Traceroute - roundtrip times from source to the given hop

Traceroute to www.msu.ru (195.208.252.130), 30 hops max, 16 byte packets
1 128.4.73.100 (128.4.73.100) 3.567 ms 1.083 ms 1.150 ms
2 128.175.137.2 (128.175.137.2) 3.056 ms 0.993 ms 10.980 ms
3 sch-br-1.ge-0-1.net.msstate.edu (128.175.113.1) 1.002 ms 1.187 ms 0.814 ms
4 sch-br-2.ge-0-1.net.msstate.edu (128.175.111.1) 0.913 ms 1.326 ms 1.436 ms
5 load.msstate.inet.rrc.net (216.27.96.37) 2.453 ms 2.519 ms 2.948 ms
6 phy-02-08.buckhome.mson.net (198.32.49.97) 2.244 ms 2.487 ms 2.251 ms
7 phy-00-02.buckhome.mson.net (216.27.100.230) 2.712 ms 3.135 ms 2.712 ms
8 remote-0c8b.abloke.mson.net (216.27.103.22) 4.759 ms 4.441 ms 4.470 ms
9 ching-myymg.abóbine.ucard.edu (198.32.8.82) 42.381 ms 24.896 ms 24.852 ms
10 Chicago-084-1-1bnet.ru (195.209.14.250) 128.863 ms 128.911 ms 129.394 ms
11 Chicago-084-1-1bnet.ru (195.209.14.261) 373.397 ms 173.266 ms 174.054 ms
12 RHB-28-4-1-1bnet.ru (195.209.14.22) 196.466 ms 195.995 ms 196.342 ms
13 ge-cc-fe-0-1-1bnet.ru (195.208.248.241) 200.495 ms 197.056 ms 197.875 ms
14 web.msru.ru (195.208.252.130) 198.378 ms 199.052 ms 198.420 ms

HTTP connections

- **HTTP runs over TCP:**
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- **HTTP is "stateless"**
  - Server maintains no information about past client requests

Nonpersistent HTTP

User enters URL `www.someSchool.edu/index.html`

- **Nonpersistent HTTP**
  - At most one object is sent over a TCP connection.
  - HTTP/1.0 uses nonpersistent HTTP
  - HTTP/1.1 uses persistent connections in default mode

Persistent HTTP

User enters URL `www.someSchool.edu/index.html`

- **Persistent HTTP**
  - Multiple objects can be sent over single TCP connection between client and server
  - HTTP/1.1 uses persistent connections

Response time modeling

- Definition of round trip time (RTT): time to send a small packet to travel from client to server and back.
  - **Response time:**
    - One RTT to initiate TCP connection
    - One RTT for HTTP request and first few bytes of HTTP response to return
    - File transmission time
  - Total = 2RTT+ file transmit time
Persistent vs. Nonpersistent HTTP

- Nonpersistent HTTP:
  - Requires 2 RTTs per object
  - OS overhead for each TCP connection
  - Browsers often open parallel TCP connections to fetch referenced objects

- Persistent HTTP:
  - Server leaves connection open after sending response
  - Subsequent HTTP messages between same client/server sent over this open connection

- Persistent without pipelining:
  - Client issues new request only when previous response has been received
  - One RTT for each object

- Persistent with pipelining:
  - Default in HTTP/1.1
  - Client sends requests as soon as it encounters an reference
  - As little as one RTT for all the referenced objects

User-server state: cookies

Many major Web sites use cookies to store user-related information at user’s computer and retrieve it when needed

- Four components:
  1) Server to client: cookie header line of HTTP response message - set cookie
  2) Later, client to server: cookie header line in HTTP request message - send cookie
  3) Cookie file is kept on user’s host, managed by user’s browser
  4) Server looks up a back-end database at Web site

Example:
  - Susan access Internet always from same PC
  - She visits a specific e-commerce site for first time
  - When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping “state”

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookie file</td>
<td>usual http request msg</td>
</tr>
<tr>
<td>share: 1578</td>
<td>usual http response msg</td>
</tr>
<tr>
<td>cookie: 1678</td>
<td>set-cookie: 1678</td>
</tr>
</tbody>
</table>

One week later:

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookie file</td>
<td>usual http request msg</td>
</tr>
<tr>
<td>share: 1578</td>
<td>usual http response msg</td>
</tr>
<tr>
<td>cookie: 1678</td>
<td>cookie: 1678</td>
</tr>
</tbody>
</table>

Cookies: Pros and Cons

- What cookies can bring:
  - Authorization
  - Shopping carts
  - Recommendations
  - User session state (Web e-mail)

- Cookies and privacy:
  - Cookies permit sites to learn a lot about you
  - You may supply name and e-mail to sites
  - Search engines use redirection & cookies to learn yet more
  - Advertising companies obtain info across sites

Web caches (proxy servers)

- Goal: satisfy client request without involving origin server
- User sets browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - Object in cache: cache returns object
  - Else cache requests object from origin server, then returns object to client

More about Web caching

- Cache acts as both client and server
- Typically cache is installed by ISP (university, company, residential ISP)
- Why Web caching?
  - Reduce response time for client request
  - Reduce traffic on an institution’s access link
  - Internet dense with caches enables “poor” content providers to effectively deliver content
Caching example
Assumptions
- Average object size = 100,000 bits
- Avg. request rate from institution’s
  browsers to origin servers = 15/sec
- Delay from institutional router to any origin
  server and back to router = 2 sec
Without caching
- LAN utilization = 15%
  (100kb*15/10Mbps)
- Access link utilization = 100%
- Delay per request = 2 sec +
milliseconds

Possible solution
- Suppose hit rate is 40%
With caching
- 40% requests will be satisfied
  almost immediately
- 60% requests satisfied by origin
  server
- Access link utilization = 60%
  (60%*100kb*15/10Mbps)
- Delay = 60%*(2sec+milliseconds)
  40%*milliseconds ~ 1.2 sec

Caching example

HTTP Conditional GET
- Goal: cache asks the server
  not to send object if it
  wasn’t modified since last
  time this cache has
  requested this object
- Cache: specify date of
  cached copy in HTTP request
  If-modified-since: <date>
- Server: response contains no
  object if cached copy is up-
to-date:
  HTTP/1.0 304 Not Modified

FTP: the file transfer protocol
- Transfer file to/from remote host
- Client/server model
  - client: side that initiates transfer
    - server: remote host
- FTP: RFC 959
- FTP server: port 21

Separate control, data connections
- FTP client contacts FTP
  server at port 21, specifying TCP
  as transport protocol
- Client logs on over
  control connection
- Client browses remote
directory over control
connection.
- When server receives a
  command for a file
  transfer, it opens a TCP
  data connection to client
- After transferring one
  file, server closes
  connection.
- Server opens a second TCP
  data connection to
  transfer another file.
- FTP server maintains
  “state”: current directory,
  earlier authentication
FTP commands, responses

Sample commands:
- Sent as ASCII text over control channel
- USER username
- PASS password
- LIST return list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores (puts) file onto remote host

Sample return codes:
- Status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file

Electronic Mail

Three major components:
- User agents
- Mail servers
- Simple mail transfer protocol: SMTP

User Agent
- Composing, editing, reading mail messages
- E.g., Eudora, Outlook, elm, Netscape Messenger
- Outgoing messages sent to user mailbox
- Incoming messages stored on server

Mail servers

- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

SMTP Protocol [RFC 2821]

- Uses TCP to reliably transfer email message from client to server, port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure
- Command/response interaction
  - commands: ASCII text
  - response: status code and phrase
- Messages must be in 7-bit ASCII

Alice sends message to Bob

1) Alice uses user agent (UA) to compose message for bob@someschool.edu
2) Alice’s UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message

Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
Try SMTP interaction for yourself:

- `telnet mail.cis.udel.edu 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands
- above lets you send email without using email client (reader)

Mail message format

- Header lines, e.g.,
  - `To`:
  - `From`:
  - `Subject`:
  - `different from SMTP command`!
- Body
  - "the "message", ASCII characters only"

Mail access protocols

- **SMTP**: delivery/storage to receiver’s server
- **Mail access protocol**: retrieval from server
  - `POP`: Post Office Protocol (RFC 1939)
    - Authorization (agent <-> server) and download
  - `IMAP`: Internet Mail Access Protocol (RFC 1730)
    - More features (more complex)
    - Manipulation of stored mail on server
  - `HTTP`: Hotmail, Yahoo Mail, etc.

SMTP: final words

- **SMTP uses persistent connections**
- **SMTP requires message (header & body) to be in 7-bit ASCII**
- **SMTP server uses CRLF, CR/LF to determine end of message**
- **Comparison with HTTP**:
  - `HTTP`: pull
  - `SMTP`: push
  - Both have ASCII command/response interaction, status codes

Multimedia extensions

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

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

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

POP3 protocol

**authorization phase**

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

**transaction phase**, client:

- `list`: list message numbers
- `retr`: retrieve message by number
- `dele`: delete
- `quit`
## POP3 vs IMAP

<table>
<thead>
<tr>
<th>POP3</th>
<th>IMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Previous example uses</td>
<td>- Keep all messages in one place: the server</td>
</tr>
<tr>
<td>&quot;download and delete&quot;</td>
<td>- Allows user to organize messages in folders</td>
</tr>
<tr>
<td>mode</td>
<td>- IMAP keeps user state across sessions:</td>
</tr>
<tr>
<td>- Bob cannot re-read</td>
<td>- Names of folders and mappings between</td>
</tr>
<tr>
<td>e-mail if he changes</td>
<td>message IDs and folder name</td>
</tr>
<tr>
<td>client</td>
<td></td>
</tr>
<tr>
<td>- &quot;Download-and-keep&quot;:</td>
<td></td>
</tr>
<tr>
<td>copies of messages on</td>
<td></td>
</tr>
<tr>
<td>different clients</td>
<td></td>
</tr>
<tr>
<td>- POP3 is stateless</td>
<td></td>
</tr>
<tr>
<td>across sessions</td>
<td></td>
</tr>
</tbody>
</table>