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
**Admin**

- 72 discretionary late hours for assignments across the semester

---

**Recap: The Big Picture of the Internet**

- **Hosts and routers:**
  - ~1 billion 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

![Diagram of the Internet stack](https://via.placeholder.com/150)
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.espl234.com (HELO mtal.espl234.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@espl234.com
Return-Path: bounce-4101674@authorscompany.espl234.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 \( e \lt n \)) 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 \( n \))

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

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

- Private Key
- ORIGINATING OR RELAYING ADMD
- Sign Message with S DID
- (paired)
- Public Key
- RELAYING OR DELIVERING ADMD
- Message Signed?
  - yes
  - no
  - Verify Signature
  - fail
  - pass
  - Assessments
  - / Check Signing Practices <
  - / Remote Sender Practices

- Reputation/Accreditation
- Message Filtering
- Local Info on Sender
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

Mapping Functions Design Alternatives

- Map from an email address server name to IP address of email server
**Mapping Functions Design Alternatives**

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

- **name** (e.g., yale.edu)
  - mapping
  - 1 IP

---

**DNS: Domain Name System**

- **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

---

*Image descriptions are not provided in the given text.*
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=TXT
  - general txt

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

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

- Type=SRV
  - general extension for services

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

![Diagram of DNS zones and authoritative servers]

denis

Email Architecture + DNS

![Diagram of email architecture and DNS integration]
Root Zone and Root Servers

- The root zone is managed by the root name servers
  - 13 root name server IPs worldwide
    - Verisign, Dulles, VA
    - Cogent, Herndon, VA (also Los Angeles)
    - U Maryland College Park, MD
    - US DoD Vienna, VA
    - ARL Aberdeen, MD
    - NASA Mt View, CA
    - Internet Software C.
    - Palo Alto, CA
    - (and 17 other locations)
    - USC-ISI Marina del Rey, CA
    - ICANN Los Angeles, CA
    - Autonomica, Stockholm
    - (plus 3 other locations)
    - RIPE London
    - (also Amsterdam, Frankfurt)
    - 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)

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
- Benefits of local name servers
  - simplifies client
  - Caches/reuses results

1. Host knows only local name server
2. Local name server is learned from DHCP, or configured, e.g. /etc/resolv.conf
3. Local DNS server helps clients resolve DNS names
4. 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:
  o query or reply
  o recursion desired
  o recursion available
  o 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:
  o force iterated query to see the trace:
    %dig +trace www.cnn.com
    • see the manual for more details

- Capture the messages
  o 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)

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

UDP Java API
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)                      Client
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
```

```
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);
        // Add more server logic here...
    }
}
```

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

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

% ifconfig -a