Applications I

TkL Markus Peuhkuri

27th Feb 2008

Lecture topics

- You should understand
  - paradigms
    - client-server
    - peer-to-peer
  - how to build services on top of transport protocols
  - some common application level protocols
- Topics
  - Application model
  - Example application: web

What are applications

- "Application" is a service or function
  - email
  - IM, VoIP, video conferencing
  - web, file transfer, file access
  - remote login, access
  - video and audio streaming
  - MMORPG, MMOFPS, MMORTS
  - grid computing
- Not an application program
  - Firefox, Word

Internet application model

- Network stupid, end systems intelligent
- Programs developed for end systems
  - communicates over network
  - e.g. web browser and web server
- No modification for network devices
  - core network intact ⇒reliability
  - rapid development
  - all bytes equal (in good and worse)

Telco communication app. model

- Network intelligent, end system stupid
- Functionality build into network
  - network devices in important role
  - client devices controlled by network
  - e.g. MMS
- Core network modified
  - additional devices or functions
  - slower and more expensive development
  - possibility to charge different amount of money for a bit
    - 1 SMS: 0.08 € ⇒599 € / MB
    - 1 minute voice call: 0.08 € ⇒0.86 € / MB
    - 20 MB GPRS/month: 5.9 € ⇒0.30 € / MB
    - "unlimited" = 3 GB/month: 14.9 € ⇒0.005 € / MB
  - also different quality can be provided

Application architectures

- Client-server
- Peer-to-peer
  - also server-to-server
- Hybrid usage

Client-server architecture

- Server
  - always available
  - permanent IP address or name
  - possibly multiple redundant servers
- Clients
  - communicate with server: know it based
    - IP address
    - name lookup
    - other reference
  - connected only when used
  - IP address not important, unless used for authentication
    - often dynamic
  - communication between clients via server

Peer-to-peer architecture

- No single, always on server
  - nodes may leave network
  - change IP address
- End systems communicate directly to each other
- How to discover other systems
  - a list of potential peers
  - a single peer is sufficient to bootstrap

Combined client-server p2p

- Server used as a directory or registry
  - works as initial contact point
  - helps finding peers
- Data transfer directly from peer to peer
- Load on server keeps limited

Processes

- Application communication is between processes
- Processes may be located in
  - same host
  - on multiple hosts
- In theory, it should not make difference where the processes are
- In practice, for performance reasons there may be a difference
  - processing
  - bandwidth
  - delay
- Client initiates communication
- Server waits for clients to contact
- In P2P, a node will have both client and server functionality

Programming communications

- The most common paradigm: sockets
- Like a hose connecting processes
  - may leak
  - throughput may wary
- Characteristics of hose depend on chosen transport protocol and tuned based on parameters
- Process addressing: refer to multiplexing on transport lecture

Defining protocols

- How communication happens
  - when to send a message
- What message types are used
  - parameters for each message
  - fields
- Message syntax
  - presentation on wire
  - encoding, delimiting
- Public vs. proprietary protocols
  - well-documented, inter-operable
  - obscure, reverse-engineered

Application requirements

- Data loss
  - some loss may be tolerated (e.g. real-time voice)
  - often, no losses or bit errors (file transfer)
- Delays and timing
  - no exact timing often needed
  - conversation, real-time applications
- Throughput
  - some applications rigid: they need minimum throughput to be any good; like media applications
  - most applications "elastic", adapt on available bandwidth
- Security needs

Service needed from transport

Application	Tolerates loss	throughput	delay
file transfer	no	elastic	yes
e-mail	no	elastic	yes
web browsing	no	elastic	yes
real-time audio	yes	5 kbit/s – 1 Mbit/s	200 ms
real-time video	yes	10 kbit/s – 5 Mbit/s	200 ms
stored audio/video	yes	see above	upto 3 – 5 s
MMOxxx games	yes	kbit/s –	200 ms or less
IM	no	elastic	partly

IP transport protocols

UDP Unreliable datagram service

- point-to-multipoint
- no connection, reliability, flow or congestion control, timing, throughput guarantee, security

TCP Byte stream service

- connection-oriented, point-to-point, flow and congestion controlled, full-duplex, buffered, reliable, in-order
- no timing, throughput guarantee, security

Applications and protocols

- UDP
  - Simple Network Management Protocol (SNMP) [2]
  - Dynamic Host Configuration Protocol (DHCP) [6] (BootP[5])
  - Trivial File Transfer Protocol (TFTP) [16]
  - Remote Procedure Call (RPC)
  - Domain Name Server (DNS) [11]
  - Network Time Protocol (NTP) [10]
  - Real-time Transport Protocol (RTP) [9]
- TCP

  - many applications text-based
  - Virtual Terminal (telnet) [15]
  - File Transfer Protocol (FTP) [14]
  - Simple Mail Transfer Protocol (SMTP) [13]
  - Post Office Protocol (POP) [12]
  - Internet Message Access Protocol (IMAP) [4, 3]
  - Hypertext Transfer Protocol (HTTP) [1, 8]
  - X Window System (X11)

Application: Web

- Web is about objects
- Web page is a collection of objects
  - main hypertext markup language (HTML) document
  - cascading style sheets (CSS)
  - javascript files
  - images (png, jpeg), objects (flash, java)
  - other html documents (iframe)
  - data files parsed by active content (javascript, flash, java)
- Objects identified by URLs:
  `http://www.example.com/fi/intro.html`

  http scheme: defines name space

  www.example.com hostname

  fi/intro.html local path

Transferring objects

- HyperText Transfer Protocol: a misleading name for an object request protocol
- Web not limited to http: can use any defined protocol that has URL syntax to request data files
  - https
  - ftp
  - file
  - nfs
  - ...
- Each URL defined by its rules
  `http://www.example.com/fi/intro.html`

  http scheme: defines name space

  www.example.com hostname

  fi/intro.html local path

Hypertext Transfer Protocol [1, 7]

- Request-response client-server protocol
  - client: web browser (or a program requesting information: machine2machine communication)
  - server: serves object that are static or generated
- Runs on top of TCP
  1. server listens on port (default 80)
  2. client establishes TCP connection with server
  3. client sends HTTP request
  4. server responds
  5. TCP connection is closed
- Stateless
  - no information about past requests
  - makes implementation simple

Persistent HTTP connections

- Original HTTP/1.0 requested one object with one TCP connection
  - 2*round trip time for connection
  - connection overhead in both client and server
  - many, parallel TCP connections
- HTTP/1.1 supports persistent HTTP
  - same connection used for multiple requests
  - request pipelining
  - just one RTT delay for all requests + transmission

HTTP requests

- Human readable format

```
GET /u/puhuri/ HTTP/1.0
Accept: text/html
User-Agent: Chinzilla/9.2 (fooniz)
Host: www.netlab.tkk.fi
```

- Request line
  - GET
  - POST
  - HEAD
  - PUT
  - DELETE
- Header fields
  - Name: Value

HTTP responses

```
HTTP/1.1 200 OK
Date: Mon, 08 Nov 1999 13:35:16 GMT
Server: Apache/1.3.4 (Unix)
Connection: keep-alive
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML><HEAD><TITLE>Markus Peuhkuri: home page</TITLE> ...
```

- Return code + clarifying text

  1yz positive initial response
  2yz operation successfull (200 OK)
  3yz redirect (301 Moved)
  4yz client error (404 Not found)
  5yz server error (505 Version not supported)

- Header lines
- Object data

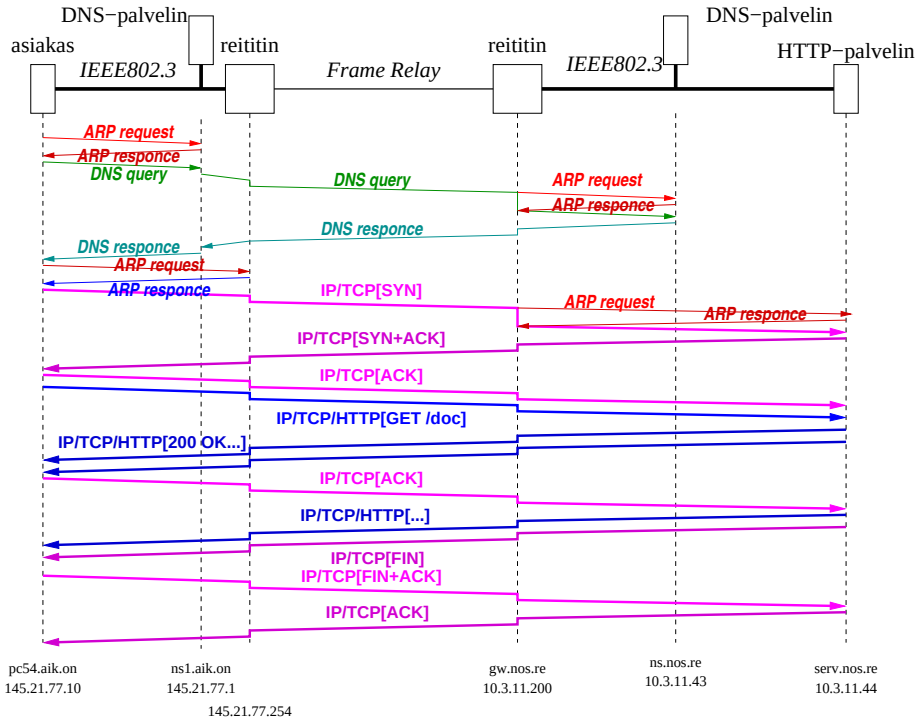
How to keep state

- HTTP as protocol stateless
- Some applications want to carry state
  - shopping cart
  - authorization
  - preferences, settings
- Could be encoded into URL
  - long URLs
  - privacy problems if links shared
- Cookies
  - `Set-Cookie` header by server
  - `Cookie` header in requests
  - lifetime can be defined

Web caching

- No need to download from server if not changed
  $\Rightarrow$ saves bandwidth and faster action
- Conditional GET
  - use `If-modified-since` header
  - response `304 Not Modified`
- Cache at local memory, disk
- Cache at network
  - shared by all users of local network
  - hit rate about 30 %
  - popular content (like youtube videos) 75 %
- Only “public” (non-authenticated) cached
- HTTPS cannot be cached

HTTP connection



Summary

- Internet application model: functionality at end systems
- Client-server or p2p
- HTTP is generic object request protocol
- Next week: some other applications and more on socket programming

Viitteet

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, May 1996. RFC 1945. URL:`http://www.ietf.org/rfc/rfc1945.txt`.

[2] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. *Simple Network Management Protocol (SNMP)*, May 1990. RFC 1157. URL:`http://www.ietf.org/rfc/rfc1157.txt`.

[3] M. Crispin. *IMAP4 Compatibility with IMAP2bis*, December 1996. RFC 2061. URL:`http://www.ietf.org/rfc/rfc2061.txt`.

[4] M. Crispin. *Internet Message Access Protocol - Version 4rev1*, December 1996. RFC 2060. URL:`http://www.ietf.org/rfc/rfc2060.txt`.

[5] W.J. Croft and J. Gilmore. *Bootstrap Protocol*, September 1985. RFC 951. URL:`http://www.ietf.org/rfc/rfc951.txt`.

[6] R. Droms. *Dynamic Host Configuration Protocol*, March 1997. RFC 2131. URL:`http://www.ietf.org/rfc/rfc2131.txt`.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, January 1997. RFC 2068. URL:`http://www.ietf.org/rfc/rfc2068.txt`.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999. RFC 2616. URL:`http://www.ietf.org/rfc/rfc2616.txt`.

[9] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, January 1996. RFC 1889. URL:`http://www.ietf.org/rfc/rfc1889.txt`.

[10] D. Mills. *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, March 1992. RFC 1305. URL:`http://www.ietf.org/rfc/rfc1305.txt`.

[11] P.V. Mockapetris. *Domain names - concepts and facilities*, November 1987. RFC 1034. URL:`http://www.ietf.org/rfc/rfc1034.txt`.

[12] J. Myers and M. Rose. *Post Office Protocol - Version 3*, May 1996. RFC 1939. URL:`http://www.ietf.org/rfc/rfc1939.txt`.

[13] J. Postel. *Simple Mail Transfer Protocol*, August 1982. RFC 821. URL:`http://www.ietf.org/rfc/rfc821.txt`.

[14] J. Postel and J. Reynolds. *File Transfer Protocol*, October 1985. RFC 959. URL:`http://www.ietf.org/rfc/rfc959.txt`.

[15] J. Postel and J.K. Reynolds. *Telnet Protocol Specification*, May 1983. RFC 854. URL:`http://www.ietf.org/rfc/rfc854.txt`.

[16] K. Sollins. *The TFTP Protocol (Revision 2)*, July 1992. RFC 1350. URL:`http://www.ietf.org/rfc/rfc1350.txt`.