Network Applications: DNS, UDP Socket

Y. Richard Yang

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

9/12/2013
Outline

- Admin and recap
- DNS
- Network application programming: UDP
72 discretionary late hours for assignments across the semester
Recap: The Big Picture of the Internet

- **Hosts and routers:**
  - ~ 1 bill. hosts (2013)
  - organized into ~45K networks
  - backbone links 100 Gbps

- **Software:**
  - datagram switching with virtual circuit support
  - layered network architecture
    - use end-to-end arguments to determine the services provided by each layer
  - the hourglass architecture of the Internet
## Protocol Formats

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit source port number</td>
<td>16-bit destination port number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit UDP length</td>
<td>16-bit UDP checksum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data (if any)

---

**UDP Protocol Format**

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit version</td>
<td>4-bit header length</td>
<td>8-bit type of service (TOS)</td>
<td>16-bit total length (in bytes)</td>
</tr>
<tr>
<td>16-bit identification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-bit time to live (TTL)</td>
<td>8-bit protocol</td>
<td>16-bit header checksum</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32-bit source IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit destination IP address</td>
</tr>
</tbody>
</table>

Options (if any)

Data

---

**Ethernet Frame**

<table>
<thead>
<tr>
<th>DA</th>
<th>SA</th>
<th>Type</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>46-1500</td>
<td>4</td>
</tr>
</tbody>
</table>

Minimum size = 64 bytes
Recap: Client-Server Paradigm

- The basic paradigm of network applications is the client-server (C-S) paradigm.

- Some key design questions to ask about a C-S application:
  - extensibility
  - scalability
  - robustness
  - security
Recap: Email App

Some nice protocol extensibility design features

- separate protocols for different functions
- simple/basic (smtp) requests to implement basic control; fine-grain control through ASCII header and message body
- status code in response makes message easy to parse
Recap: A Major Challenge to Email

- Spam (Google)

https://mail.google.com/intl/en/mail/help/fightspam/spamexplained.html
Recap: Spam Detection Methods by GMail

- Known phishing scams
- Message from unconfirmed sender identity
- Message you sent to Spam/similarity to suspicious messages
- Administrator-set policies
- Empty message content

https://support.google.com/mail/answer/1366858?hl=en
Confirming Sender Identity

- Ideal case:
  - RFC 822 From: Header Field
    - Content author

- Other identifies
  - Peer MTA Host IP Address
    - Neighbor SMTP client host
  - SMTP EHLO Command
    - Neighbor SMTP client organization
  - SMTP MAIL FROM Command
    - Notification return address
  - RFC 822 Sender: Header Field
    - Message posting agent
Current Email Authentication Approaches

Sender Policy Frame (SPF)  DomainKeys Identified Mail (DKIM)
Sender Policy Framework (SPF RFC4408)

Is my neighbor m a permitted sender for the domain?
SPF Example

Received: from mtal.esp1234.com (HELO mtal.esp1234.com) (10.0.0.1)
   by mailserver.company.com with SMTP; 28 Mar 2008 19:53:28 -0000
Date: Fri, 28 Mar 2008 14:53:27 -0500 (CDT)
From: "Author" <author@authorscompany.com>
To: Recipient@company.com
Subject: March Newsletter
Sender: authorscompany@esp1234.com
Return-Path: bounce-4101674@authorscompany.esp1234.com
...
DomainKeys Identified Mail (DKIM; RFC 5585)

- A domain-level digital signature authentication framework for email, using public key crypto
  - E.g., gmail.com signs that the message is sent by gmail server

- Basic idea of public key signature
  - Owner has both public and private keys
  - Owner uses private key to sign a message to generate a signature
  - Others with public key can verify signature
Example: RSA

1. Choose two large prime numbers $p, q$. (e.g., 1024 bits each)

2. Compute $n = pq$, $z = (p-1)(q-1)$

3. Choose $e$ (with $e < n$) that has no common factors with $z$. ($e, z$ are “relatively prime”).

4. Choose $d$ such that $ed - 1$ is exactly divisible by $z$. (in other words: $ed \mod z = 1$).

5. Public key is $(n,e)$. Private key is $(n,d)$. 
RSA: Signing/Verification

0. Given \((n,e)\) and \((n,d)\) as computed above

1. To sign message, \(m\), compute \(h = \text{hash}(m)\), then sign with private key
   \[ s = h^d \mod n \] (i.e., remainder when \(h^d\) is divided by \(n\))

2. To verify signature \(s\), compute
   \[ h' = s^e \mod n \] (i.e., remainder when \(s^e\) is divided by \(n\))

   \[ h = (h^d \mod n)^e \mod n \]

   Magic happens!

   The magic is a simple application of Euler's generalization of Fermat's little theorem
DomainKeys Identified Mail (DKIM)

Is the message signed by the private key of the signing domain?
DKIM Architecture
Remaining Questions?

- How does SPF know if its neighbor MTA is a permitted sender of the domain?

- How does DKIM retrieve the public key of the author domain?
Recall: Client-Server Paradigm

- The basic paradigm of network applications is the client-server (C-S) paradigm

- Some key design questions to ask about a C-S application:
  - extensibility
  - scalability
  - robustness
  - security
Scalability/Robustness

- High scalability and robustness fundamentally require that multiple email servers serve the same email address

Client need an email server’s IP address

Mapping

- yale.edu
- 130.132.50.7
- 130.132.50.8
- 130.132.50.9
Mapping Functions Design Alternatives

- Map from an email address server name to IP address of email server

![Diagram showing two mapping scenarios: one with a single IP and one with multiple IPs.](attachment:image.png)
Mapping Functions Design Alternatives

- **load balancer** (routing)
- **switch**
- **name** (e.g., yale.edu)
- **mapping**
- **1 IP**

- **load balancer** (routing)
- **switch**
- **name** (e.g., yale.edu)
- **mapping**
- **1 IP**
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
DNS Records

**DNS:** stores resource records (RR)

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

- **Type=A**
  - name is hostname
  - value is IP address

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

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

- **Type=NS**
  - name is domain (e.g. yale.edu)
  - value is canonical name

- **Type=CNAME**
  - name is an alias name for some “canonical” (the real) name

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

- **Type=TXT**
  - general txt
DNS Examples

- nslookup
- `set type=<type>`
  - `type = MX`
    - cs.yale.edu
  - `type = TXT`
    - yale.edu
    - gmail.com
    - 20120113._domainkey.gmail.com
DNS Design: Dummy Design

- DNS itself can be considered as a client-server system as well
- How about a dummy design: introducing one super Internet DNS server?

THE DNS server of the Internet
Problems of a Single DNS Server

- Scalability and robustness bottleneck
- Administrative bottleneck
DNS: Distributed Management of the Domain Name Space

- A distributed database managed by authoritative name servers
  - divided into zones, where each zone is a sub-tree of the global tree
  - each zone has its own 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
Email Architecture + DNS
Root Zone and Root Servers

- The root zone is managed by the root name servers
  - 13 root name server IPs worldwide
    - 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.
    - 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

Linking the Name Servers

- Each name server knows the addresses of the root servers
- Each name server knows the addresses of its immediate children (i.e., those it delegates)

Top level domain (TLD)

Q: how to query a hierarchy?
DNS Message Flow: Two 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 Extreme 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
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
  - simplifies client
  - Caches/reuses results

Diagram:
- Requesting host: cyndra.cs.yale.edu
- Local name server: 130.132.1.9
- Root name server
- Iterated query
- TLD name server
- Authoritative name server: dns.cs.umass.edu
- Gaia.cs.umass.edu
- 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
  - simplifies client
  - Caches/reuses results
**DNS Protocol, Messages**

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

**DNS Msg header:**
- **identification**: 16 bit # for query, the reply to a query uses the same #
- **flags**:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

<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>
Observing DNS

- Use 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
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](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 improve robustness
Problems of DNS

- Domain names may not be the best way to name other resources, e.g. files

- 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

- Simple query model makes it hard to implement advanced query

- 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
- Email
- DNS
- Network application 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

---

```
Host A
Hello
Host B
I am ready
DATA
ACK
```
Big Picture: Socket
UDP Java API

Host or server

Process

Buffers, states

Socket

Controlled by application developer

Controlled by operating system

Internet

Host or server

Process

Buffers, states

Socket

Controlled by application developer

Controlled by operating system
DatagramSocket (Java)

- **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.

- **DatagramSocket(int lport, InetAddress laddr)**
  creates a datagram socket and binds to the specified local port and laddress.

- **DatagramSocket(SocketAddress bindaddr)**
  creates a datagram socket and binds to the specified local socket address.

- **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 = 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

- Create socket with port number:
  DatagramSocket sSock = new DatagramSocket(9876);
- If no port number is specified, the OS will pick one
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);
    }

    Check socket state:
    %netstat -p udp -n
```
System State after the Call

server
128.36.232.5
128.36.230.2

UDP socket space

address: {*:9876}
snd/recv buf:

local address
why shown as "*"?

local port

address: *(128.36.232.5:53)
snd/recv buf:

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

% ifconfig -a