Network Applications: DNS; Socket Programming

Y. Richard Yang

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

9/20/2018
Assignment Two to be linked on the Schedule page

Pace slow down?
Recap: Domain-based Message Authentication, Reporting, and Conformance (DMARC) [RFC7489]
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
Summary: DNS Design Features

- Hierarchical name space and hierarchical delegation avoids administrative bottleneck/central control, improving manageability and scalability
- Multiple domain servers improve scalability/robustness
- Native caching (control) reduces workload and improves robustness
- Flexible recursive and iterative query allows structure such as local resolver to simplify client and enable caching
- Using UDP to reduce overhead but also support TCP using the same format
- Same query and response format can make simplify basically servers
- Domain name encoding compression reduces query/response overhead
- Proactive answers of anticipated queries (server push) reduce # queries on server and latency on client

Today: approximately 1.3 million authoritative name servers listed in the .COM, .NET and .ORG zone files.
Grown from a few thousand entries to over 100 million entries. – That's scaling!
Many Other Uses of DNS

- DNSBL (black list) or RBL (realtime)
  - See changes: https://www.spamhaus.org/sbl/latest/
  - Query dig <reverse>.zen.spamhaus.org
    - https://www.spamhaus.org/zen/
Problems/Remaining Issues of DNS

- Security of DNS itself

- Limited extensibility
  - limited query model
  - Mixed, limited query cmd and query type
    - See https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters

- Largely a read data store, although theoretically you can update the values of the records, it is rarely enabled

- Each local domain needs servers, but an ad hoc domain may not have a DNS server
Outline

- Admin and recap
- DNS
  - Interface
  - Architecture design
  - Message design
  - Extensions/alternatives
    - service discovery
What do we need to extend standard DNS to support service discovery, say to implement Bonjour-type service discovery (discover local printers, local apple tv, file-share...)?
DNS-Service Discovery Component: Multicast DNS [RFC6762]

- Utilize IP multicast (broadcast medium)
  - link-local addressing
    - send to multicast address: 224.0.0.251 (address as a group name, and any host can specify that it joins the group)

- Implication:
  - each node (host) can become a responder
  - each node (host) can use multicast to announce (write) its values
Avoid continuous adding to DNS Resource Record Type, use
- ptr as as the only type
- introduce an extensible service naming convention (service in name)
"My Test" _printer._tcp dns-sd.org.

Example: dig _http._tcp.dns-sd.org. ptr
DNS-SD Local Service Discovery

- Use the `dns-sd` command on Mac as example
  - Advertise (register) an LPR printer on port 515

```
dns-sd -R "My Test" _printer._tcp. 515
pdl=application/postscript
```

Capture packets using wireshark

Name of instance providing the service

Txt for additional data

domain (default is local)

Port

Type of service

Transport
<table>
<thead>
<tr>
<th>MDNS</th>
<th>24 15.749230 172.27.21.251 224.0.0.251</th>
<th>232 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca...</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>15.758136 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 252 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca...</td>
</tr>
<tr>
<td>26</td>
<td>15.946447 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 172 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;QU&quot; question ANY Ys-MacBook-Pro.local, &quot;QU&quot; question SRV 0...</td>
</tr>
<tr>
<td>27</td>
<td>15.946445 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 192 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;QU&quot; question ANY Ys-MacBook-Pro.local, &quot;QU&quot; question SRV 0...</td>
</tr>
<tr>
<td>28</td>
<td>16.197838 172.27.21.251 224.0.0.251</td>
<td>MDNS 172 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;OM&quot; question ANY Ys-MacBook-Pro.local, &quot;OM&quot; question SRV 0...</td>
</tr>
<tr>
<td>29</td>
<td>16.197896 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 192 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;OM&quot; question ANY Ys-MacBook-Pro.local, &quot;OM&quot; question SRV 0...</td>
</tr>
<tr>
<td>30</td>
<td>16.450462 172.27.21.251 224.0.0.251</td>
<td>MDNS 172 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;OM&quot; question ANY Ys-MacBook-Pro.local, &quot;OM&quot; question SRV 0...</td>
</tr>
<tr>
<td>31</td>
<td>16.450508 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 192 Standard query 0x0000 ANY My Test_printer.tcp.local, &quot;OM&quot; question ANY Ys-MacBook-Pro.local, &quot;OM&quot; question SRV 0...</td>
</tr>
<tr>
<td>32</td>
<td>16.700958 172.27.21.251 224.0.0.251</td>
<td>MDNS 291 Standard query response 0x0000 TXT, cache flush PTR_printer.tcp.local PTR My Test_printer.tcp.local SRV, cache ...</td>
</tr>
<tr>
<td>33</td>
<td>16.700999 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 311 Standard query response 0x0000 TXT, cache flush PTR_printer.tcp.local PTR My Test_printer.tcp.local SRV, cache ...</td>
</tr>
<tr>
<td>34</td>
<td>16.805259 172.27.21.251 224.0.0.251</td>
<td>MDNS 232 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca...</td>
</tr>
<tr>
<td>35</td>
<td>16.805318 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 252 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca...</td>
</tr>
<tr>
<td>36</td>
<td>17.703216 172.27.21.251 224.0.0.251</td>
<td>MDNS 291 Standard query response 0x0000 TXT, cache flush PTR_printer.tcp.local PTR My Test_printer.tcp.local SRV, cache ...</td>
</tr>
<tr>
<td>37</td>
<td>17.704185 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 311 Standard query response 0x0000 TXT, cache flush PTR_printer.tcp.local PTR My Test_printer.tcp.local SRV, cache ...</td>
</tr>
<tr>
<td>38</td>
<td>18.888877 fe80:1c99:22de:9a::ff02::ff</td>
<td>MDNS 469 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local TXT, cac...</td>
</tr>
<tr>
<td>39</td>
<td>18.889057 172.27.21.251 224.0.0.251</td>
<td>MDNS 449 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local TXT, cac...</td>
</tr>
</tbody>
</table>
Exercise

- Use the `dns-sd` command on Mac as example
  - Browse web pages on local machines
    - `dns-sd -B _http._tcp`
      - Advertise (register) a web page on local machine
        - `dns-sd -R "My Test" _http._tcp . 80 path=/path-to-page.html`
      - Kill the command
Network Service Discovery in Android

- Based on DNS-SD/mDNS
- Foundation for peer-to-peer/Wi-Fi Direct in Android

- See https://developer.android.com/training/connect-devices-wirelessly/nsd.html for programming using nsd
General Service/Naming Discovery Paradigm: Linda

- “Distributed workspace” by David Gelernter in the 80’s at Yale

- Very influential in naming and resource discovery

- Key issues
  - How to name services/resources
  - How to write/update into name space
  - How to resolve names
The Linda Paradigm

- Naming scheme:
  - arbitrary tuples (heterogeneous-type vectors)

- Name resolution:
  - Nodes write into shared memory
  - Nodes read matching tuples from shared memory
    - exact matching is required for extraction
Linda: Core API

- **out()**: writes tuples to shared space
  - example: `out("abc", 1.5, 12).`
  - result: insert (“abc”, 1.5, 12) into space

- **read()**: retrieves tuple copy matching arg list (blocking)
  - example: `read(“abc”, ? A, ? B)`
  - result: finds (“abc”, 1.5, 12) and sets local variables
    - $A = 1.5$, $B = 12$. Tuple (“abc”, 1.5, 12) is still resident in space.

- **in()**: retrieves and deletes matching tuple from space (blocking)
  - example: same as above except (“abc”, 1.5, 12) is deleted

- **eval(expression)**: similar to out except that the tuple argument to eval is evaluated
  - example: `eval("ab", -6, abs(-6))` creates tuple (“ab”, -6, 6)
Linda Extension: JavaSpaces

- Industry took Linda principles and made modifications
  - add transactions, leases, events
  - store Java objects instead of tuples
  - a very comprehensive service discovery system

- Definitive book, “JavaSpaces Principles, Patterns, and Practice”
  - 2 of 3 authors got Ph.D.’s from Yale
Additional Pointers

- *Grapevine: Xerox PARC early 1980’s* Birrell, Levin, Needham, Schroeder *CACM* 25(1)

- The MAIN name system, an exercise in centralized computing, Deegan, Crowcroft and Warfield, *ACM SIGCOMM* 35(5), Oct 2005
Outline

- Admin and recap
- DNS
  - Implementation/programming: UDP 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
Big Picture: Socket

- Host or server
  - Process
    - Socket
      - buffers, states
    - Controlled by operating system
  - Controlled by application developer
- Internet
- Host or server
  - Process
    - Socket
      - buffers, states
    - Controlled by operating system
  - Controlled by application developer
Outline

- Admin and recap
- DNS
- Basic network application programming
  - Overview
  - UDP (Datagram Socket)
Discussion

- What might the UDP API look like if you were to design it?
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.

  // more methods on multiplexing control: bind, connect; see demos

- **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.

// socket state control

- **close()**
  closes this datagram socket.

https://docs.oracle.com/javase/9/docs/api/java/net/DatagramSocket.html
Connectionless UDP: Big Picture (Java version)

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

Client
- Create socket, 
  clientSocket = DatagramSocket()
- 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.
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:

% ifconfig -a
Binding to Specific IP Addresses

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

UDP socket space

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

```java
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);
```
Exercise: UDP Port Scanner

- Try to test all UDP bindings
- `[sudo] lsof -i4UDP -n -P`
Exercise: UDP Demultiplexing

**Server**

Public address: 128.36.59.2  
Local address: 127.0.0.1  
UDP socket space

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

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

- Address: {128.36.232.5:53}  
  snd/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:**
- P1
  - IP: A

- P2
  - IP: B

UDP demultiplexing is based on matching state
Exercise: 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: {127.0.0.1:9876}
- Send/recv buf:
- Address: {*:9876}
- Send/recv buf:
- Address: {128.36.232.5:53}
- Send/recv buf:

UDP demultiplexing is based on matching state

**Client on server**
- P1
  - SP: x
  - DP: 9876
  - S-IP: A
  - D-IP: 127.0.0.1

**Client IP: C**
- P3
  - SP: y
  - DP: 9876
  - S-IP: C
  - D-IP: 128.36.59.2
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

Example: socket state after clients sent msgs to the server
Exercise: UDPClient

- Send messages to UDPServer from local, from a zoo machine
- Use wireshark to capture traffic
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);
        }
    }
}
```
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`.

- **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)`
Java Server (UDP): Processing

```java
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.

**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);
```
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");
        DatagramPacket packet = new DatagramPacket(sendData, sendData.length, sIPAddress, 1234);
        clientSocket.send(packet);
    }
}
```
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();
}
A simple upper-case UDP echo service is among the simplest network service. Are there any problems with the processing?
Data Encoding/Decoding

- Rule: ALWAYS pay attention to encoding/decoding of data

If not careful, query sent != query received (how?)
Example: Endianness of Numbers

- `int var = 0x0A0B0C0D`

- ARM, Power PC, Motorola 68k, IA-64
- Intel x86

- `sent != received: take an int on a big-endian machine and send a little-endian machine`
Example: String and Chars

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:
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 2 (Streams) of Java Network Programming for more details
  - Java stream, reader/writer can always be confusing, but it is good to finally understand

- Common mistake even in many (textbook) examples:
Modify the example UDP server code to implement a local DNS server.

<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>Name, type fields for a query</td>
</tr>
<tr>
<td>Answers (variable number of resource records)</td>
<td>RRs in response to query</td>
</tr>
<tr>
<td>Authority (variable number of resource records)</td>
<td>Records for authoritative servers</td>
</tr>
<tr>
<td>Additional information (variable number of resource records)</td>
<td>Additional &quot;helpful&quot; info that may be used</td>
</tr>
</tbody>
</table>

root name server

dns.cs.umass.edu

TLD name server

130.132.1.9

local name server

TLD name server

authoritative name server
dns.cs.umass.edu

requesting host
cyndra.cs.yale.edu

gaia.cs.umass.edu