Communication over the Network

Principles

Services provided by Internet transport protocols

TCP service:
- Connection-oriented: setup required between client, server
- Reliable transport between sending and receiving process
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network overloaded
- Does not provide: timing, minimum bandwidth guarantees

UDP service:
- Unreliable data transfer between sending and receiving process
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?

WWW: the HTTP protocol

HTTP: hypertext transfer protocol
- WWW's application layer protocol
- Client/server model
  ❍ Client: browser that requests, receives, "displays" WWW objects
  ❍ Server: WWW server sends objects in response to requests

WWW Server
PC running Explorer

http message format: request

Two types of http messages: request, response

http request message:
- ASCII (human-readable format)

GET /somemdir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr

(extra carriage return, line feed)

HTTP Beispiel

Annahme der Benutzer gibt folgende URL ein
www.someSchool.edu/someDepartment/home.index

1b. http Server auf Host www.someSchool.edu wartet auf TCP Verbindungen auf Port 80. "akzeptiert" Verbindung, benachrichtigt Client
2. http Client sendet Request message (mit URL) in den TCP Verbindungs's Socket
3. http Server erhält Request message, bildet Response message mit angefragtem Objekt (someDepartment/home/index), sendet Message in den Socket

http Beispiel (2)

6. Wiederholung von Schritten 1-5 für jedes der 10 jpeg Objekte

Nicht-persistente Verbindungen: ein Objekt in jeder TCP Verbindung
- Einige Browsers benutzen mehrere TCP Verbindungen gleichzeitig – eine pro Objekt
- Persistente Verbindungen: mehrere Objekte können über eine TCP Verbindung transferiert werden
http request Nachricht: generelles Format

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td>request line</td>
</tr>
</tbody>
</table>

Entity Body

http message format: reply

HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ....
Content-Length: 6821
Content-Type: text/html

data data data data data ...

status line
(protocol status code status phrase)
header lines
data, e.g., requested html file

http reply status codes
In first line in server — client response message.
A few sample codes:

200 OK
request succeeded, requested object later in this message

301 Moved Permanently
requested object moved, new location specified later in this message (Location:)

400 Bad Request
request message not understood by server

404 Not Found
requested document not found on this server

505 HTTP Version Not Supported

Selber ausprobieren von http (client side)
1. Telnet zu Ihrem favorisierten WWW Server:
telnet www.eurecom.fr 80

2. Eintippen eines GET http Request:
GET /~ross/index.html HTTP/1.0
Indem man diese eintippet (drücke enter zweimal), sendet man diesen minimalen (aber kompletten) GET Request zum http Server

3. Beachte die Response Nachricht, die vom http Server gesandt wird!

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another

controlled by application developer
controlled by operating system

TCP with buffers variables

host or server

Internet
### Socket programming with TCP

**Client must contact server**
- Server process must first be running
- Server must have created socket (door) that welcomes client's contact

**Client contacts server by:**
- Creating client-local TCP socket
- Specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
- Allows server to talk with multiple clients

**Application viewpoint**
- TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

---

### Example client-server app:

- Client reads line from standard input (\texttt{inFromUser stream}), sends to server via socket (\texttt{outToServer stream})
- Server reads line from socket
- Server converts line to uppercase, sends back to client
- Client reads, prints modified line from socket (\texttt{inFromServer stream})

**Input stream:** sequence of bytes into process
**Output stream:** sequence of bytes out of process

---

### Client/server socket interaction: TCP

**Server** (running on hostid)

- Create socket, port=\texttt{x}, for incoming request:
  
  ```
  welcomeSocket = ServerSocket()
  ```

- Wait for incoming connection request
  
  ```
  connectionSocket = welcomeSocket.accept()
  ```

**Client**

- Send request using \texttt{clientSocket}
  
  ```
  connect(TO_SERVER, $paddr)
  ```

---

### Basic Elements

- **Packing of host and port into C—like structure**
  
  ```
  use Socket;
  $packed_ip = inet_aton("208.146.240.1");
  $socket_name = sockaddr_in($port, $packed_ip);
  ```

- **Extraction of host and port out of structure**
  
  ```
  ($port, $packed_ip) = sockaddr_in($socket_name);
  ```

- **Manipulation of IP addresses**
  
  ```
  $ip_address = inet_ntoa($packed_ip);
  $packed_ip = inet_aton("204.148.40.9");
  $packed_ip = inet_aton("www.oreilly.com");
  ```

---

### Socket Programming

**In Perl**

```perl
use Socket;
socket(TO_SERVER, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
$remote_addr = inet_aton($remote_host) or die "Couldn't convert $remote_host into an Internet address: $!
";
$paddr = sockaddr_in($remote_port, $remote_addr);
connect(TO_SERVER, $paddr) or die "Couldn't connect to $remote_host:$remote_port : $!
";
print TO_SERVER "Why don't you call me anymore?
"
close(TO_SERVER);
```
TCP Client (alternative)

use IO::Socket;
Sock = IO::Socket::INET->new(PeerAddr => $remote_host,
PeerPort => $remote_port,
Proto => "tcp",
Type => SOCK_STREAM)
or die "Couldn't connect to $remote_host:$remote_port : $@
";
# ... do something with the socket
print Sock "Why don't you call me anymore?\n"
$answer = <Sock>
# and terminate the connection when we're done
close(Sock);

TCP Sockets SOCK_STREAM

# Es ist möglich den Port und die Adresse zu kombinieren
$client = IO::Socket::INET->new("www.yahoo.com:80")
or die $@;

# Aufpassen: Return Werte nach Fehler: undef and $@
$s = IO::Socket::INET->new(PeerAddr => "Does not Exist",
PeerPort => 80,
Type => SOCK_STREAM)
or die $@;

# Verringern des TCP_WAIT Timeouts
$s = IO::Socket::INET->new(PeerAddr => "bad.host.com",
PeerPort => 80,
Type => SOCK_STREAM,
Timeout => 5 ) or die $@;

TCP Server (low-level)

use Socket;
# make the socket
socket(SERVER, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
# so we can restart our server quickly
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1);
# build up my socket address
$my_addr = sockaddr_in($server_port, INADDR_ANY);
bind(SERVER, $my_addr) or die "Couldn't bind to port $server_port : $!
"
# establish a queue for incoming connections
listen(SERVER, SOMAXCONN) or die "Couldn't listen on port $server_port :
$!
"
# accept and process connections
while ($client_address = accept(CLIENT, SERVER)) {
# do something with CLIENT
}
close(SERVER);

Accept

# accept() takes 2 filehandles as argument: remote client and remote server
# Returns port and IP address of the client
use Socket;
socket(SERVER, ...);
bind(...);
listen(...);
while ($client_address = accept(CLIENT, SERVER)) {
($port, $packed_ip) = sockaddr_in($client_address);
$client_ip = inet_ntoa($packed_ip);
# do as thou wilt
}

TCP Server (Alternative)

use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => $server_port,
Type => SOCK_STREAM,
Reuse => 1,
Listen => 10) or die "Can't bind to port $server_port : $@
"
while ($client = $server->accept()) {
# $client is the new connection
}
close($server);

Sending/Receiving of Data

# Print or <>
print SERVER "What is your name?\n";
chomp ($response = <SERVER>);
# use send() and recv()
defined (send(SERVER, $data_to_send, $flags)) ||
die "Can't send : $!
";
defined(recv(SERVER, $data_read, $maxlen, $flags)) ||
die "Can't receive: $!
";
# or use IO::Socket methods
use IO::Socket;
---
$server->send($data_to_send, $flags) ||
die "Can't send: $!
"
$server->recv($data_read, $flags) ||
die "Can't recv: $!
";
Socket Programming with UDP

UDP: no "connection" between client and server

- No Handshake
- Sender explicitly specifies destination IP address and port number
- Receiver has to extract IP address and port number of the sender out of the received datagram

UDP: transmitted data can arrive out of order or be lost entirely

Application perspective

UDP provides unreliable transfer of groups of bytes ("datagrams") between sender and receiver

Client/server socket interaction: UDP

**Server** (running on hostid)

- create socket, portname, for incoming request
  - serverSocket = DatagramSocket()
- read request from serverSocket
- write reply to serverSocket specifying client host address, port number

**Client**

- create socket, clientSocket = DatagramSocket()
- Create, address (hostid, port=x)
  - send datagram request using clientSocket
- read reply from clientSocket
- Client

UDP Client (low-level)

```perl
# Creation of UDP sockets
socket(SOCKET, PF_INET, SOCK_DGRAM, getprotobyname("udp"))
or die "socket: $!";
# Sending a message to host $HOSTNAME and port $PORTNO
$packed_ipaddr = inet_aton($HOSTNAME);
$destaddr = sockaddr_in($PORTNO, $packed_ipaddr);
send(SOCKET, $MSG, 0, $destaddr) == length($MSG)
or die "cannot send to $HOSTNAME($PORTNO): $!";
# Receiving data of no more than $MAXLEN bytes
$packed_addr = recv(SOCKET, $MSG, $MAXLEN, 0) or die "recv: $!";
($portno, $ipaddr) = sockaddr_in($packed_addr);
$hostname = gethostbyaddr($ipaddr, AF_INET);
print "$hostname : $ipaddr:$portno said '$MSG'
```

UDP Server (alternativ)

```perl
# Creation of a UDP Socket (anonymous)
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => $server_port,
Proto     => "udp")
or die "Couldn't be a udp server on port $server_port : $@"
# Receiving data from socket
while ($him = $server->recv($datagram, $MAX_TO_READ,
$flags)) {
# do something
}
```

UDP Server (Example)

Wait for messages and sends another message based on the last received one

```perl
#!/usr/bin/perl –w
use strict; use IO::Socket;
my($sock, $oldmsg, $newmsg, $hisaddr, $hishost, $MAXLEN, $PORTNO);
$MAXLEN = 1024;  $PORTNO = 5151;
$sock = IO::Socket::INET->new(LocalPort => $PORTNO, Proto => 'udp')
or die "socket: $@"
print "Awaiting UDP messages ($PORTNO)
oldmsg = "Starting message."
while (defined($sock->recv($newmsg, $MAXLEN))) {
my($port, $ipaddr) = sockaddr_in($sock->peername);
$hishost = gethostbyaddr($ipaddr, AF_INET);
print "Client $hishost said " $newmsg "
$sock->send($oldmsg, 0, sockaddr_in($port, $ipaddr));
oldtmsg = "$hishost $newmsg";
}
```

Socket Programming using Select

Example server application:

- Server wants to handle multiple TCP connections simultaneously
- But server blocks with every recv() call for one connection while there might be data ready on another one

Solution: select()
Select

select() determines from which file handles one can read (write) without blocking or where exception are pending.

Arguments for select():
- Bitmask readable, bitmask writeable, bitmask exceptions, Timeout

$rin = "\*";
vec($rin, fileno(SOCKET), 1) = 1;
# mark SOCKET in $rin
# repeat calls to vec() for each socket to check

$timeout = 10;
# wait ten seconds
$nfound = select($rin, undef, undef, $timeout);

if (vec($rin, fileno(socket),1)){
  # data to be read on SOCKET
}

Select (alternative)

use IO::Select;

$select = IO::Select->new();
$select->add(FROM_SERVER);
$select->add(*STDIN);

@read_from = $select->can_read($timeout);
foreach $handle (@read_from) {
  if($handle == *STDIN) {
    # read from keyboard
  } elsif($handle == *FROM_SERVER) {
    # read the pending data from $socket
  }
}

Where to find documentation

man perlpc
perldoc -f {socket, bind, listen, send, recv}
perldoc Socket
perldoc IO::Socket
perldoc IO::Select