}
```

What happens if multiple clients try to connect at once?

Lecture Outline

1. TCP Clients
2. TCP Servers
3. Framing

Messages on streams

Suppose a client sends two consecutive messages, and the server does a `read`. Will it get

1. both messages at once?
2. the first message only?
3. part of the first message?

It depends on network conditions and the TCP/IP stacks!

Framing

TCP can only send to and receive from a byte stream, but application protocols are built with discrete messages.

We must define a method of *framing* application messages, so that message boundaries are unambiguous.

Method used depends on the kind of data

- Text
- Binary

Text protocols

Most application protocols on the Internet are textual.

- ★ Human readable — so easy to debug.
- ★ Historically, the most applications were textual e.g., Telnet, Email

Disadvantages:

- ★ Vulnerable to security attacks like buffer overflows.
- ★ Bandwidth and CPU inefficient.

Example: SMTP

```
S: 220 smtp.server.com Simple Mail Transfer Servi
C: HELO client.example.com
S: 250 Hello client.example.com
C: MAIL FROM:<jane@yahoo.com>
S: 250 OK
C: RCPT TO:<john@gmail.com>
S: 250 OK
C: DATA
S: 354 Send message content; end with <CRLF>.<CRL
C: <The message data (body text, subject, e-mail
C: .
S: 250 OK, message accepted for delivery: queued
C: QUIT
S: 221 Bye
```

How are non-text mail attachments handled?

Delimiters

We can *delimit* text protocol messages using a special character. The usual delimiter used in Internet protocols are the line termination characters `CR`, `LF` or `CRLF`.

```
Socket socket = new Socket(address, PORT);

BufferedReader sin = new BufferedReader (
  new InputStreamReader(socket.getInputStream() ));

BufferedWriter sout = new BufferedWriter (
  new OutputStreamWriter(socket.getOutputStream() ));

sout.write("hello world\n");
sin.readLine();
```

`BufferedReader.readline` splits the apart newline delimited messages in a stream.

Binary protocols

Binary protocols support transmission of arbitrary data. Usually contains a fixed-format *header* that describes the *payload*.

- Suited to describing structured data.
- Easier to *parse* — metadata received before payload.
- Efficient use of bandwidth.

Example: Basic encoding rules for ASN.1, an OSI standard used in protocols such as LDAP.

Type	Length	Value	End-of-content