TCP/IP
Transmission Control Protocol / Internet Protocol

Based on Notes by D. Hollinger
Topics

• IPv6

• TCP

• Java TCP Programming
IPv6 availability

• Generally available with (new) versions of most operating systems.
  • BSD, Linux 2.2 Solaris 8
• An option with Windows 2000/NT
• Most routers can support IPV6
• Supported in J2SDK/JRE 1.4
IPv6 Design Issues

• Overcome IPv4 scaling problem
  • lack of address space.
• Flexible transition mechanism.
• New routing capabilities.
• Quality of service.
• Security.
• Ability to add features in the future.
IPv6 Headers

- Simpler header - faster processing by routers.
  - No optional fields - fixed size (40 bytes)
  - No fragmentation fields.
  - No checksum
- Support for multiple headers
  - more flexible than simple “protocol” field.
### IPv4 Header

![IPv4 Header Diagram](image)

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (VERS)</td>
<td>4</td>
</tr>
<tr>
<td>Header Length (HL)</td>
<td>4</td>
</tr>
<tr>
<td>Service</td>
<td>1</td>
</tr>
<tr>
<td>Datagram ID</td>
<td>4</td>
</tr>
<tr>
<td>FLAG</td>
<td>4</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>4</td>
</tr>
<tr>
<td>Protocol</td>
<td>4</td>
</tr>
<tr>
<td>Header Checksum</td>
<td>4</td>
</tr>
<tr>
<td>Source Address</td>
<td>4</td>
</tr>
<tr>
<td>Destination Address</td>
<td>4</td>
</tr>
<tr>
<td>Options (if any)</td>
<td>1-4</td>
</tr>
<tr>
<td>Data</td>
<td>Variable</td>
</tr>
</tbody>
</table>

4 for IPv4
### IPv6 Header

<table>
<thead>
<tr>
<th>VERS</th>
<th>PRIO</th>
<th>Flow Label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Payload Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Next Header</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hop Limit</td>
</tr>
</tbody>
</table>

- **VERS**: version number (6 for IPv6)
- **PRIO**: priority
- **Flow Label**: 1 byte
- **Payload Length**: 1 byte
- **Next Header**: 1 byte
- **Hop Limit**: 1 byte

- **Source Address (128 bits - 16 bytes)**

- **Dest. Address (128 bits - 16 bytes)**
IPv6 Header Fields

• VERS: 6 (IP version number)
• Priority: will be used in congestion control
• Flow Label: experimental - sender can label a sequence of packets as being in the same flow.
• Payload Length: number of bytes in everything following the 40 byte header (up to 64 Kb), or 0 for a Jumbogram (up to 4 Gb).
IPv6 Header Fields

• Next Header is similar to the IPv4 “protocol” field - indicates what type of header follows the IPv6 header.

• Hop Limit is similar to the IPv4 TTL field (but now it really means hops, not time).
Extension Headers

- Routing Header - source routing
- Fragmentation Header - supports fragmentation of IPv6 datagrams.
- Authentication Header
- Encapsulating Security Payload Header
IPv6 Addresses

• 128 bits - written as eight 16-bit hex numbers.
  5f1b:df00:ce3e:e200:0020:0800:2078:e3e3

• High order bits determine the type of address.
IPv6
Aggregate Global Unicast Address

<table>
<thead>
<tr>
<th>3</th>
<th>13</th>
<th>32</th>
<th>16</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>TLA ID</td>
<td>NLA ID</td>
<td>SLA ID</td>
<td>Interface ID</td>
</tr>
</tbody>
</table>

TLA: top-level aggregation (provider)
NLA: next-level (subscriber)
SLA: site-level (subnet)

Interface ID is (typically) based on hardware MAC address
IPv4-Mapped IPv6 Address

- IPv4-Mapped addresses allow a host that support both IPv4 and IPv6 to communicate with a host that supports only IPv4.

- The IPv6 address is based completely on the IPv4 address.
## IPv4-Mapped IPv6 Address

- 80 bits of 0s followed by 16 bits of ones, followed by a 32 bit IPv4 Address:

<table>
<thead>
<tr>
<th>0000 ... 0000</th>
<th>FFFF</th>
<th>IPv4 Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 bits</td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
</tbody>
</table>
Works with DNS

- An IPv6 application asks DNS for the address of a host, but the host only has an IPv4 address.
- DNS creates the IPv4-Mapped IPv6 address automatically.
- Kernel understands this is a special address and really uses IPv4 communication.
IPv4-Compatible IPv6 Address

• An IPv4 compatible address allows a host supporting IPv6 to talk IPv6 even if the local router(s) don’t talk IPv6.

• IPv4 compatible addresses tell endpoint software to create a tunnel by encapsulating the IPv6 packet in an IPv4 packet.
**IPv4-Compatible IPv6 Address**

- 80 bits of 0s followed by 16 bits of 0s, followed by a 32 bit IPv4 Address:

<table>
<thead>
<tr>
<th>80 bits</th>
<th>16 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 ... 0000</td>
<td>0000</td>
<td>IPv4 Address</td>
</tr>
</tbody>
</table>

**IPv4 Address**

Netprog 2002  TCP/IP
Tunneling
(done automatically by kernel when IPv4-Compatible IPv6 addresses used)

IPv6 Host       IPv4 Routers       IPv6 Host
IPv4 Datagram   IPv6 Datagram

Netprog 2002  TCP/IP
IPv6 in Java 1.4

• Inet6Address class
  • :: corresponds to 0.0.0.0 (unspecified) in IPv4
  • ::1 corresponds to 127.0.0.1 (loopback) in IPv4
  • ::ffff:w.x.y.z IPv4-mapped address
  • ::w.x.y.z to tunnel IPv6 packets over IPv4 routing

• For details, see:
  http://java.sun.com/j2se/1.4/docs/guide/net/ipv6_guide/
TCP
*Transmission Control Protocol*

- TCP is an alternative transport layer protocol over IP.
- TCP provides:
  - Connection-oriented
  - Reliable
  - Full-duplex
  - Byte-Stream
Connection-Oriented

• *Connection oriented* means that a virtual connection is established before any user data is transferred.

• If the connection cannot be established - the user program is notified.

• If the connection is ever interrupted - the user program(s) is notified.
Reliable

• *Reliable* means that every transmission of data is acknowledged by the receiver.

• If the sender does not receive acknowledgement within a specified amount of time, the sender retransmits the data.
Byte Stream

- *Stream* means that the connection is treated as a stream of bytes.

- The user application does not need to package data in individual datagrams (as with UDP).
Buffering

• TCP is responsible for buffering data and determining when it is time to send a datagram.

• It is possible for an application to tell TCP to send the data it has buffered without waiting for a buffer to fill up.
Full Duplex

- TCP provides transfer in both directions.

- To the application program these appear as 2 unrelated data streams, although TCP can piggyback control and data communication by providing control information (such as an ACK) along with user data.
TCP Ports

• Interprocess communication via TCP is achieved with the use of ports (just like UDP).

• UDP ports have no relation to TCP ports (different name spaces).
TCP Segments

• The chunk of data that TCP asks IP to deliver is called a **TCP segment**.

• Each segment contains:
  • data bytes from the byte stream
  • control information that identifies the data bytes
## TCP Segment Format

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment Number</td>
</tr>
<tr>
<td>offset</td>
<td>Reser.</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP Lingo

• When a client requests a connection it sends a “SYN” segment (a special TCP segment) to the server port.
• SYN stands for synchronize. The SYN message includes the client’s ISN.
• ISN is Initial Sequence Number.
• Every TCP segment includes a **Sequence Number** that refers to the first byte of *data* included in the segment.
• Every TCP segment includes an **Acknowledgement Number** that indicates the byte number of the next data that is expected to be received.
  • All bytes up through this number have already been received.
And more...

- There are a bunch of control flags:
  - URG: urgent data included.
  - ACK: this segment is (among other things) an acknowledgement.
  - RST: error – connection must be reset.
  - SYN: synchronize Sequence Numbers (setup)
  - FIN: polite connection termination.
And more...

- MSS: Maximum segment size (A TCP option)
- Window: Every ACK includes a Window field that tells the sender how many bytes it can send before the receiver will have to toss it away (due to fixed buffer size).
TCP Connection Creation

• Programming details later - for now we are concerned with the actual communication.

• A server accepts a connection.
  • Must be looking for new connections!

• A client requests a connection.
  • Must know where the server is!
Client Starts

- A client starts by sending a SYN segment with the following information:
  - Client’s ISN (generated pseudo-randomly)
  - Maximum Receive Window for client.
  - Optionally (but usually) MSS (largest datagram accepted).
  - No payload! (Only TCP headers)
Server Response

- When a waiting server sees a new connection request, the server sends back a SYN segment with:
  - Server’s ISN (generated pseudo-randomly)
  - Request Number is Client ISN+1
  - Maximum Receive Window for server.
  - Optionally (but usually) MSS
  - No payload! (Only TCP headers)
Finally

• When the Server’s SYN is received, the client sends back an ACK with:
  • Acknowledgment Number is Server’s ISN+1
Client

1. SYN
   ISN=\text{x}

2. SYN
   ISN=\text{y}
   ACK=\text{x+1}

3. ACK=\text{y+1}

Server

Netprog 2002  TCP/IP
TCP 3-way handshake

1. Client: “I want to talk, and I’m starting with byte number $X$”.

2. Server: “OK, I’m here and I’ll talk. My first byte will be called number $Y$, and I know your first byte will be number $X+1$”.

3. Client: “Got it - you start at byte number $Y+1$”.

? Bill: “Monica, I’m afraid I’ll syn and byte your ack”
Why 3-Way?

- Why is the third message necessary?

HINTS:
- TCP is a reliable service.
- IP delivers each TCP segment.
- IP is not reliable.
TCP Data and ACK

- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each segment (data or empty) includes a request number indicating what data has been received.
Buffering

• Keep in mind that TCP is part of the Operating System. The O.S. takes care of all these details asynchronously.

• The TCP layer doesn’t know when the application will ask for any received data.

• TCP buffers incoming data so it’s ready when we ask for it.
TCP Buffers

• Both the client and server allocate buffers to hold incoming and outgoing data
  • The TCP layer does this.

• Both the client and server announce with every ACK how much buffer space remains (the Window field in a TCP segment).
Send Buffers

• The application gives the TCP layer some data to send.
• The data is put in a send buffer, where it stays until the data is ACK’d.
• The TCP layer won’t accept data from the application unless (or until) there is buffer space.
ACKs

• A receiver doesn’t have to ACK every segment (it can ACK many segments with a single ACK segment).
• Each ACK can also contain outgoing data (piggybacking).
• If a sender doesn’t get an ACK after some time limit, it resends the data.
TCP Segment Order

- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP is not reliable - IP datagrams can be lost or arrive out of order.
Termination

• The TCP layer can send a RST segment that terminates a connection if something is wrong.
• Usually the application tells TCP to terminate the connection politely with a FIN segment.
FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK’d.
- The other end must now send a FIN.
- That FIN must be ACK’d.
Netprog 2002  TCP/IP

App1

1. FIN
   SN=X

2. ACK=X+1

3. FIN
   SN=Y

4. ACK=Y+1

App2

Netprog 2002  TCP/IP
TCP Termination

1. App1: “I have no more data for you”.

2. App2: “OK, I understand you are done sending.”
   
   dramatic pause…

3. App2: “OK - Now I’m also done sending data”.

4. App1: “Roger, Over and Out, Goodbye, Hastalavista Baby, Adios, It’s been real …”
   
   camera fades to black …
TCP TIME_WAIT

- Once a TCP connection has been terminated (the last ACK sent) there is some unfinished business:
  - What if the ACK is lost? The last FIN will be resent and it must be ACK’d.
  - What if there are lost or duplicated segments that finally reach the destination after a long delay?
- TCP hangs out for a while to handle these situations.
TCP Sockets Programming

- Creating a *passive mode* (server) socket.
- Establishing an application-level *connection*.
- Sending/receiving data.
- Terminating a connection.
Establishing a passive mode TCP socket

Passive mode:
- Address already determined.
- Tell the kernel to accept incoming connection requests directed at the socket address.
  - 3-way handshake
- Tell the kernel to queue incoming connections for us.
Accepting an incoming connection.

- Once we start listening on a socket, the O.S. will queue incoming connections
  - Handles the 3-way handshake
  - Queues up multiple connections.

- When our application is ready to handle a new connection, we need to ask the O.S. for the next connection.
Terminating a TCP connection

• Either end of the connection can call the `close()` system call.
• If the other end has closed the connection, and there is no buffered data, reading from a TCP socket returns 0 to indicate EOF.
Client Code

- TCP clients can connect to a server, which:
  - takes care of establishing an endpoint address for the client socket.
    - don’t need to call bind first, the O.S. will take care of assigning the local endpoint address (TCP port number, IP address).
  - Attempts to establish a connection to the specified server.
    - 3-way handshake
Reading from a TCP socket

- By default `read()` will block until data is available.
- Reading from a TCP socket may return less than max bytes (whatever is available).
- You must be prepared to read data 1 byte at a time!
Writing to a TCP socket

• write might not be able to write all bytes (on a nonblocking socket).
Metaphor for Good Relationships

To succeed in relationships:

- you need to establish your own identity.
- you need to be open & accepting.
- you need to establish contacts.
- you need to take things as they come, not as you expect them.
- you need to handle problems as they arise.

Netprog 2002  TCP/IP
Java Sockets Programming

- The package java.net provides support for sockets programming.
- Typically you import everything defined in this package with:

  ```java
  import java.net.*;
  ```
Classes

InetAddress
Socket
ServerSocket
DatagramSocket
DatagramPacket
UDP
Socket class

• Corresponds to active TCP sockets only!
  • client sockets
  • socket returned by accept();

• Passive sockets are supported by a different class: ServerSocket
Socket Constructors

• Constructor creates a TCP connection to a named TCP server.
  • There are a number of constructors:

    Socket(InetAddress server, int port);

    Socket(InetAddress server, int port,
    InetAddress local, int localport);

    Socket(String hostname, int port);
Socket Methods

void close();
InetAddress getInetAddress(); \textit{getpeername}
InetAddress getLocalAddress(); \textit{getsockname}
InputStream getInputStream();
OutputStream getOutputStream();

• Lots more (setting/getting socket options, partial close, etc.)
Socket I/O

• Socket I/O is based on the Java I/O support (in the package `java.io`).

• `InputStream` and `OutputStream` are abstract classes
  • common operations defined for all kinds of `InputStreams`, `OutputStreams`…
ServerSocket Class
(TCP Passive Socket)

• Constructors:

ServerSocket(int port);

ServerSocket(int port, int backlog);

ServerSocket(int port, int backlog, InetAddress bindAddr);
ServerSocket Methods

Socket accept();

void close();

InetAddress getInetAddress();

int getLocalPort();

throw IOException, SecurityException
Sample Echo Server

TCPEchoServer.java, EchoClient.java, GenericClient.java

Simple TCP Echo server.

Based on code from:
  TCP/IP Sockets in Java, Java Online Tutorial