Introduction to TCP/IP Sockets

ECE 454

Stefan Birrer

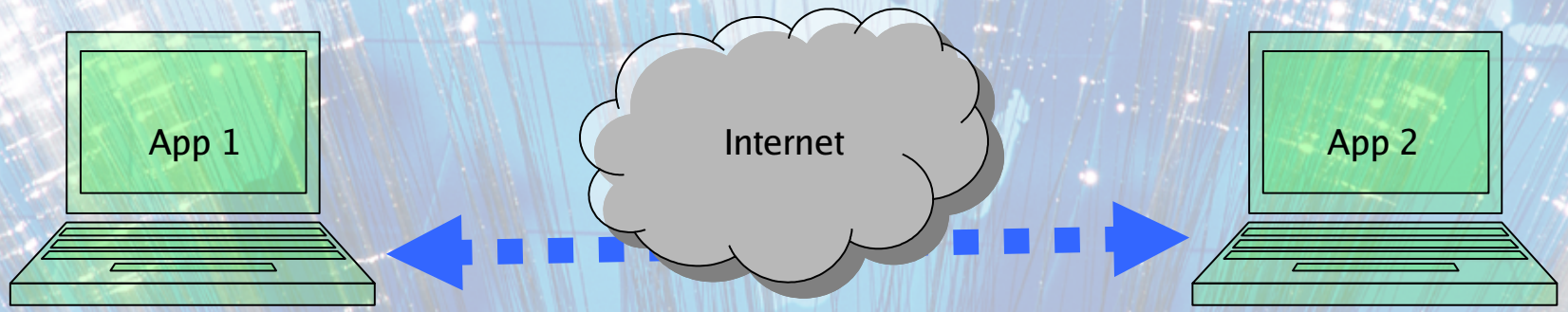
1/23/2006

Slides with minor modifications from Sasha Jevtic

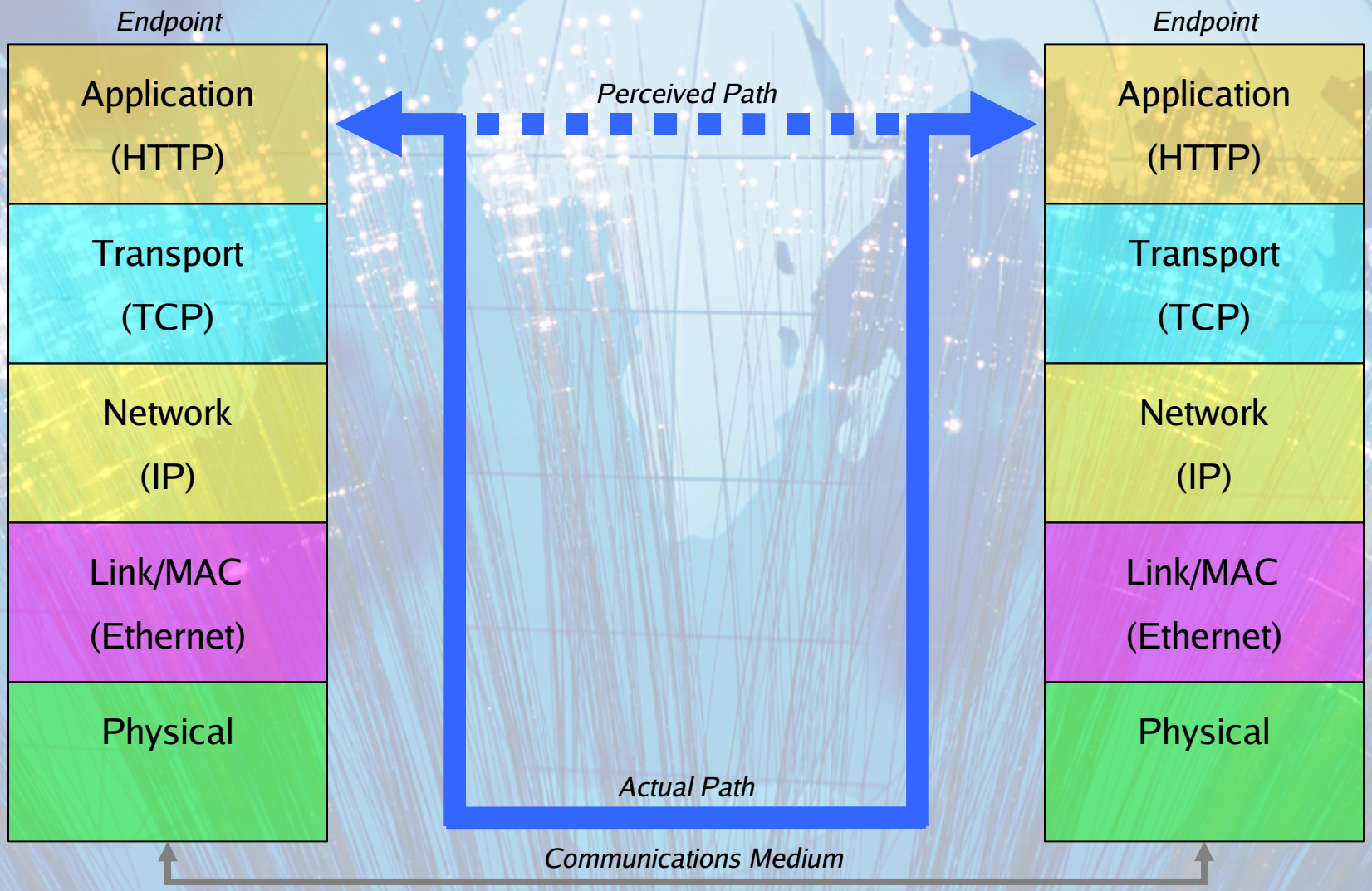
Agenda

- The protocol stack
- What is a TCP connection?
- The Domain Name System
- Socket: just another file?
- Client vs. server
- Socket client API
- Socket server API
- Compiling sockets code
- Multi-connection servers
- Multi-platform issues

Objective



Protocol Stack: TCP/IP Model



Application Protocols

Client: How are you?
Server: Fine, and yourself?
Client: Excellent.
Server: Shall we meet for dinner?
Client: Yes, 6:30PM would be good.
Server: Ok, see you then; must go.

Application Protocols: HTTP

```
Client: GET index.html HTTP/1.0
Client: -- request complete; please begin serving --
Server: HTTP/1.0 200 OK
Server: Content-type: text/html
Server: Content-length: 83
Server: -- header complete; starting document --
Server: <HTML>
Server: <HEAD>
Server: <TITLE>Hello</TITLE>
Server: </HEAD>
Server: <BODY>
Server: What's going on?
Server: </BODY>
Server: </HTML>
Server: -- document complete; done --
```

What is a TCP Connection?

Source IP Addr. (129.105.6.184)	Dest. IP Addr. (164.43.121.54)	Protocol (TCP)	Source Port (5501)	Destination Port (80)

- IP Address: 32-bit value
  - Dotted Decimal: 129.105.6.184
  - Bin: 10000001 01101001 00000110 11011000
- Protocol: TCP or UDP
- Protocol specific fields
  - Port: 16-bit value (0-65535)
    - Privileged: < 1024
    - Non-privileged: >= 1024

The Domain Name System

- Distributed database: DNS name $\leftrightarrow$ IP address
  - Root servers know who owns domains (i.e., northwestern.edu)
  - Servers in each domain know other machines inside
- Automatic recursive queries
- Ubiquitous; very easy to use
  - Incomplete
  - Can contain lies/inconsistencies
  - Mappings not all one-to-one

Socket: Just Another File?

- Socket: abstraction through which application may send/receive data (often over a network)
- UNIX treats things as files
  - Plain text files
  - Sound cards
  - Serial ports
  - Sockets

Client vs. Server

- Client
  - Initiates requests
  - Examples
    - Netscape, IE
    - ssh
    - Windows Media Player
- Server
  - Passively waits for requests
  - Examples
    - Apache, IIS
    - sshd
    - Windows Media Services

Sockets Client-Side API

- `int socket (int protocolFamily, int type, int protocol)`
- `int connect(int socket, struct sockaddr* foreignAddress, unsigned int addressLength)`
- `int close(int socket)`
- `ssize_t read(int filedes, void* buf, size_t nbytes)`
- `ssize_t write(int filedes, const void* buf, size_t nbyte)`

Specifying Addresses/1

- Generic

```
struct sockaddr {
  unsigned short sa_family; /* Address family          */

  char sa_data[14];         /* Family specific address */
};
```

- Internet-friendly

```
struct in_addr {
  unsigned long s_addr;      /* Inet address (32 bits) */
};
struct sockaddr_in {
  unsigned short sin_family; /* Inet protocol (AF_INET)*/
  unsigned short sin_port;   /* Address port (16 bits) */
  struct in_addr sin_addr;   /* Inet address (32 bits) */

  char sin_zero[8];          /* Leftover space; zeros  */
};
```

Specifying Addresses/2

- Sample initialization

```
struct sockaddr_in myAddrObj;

memset(&myAddrObj, 0, sizeof(myAddrObj);
myAddrObj.sin_family      = AF_INET;
myAddrObj.sin_addr.s_addr = inet_addr(myIP);
myAddrObj.sin_port        = htons(myPort);
```

Typical Client Structure

1. Create a TCP socket
2. Establish connection with server
3. Communicate with server
4. Close connection

Sockets Server-Side API

- `int socket (int protocolFamily, int type, int protocol)`
- `int bind(int socket, struct sockaddr* localAddress, unsigned int addressLength)`
- `int listen(int socket, int queueLimit)`
- `int accept(int socket, struct sockaddr* clientAddress, unsigned int* addressLength)`
- `int close(int socket)`
- `ssize_t read(int filedes, void* buf, size_t nbytes)`
- `ssize_t write(int filedes, const void* buf, size_t nbyte)`

Typical Server Structure

1. Create a TCP socket
2. Assign port number to socket
3. Start allowing connections on port
4. Continuously
    1. Accept a new client connection
    2. Communicate with client
    3. Close connection

Compiling Sockets Code

- Compiles with ANSI C compiler on UNIX system
- Use proper includes
- Link in the appropriate libraries
  - socket for all sockets applications
  - nsl if you are using DNS-related functionality
- Use a makefile

Multi-Connection Servers

- Single connection server issues
  - May not serve that user as quickly as possible
  - Only serves one user at a time
- Solution: multi-connection server
  - select() and non-blocking I/O
    - Java: NIO package
  - Process per connection
  - Thread per connection
  - Some combination of/twist on above

Multi-Platform Issues

- Byte ordering

```
long int htonl(long int hostLong)
short int htons(short int hostShort)
long int notohl(long int netLong)
short int ntohs(long netShort)
```

- Data alignment
  - 32 bit systems can use 32-bit alignment
  - 64-bit systems can use 64-bit alignment

Java Socket Programming

- Everything is an object!
  - class Socket
    - connect()
    - close()
    - getInputStream()
    - getOutputStream()
  - class ServerSocket
    - accept() (returns Socket; see above)
- Java NIO (Non-blocking I/O) package
  - One thread serves multiple connections
  - High performance

Things to Explore

- TCP/IP Sockets in C: Practical Guide for Programmers (Morgan Kaufmann, 2003)
- http://books.elsevier.com/companions/1558608265?country=United+States
- http://www.cs.northwestern.edu/~pdinda/netclass-w02/sockets_nutshell.pdf
- http://www.cs.northwestern.edu/~pdinda/netclass-w02/unix_nutshell.pdf
- http://www.cs.northwestern.edu/~pdinda/minet/
- UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API (Addison-Wesley, 2003)

- http://java.sun.com/docs/books/tutorial/networking/sockets/index.html
- http://java.sun.com/j2se/1.4.2/docs/guide/nio/
- Thinking in Java, 3rd Edition (Free Electronic Book) http://www.mindview.net/Books/TIJ/

- Ethereal (http://www.ethereal.com/)