

Network Applications and Network Programming

9/18/2009

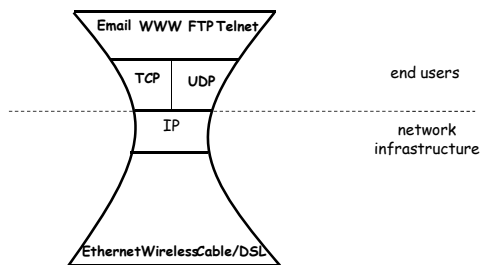
1

Outline

- Recap
- Email app.
- DNS
- Network application programming

2

Recap: The Internet Hourglass Architecture



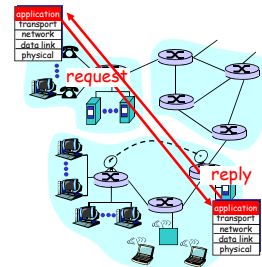
3

Recap: Client-Server Paradigm

Typical network app has two pieces: *client* and *server*

Key questions to ask about a C-S application

- Is the application **extensible**?
- Is the application **scalable**?
- How does the application handle server failures (being **robust**)?
- How does the application provide **security**?



4

Outline

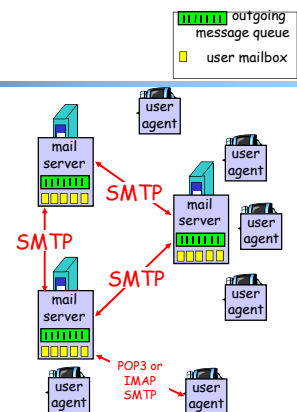
- Recap
- Email app.
- DNS
- Network application programming

5

Electronic Mail

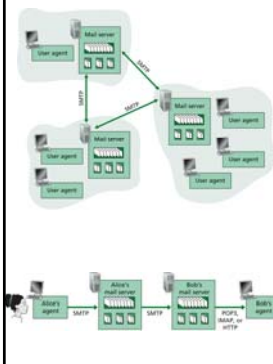
Three major components:

- User agents
- Mail servers
- Protocols
 - Outgoing email
 - SMTP
 - Retrieving email
 - POP3: Post Office Protocol [RFC 1939]
 - IMAP: Internet Mail Access Protocol [RFC 1730]



6

SMTP: Outgoing Email as a Client-Server Application



```

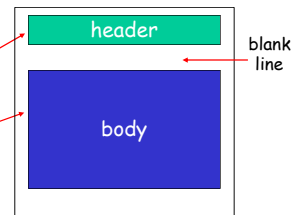
S: 220 mrl.its.yale.edu
C: HELO cyndra.yale.edu
S: 250 Hello cyndra.cs.yale.edu, pleased to meet you
C: MAIL FROM: <spoofofcs.yale.edu>
S: 250 spoofofcs.yale.edu... Sender ok
C: RCPT TO: <yry@yale.edu>
S: 250 yry@yale.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Date: Wed, 23 Jan 2008 11:20:27 -0500 (EST)
C: From: "Y. R. Yang" <yry@cs.yale.edu>
C: To: "Y. R. Yang" <yry@cs.yale.edu>
C: Subject: This is subject
C:
C: This is the message body!
C: Please don't spoof!
C:
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 mrl.its.yale.edu closing connection
    
```

7

Mail Message Data Format

SMTP: protocol for exchanging email msgs
 RFC 822: standard for text message format:

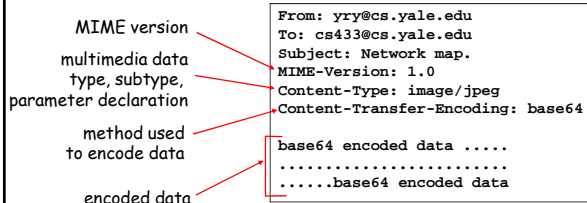
- Header lines, e.g.,
 - To:
 - From:
 - Subject:
- Body
 - the "message", ASCII characters only



8

Message Format: Multimedia Extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- Additional lines in msg header declare MIME content type



```

From: yry@cs.yale.edu
To: cs433@cs.yale.edu
Subject: Network map.
MIME-Version: 1.0
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

base64 encoded data .....
.....base64 encoded data
    
```

9

Multipart Type: How Attachment Works

```

From: yry@cs.yale.edu
To: cs433@cs.yale.edu
Subject: Network map.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Hi,
Attached is network topology map.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....base64 encoded data
--98766789--
    
```

10

POP3 Protocol: Mail Access

Authorization phase

- client commands:
 - user: declare username
 - pass: password
- server responses
 - +OK
 - -ERR

```

S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
    
```

Transaction phase, client:

- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
    
```

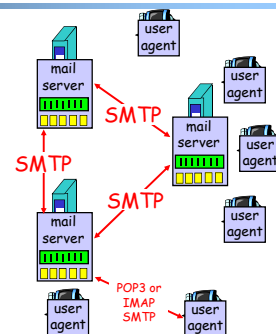
%telnet <netid>.mail.yale.edu 110
 %openssl s_client -connect pop.gmail.com:995

11

Evaluation of SMTP//POP/IMAP

Key questions to ask about a C-S application

- extensible?
- scalable?
- handle server failures (being robust)?
- security?



12

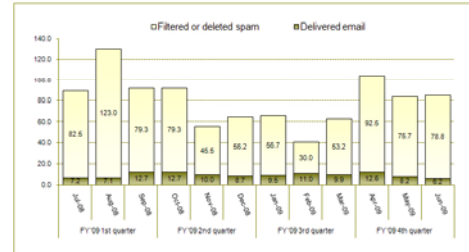
Email: Positive

- Some nice designs we can learn from the design of the email application
 - separate protocols for different functions
 - email retrieval (e.g., POP3, IMAP)
 - email transmission (SMTP)
 - simple/basic requests to implement basic control; fine-grain control through ASCII header and message body
 - make the protocol easy to read/debug/extend (analogy with end-to-end layered design?)
 - status code in response makes message easy to parse

13

Email: Negative

- Some design challenges
 - handling spam



<http://www.yale.edu/its/metrics/email/index.html>

14

Preview

- Scalability (add more email servers to help and robustness (multiple email servers serve as back up) depend on the mapping from email address to server process(es).

15

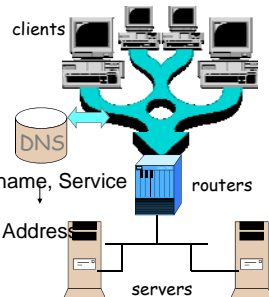
Outline

- Recap
- Email app.
- DNS

16

DNS: Domain Name System

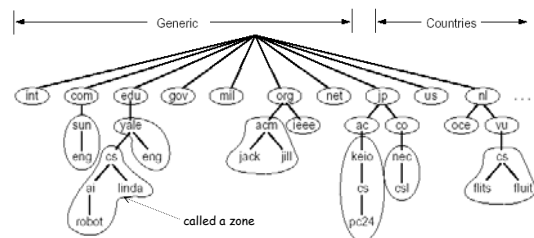
- Function
 - map between (domain name, service) to value, e.g.,
 - (www.cs.yale.edu, Addr) → 128.36.229.30
 - (cs.yale.edu, Email) → netra.cs.yale.edu
- Why use name, instead of IP address directly?



17

DNS: Domain Name System

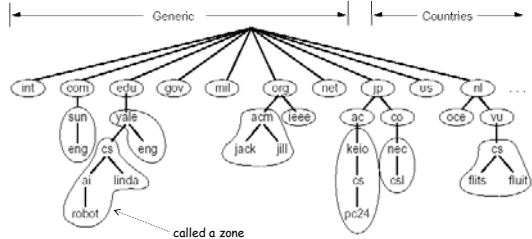
- Naming scheme
 - a hierarchical naming space to avoid name conflict



18

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



19

Root Zone and Root Servers

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

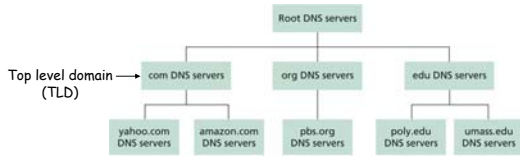


See <http://root-servers.org/> for more details

20

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?

21

DNS Message Flow: Two Types of Queries

Recursive query:

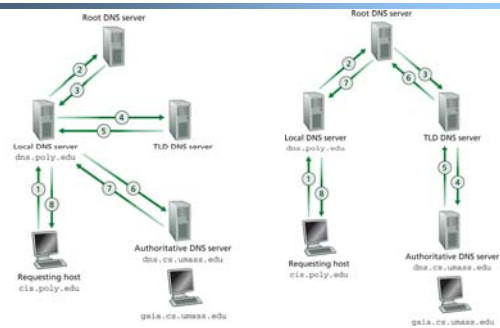
- Puts burden of name resolution on contacted name server
 - the contacted name server resolves the name completely

Iterated query:

- Contacted server replies with name of server to contact
 - "I don't know this name, but ask this server"

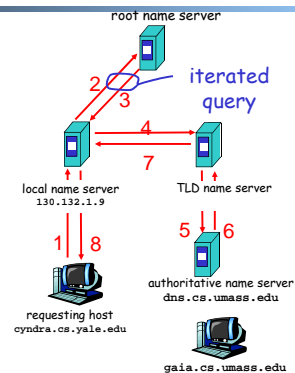
22

DNS Message Flow: Examples



Local DNS server helps requesting hosts to query the DNS system
Local DNS server is learned from DHCP, or configured, e.g. /etc/resolv.conf

DNS Message Flow: The Hybrid Case



24

DNS Records

DNS: distributed db storing 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=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

25

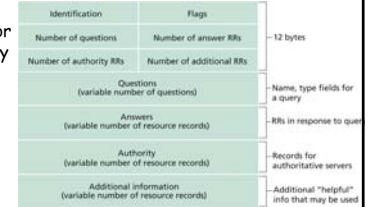
DNS Protocol, Messages

DNS protocol: over UDP/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



26

Observing DNS

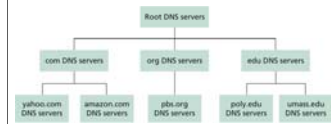
- Use the command dig (or nslookup):
 - force iterated query to see the trace:
 - %dig +trace www.cnn.com
 - see the manual for more details
- Capture the messages
 - DNS server is at port 53

27

Evaluation of DNS

Key questions to ask about a C-S application

- **scalable?**
- **handle server failures (being robust)?**
- **extensible?**
- **security?**



28

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

29

Problems of DNS

- How does a client locate a server?
- Is the application secure extensible, robust, scalable?

- 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

30

Outline

- Recap
- Email app.
- DNS
- Network application programming

31

Socket Programming

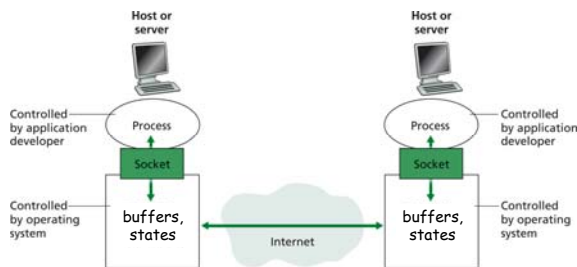
Socket API

- introduced in BSD4.1 UNIX, 1981
- Two types of sockets
 - connectionless
 - connection-oriented

socket
 an interface (a "door") into which one application process can both send and receive messages to/from another (remote or local) application process

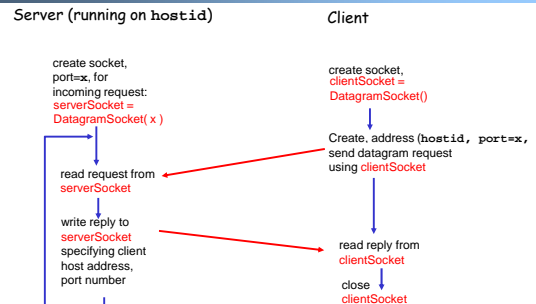
32

Big Picture



33

Connectionless: Big Picture (Java version)



34

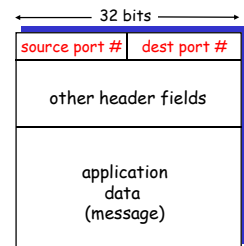
DatagramSocket (Java)

- `DatagramSocket()`
constructs a datagram socket and binds it to any available port on the local host
- `DatagramSocket(int port)`
constructs a datagram socket and binds it to the specified port on the local host machine.
- `DatagramSocket(int port, InetAddress laddr)`
creates a datagram socket and binds to the specified local 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 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.

35

How demultiplexing works

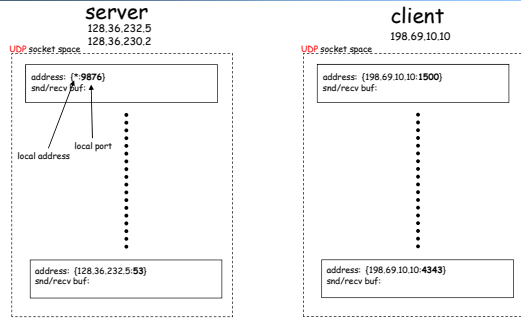
- **Dest. receives IP datagrams**
 - each datagram has source IP address, destination IP address
 - each datagram carries 1 transport-layer segment
 - each segment has source, destination port number (recall: well-known port numbers for specific applications)
- **Dest. uses IP addresses & port numbers to direct segment to appropriate socket**



TCP/UDP segment format

36

UDP Provides Multiplexing/Demultiplexing



Packet demultiplexing is based on (dst address, dst port) at dst
%netstat --udp -n -a