Infrastructure Supporting Multiple Servers: 
Domain Name System (DNS)

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs434/

03/08/2021
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
    - Basic DNS
    - mDNS and DNS-SD
Admin

- Remaining office hour today: after class from 5:15 to 6:30 or later if needed
- I will post slot later this week and next to schedule one-on-one grading meetings
Recap: Operational Laws

- Little’s Law: \( Q = X R \)
- Utilization law: \( U = XS \)
- Response time law (Poisson arrival): \( R = S/(1-U) \)
- Forced flow law: \( X_i = V_i X \)
- Bottleneck device: largest \( D_i = V_i S_i \)
- Bottleneck bound of interactive response (for the given closed model):

\[
X(N) \leq \min\left\{ \frac{1}{D_{\text{max}}} , \frac{N}{D+Z} \right\}
\]

\[
R(N) \geq \max\{D, ND_{\text{max}} - Z\}
\]
Recap: The High-Performance, Robust Network Server Journey

- Avoid blocking (so that we can reach bottleneck throughput)
  - introduce threads
- Limit unlimited thread overhead
  - thread pool, select, async io
- Coordinating data access
  - synchronization (lock, synchronized)
- Coordinating behavior: avoid busy-wait
  - wait/notify; select FSM, future/listener/callback
- “scheduler”/eventloop design
  - Select/eventloop as a scheduling loop, mixing io/tasks
- Extensibility/robustness
  - language support/design for interfaces (handler interface)
  - handler pipeline (Netty, nginx)
- Data driven architecture analysis
  - operational analysis
Recap: HTTP Protocol

- Representation can be based from different sources
  - URI can be a static file
  - URI can be an external program (CGI, FastCGI, ...)
  - URI can be an internal component of the server

- Representation can be computed from both server and client attributes

- Representation can have meta data
  - Last modified, ...
Recap: Common Benefits of Multiple Servers

- **Scalability**
  - Scaling beyond single server **throughput**
  - Scaling beyond single geo location **latency**

- **Redundancy/fault tolerance**
  - **Redundancy** (e.g., to handle failures)
  - **Operation**/management (e.g., incremental upgrade)

- **System/software architecture**
  - Software constraint (e.g., run a single copy of a database server due to single license)
  - Software modularity (e.g., front end, business logic, and database; microservice architectures)
Recap: Roadmap

- We first focus on using multiple largely homogenous (similar) servers providing resource/latency scaling.
Basic Notification/Direction Abstractions

- Name Abstraction (DNS) - mapping
  - 1 IP
  - 1 IP
  - Name (e.g., www.yale.edu)

- IP/Locator Abstraction (LB, IP anycast) - mapping
  - 1 IP
  - 1 IP
  - IP (e.g., 8.8.8.8)
Recap: DNS Basic Architecture:
Distributed Management of the Domain Name Space

- A distributed name space managed by authoritative name servers
  - divided into zones, where each zone is a sub-tree of the global tree
  - each zone has its own (one or more) authoritative name servers
  - an authoritative name server of a zone may delegate a subset (i.e. a sub-tree) of its zone to another name server

Design to remove
- Scalability and robustness bottleneck
- Administrative bottleneck
Information Model

DNS: stores resource records (RR) for each name (node in the tree)

RR format: (name, type, value, ttl)

- **Type=SOA**
  - name is domain name
  - value is SOA record of domain

- **Type=A (AAAA)**
  - name is hostname
  - value is IPv4 (IPv6) address

- **Type=NS**
  - name is domain (e.g. yale.edu)
  - value is name of the domains’ authoritative name server

- **Type=MX**
  - value is hostname of mail server associated with name

- **Type=CNAME**
  - name is an alias of a “canonical” (real) name
  - value is canonical name

- **Type=SRV**
  - general extension for services

- **Type=PTR**
  - a pointer to another name

- **Type=TXT**
  - general txt

see [http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml](http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml)
Distributed Information Mapping: Zone and its Name Servers

- A zone is defined as a subtree of a set of nodes in the global tree.
- Exercise: What does the name server of a zone know?
Zone and its Name Server

- Each name server of a zone should know
  - the mapping of each node (name) in its zone to that node’s properties (authoritative answer)
  - the name(s) of the name server(s) of each of its subzone
Root Zone

- The root zone is managed by the root name servers

  a. Verisign, Dulles, VA
  b. USC-ISI Marina del Rey, CA
  c. Cogent, Herndon, VA (also Los Angeles)
  d. U Maryland College Park, MD
  e. NASA Mt View, CA
  f. Internet Software C.
     Palo Alto, CA
     (and 17 other locations)
  g. US DoD Vienna, VA
  h. ARL Aberdeen, MD
  i. Autonomica, Stockholm
     (plus 3 other locations)
  j. Verisign, (11 locations)
  k. RIPE London
     (also Amsterdam,
     Frankfurt)
  l. ICANN Los Angeles, CA
  m. WIDE Tokyo

Demo (API): Try DNS using dig

- `dig @dns-server <name> <type>`
  - `<name>`: a sequence of **labels** combined by "." , where each label is the string assigned to a node on the path; root has an empty label
    - Q: what is the official name of yale node?
  - `<type>` can be one of A, AAAA, NS, TXT, CNAME, MX, SRV, PTR, SOA
  - Try yale.edu, for each of the types

- **Offline**
  - Try mail.google.com
  - Try cnn.com and see an image and try to query related information
Behind the Scene: The Zone File

- The content of a DNS server often is initialized by a zone file
- See zone-example.txt

- Exercise: add a subzone
Note

- One of the challenges in setting up DNS is the glue record
- See https://www.howtoforge.com/troubleshooting-common-dns-misconfiguration-errors
DNS Message Flow: Two Basic Types of Queries

Recursive query:
- The contacted name server resolves the name completely

Iterated query:
- Contacted server replies with name of server to contact
  - “I don’t know this name, but ask this server”
Two Basic DNS Message Flows

Issues of the two approaches?
Typical DNS Message Flow: The Hybrid Case

- Host knows only local name server
- Local name server is learned from DHCP, or configured, e.g. /etc/resolv.conf
- Local DNS server helps clients resolve DNS names
- Benefits of local name servers (often called resolvers)
  - simplifies client
  - caches/reuses results
DNS Message Format?

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

- 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., MacOS sudo killall -HUP mDNSResponder)
  - visit gmail.com
  - Try to load the dns-capture file from class Schedule page, if you do not want live capture
DNS Messages

DNS messages 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>
</tbody>
</table>

| Questions             | 12 bytes |
| (variable number of questions) | Name, type fields for a query |
| Answers               | RRs in response to query |
| (variable number of resource records) | Records for authoritative servers |
| Authority             | Additional “helpful” info that may be used |
| (variable number of resource records) | |
DNS Messages Detail: Name Encoding

<table>
<thead>
<tr>
<th>Name: gmail.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Name Length: 9]</td>
</tr>
<tr>
<td>[Label Count: 2]</td>
</tr>
<tr>
<td>Type: A (Host Address) (1)</td>
</tr>
<tr>
<td>Class: IN (0x0001)</td>
</tr>
</tbody>
</table>

- `<length-byte><string>` until zero length
- `<length-byte>` first be 00; 01/10 reserved; 11 indicates pointer
Message Compression (Label Pointer)
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)
- DNSsec
  - https://metebalci.com/blog/a-minimum-complete-tutorial-of-dnssec/
  - http://dnsinstitute.com/documentation/dnssec-guide/ch06s02.html
Exercise: Implications of Encoding

- What is the maximum DNS label length?
- Does the protocol have a problem when . appears in label name?
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 simplify result forwarding
- Interesting \(<\text{length}\><\text{content}\>\) encoding and pointer for compression
- 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!
Problems/Remaining Issues of Basic DNS

- **Security**
  - Security of DNS information [see DNSSEC]

- **Architecture**
  - Each local domain needs servers, but an ad hoc domain may not have a DNS server
    - Example: typical home network does not have a DNS server

- **Information/data/query model**
  - Limited key (fixed harder to extend qtype; see [1])
  - One dimension (nodes on the tree, not flow from src to dst)

- **Update model**
  - Largely a read data store
    - secondary NS server sync up w/ primary [AXFR query]
    - although theoretically you can update the values of the records, it is rarely enabled

Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
    - Basic DNS
    - mDNS and DNS-SD
**DNS-Service Discovery and Multicast DNS**

- Form the foundation of Apple Bonjour/Airplay
- Often used in home network for zero-touch management (no DNS server set up)

**Demo on Mac**
- See list of commands `dns-sd`
- List Airplay in my home
  - `dns-sd -B _airplay._tcp`
- See a particular instance
  - `dns-sd -L <instance> <service> <domain>`
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
DNS-Service Discovery Component: DNS-based Service Discovery [RFC6763]

- Based on DNS SRV record type [RFC2782], which avoids continuous adding to DNS Resource Record Type, use
  - `<service> . <protocol> . <domain>` in SRV record
  - `_printer._tcp.example.com`.

- DNS-SD extends to `<instance> . <service> . <protocol> . <domain>`; using PTR record

"My Test" _printer._tcp dns-sd.org.

Name of instance providing service

`<type_service>`. `<transport>`

domain (. means default, which is local
Exercise: DNS-SD Local Service Registration

- Exercise: Advertise (register) an LPR printer on port 515

```

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

- Name of instance providing the service
- `<type_service>`, `<transport>`
- Domain (`.` means default, which is local)
- Port
- Txt for additional data

- `dns-sd -B _printer._tcp`
- Capture packets using wireshark
- Check System Preference/Printer +
Backup Slides: UDP Socket Programming
UDP Java Socket Programming
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
    }
}
“*” 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
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: UDPPortScanner

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

UDP demultiplexing is based on matching state.

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:

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: A

client IP: B
Exercise: UDP Demultiplexing

UDP demultiplexing is based on matching state.
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
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();
    }
}
```

data methods:

- `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);

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");

        DatagramPacket packet = new DatagramPacket(sendData, sendData.length, sIPAddress, 25000);

        clientSocket.send(packet);

        System.out.println("Data sent to Server");
    }
}
```
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();
}
Java Server (UDP): Processing

A simple upper-case UDP echo service is among the simplest network service. Are there any problems with the processing?

```java
class UDPServer {
    public static void main(String args[]) throws Exception {
        …
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);

        // process
        String sentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();

        // send
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
        serverSocket.send(sendPacket);
    }
}
```
Basic DNS Indirection and Rotation

%dig cnn.com

router

IP address of cnn.com
157.166.226.25
157.166.226.26

DNS server for cnn.com
157.166.226.26
157.166.226.25
157.166.255.18

IP address of cnn.com
157.166.226.25
157.166.226.26
Summary: Basic DNS Protocol, Messages

Many features: typically over **UDP** (can use **TCP**); query and **reply** messages with the **same message format**; **length/content encoding of names**; **simple compression**; additional info as **server push**

<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>
</tbody>
</table>

- **12 bytes**
- Name, type fields for a query
- RRs in response to query
- Records for authoritative servers
- Additional “helpful” info that may be used