Computer Network Programming

Socket Programming

Dr. Petru Florin Mihancea

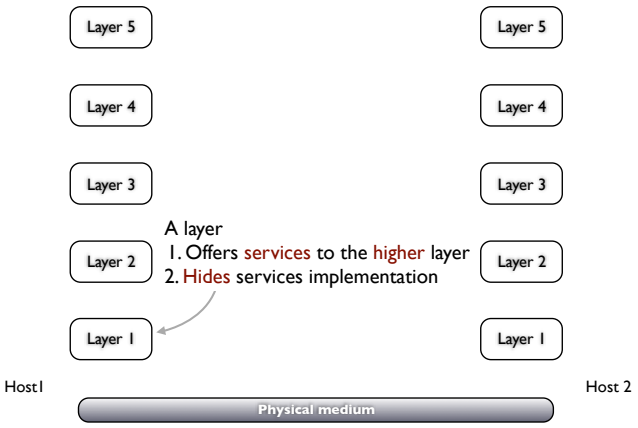
V20180301

1

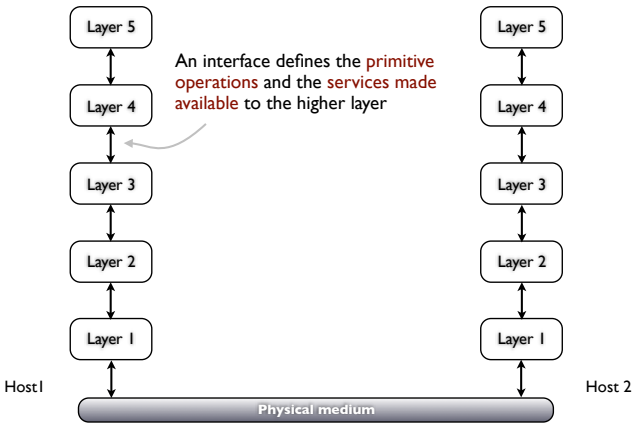
TCP/IP

the primary computer network communication infrastructure

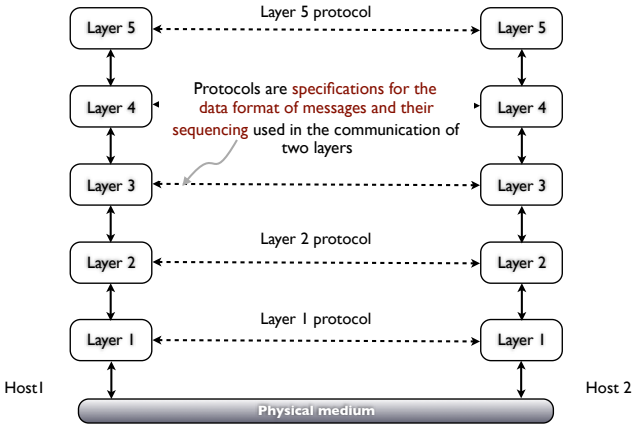
Protocol Hierarchies



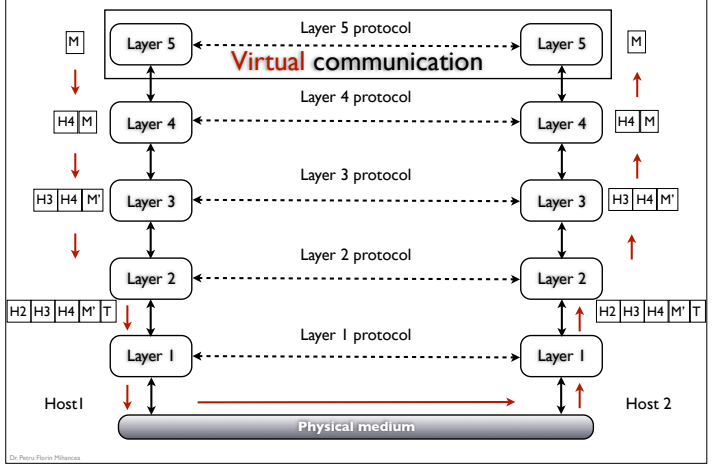
Protocol Hierarchies



Protocol Hierarchies



Protocol Hierarchies



A

Taxonomy of Services

Connection-Oriented

It's like making a phone call

1. **Service user establishes a connection**
2. **Use the connection**
3. **Release the connection**

Preserves information order

Taxonomy of Services

Connectionless

It's like sending postcards

1. **Service user sends two messages to the same receiver**
2. **Messages travel independently (maybe various routs)**

The order may not be preserved

B

Taxonomy of Services

Reliable

No data is lost

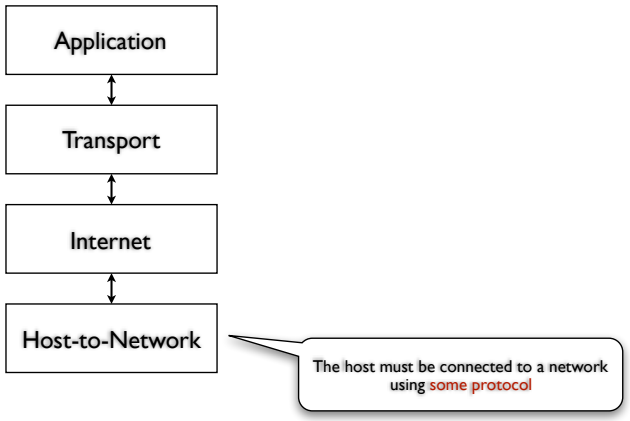
Unreliable

Some data may be lost

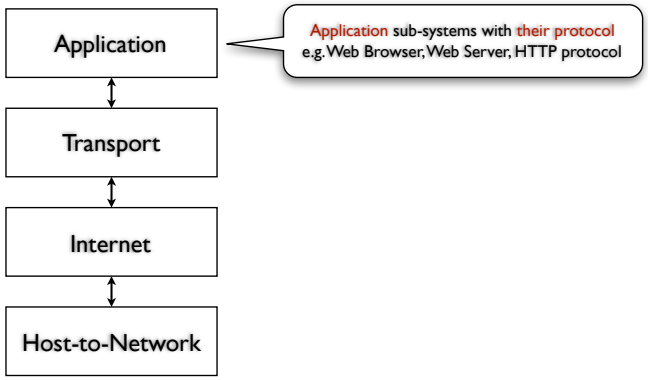
Taxonomy of Services

1. Reliable Connection-Oriented
2. Unreliable Connection-Oriented
3. Unreliable Connectionless
4. Reliable Connectionless

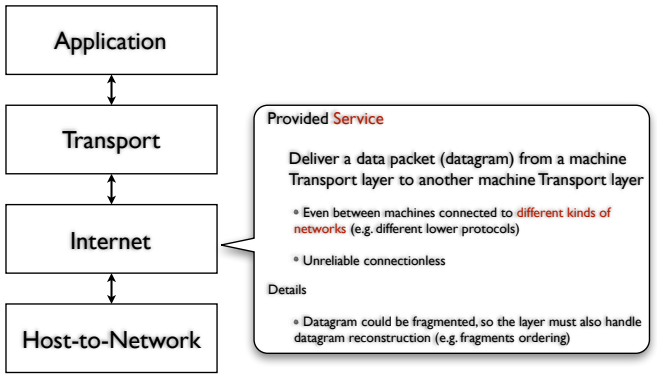
TCP/IP Reference Model



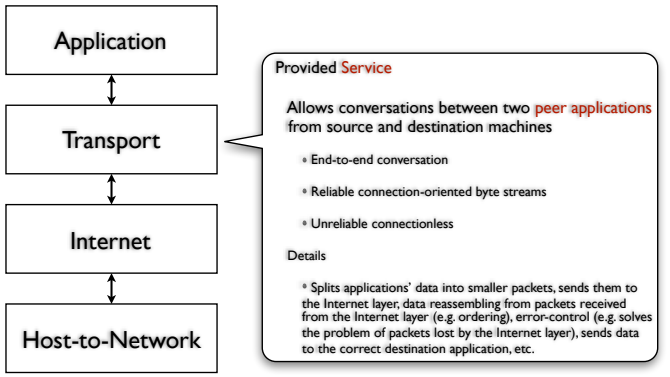
TCP/IP Reference Model



TCP/IP Reference Model



TCP/IP Reference Model



Provided Service

Allows conversations between two peer applications from source and destination machines

- End-to-end conversation
- Reliable connection-oriented byte streams
- Unreliable connectionless

Details

- Splits applications' data into smaller packets, sends them to the Internet layer, data reassembling from packets received from the Internet layer (e.g. ordering), error-control (e.g. solves the problem of packets lost by the Internet layer), sends data to the correct destination application, etc.

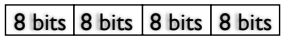
Internet Protocol

Data Transfer Protocol in Internet layer
Unreliable connectionless

IP Address

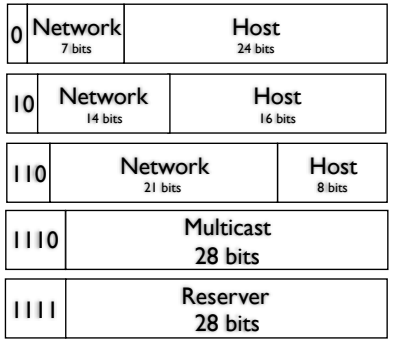
"Uniquely" identifies a network interface of a host in Internet

IPv4



e.g. 74.125.87.104

Internet Addresses



Class A

Class B

Class C

Class D

Class E

Special Internet Address

127	0	0	1

Loopback

Internet Address Issues

Running out of addresses

Solutions

- **Classless InterDomain Routing**
- **Network Address Translation**
  - Some addresses are not routed in Internet
    but translated into a really unique IP address
    - e.g. 192.168.0.0 - 192.168.255.255
- **IPv6**

Transport Protocols

UDP - User Datagram Protocol

Transport layer
Unreliable connectionless

TCP - Transmission Control Protocol

Transport layer
Reliable connection-oriented byte streams

Transport Addresses

TCP & UDP Ports

Helps identifying an application process running on a machine

An integer on 16 bits

Different address spaces for TCP and UDP

Ports < 1024 are reserved for well know servers
e.g. TCP 80 - Web Servers

Socket Programming

API to access the TCP/IP services

Depends on the service but also on the OS and programming language

However, the elementary operations and concepts are very similar

2

Java Socket Programming

Dr. Petru Florin Mihancea

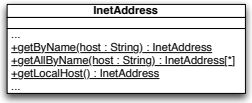
Socket

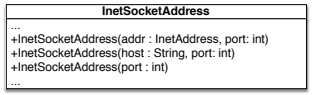
... is one endpoint of a two-way communication link between two programs running on the network

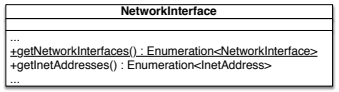
Modelling Addresses

IP

IP + Port

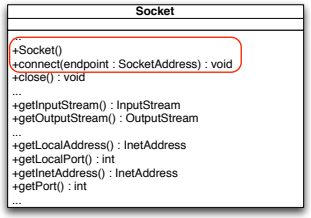


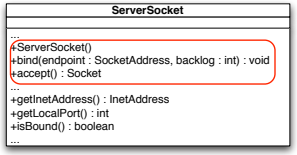




Using the TCP Service

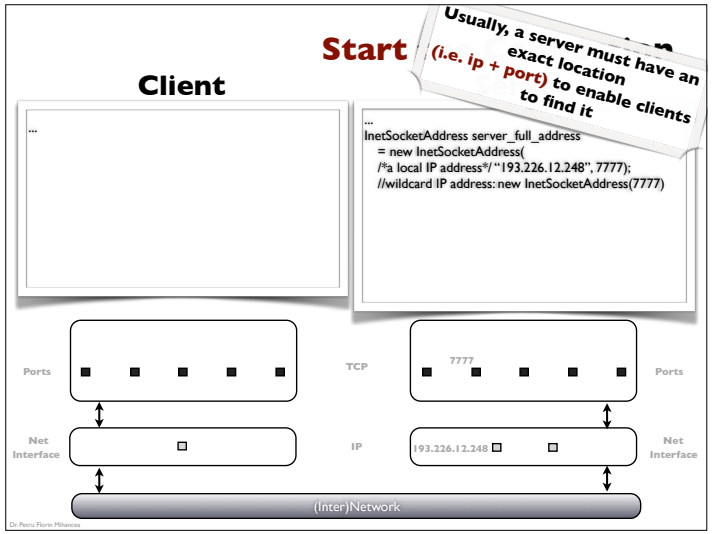
TCP Sockets



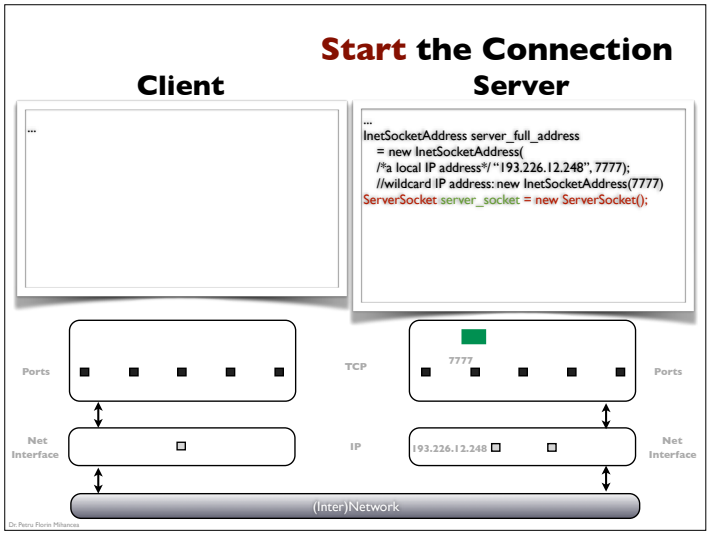


Client & Server

Server


Start
Usually, a server must have an exact location (i.e. ip + port) to enable clients to find it
Client
...
...
InetSocketAddress server_full_address
= new InetSocketAddress(
/*a local IP address*/ "193.226.12.248", 7777);
//wildcard IP address: new InetSocketAddress(7777)
Ports
TCP
7777
Ports
Net Interface
IP
193.226.12.248
Net Interface
(Inter)Network
Dr. Petru Florin Mihancea

Start the Connection



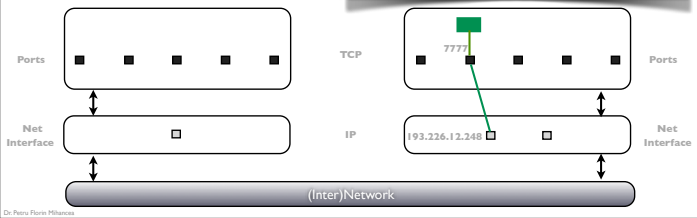
Start the Connection

Client

```
...
```

Server

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
```



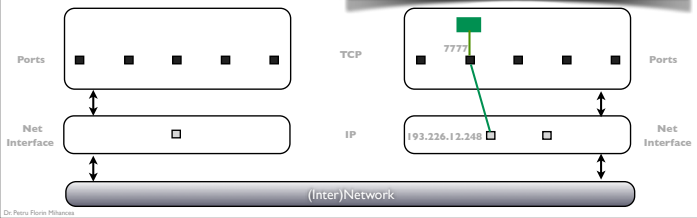
Start the Connection

Client

```
...
```

Server

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
while(true) {
     Socket to_sclient = server_socket.accept(); //blocking
```



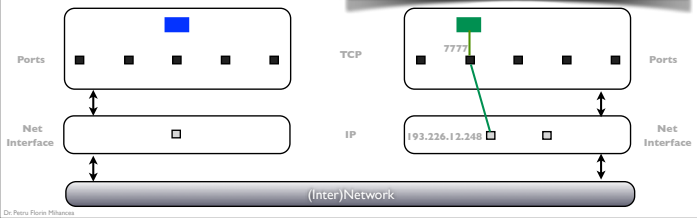
Start the Connection

Client

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress("193.226.12.248", 7777);

Socket client_socket = new Socket();

client_socket.connect(server_full_address); //blocking
```

Server

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
while(true) {
    Socket to_sclient = server_socket.accept(); //blocking
```



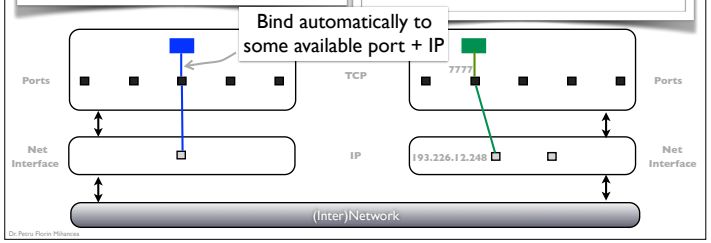
Start the Connection

Client

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress("193.226.12.248", 7777);

Socket client_socket = new Socket();

client_socket.connect(server_full_address); //blocking
```

Server

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
while(true) {
    Socket to_sclient = server_socket.accept(); //blocking
```



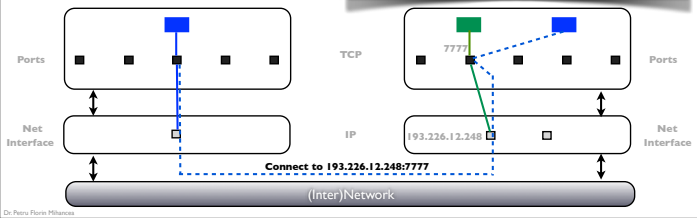
Start the Connection

Client

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress("193.226.12.248", 7777);

Socket client_socket = new Socket();

client_socket.connect(server_full_address); //blocking
```

Server

```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
while(true) {
    Socket to_sclient = server_socket.accept(); //blocking
    // Satisfy client request via to_client (along the blue
    // line) by reading / writing data from / in this socket
}
```



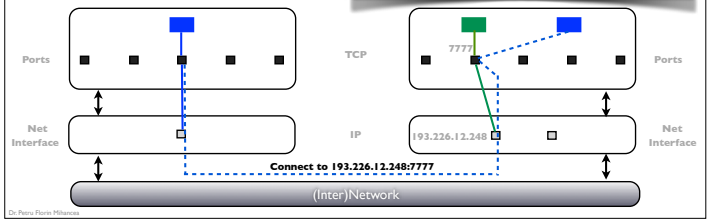
Start the Connection

Client

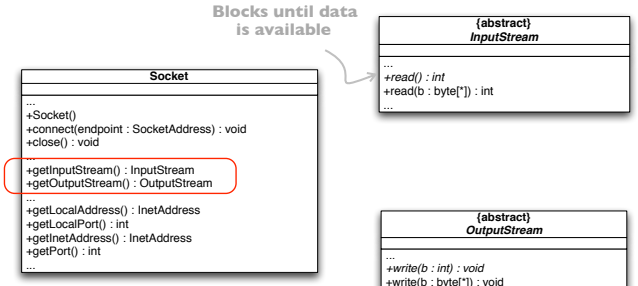
```
...
InetSocketAddress server_full_address
   = new InetSocketAddress("193.226.12.248", 7777);

Socket client_socket = new Socket();

client_socket.connect(server_full_address); //blocking

//Communicate with the server by
// writing / reading data in / from client_socket
```

Server

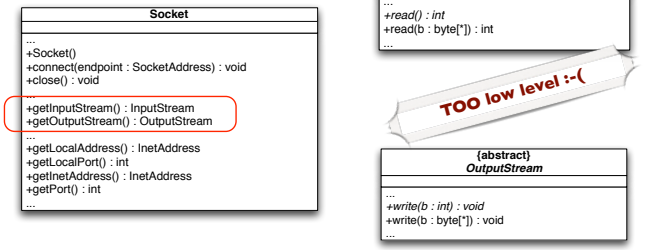
```
...
InetSocketAddress server_full_address
   = new InetSocketAddress(
   /*a local IP address*/ "193.226.12.248", 7777);
   //wildcard IP address: new InetSocketAddress(7777)
ServerSocket server_socket = new ServerSocket();
server_socket.bind(server_full_address, 10);
while(true) {
     Socket to_sclient = server_socket.accept(); //blocking
     // Satisfy client request via to_client (along the blue
     // line) by reading / writing data from / in this socket
}
```



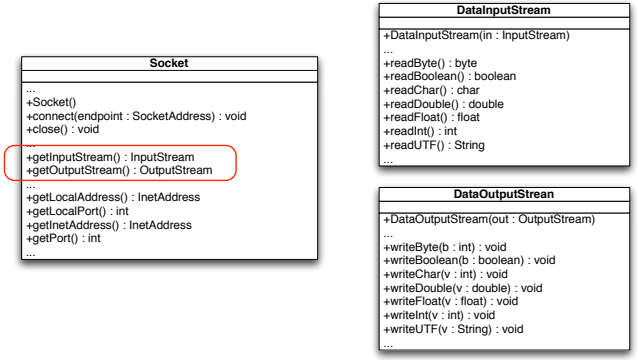
Exchange Data



Exchange Data

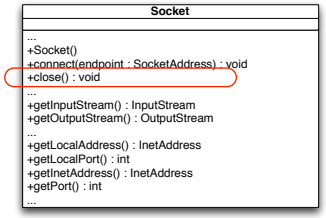


Exchange Data



data_in = new DataInputStream(theSocket.getInputStream())
data_out = new DataOutputStream(theSocket.getOutputStream())

End the Connection



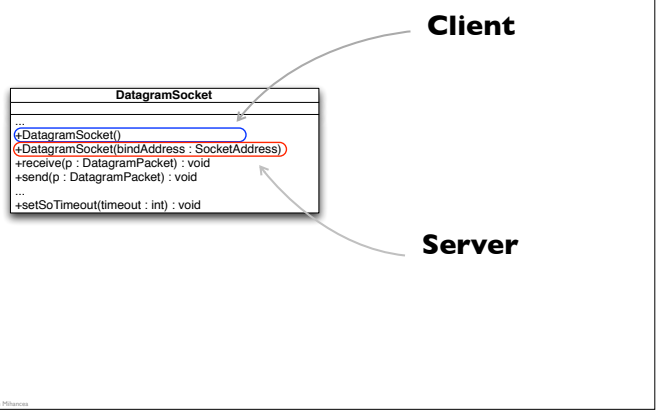
Be careful on what might the partner do !

e.g. low level read returns -1 in case the stream ended

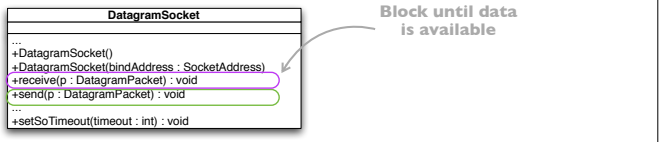
B

Using the UDP Service

UDP Sockets

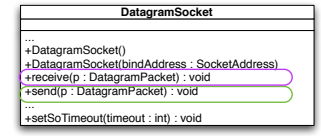

Client
DatagramSocket
...
+DatagramSocket()
+DatagramSocket(bindAddress : SocketAddress)
+receive(p : DatagramPacket) : void
+send(p : DatagramPacket) : void
...
+setSoTimeout(timeout : int) : void
Server

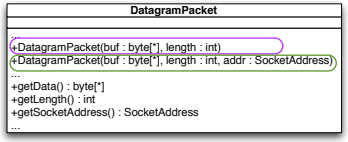
Exchange Data



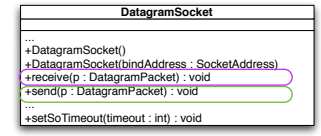
Block until data is available

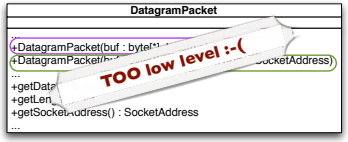
Exchange Data



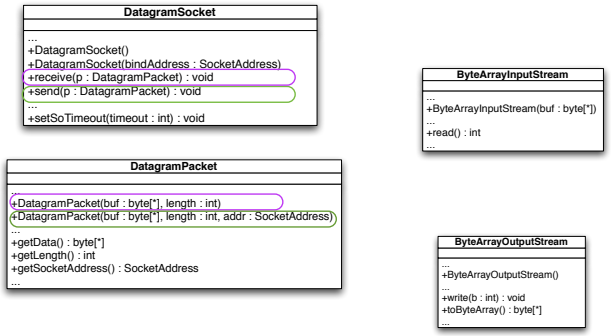


Exchange Data





Exchange Data




```
ByteArrayOutputStream byte_out = new ByteArrayOutputStream();
DataOutputStream data_out = new DataOutputStream(byte_out);
// use previous write methods (e.g. writeByte, writeBoolean, etc.)
DatagramPacket out_packet = new DatagramPacket(byte_out.toByteArray(),
    byte_out.toByteArray().length, dest_address);
```

3

BSD Sockets

A Way to Model Addresses

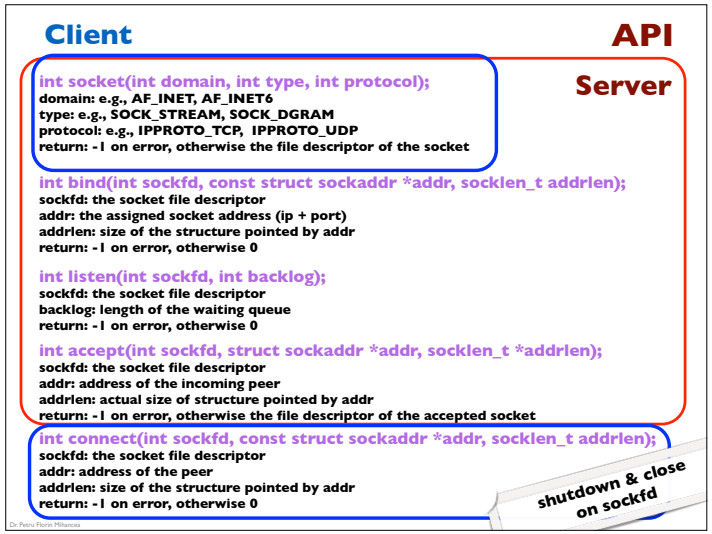
IP

```
struct in_addr {
   uint32_t          s_addr;     /* address in network byte order - may use inet_aton-like funcs*/
                                 /* INADDR_ANY - bind to all addresses
};
```

IP + Port

```
struct sockaddr_in {
   sa_family_t      sin_family; /* address family: e.g., AF_INET, AF_INET6 */
   in_port_t        sin_port;    /* port in network byte order -use htons-like functions */
   struct in_addr   sin_addr;    /* internet address */
};
```

struct hostent * gethostbyname(const char *name);
name: name of the host
return: NULL on error, or it points to a structure in which h_addr_list is a list of addresses in network byte order


Client
API
Server
int socket(int domain, int type, int protocol);
domain: e.g., AF_INET, AF_INET6
type: e.g., SOCK_STREAM, SOCK_DGRAM
protocol: e.g., IPPROTO_TCP, IPPROTO_UDP
return: -1 on error, otherwise the file descriptor of the socket
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
sockfd: the socket file descriptor
addr: the assigned socket address (ip + port)
addrlen: size of the structure pointed by addr
return: -1 on error, otherwise 0
int listen(int sockfd, int backlog);
sockfd: the socket file descriptor
backlog: length of the waiting queue
return: -1 on error, otherwise 0
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
sockfd: the socket file descriptor
addr: address of the incoming peer
addrlen: actual size of structure pointed by addr
return: -1 on error, otherwise the file descriptor of the accepted socket
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
sockfd: the socket file descriptor
addr: address of the peer
addrlen: size of the structure pointed by addr
return: -1 on error, otherwise 0
shutdown & close on sockfd
Dr. Petru Florin Mihancea

Exchange Data

Usually for connection-orientated sockets

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);

ssize_t write(int fd, const void *buf, size_t count);
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

Usually for connectionless sockets

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
    const struct sockaddr *dest_addr, socklen_t addrlen);

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
    struct sockaddr *src_addr, socklen_t *addrlen);
```

Return: -1 on error or number of bytes received/written

We can control various aspects using flags (e.g., make a non-blocking invocation)

4

Demo Application

Server

1. Convert an int representing a digit into the corresponding string
   e.g. 1 - “one”, 2 - “two”, 11 - “Not a digit!”

2. The reverse
   e.g. “one” - 1, “zer” - -1

Client

Uses these operations

But first ...

The Application Protocol

Request messages		Result message
Command	Data (if required)	Result
1. Int Digit - to - String		
1	Int	String
2. String Digit - to - Int		
2	String	Int
3. Ends connection (where required)		
3		