Network Applications:
DNS Details;
UDP Network App Programming

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

2/3/2016
Outline

- Admin and recap
- Network app programming
Recap: Domain Name System (DNS)

- **Function**
  - map between (domain name, service) to value, e.g.,
    - (www.cs.yale.edu, Addr) → 128.36.229.30
    - (cs.yale.edu, Email) → netra.cs.yale.edu
Recap: DNS

Key design features of DNS
- Hierarchical domain name space allowing delegation
- Recursive or iterative queries
DNS Message Format?

Basic encoding decisions: UDP/TCP, how to encode domain name, how to encode answers...
Observing DNS Messages

- Issue DNS query using the command `dig`:
  - force iterated query to see the trace:
    `%dig +trace www.cnn.com`
    - see the manual for more details

- Capture the messages
  - DNS server is at port 53
    - Display and clear DNS cache
      - [https://support.apple.com/en-us/HT202516](https://support.apple.com/en-us/HT202516) (e.g., MAC)
  - Try to load the dns-capture file from class Schedule page, if you do not want live capture
DNS Protocol, Messages

DNS protocol: typically over UDP (can use TCP); query and reply messages, both with the same message format

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of questions</td>
<td>Number of answer RRs</td>
</tr>
<tr>
<td>Number of authority RRs</td>
<td>Number of additional RRs</td>
</tr>
<tr>
<td>Questions (variable number of questions)</td>
<td></td>
</tr>
<tr>
<td>Answers (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>Authority (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>Additional information (variable number of resource records)</td>
<td></td>
</tr>
</tbody>
</table>

https://www.ietf.org/rfc/rfc1035.txt
DNS Details

- Header (Sec. 4.1.1 of https://www.ietf.org/rfc/rfc1035.txt)
- Encoding of questions (Sec. 4.1.2):
  - [Label-length label-chars]
- Encoding of answers (Sec. 4.1.3)
  - Pointer format (http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml)
- See example DNS packets
Evaluation of DNS

Key questions to ask about a C-S application

- extensible?
- scalable?
- robust?
- security?
What DNS did Right?

- Hierarchical delegation avoids central control, improving manageability and scalability

- Redundant servers improve robustness
  - see http://www.internetnews.com/dev-news/article.php/1486981 for DDoS attack on root servers in Oct. 2002 (9 of the 13 root servers were crippled, but only slowed the network)

- Caching reduces workload and improves robustness
Problems of DNS

- Domain names may not be the best way to name other resources, e.g. files
- Simple query model makes it hard to implement advanced query
- Relatively static resource types make it hard to introduce new services or handle mobility
- Although theoretically you can update the values of the records, it is rarely enabled
- Early binding (separation of DNS query from application query) does not work well in mobile, dynamic environments
  - e.g., load balancing, locate the nearest printer
Outline

- Recap
- Network app programming
Socket Programming

Socket API

- introduced in BSD4.1 UNIX, 1981

- Two types of sockets
  - Connectionless (UDP)
  - connection-oriented (TCP)

socket

an interface (a “door”) into which one application process can both send and receive messages to/from another (remote or local) application process
Services Provided by Transport

- **User data protocol (UDP)**
  - multiplexing/demultiplexing

- **Transmission control protocol (TCP)**
  - multiplexing/demultiplexing
  - reliable data transfer
  - rate control: flow control and congestion control

![Diagram of data transmission between Host A and Host B]

Host A

Hello

I am ready

DATA

ACK

Host B
Big Picture: Socket

Host or server

Process

Socket

buffers, states

Internet

Host or server

Process

Socket

buffers, states

Controlled by application developer

Controlled by operating system

Controlled by application developer

Controlled by operating system
Outline

- Recap
- Basic network application programming
  - Overview
  - UDP (Datagram Socket)
DatagramSocket (Java) (Basic)

- **DatagramSocket()**
  constructs a datagram socket and binds it to any available port on the local host.

- **DatagramSocket(int lport)**
  constructs a datagram socket and binds it to the specified port on the local host machine.

- **DatagramPacket(byte[] buf, int length)**
  constructs a DatagramPacket for receiving packets of length length.

- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
  constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

- **receive(DatagramPacket p)**
  receives a datagram packet from this socket.

- **send(DatagramPacket p)**
  sends a datagram packet from this socket.

- **close()**
  closes this datagram socket.
Connectionless UDP: Big Picture (Java version)

Server (running on serv)
- Create socket, port=x, for incoming request:
  `serverSocket = new DatagramSocket(x)`
- Read request from `serverSocket`
- Generate reply, create datagram using client host address, port number
- Write reply to `serverSocket`

Client
- Create datagram using (serv, x) as (dest addr. port), send request using `clientSocket`
- Read reply from `clientSocket`
- Close `clientSocket`
Example: UDPServer.java

- A simple UDP server which changes any received sentence to upper case.
Java Server (UDP): Create Socket

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

class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
    
        // Create datagram socket
        // bind at port 9876
    
        // Check socket state:
        // %netstat -a -p udp -n
    }
}
```
System State after the Call

“*” indicates that the socket binds to all IP addresses of the machine:

```bash
% ifconfig -a
```
Binding to Specific IP Addresses

Server
Public address: 128.36.59.2
Local address: 127.0.0.1

UDP socket space

InetAddress sIP1 = InetAddress.getByName("localhost");
DatagramSocket ssock1 = new DatagramSocket(9876, sIP1);

InetAddress sIP2 = InetAddress.getByName("128.36.59.2");
DatagramSocket ssock2 = new DatagramSocket(9876, sIP2);

DatagramSocket serverSocket = new DatagramSocket(6789);
UDP Demultiplexing

**Server**
Public address: 128.36.59.2
Local address: 127.0.0.1

UDP socket space

- Address: {127.0.0.1:9876}
  Send/recv buf:

- Address: {128.36.59.2:9876}
  Send/recv buf:

- Address: {128.36.232.5:53}
  Send/recv buf:

**Client on server**

P1

- SP: x
- DP: 9876
- S-IP: A
- D-IP: 127.0.0.1

P2

- SP: y
- DP: 9876
- S-IP: B
- D-IP: 128.36.59.2

**Client IP: B**

UDP demultiplexing is based on matching (dst address, dst port)
**UDP Demultiplexing**

**Server**
- Public address: 128.36.59.2
- Local address: 127.0.0.1

UDP socket space:
- Address: {127.0.0.1:9876}
  - Send/recv buf:

- Address: {128.36.59.2:9876}
  - Send/recv buf:

- Address: {*:6789}
  - Send/recv buf:

- Address: {128.36.232.5:53}
  - Send/recv buf:

**Client on server**

- P1
  - SP: x
  - DP: 9876
  - S-IP: A
  - D-IP: 127.0.0.1

**Client IP: C**

- P3
  - SP: y
  - DP: 6789
  - S-IP: C
  - D-IP: 128.36.59.2

UDP demultiplexing is based on matching (dst address, dst port)
Per Socket State

- Each Datagram socket has a set of states:
  - local address
  - send buffer size
  - receive buffer size
  - timeout
  - traffic class

See [http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramSocket.html](http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramSocket.html)

Example: socket state after clients sent msgs to the server
Java Server (UDP): Receiving

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

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = null;

        while(true) {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);

            // Process the received datagram
            // Do something with receiveData
        }
    }
}
```
DatagramPacket

- **Receiving**
  - `DatagramPacket(byte[] buf, int length)` constructs a DatagramPacket for receiving packets of length length.
  - `DatagramPacket(byte[] buf, int offset, int length)` constructs a DatagramPacket for receiving packets starting at offset, length length.

- **Sending**
  - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)` constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
  - `DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)`
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception {
        ... 
        // process data
        String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
    }

    getData() returns a pointer to an underlying buffer array; for efficiency, don't assume receive() will reset the rest of the array.

    getLength() returns how much data is valid.
Java Server (UDP): Response

- **Java DatagramPacket:**
  - `getAddress()` / `getPort()` returns the **source** address/port
Java server (UDP): Reply

Get IP addr port #, of sender

InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

Create datagram to send to client

DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length,
IPAddress, port);

Write out datagram to socket

serverSocket.send(sendPacket);

End of while loop, loop back and wait for another datagram
Example: UDPClient.java

- A simple UDP client which reads input from keyboard, sends the input to server, and reads the reply back from the server.
Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        String sentence = inFromUser.readLine();
        byte[] sendData = sentence.getBytes();

        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress sIPAddress = InetAddress.getByName("servname");
    }
}
```
Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length, sIPAddress, 9876);

clientSocket.send(sendPacket);

Send datagram to server

byte[] receiveData = new byte[1024];
DatagramPacket receivePacket =
new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

Read datagram from server

String modifiedSentence =
new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
Demo

%mac: java UDPServer
% netstat to see buffer

%cicada: java UDPClient <server>

% wireshark to capture traffic
Discussion on Example Code

- A simple upper-case UDP echo service is among the simplest network service.

- Are there any problems with the program?
Data Encoding/Decoding

- Pay attention to encoding/decoding of data: transport layer handles only a sequence of bytes

if not careful, query sent != query received (how?)

client

query

encoding

byte array

server

result

decoding
Example: Endianness of Numbers

- `int var = 0x0A0B0C0D`

ARM, Power PC, Motorola 68k, IA-64

Intel x86
Example: String andChars

**Will we always get back the same string?**

- **client**
  - String (UTF-16)
  - `String.getBytes()`
  - byte array

- **server**
  - String (UTF-16)
  - `String(rcvPkt, 0, rcvPkt.getLength())`

**Depends on default local platform char set (why?):**
```
java.nio.charset.Charset.defaultCharset()
```
Example: Charset Troubles

- Try
  - java EncodingDecoding US-ASCII UTF-8
Encoding/Decoding as a Common Source of Errors

- Please read chapter 4 of Java Network Programming for more details.

- Common mistake even in many (textbook) examples:
**DataStream**

DataStream Diagram:

- **DataOutputStream**
  - 3.14 (8 bytes)
- **BufferedOutputStream**
  - 3.14
  - 343
  - 800
- **OutputStream**
  - 14 bytes

DataStream Diagram:

- **DataInputStream**
  - 3.14 (8 bytes)
- **BufferedInputStream**
  - 3.14
  - 343
  - 800
- **InputStream**
  - 14 bytes