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

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

Protocol Formats

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 Framework (SPF RFC4408)
**SPF Example**

Received: from mta1.expl234.com (HELO mta1.expl234.com) [10.0.0.1]
by mailserver.com with SMTP 28 Mar 2008 19:53:28 -0000
Date: Tue, 28 Mar 2008 14:10:21 -0000 (UTC)
From: "Author" <author@authorscompany.com>
To: Recipient@company.com
Subject: March Newsletter
Sender: authorscompany@smtp1234.com
Return-Path: bounce-41036748@authorscompany.expl234.com

---

**DomainKeys Identified Mail (DKIM: RFC 585)**

- 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.g., $e$ and $z$ 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 $n$)

The magic is a simple application of Euler’s generalization of Fermat’s little theorem.

---

**DomainKeys Identified Mail (DKIM)**

- **MUA**
  - smtp/submission
- **Signing MTA**
  - smtp
- **MTA**
  - smtp
- **Verifying MTA**
  - pop/mapi
- **MUA**

Is the message signed by the private key of the signing domain?
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.

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.

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
DNS Records

DNS: stores resource records (RR)

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

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

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

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

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

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

- **Type=TXT**
  - general text

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服务器 IP addresses worldwide
  - NASA, NY, US
  - Internet Software Center, PA, US
  - NLANR, CA
  - 17 other locations
- See http://root-servers.org/ for more details

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)

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

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
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

---

Observing DNS

- **Use the command** `dig`:
  - `dig +trace www.cnn.com`
  - see the manual for more details
- **Capture the messages**
  - DNS server is at port 53

---

Evalutation 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 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 queries
- **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)
- Transmission control protocol (TCP)

**Big Picture: Socket**

**UDP Java API**

**DatagramSocket (Java)**
- `DatagramSocket(int lport)` constructs a datagram socket and binds it to the specified port on the local host machine.
- `DatagramSocket(InetAddress bindaddr)` creates a datagram socket and binds to the specified local port and address.
- `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 given 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)**

- Create socket with port number:
  ```java
  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);
        // Create datagram socket
        // bind at port 9876

        System State after the Call

        **"*"** indicates that the socket binds to all IP addresses of the machine:
        % ifconfig -a
```