Lecture 3
Networking Fundamentals (cont’d)
Sockets API

Sources:

Computer Networks (Andrew S. Tanenbaum)
UNIX Network Programming, Volume 1 (W. Richard Stevens)
Internet RFCs, …
Announcements

• Lab 1 handed out today
  – Due back September 23rd
  – No extensions
  – Use the mailing list for questions/clarifications

• E-mail about MSDNAA accounts should be sent to comment@cs
  – I will not be able to help you

• Have requested ITS to enable your i5 accounts
  – I was not aware that these accounts are enabled only when requested
(Review) The OSI Model in the Internet

OSI

Application
Presentation
Session
Transport
Network
Data Link
Physical

Internet

Application

Transport

Network

Physical + Data Link

Protocols

TELNET, FTP, SMTP, DNS, HTTP, ...

TCP (Trans. Control P.), UDP (User Datagram P.)

IP (Internet Protocol), ICMP, ARP, RARP

Ethernet LAN (802.3), Wireless LAN (802.11) Packet Radio (GPRS)

Networks
(Review) Transport and Higher-Level Layers

• Last lecture: Discussed sending an IP datagram
  – Standalone packet

• Applications require higher-level abstractions
  – Services: Some way of identifying different programs on the recipient host that will deal with the packet
    • Addressing/naming handled using port numbers
      – Networking code responsible for demultiplexing
    • Realized as the User Datagram Protocol (UDP)
  – Connections
    • A continuous stream of packets
    • In-order, exactly-once delivery semantics, plus flow and congestion control
    • Realized as the Transmission Control Protocol (TCP)
Ports

- **End-point of a communication operation**
  - A 16-bit number
  - Alternatives: String (name of the application), URL
  - **Tags** packets as belonging to different services/streams
    - Note that there is no assumption that these packets will be picked up and/or serviced appropriately
    - Need network-aware programs that can do this – Rest of the Course

- Port numbers < 1024 are reserved for privileged programs (convention)
- Ports of publicly accessible services need to be widely advertised
  - On Unix, the `/etc/services` file

<table>
<thead>
<tr>
<th>Service</th>
<th>Port/Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp</td>
<td>20/tcp, 21/tcp</td>
</tr>
<tr>
<td>ssh</td>
<td>22/tcp</td>
</tr>
<tr>
<td>smtp</td>
<td>25/tcp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service</th>
<th>Port/Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>53/udp, 53/tcp</td>
</tr>
<tr>
<td>http</td>
<td>80/tcp</td>
</tr>
<tr>
<td>pop3</td>
<td>110/tcp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service</th>
<th>Port/Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms-sql</td>
<td>1433/tcp, udp</td>
</tr>
<tr>
<td>nfsd</td>
<td>2049/udp</td>
</tr>
<tr>
<td>rdp</td>
<td>3389/tcp</td>
</tr>
</tbody>
</table>
UDP

- A **connectionless** transport protocol
  - Send IP datagrams without establishing a connection
  - No guarantees of delivery (in- or out-of-order)
  - Used by applications whose interactions involve one request, one response
    - E.g., Domain Name Service (DNS)

- Packet format

```
<table>
<thead>
<tr>
<th>IP header</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>Destination port</td>
</tr>
<tr>
<td>UDP length</td>
<td>UDP checksum</td>
</tr>
</tbody>
</table>
```

32 bits
TCP

• A connection-based transport protocol
  – Connection identified by \((\text{src:sport, dst:dport})\) pair
  – Provides abstraction of a reliable byte stream

Three major components
• **Connection setup**
  – Permits ends of connection to synchronize on sequence numbers
  – Basis for reliable, in-order delivery
• **Window management**
  – Allows receiver to control rate at which sender can send data
  – Ensures sufficient space at receiver to store packets \(\rightarrow\) reliable delivery
• **Congestion management**
  – Allows sender to detect and cope with congestion in the network
  – Prevents fill-up of buffer space at sender \(\rightarrow\) reliable delivery
    • Improved throughput
TCP – Details

Connection Setup

Host 1

SYN(SEQ=x)

Host 2

SYN(SEQ=y, ACK=x+1)

(SEQ=x+1,ACK=y+1)

Agreeing on SEQ # allows hosts to discover out-of-order, dropped packets (and request retransmission)

Window Management

Block sender

Sender

2K

ACK=2048, WIN=2048

2K, SEQ=2048

2K, SEQ=2048

Full

Receiver

4K

2K

Full

2K

1K

1K, SEQ=4096

2K

WIN=0

WIN=0

WIN=2048

WIN=2048
TCP – Congestion Management

Why worry?
- Congestion may result in packet getting dropped
- Sender times out (waiting for an acknowledgement) and retransmits
- Retransmitted packet is also dropped, sender repeats
- Now assume, everybody is doing the same thing …

Solution
- Sender rate controlled by an additional parameter:
  \[ \text{Rate} = \min(\text{receiver window}, \text{congestion window}) \]
- Congestion window is dynamically adjusted based on time to receive acknowledgements
Helper Applications – Domain Name Service

- Applications prefer to work with symbolic host names
  - E.g., netserver1.pdsg.cs.nyu.edu, localhost
- The **Domain Name Service (DNS)** translates these into IP addresses
  - Sometimes, the reverse translation is also useful
Network Programming

• Builds on top of networking protocols (primarily: UDP, TCP, HTTP)
  – Lowest-level API just provides user-level abstractions for TCP and UDP

• Sockets: Application-level end-point of communication

• Operations often described by drawing analogy of a telephone
  – Call (Connection) setup
  – Conversation (Sending and receiving data packets)
  – Hangup (Disconnection)
Sockets API

- What does the API include?
  - A data structure, called a socket, that serves as an application-level end-point for networking operations
  - A set of functions for operating on this data structure

- Operation
  - Create a socket
    - Specify the kind of communication operation (e.g., TCP, UDP)
  - Setup the socket
    - In case of a protocol like TCP, establish the underlying connection
      - Which requires OS involvement, hence involves a system call
  - Use the socket
    - Provides a file-like interface: read and write bytes against the socket
  - Close the socket
    - Release any OS resources, free up memory used by the structure
Associating Sockets with (IP Address, Port #)

- There could be potentially many networking applications concurrently running on a node
  - OS needs to demultiplex incoming traffic for these different services
  - UDP and TCP networking code relies upon port numbers to distinguish different messages
- This means: Need to associate a port number with a socket
  - More generally, a network end-point: (IP Address, Port #)
- Different ways of associating network end-points with sockets
  - Implicit: Client-end of an application
    - As part of using the socket
  - Explicit: Server-end of an application
    - As part of setting up the socket, via an explicit call
Creating a Socket

- Functions of the Sockets API are protocol-neutral, however the OS needs to know what kind of operations are being done.

- So, three pieces of information communicated at creation time:
  - **Address family**: IPv4, IPv6, Unix, AppleTalk, IrDA, …
  - **Socket type**: Dgram, Stream, Raw, several extensions
    - First two correspond to UDP and TCP protocols
  - **Protocol**: UDP, TCP, …
    - Needs to be consistent with the socket type chosen above.

- .NET Framework Library
  - Create a new instance of the **Socket** class
    - Constructor takes three arguments corresponding to the above:

        ```csharp
        Socket s = new Socket(AddressFamily.InternetNetwork,
                               SocketType.Stream, ProtocolType.Tcp);
        ```
Setting Up and Using Sockets

• Different functions depending on protocol and whether client or server-end of an application

• UDP Sockets

Client

```
t = new Socket(...);
```

```
t.SendTo( [Server, Sport] );
```

```
t.ReceiveFrom( [?, ?] );
```

Client sends data to server at known port OS (implicitly) associates a port with socket

Client blocks

Server

```
s = new Socket(...);
```

```
s.Bind( [Server, SPort] );
```

Associate a well-known port number with the server-end of application

```
s.ReceiveFrom( [?, ?] );
```

```
s.SendTo( [Client, CPort] );
```

OS will deliver incoming packets with specified port number to application
Setting Up and Using Sockets – TCP

Client

```
t = new Socket(...);
```

```
t.Connect( [Server, SPort] );
```

Try connecting with server: OS implicitly associates a port, sends SYN packet

Client blocks

Server

```
s = new Socket(...);
```

```
s.Bind( [Server, SPort] );
```

```
s.Listen( #connections );
```

Associate a well-known port number with the server-end of application

Places the socket in a listening state: OS starts queuing SYN packets

```
u = s.Accept( );
```

Waits for handshake, and returns a new socket (bound to a different port)

Server blocks
Setting Up and **Using** Sockets – TCP

Client

```
\texttt{t.Send( ... );}
```

```
\texttt{t.Recv( ... );}
```

Server

\texttt{s: listening socket}

\texttt{u: accepted socket}

```
\texttt{u.Recv( ... );}
```

```
\texttt{u.Send( ... );}
```

```
\texttt{v = s.Accept( ... );}
```

Whether or not Send blocks is determined by buffer associated with socket

Recv blocks till sufficient data has been read (or connection closed)

Accept a new connection. Again, returns a new socket (bound to a different port)
Closing a Socket

• .NET Framework Library: `Close()`

• No operations are permitted on a closed socket
  – TCP will try and send any queued data to the other end
  – Only when both ends call close() is the TCP connection termination sequence initiated

• .NET: The user-level data structure can be garbage collected
  – Kernel storage associated with socket remains
Why Does Accept Return A New Socket?

- To keep packets belonging to different TCP connections separate
  - TCP sockets uniquely identified by a pair of end-points

- Consider a scenario where multiple processes are attempting to connect to a server process on a remote machine

```
216.165.111.8
(216.165.111.8:1500, 122.128.140.144:22)
process1
(216.165.111.8:1600, 122.128.140.144:22)
process2

122.128.140.144
(122.128.140.144:22, *:*).

OS

(122.128.140.144:22, 216.165.111.8:1500)
(122.128.140.144:22, 216.165.111.8:1600)
Process is listening on port 22
```
Example: A Simple Server Program (StringServer)

[Code walk-through using Remote Desktop Connection to netserver1.pdsg.cs.nyu.edu
StringServer code available as part of Lab1 helper files.]
Programming With The Sockets API

**Byte streams**, no boundaries preserved

- Send’s and Recv’s can line up arbitrarily
- Therefore, need a convention about data format
  - Agreeing on this convention is one of the hardest things
  - For our StringServer app: <length> <sequence of bytes>
  - For HTTP packets: server parses client requests
    - HTTP standard defines the format of these requests
- Your networking programs need to work in heterogeneous environments
  - **Byte-order** (Endian-ness) matters: network byte order is big-endian
    - HostToNetworkOrder, NetworkToHostOrder functions
- Errors can arise because of a number of reasons
  - Connect request to a socket that is not being listened to, early close, disconnected hosts, …
  - In C#, .NET, all of these delivered to the application as exceptions