SOCKET-BASED COMMUNICATION

Classic view of network API

- Start with host name (maybe)

foo.bar.com

Classic view of network API

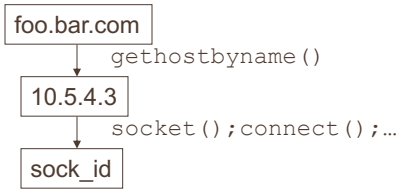
- Start with host name
- Get an IP address

23

23

Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)

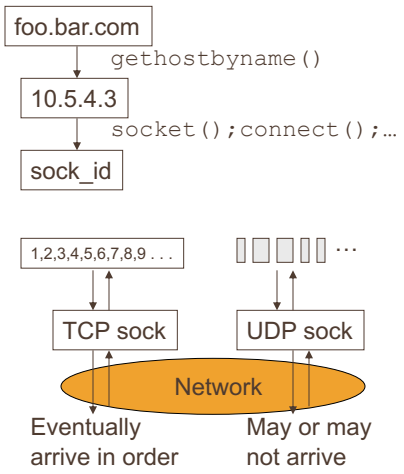


24

24

Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)
- Send byte stream (TCP) or packets (UDP)



25

25

Sockets

- Berkeley Unix 4.2BSD
- API for Internet transport protocols
- The foundation of the Internet
- Two main kinds:
  - Datagram sockets (UDP/IP)
  - Stream sockets (TCP/IP)
  - Also raw sockets (IP, ICMP)

26

26

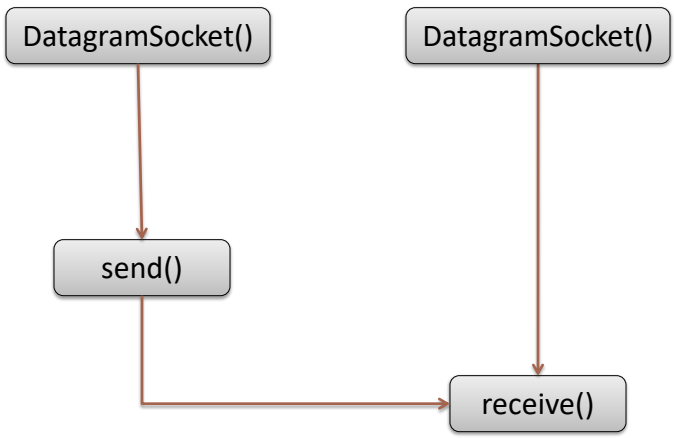
Datagram Sockets

- Unreliable
  - In practice, reliable in LANs
- Low overhead
- Applications
  - Queries
  - Notifications
  - Small messages

27

27

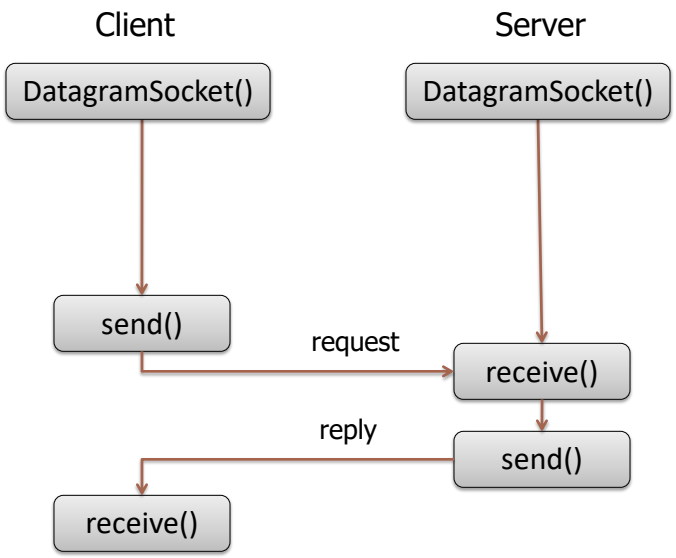
Simple Message Passing (Java)



28

28

Client-Server (Java)



29

29

Network Addresses in Java

```
public final class InetAddress implements Serializable {
  // host may be DNS address
  static InetAddress getByName (String host);
  static InetAddress getLocalHost ();

  byte[] getAddress ();
  String getHostName ();
  String getHostAddress ();
  boolean isMulticastAddress ();
}
```

30

30

Datagram Packets in Java

```
public class DatagramPacket {
   public DatagramPacket (byte ibuf[], int ilen);
   public DatagramPacket (byte ibuf[], int ilen,
                          InetAddress dest, int port);

   synchronized InetAddress getAddress ();
   synchronized int getPort ();
   synchronized byte[] getData ();
   synchronized int getLength ();
}
```

31

31

Datagram Sockets in Java

```
Public class DatagramSocket {
   public DatagramSocket ();
   public DatagramSocket (int port);

   void send (DatagramPacket p);
   void receive (DatagramPacket p);
   void close ();

   synchronized void setSoTimeout (int timeout);
}
```

32

32

STREAM SOCKETS

33

33

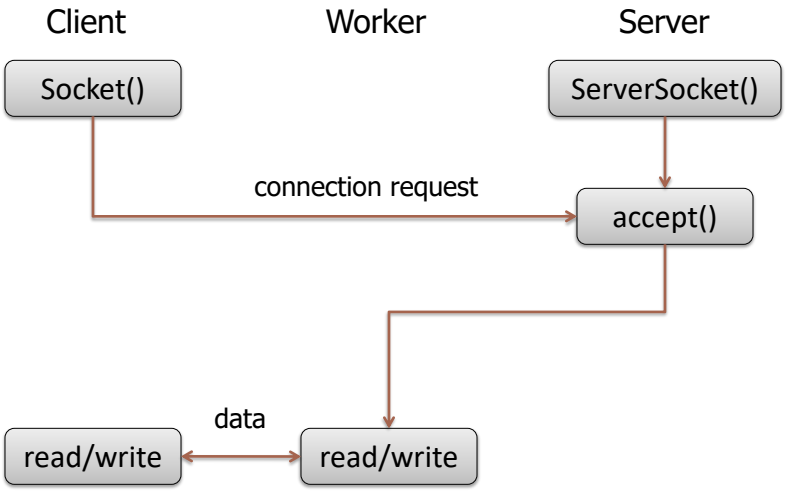
Stream Sockets

- Based on TCP/IP protocol
- Client makes connection request
- Server listens for connection requests
  - stream socket returned to client and server
- Stream socket looks/smells/feels like a file
  - Reliable, ordered byte stream
- New server thread

34

34

Stream Sockets (Java)



35

35

Client Stream Sockets in Java

```
public class Socket {
   public Socket (InetAddress address, int port);
   InputStream getInputStream ();
   OutputStream getOutputStream ();
   synchronized void close();

   void setTcpNoDelay (boolean on);
   void setSoLinger (boolean on, int val);
   void setSoTimeout (int timeout);
   static setSocketImplFactory (SocketImplFactory fac);
```

36

36

Server Stream Sockets in Java

```
public class ServerSocket {
   public ServerSocket (int port);
   public ServerSocket (int port, int backlog);
   void Socket accept ();
   void close ();

   void setToTimeout (int timeout);
   protected final void implAccept (Socket s);
   static setSocketImplFactory (SocketImplFactory fac);
}
```

37

37

Extended Server Sockets

```
class SSLServerSocket extends ServerSocket { ...
   public Socket accept () {
        SSLSocket s = new SSLSocket (certChain, privateKey);
        // create an unconnected client SSLSocket, that we'll
        // return from accept
        implAccept (s);
        s.handshake ();
        return s; }
}

class SSLSocket extends Socket { ...
   public SSLSocket(CertChain c, PrivateKey k) {
        super(); ...
   }
}
```

38

38

FTP EXAMPLE

39

39

FTP: File Transfer Protocol

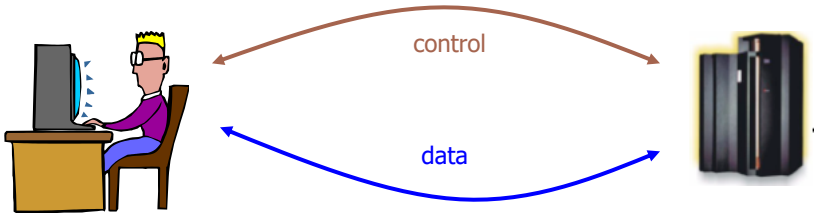
- Client connected to ftp server (ftpd)
  - List directory contents
  - Change directory
  - Get file contents
  - Put file contents
- Stateful protocol
  - Server maintains client state

40

40

FTP Data Transfer

- Control connection for commands
- Use separate data connection to:
  - Send lists of files (LIST)
  - Retrieve a file (RETR)
  - Upload a file (STOR)

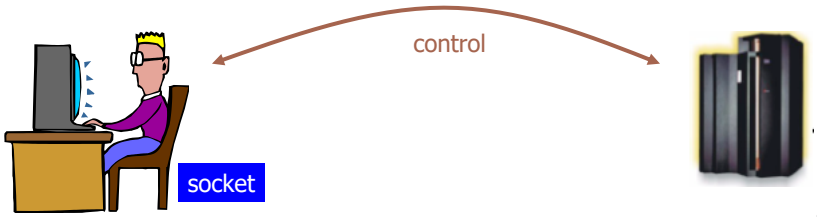


41

41

Creating the data connection: active mode

- Client acts like a server
  - Creates a socket
    - Assigned an ephemeral port number by the OS
  - Listens on socket
  - Waits to hear from FTP server

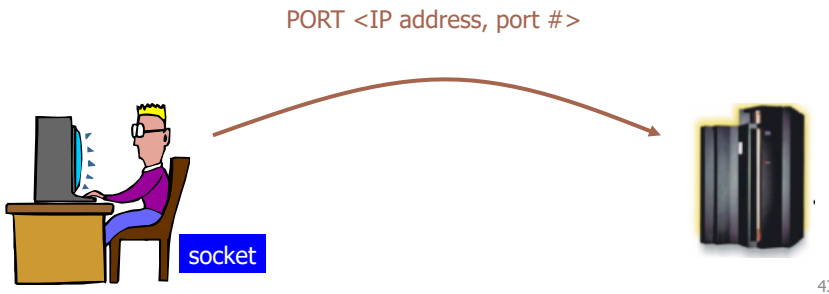


42

42

Creating the data connection: active mode

- Client tells port number to the server
  - Via PORT command on control connection

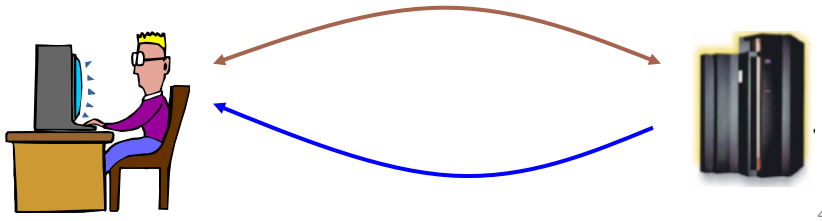


43

43

Creating the data connection: active mode

- Server initiates the data connection
  - Connects to the socket on the client machine
  - Client accepts, to complete the connection
- Data now flows along second connection

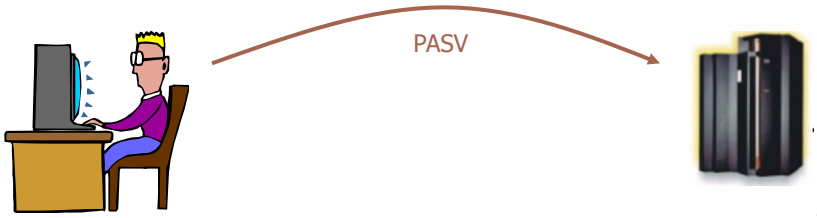


44

44

Creating the data connection: passive mode

- Client tells the server to go into passive mode

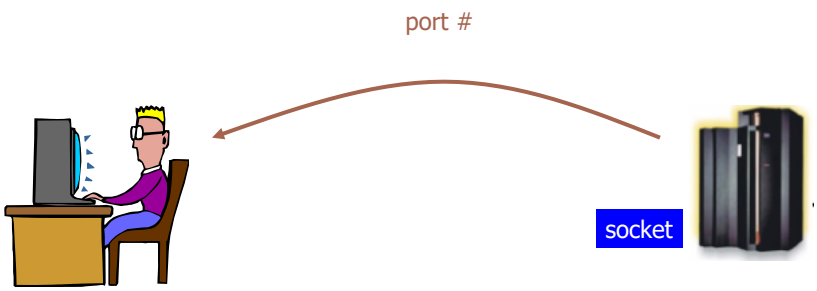


45

45

Creating the data connection: passive mode

- Server creates socket, responds with port number

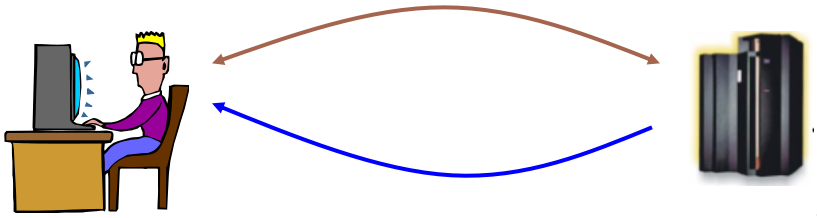


46

46

Creating the data connection: passive mode

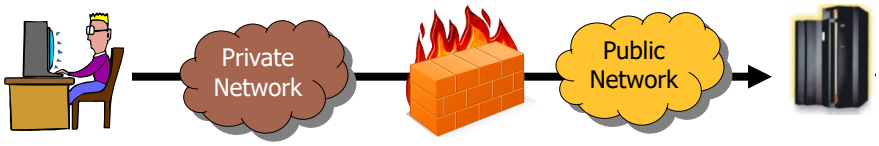
- Client initiates the data connection
  - Connects to the socket on the server machine
  - Server accepts, to complete the connection
- Data now flows along second connection



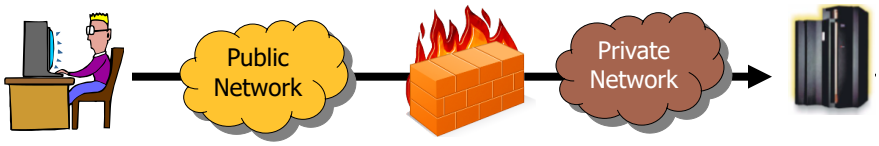
47

47

Active vs Passive Mode



PASV reply returns a public IP address



PORT command includes a public IP address

48

48

Server File Transfer

```
ServerSocket listenTo = new ServerSocket (0, 1);
// 0 means any port
…. send listenTo.getLocalPort() to client….

Socket xfer = listenTo.accept ();
InputStream is = new FileOutputStream (file);
OutputStream os = xfer.getOutputStream ();
byte[] buf = new [512] byte ();
int nbytes = is.read (buf, 0, 512);
while (nbytes > 0) {
    os.write (buf, 0, nbytes);
    nbytes = is.read (buf, 0, 512);
}
is.close();  os.close();
```

49

49

Client File Transfer

```
String file;
int setupPort;

Socket xfer = new Socket (this.server, setupPort);
InputStream is = xfer.getInputStream ();
OutputStream os = new FileOutputStream (file);
byte[] buf = new [512] byte ();
int nbytes = is.read (buf, 0, 512);
while (nbytes > 0) {
   os.write (buf, 0, nbytes);
   nbytes = is.read (buf, 0, 512);
}
is.close();  os.close();
}
```

50

50