



Computer Networks 1

(Mạng Máy Tính 1)

Lectured by: Dr. Phạm Trần Vũ

[cuu duong than cong . com](http://cuuduongthancong.com)



Lecture 8: Transport Layer and Socket Programming with Java

Reference:

Chapter 6 - “*Computer Networks*”,
Andrew S. Tanenbaum, 4th Edition, Prentice Hall, 2003.



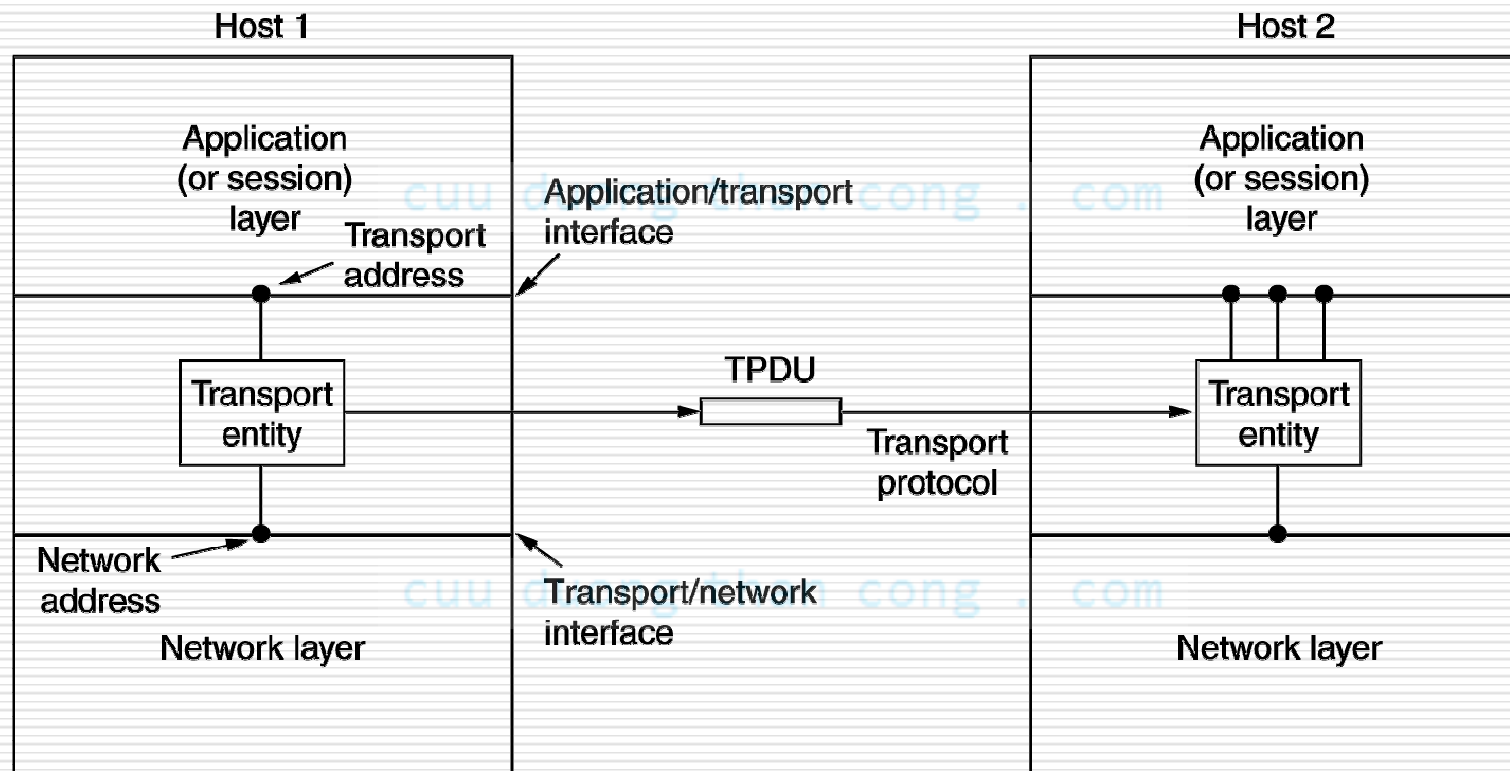
The Transport Service

- Services Provided to the Upper Layers
- Transport Service Primitives
- Berkeley Sockets
- An Example of Socket Programming:
 - An Internet File Server



Services Provided to the Upper Layers

The network, transport, and application layers.





Transport Service Primitives

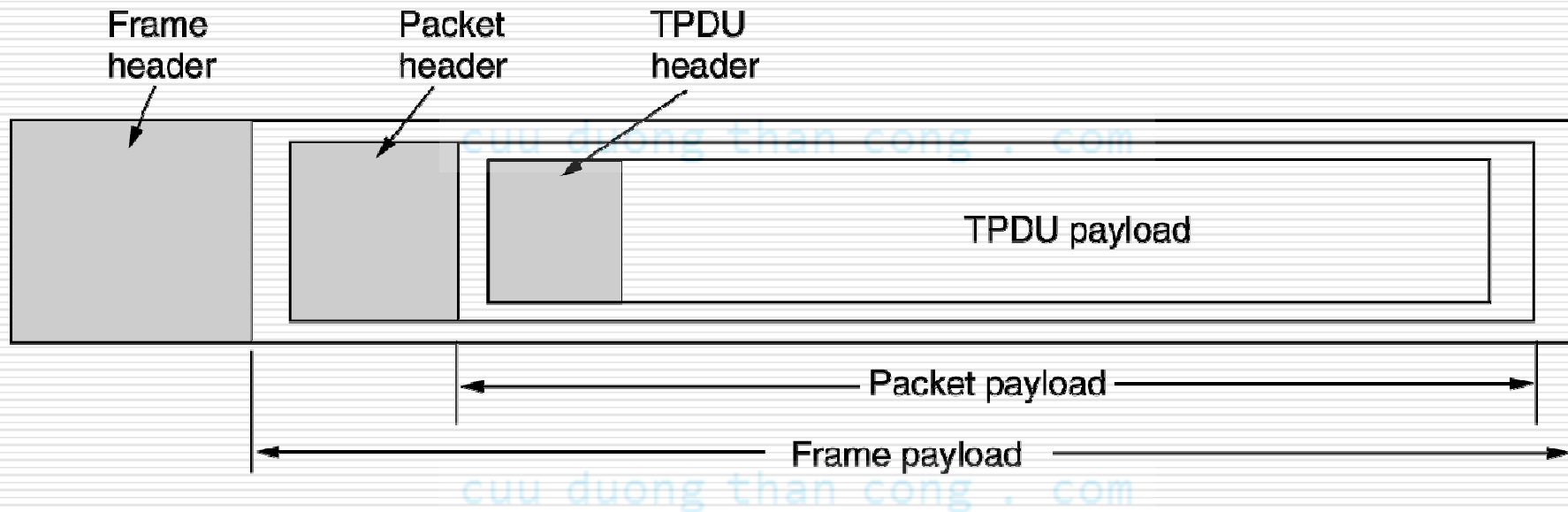
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service.



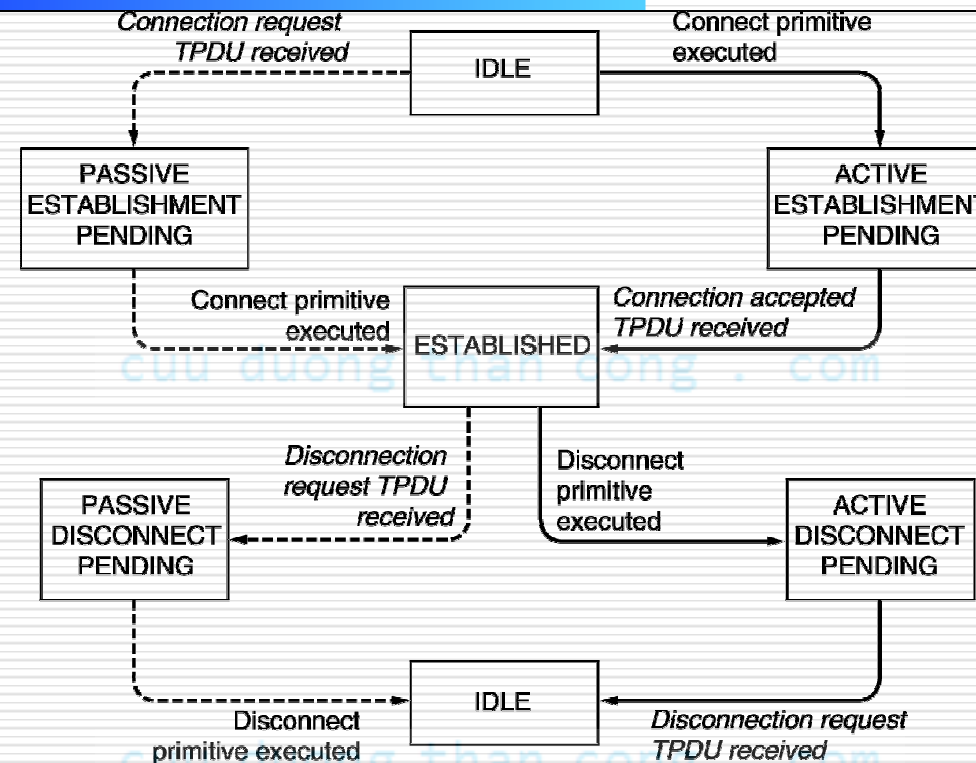
Transport Service Primitives (2)

The nesting of TPDU, packets, and frames.





Transport Service Primitives (3)



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.



Berkeley Sockets

The socket primitives for TCP.

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection



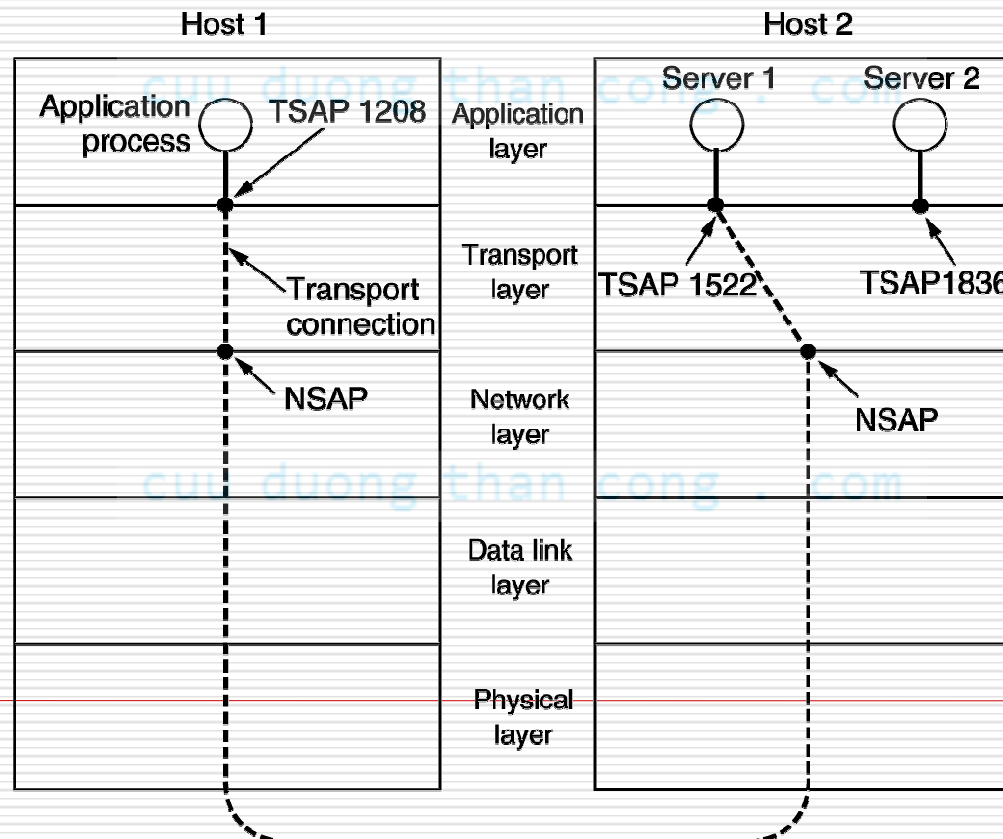
Elements of Transport Protocols

- **Addressing**
- **Connection Establishment**
- **Connection Release**
- Flow Control and Buffering
- Multiplexing
- Crash Recovery



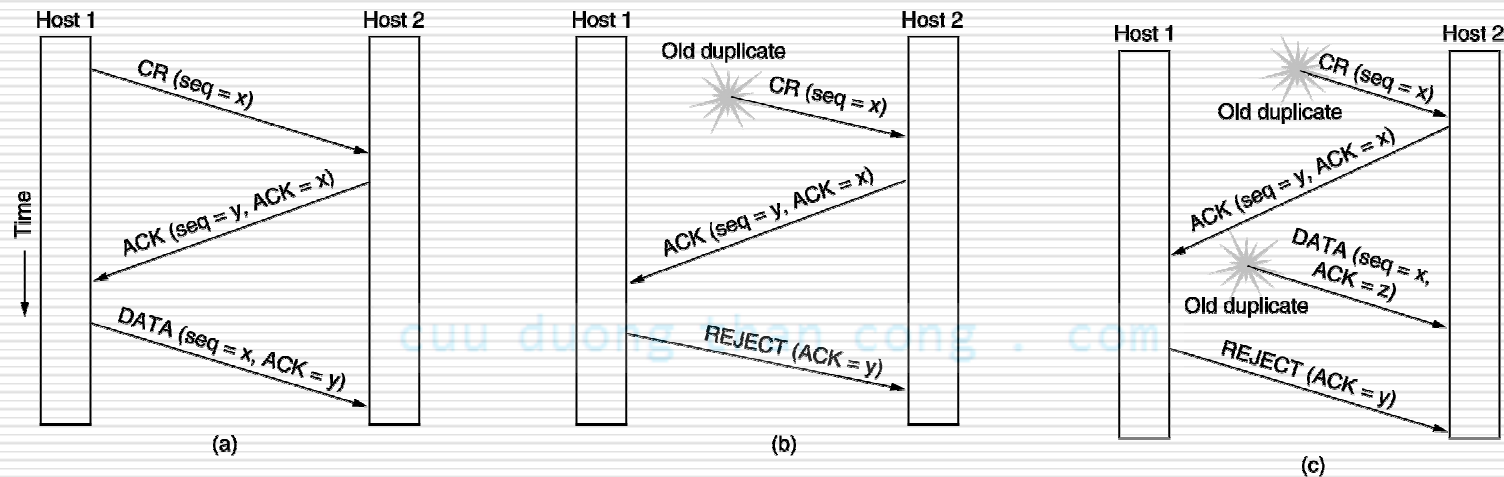
Addressing

- Application addresses on a host: Ports
TSAPs, NSAPs and transport connections.





Connection Establishment

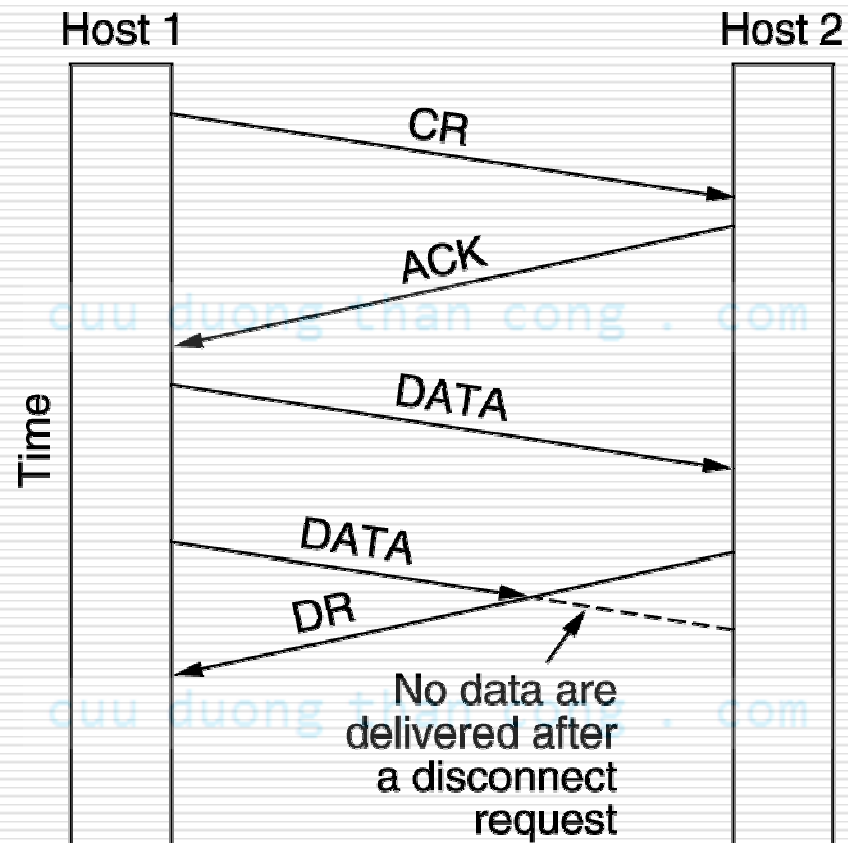


Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.

- (a) Normal operation,
- (b) Old CONNECTION REQUEST appearing out of nowhere.
- (c) Duplicate CONNECTION REQUEST and duplicate ACK.



Connection Release

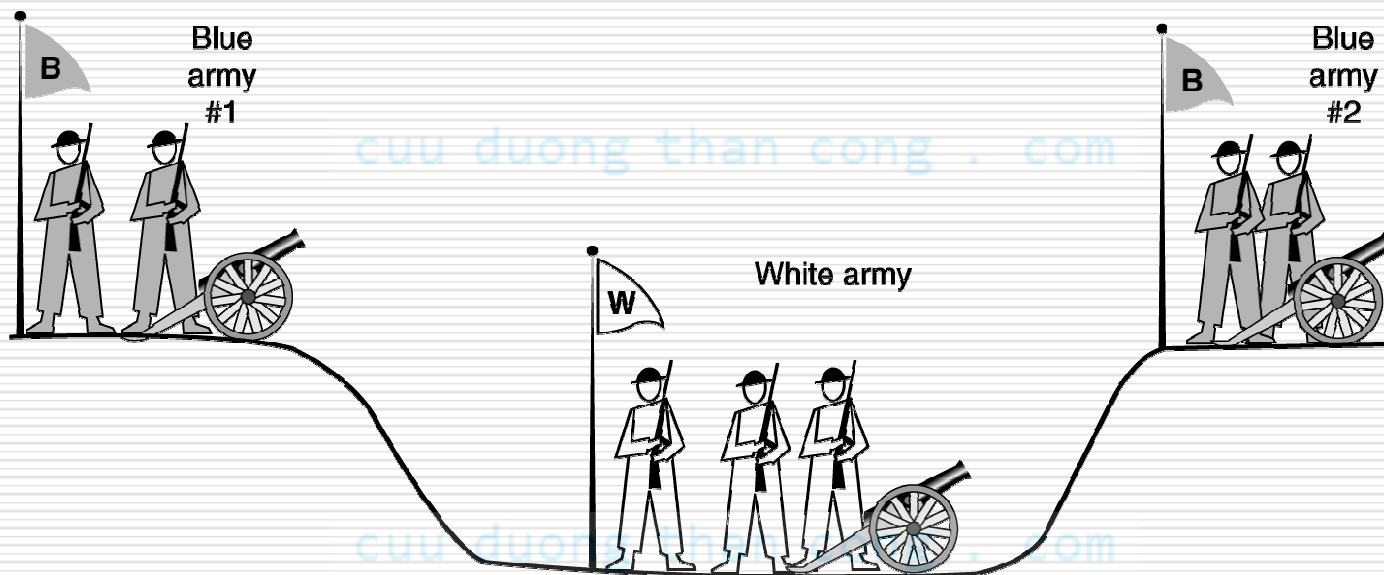


Abrupt disconnection with loss of data.



Connection Release (2)

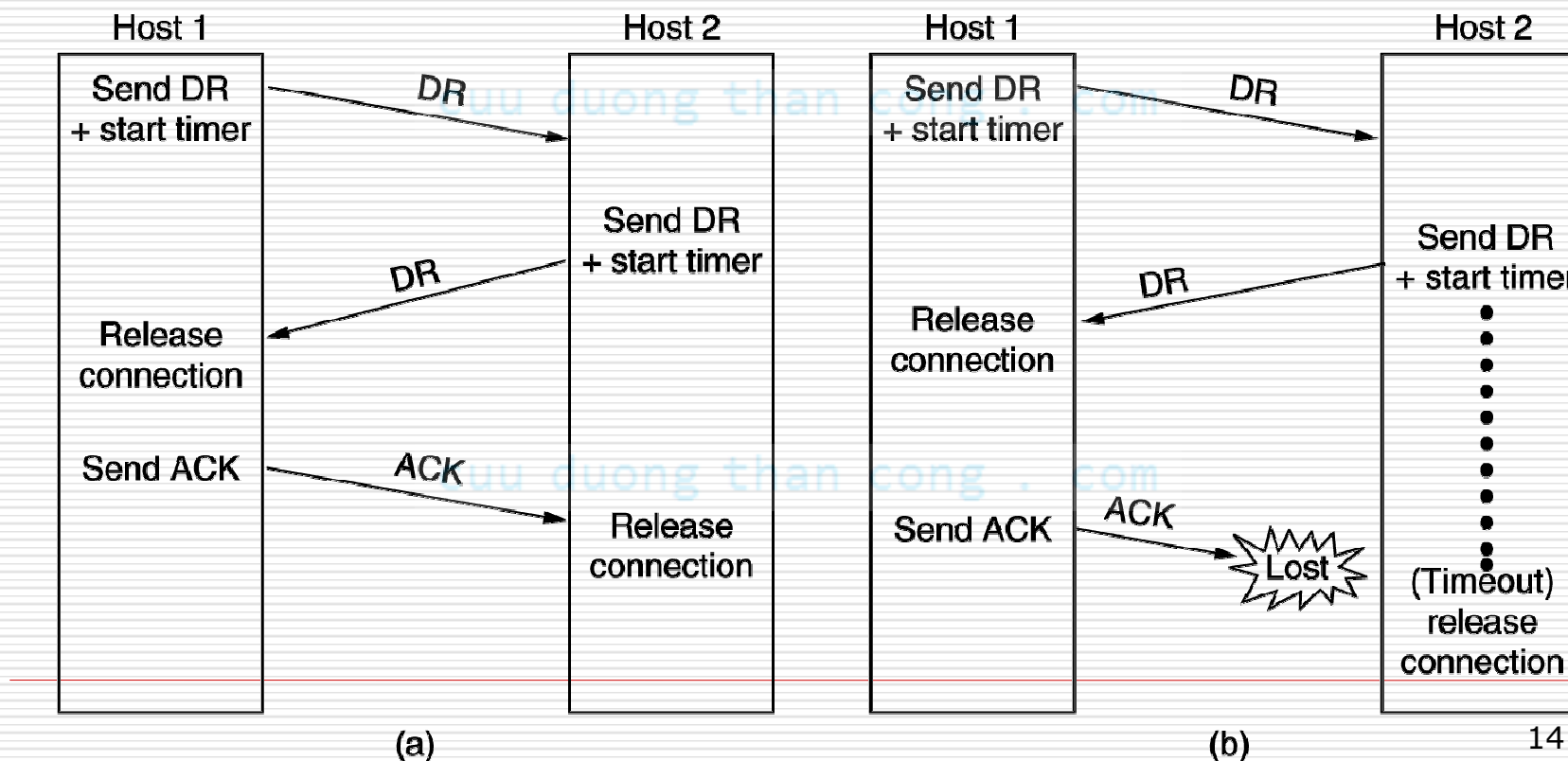
The two-army problem.





Connection Release (3)

Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) final ACK lost.



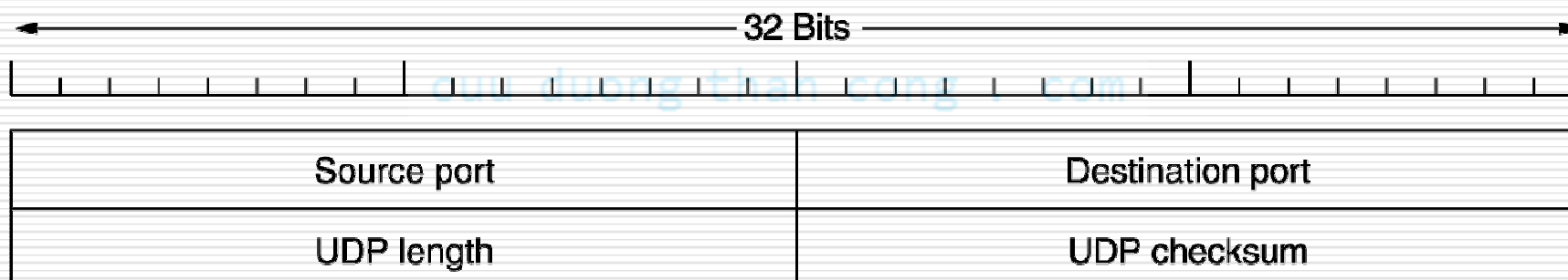


The Internet Transport Protocols

- ❑ UDP – User Datagram Protocol
 - Connection-less service
 - Useful in client-server situations: Remote Procedure Call, real-time AV streaming
 - ❑ TCP – Transmission Control Protocol
 - Connection oriented service
 - Reliable byte stream services over unreliable network
 - Most widely used in Internet
-



UDP Header



cuu duong than cong . com



TCP Service Model

- ❑ Sender and receiver need to create connection end-points first, called sockets
- ❑ Each socket is addressed by the host IP address and a port number
- ❑ Port numbers < 1024 are reserved
- ❑ TCP connections are full-duplex

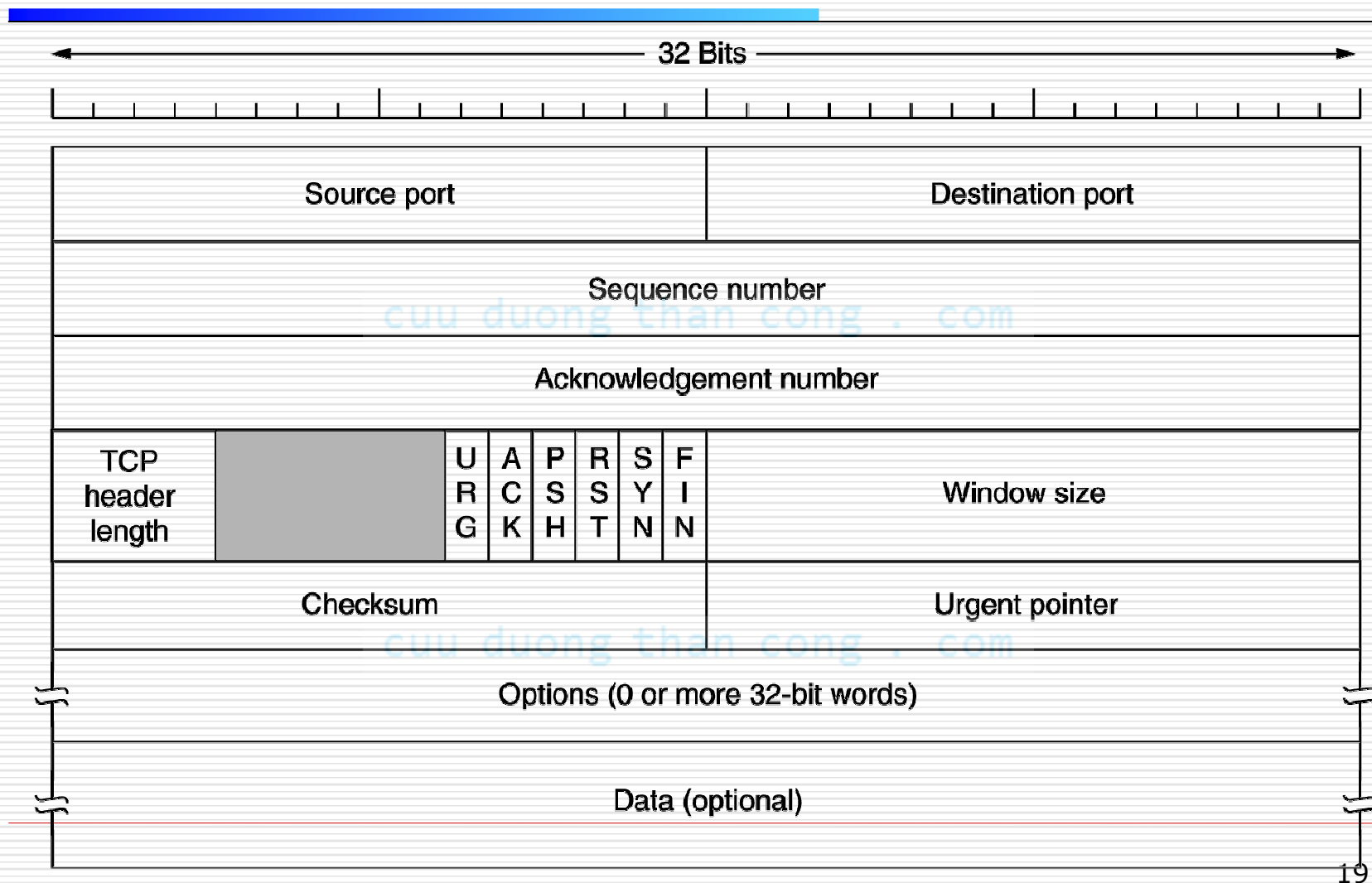


Typical TCP Applications and Ports

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

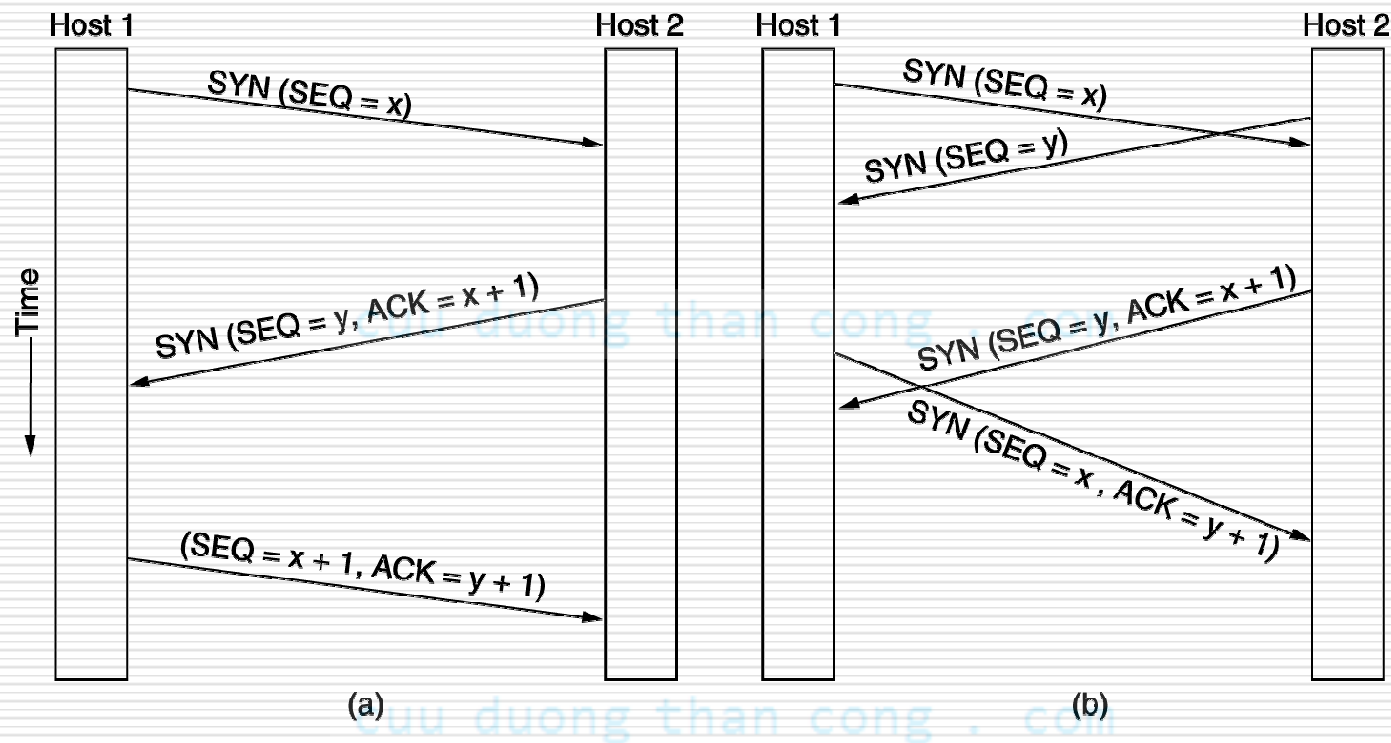


The TCP Segment Header





TCP Connection Establishment



- (a) TCP connection establishment in the normal case.
- (b) Call collision.



Socket Programming in Java

cuu duong than cong . com

cuu duong than cong . com



Client-Server Application with UDP (1)

□ Client operations

- Identify server IP and port
- Create UDP socket
- Send/receive data to server
- Close socket

□ Server operations

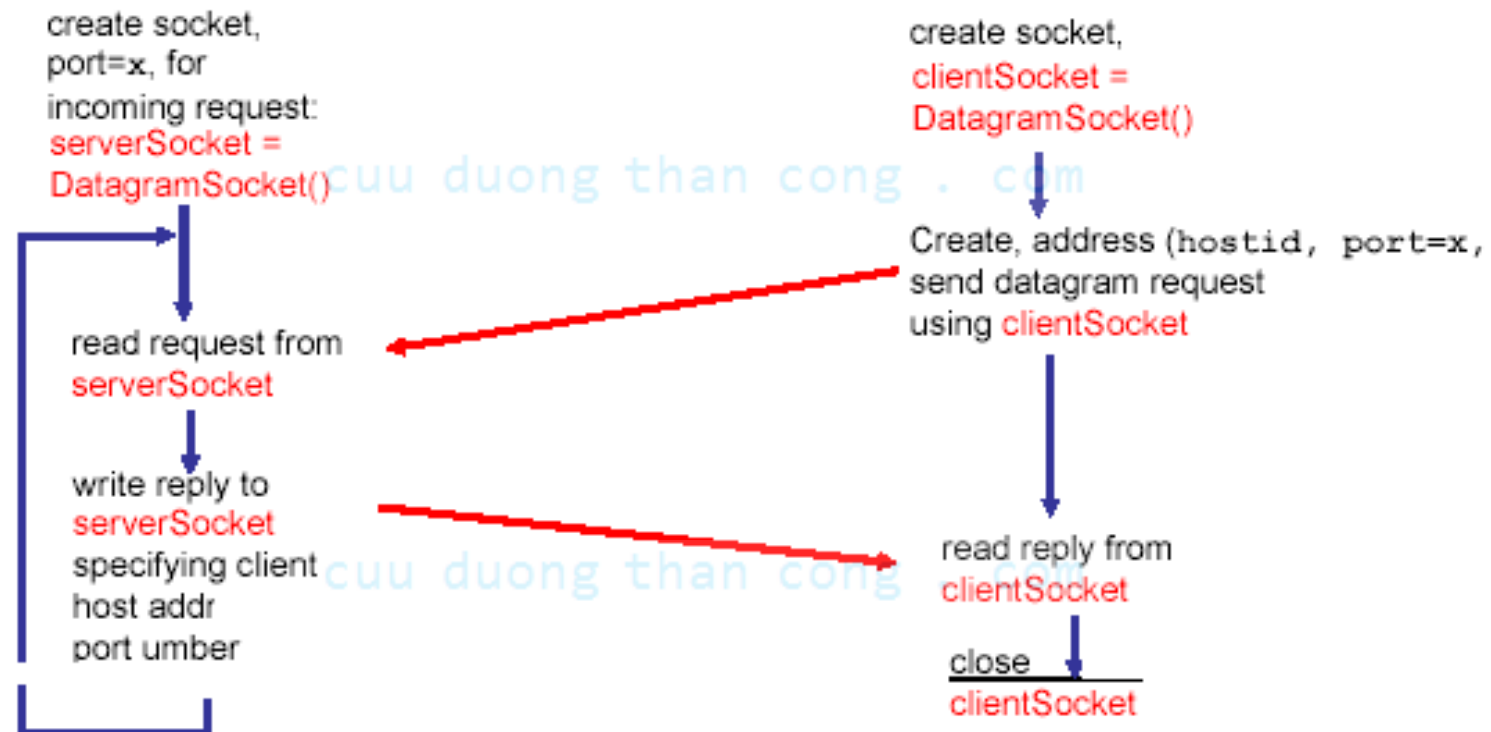
- Create socket and register with the system
- Read client messages and respond to client



Client-Server Application with UDP (2)

Server (running on `hostid`)

Client





Client-Server Application with TCP (1)

- ❑ Client operations
 - Identify server IP and port
 - Create UDP socket
 - Setup connection to server
 - Send/receive data
 - Close connection



Client-Server Application with TCP (2)

- ❑ Server operations
 - Create and register socket
 - Listen and wait for incoming connections
 - Accept connection
 - Send/receive data
 - Close connection

cuu duong than cong . com

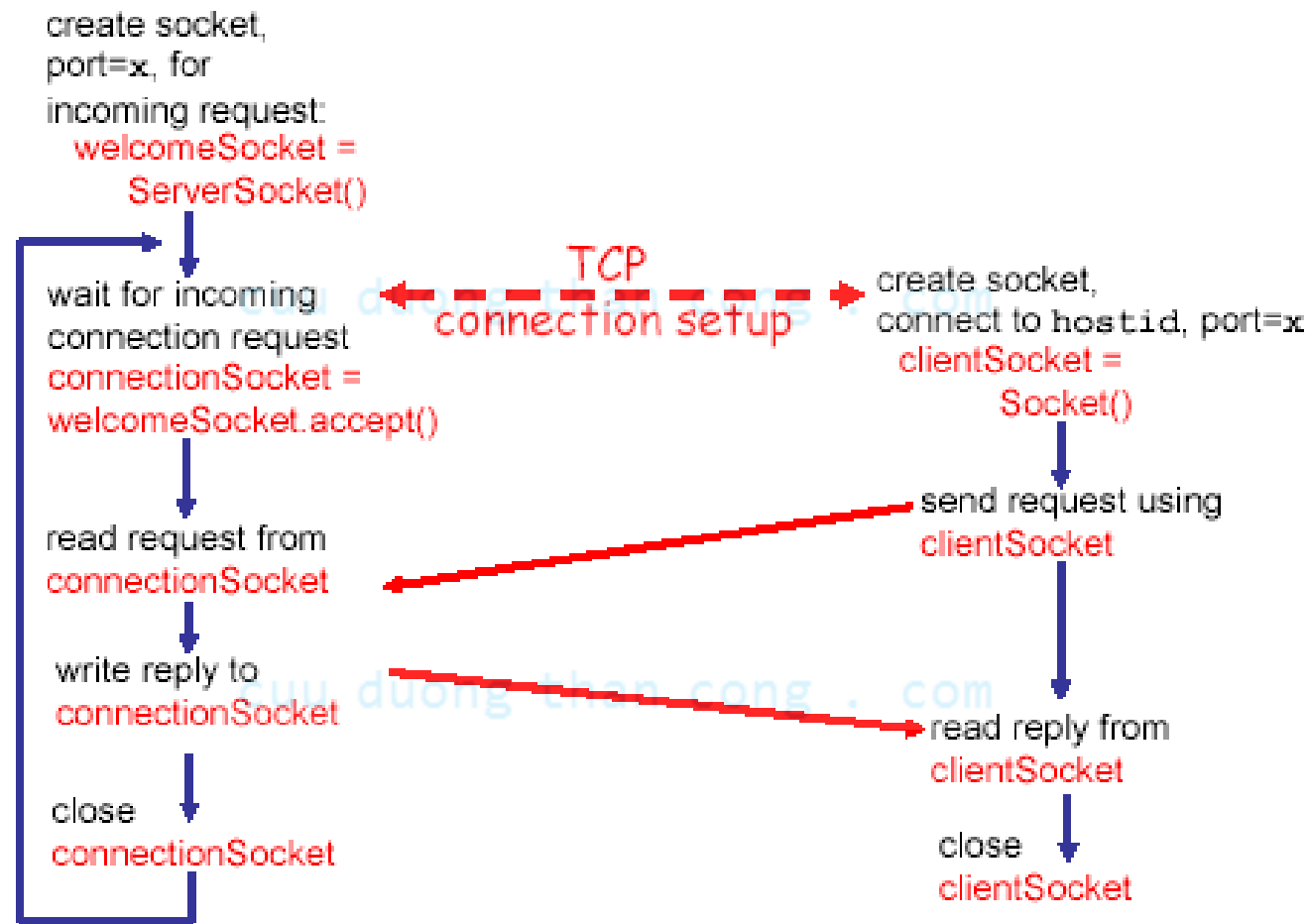


Client-Server Application with TCP (3)

- ❑ Concurrent server operations
 - Create and register socket
 - Listen and wait for incoming connections
 - Accept connection and spawn new thread to handle the connection
 - Listen and wait for new connection
 - Thread operations
 - ❑ Send/receive data through connection
 - ❑ Close connection



Client-Server Application with TCP (4)





Java API: java.net package

- ❑ *InetAddress*
- ❑ *ServerSocket*
- ❑ *Socket*
- ❑ *URL*
- ❑ *URLConnection*
- ❑ *DatagramSocket*



InetAddress class

- ❑ Class used for internet addresses (Internet Protocol)
- ❑ Use methods: *getLocalHost*, *getByName*, or *getAllByName* to create an *InetAddress* instance:
 - *public static InetAddress InetAddress.getByName(String hostname)*
 - *public static InetAddress [] InetAddress.getAllByName(String hostname)*
 - *public static InetAddress InetAddress.getLocalHost()*
- ❑ To get the host IP address or host name:
 - *getHostAddress()*
 - *getHostName()*



Socket class (1)

- ❑ To describe a *socket*
- ❑ To create a *socket*
 - **Socket**(InetAddress address, int port)
 - **Socket**(String host, int port)
 - **Socket**(InetAddress address, int port, InetAddress, localAddr, int localPort)
 - **Socket**(String host, int port, InetAddress, localAddr, int localPort)
 - **Socket**()



Socket class (2)

- Get socket information
 - InetAddress **getInetAddress()**
 - int **getPort()**
 - InetAddress **getLocalAddress()**
 - int **getLocalPort()**
- Using output and input Streams
 - public OutputStream **getOutputStream()** throws IOException
 - public InputStream **getInputStream()** throws IOException



ServerSocket Class (1)

- ❑ Used for a server side socket
- ❑ Create a *ServerSocket*
 - **ServerSocket**(int port) throws IOException
 - **ServerSocket**(int port, int backlog) throws IOException
 - **ServerSocket**(int port, int backlog, InetAddress bindAddr) throws IOException



ServerSocket Class (2)

- ❑ Socket **accept()** throws IOException.
- ❑ void **close()** throws IOException
- ❑ InetAddress **getInetAddress()**
- ❑ int **getLocalPort()**
- ❑ void **setSoTimeout**(int timeout) throws SocketException