Application Level Protocol Design

Concurrent & Distributed Software Systems

Application Layer Protocol Design

- Steps in design
  - Services
  - Protocol Data Unit (PDU) structure and encoding
  - Protocol
  - Client, Server, interaction with environment (DNS, NFS, etc.)

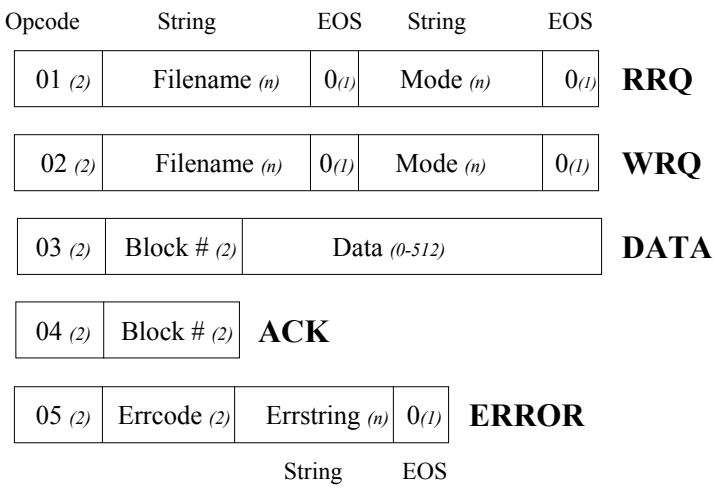
Trivial FTP (TFTP)

- RFC 1350
- a simple file transfer protocol, often used for booting from a remote file system
- no access control, directory manipulation, etc.
- UDP transport
- files are netascii or binary

Services & PDU design

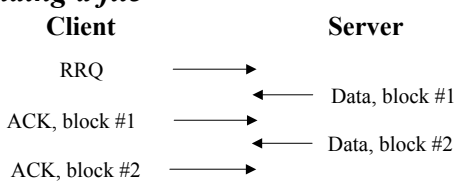
- Services
  - Read, write a file
- PDUs
  - RRQ, read request: read a file
  - WRQ, write request: write a file
  - DATA
  - ACK
  - ERROR

PDU structure and encoding

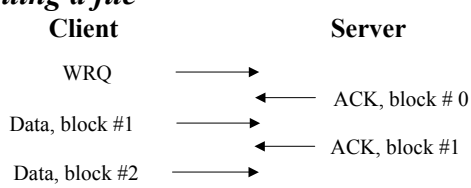


TFTP Protcol

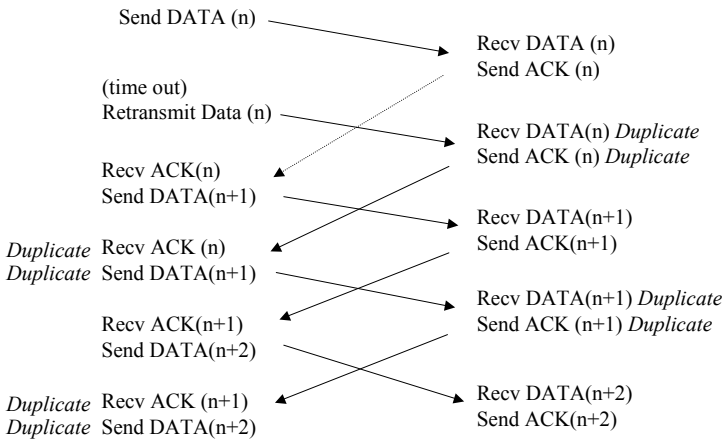
Reading a file



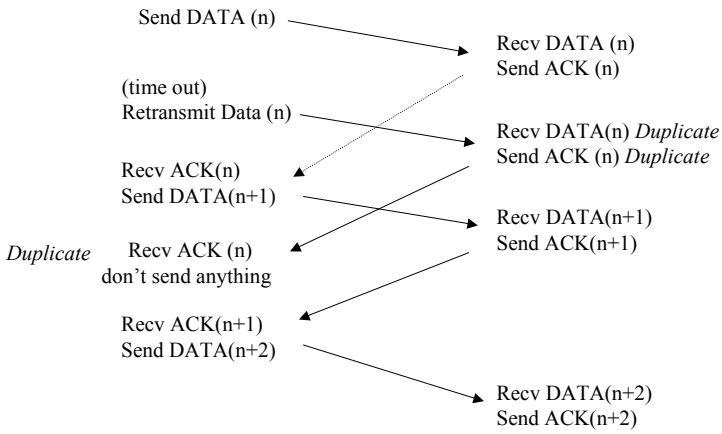
Writing a file



Sorcerer's apprentice syndrome



Correction



Implementation Issues

- How does server keep track of client address?
  - New socket created by child process on server side
  - Child process *binds* socket to a local address
  - All subsequent messages from client sent to new address

Implementation cont'd

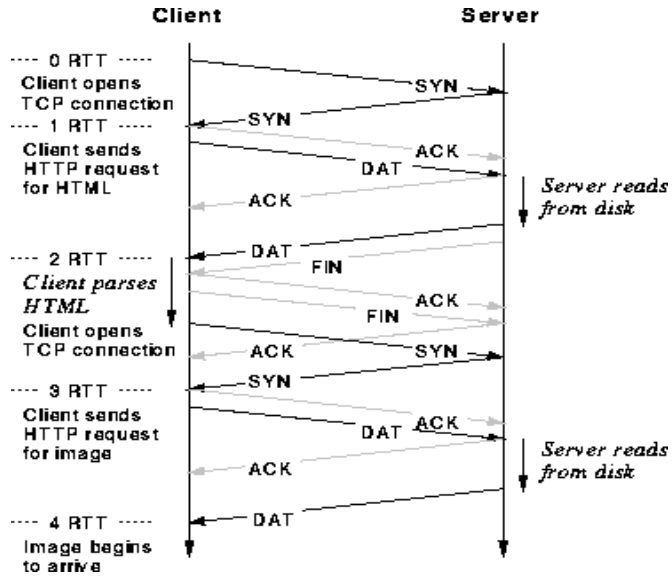
- Data formats
  - netascii -responsibility of the client & server to convert from netascii to local representation (and vice versa)
- Protocol processing
  - Finite state machine
- Security

HTTP 1.0

- Services
  - read web pages, append to web pages, write pages, etc.
  - GET, HEAD, POST, PUT, DELETE, etc.
- PDU design and encoding
  - ASCII request followed by MIME-like response
  - format specified in BNF

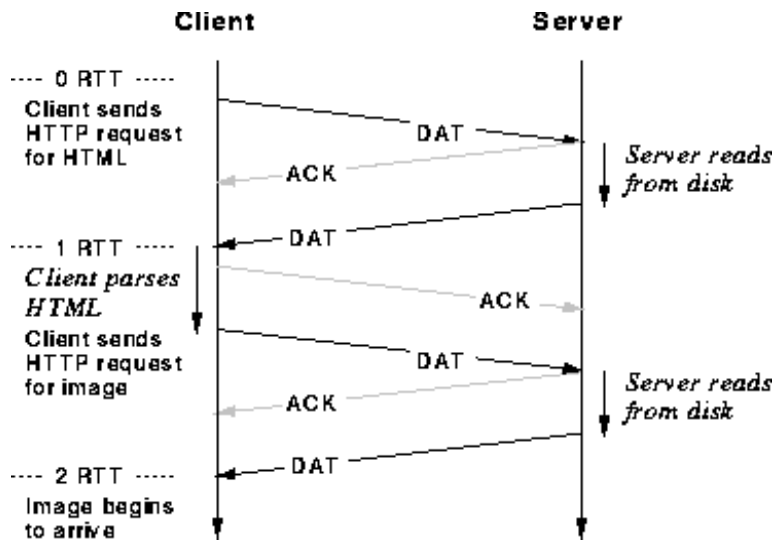
HTTP 1.0 cont'd

- Protocol
  - TCP used as transport
  - new connection for every file retrieved
    - poor performance
- Other issues
  - active content, caching, proxy servers, security



HTTP 1.1

- Several changes
  - Persistent connections - default behavior
  - Host header field in request
  - Connection header
  - See also link on class web page for a more detailed discussion
- Assignment
  - GETLIST method (not part of HTTP 1.1)
    - server can supply a list of embedded files as an optional header in the response to a GET request
    - *you have to design the request/response formats*



Application-level Protocols

- Session-layer
  - communication between processes (as opposed to hosts)
  - Example:
    - error-recovery and check-pointing for long-lived connections
    - creation and management of synchronized streams for audio and video

Presentation Layer

- Handle issues related to different data representations on communicating hosts
  - big endian vs little endian, 32 bit vs 64 bit, different formats for characters, etc.
  - Usually handled in one of two ways
    - Canonical representation of data
      - XDR, ASN.1, netascii (TFTP), CORBA CDR
      - ASCII (HTTP)
      - Java object serialization
    - "receiver makes right"
- Also handles encryption and decryption

CORBA CDR for constructed types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

CORBA CDR message

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on__"	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

Indication of Java serialized form

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles

MIME

- Internet email standard (RFC 822) specifies ASCII message format
- Multipurpose Internet Mail Extensions
  - continue to use the RFC 822 format but allow non-ASCII messages
  - New message headers
  - binary messages encoded using **base64** encoding or **quoted-printable** encoding