Objectives

- How the HTTP protocol works
- The SSL security extension from a programmer’s point of view
- How to write servers and clients in Java

HTTP

- HTTP: HyperText Transfer Protocol
- Client-Server model
- Request-Response pattern

Network Layers

- OUR APPLICATIONS
  - HTTP, FTP, SMTP, DNS
- THE TRANSPORT LAYER
  - TCP, UDP
- THE INTERNET LAYER
  - IP
- THE NETWORK INTERFACE LAYER
  - Ethernet
An Introduction to XML and Web Technologies

**IP**

- **IP:** Internet Protocol
- **Unreliable** communication of **limited size data packets** (datagrams)
- **IP addresses** (e.g. 165.193.130.107) identify machines
- Handles **routing** using the underlying physical network (e.g. Ethernet)

**TCP**

- **TCP:** Transmission Control Protocol
- Layer on top of IP
- Data is transmitted in **streams**
- **Reliability** ensured by retransmitting lost datagrams, reordering, etc.
- **Connection-oriented**
  - establish connection between client and server
  - data streaming in **both directions**
  - close connection
- **Socket:** end point of connection, associated a pair of (IP address, **port number**)

**HTTP**

- **HTTP:** HyperText Transfer Protocol
- Layer on top of TCP
- Request and response sent using TCP streams

**HTTP Requests**

```
GET /search?q=Introduction+to+XML+and+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.2)
  Gecko/20040803
Accept: text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: da,en-us;q=0.8,en;q=0.5,sw;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/
```

- **Request line** (methods: GET, POST, ...)
- **Header lines**
- **Request body** (empty here)
HTTP Responses

- Status line
- Header lines
- Response body

Status Codes

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 503 Service Unavailable
- ...

Encoding of Form Data

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bet</td>
<td>other_country</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:zacharias_doe@notmail.com">zacharias_doe@notmail.com</a></td>
</tr>
<tr>
<td>send</td>
<td>Go!</td>
</tr>
</tbody>
</table>

- Encoding to query string (URL encoding):
  - GET: place query string in request URI
    - http://.../soccer.jsp?bet=other+country&email=zacharias_doe@notmail.com&send=Go!
  - POST: place query string in request body
GET vs. POST?

- The client should not be held responsible for the consequences of a GET request
  - useful for retrieving data, not for submitting orders to an online shop
- Limits on request URI length
- POST allows other encodings (e.g. for file upload)
- Cachability

Authentication

- Restricting access to authorized users
  
  Common techniques:
  - IP-address
  - Form (with username/password fields)
  - HTTP Basic
  - HTTP Digest

HTTP Basic Authentication

- Challenge:
  HTTP/1.1 401 Authorization Required
  WWW-Authenticate: Basic realm="The Doe Family Site"

- Response:
  Authorization: Basic emFjaGFyWzOmmFwGxlG1lCg==

Advanced Features in HTTP

- Cache control
- Range requests
- Persistent connections, pipelining
- ...
**Cache Control**

- Caches used in clients, servers, and network (proxy servers, content delivery networks)

**Cache-Control:**
- no-store: never cache this message
- no-cache: may cache but need revalidation
- public: may cache
- private: intended for single user
- max-age: set expiration
- must-revalidate: require revalidation

**HTTP/1.0:**
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache

**Range Requests**

- Range: bytes=387-
- 206 Partial Content

**Persistent Connections**

- Multiple request-response pairs on a single TCP connection
  - Content-Length: now important!
  - Connection: close: persistent by default in HTTP/1.1
  - Connection: keep-alive: compatibility
  - Keep-Alive: 300: control timeout, compatibility

- Pipelining
  - send multiple requests before receiving the responses
  - fewer TCP/IP packets
  - only for idempotent requests (e.g. GET)
  - supported by newer browsers

**Limitations of HTTP**

- **Stateless**, no built-in support for tracking clients (session management)
- No built-in **security** mechanisms
### Session Management

Techniques
- URL rewriting
- Hidden form fields
- Cookies
- SSL sessions

### Cookies

- Extension of HTTP that allows servers to **store data on the clients**
  - limited size and number
  - may be disabled by the client

  - Set-Cookie: sessionid=21A9A8089C305319; path=/
  - Cookie: sessionid=21A9A8089C305319

### Security

Desirable properties:
- confidentiality
- integrity
- authenticity
- non-repudiation

### SSL

- **SSL**: Secure Sockets Layer
- **TLS**: Transport Layer Security (newer version)

- Layer between HTTP and TCP, accessed by https://...

- Based on public-key cryptography
  - private key + public key
  - certificate (usually for server authentication only)
An Introduction to XML and Web Technologies

Web Programming with Java

Why Java?
- platform independence
- safe runtime model
- multi-threading
- sandboxing
- Unicode
- serialization, dynamic class loading
- powerful standard libraries
  - java.net
  - java.nio.channels
  - javax.net.ssl

TCP/IP: DomainName2IPNumbers

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

public class DomainName2IPNumbers {
    public static void main(String[] args) {
        try {
            InetAddress[] a = InetAddress.getAllByName(args[0]);
            for (int i = 0; i < a.length; i++)
                System.out.println(a[i].getHostAddress());
        } catch (UnknownHostException e) {
            System.out.println("Unknown host!");
        }
    }
}
```

```
java DomainName2IPNumbers www.google.com
java DomainName2IPNumbers www.google.com
66.102.9.104
66.102.9.99
```

TCP/IP: SimpleServer (1/2)

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

public class SimpleServer {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));
            while (true) {
                Socket con = ss.accept();
                InputStreamReader in = new InputStreamReader(con.getInputStream());
                StringBuffer msg = new StringBuffer();
                int c;
                while ((c = in.read()) != 0)
                    msg.append((char)c);
                PrintWriter out = new PrintWriter(con.getOutputStream());
                out.print("Simon says: " + msg);
                out.flush();
                con.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
java SimpleServer 66.102.9.104
```

TCP/IP: SimpleServer (2/2)
TCP/IP: SimpleClient (1/2)

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

public class SimpleClient {
    public static void main(String[] args) {
        try {
            Socket con = new Socket(args[0], Integer.parseInt(args[1]));
            PrintStream out = new PrintStream(con.getOutputStream());
            out.print(args[2]);
            out.write(0);
            out.flush();

            InputStreamReader in = new InputStreamReader(con.getInputStream());
            int c;
            while ((c = in.read()) != -1)
                System.out.print((char)c);
            con.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

TCP/IP: SimpleClient (2/2)

```java
java SimpleServer 1234
java SimpleClient localhost 1234 "Hello World"
Simon says: Hello World
```

Non-Blocking I/O

- Support for **concurrent connections** and **buffering**
- Packages: java.nio.channels, java.nio
- Central classes:
  - ServerSocketChannel, SocketChannel
  - Selector
  - ByteBuffer
- See example in the book...

HTTP in Java

Two approaches:
1. Use the TCP/IP features in Java “manually”
2. Use the HTTP features
HTTP: ImFeelingLucky2 (1/2)

```java
import java.net.*;
import java.io.*;
public class ImFeelingLucky2 {
    public static void main(String[] args) {
        try {
            String req = "http://www.google.com/search?q=" + URLEncoder.encode(args[0], "UTF8") + "&btnI=" + URLEncoder.encode("I'm Feeling Lucky", "UTF8");
            HttpURLConnection con = (HttpURLConnection) (new URL(req)).openConnection();
            con.setRequestProperty("User-Agent", "IXWT");
            con.setInstanceFollowRedirects(false);
            String loc = con.getHeaderField("Location");
            System.out.print("The prophet spoke thus: ");
            if (loc!=null)
                System.out.println("Direct your browser to "+loc+
                            " and you shall find great happiness in life.");
            else
                System.out.println("I am sorry - my crystal ball is blank.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

SSL in Java (JSSE)

- `javax.net.ssl`, `java.security.cert`
- `SSLServerSocketFactory`, `SSLSocket`
- `SSLSession`, `Certificate`, `HttpsURLConnection`
- `keytool`
- `java -Djavax.net.ssl.trustStore=...`  
  `-Djavax.net.ssl.trustStorePassword=...` ...

- See example in the book...

A Web Server in 145 Lines of Code

- Listens for HTTP requests on a port
- Parses the requests
- Returns files from the server's file system

[ DEMO ]

- Source code in the book...
## Summary

- Communication protocols:
  - IP
  - TCP
  - HTTP
  - SSL

- Programming Web servers and clients with Java

## Essential Online Resources

- HTTP/1.1:
  [http://www.w3.org/Protocols/rfc2616/rfc2616.html](http://www.w3.org/Protocols/rfc2616/rfc2616.html)

- Java API (java.net and others):
  [http://java.sun.com/j2se/1.5.0/docs/api/](http://java.sun.com/j2se/1.5.0/docs/api/)