

SSD8: Networks and Distributed Computing

M.T. Ipalakova

My Info

- Madina Tulegenovna Ipalakova
- Мадина Тулегеновна Ипалакова

- CE&T Department, of. 409

- m.ipalakova@iitu.kz



Description

- The subject focuses on the principles and practices of network-based computing
 - Network technology in support of data and multimedia communication
 - Application-oriented protocol and approaches to distributed object-oriented programming using Java



Topics


- Networking protocols
- Technology
- Multimedia networking
- Client/server design including thick and thin clients
- CORBA and related tools
- WWW implementation issue
- Electronic mail
- Security issues
- Privacy issues



Required Texts

- James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Second edition, Boston: Addison Wesley Longman, Inc., 2003
- OR
- James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Third edition, Boston: Addison Wesley, 2004
- David Reilly and Michael Reilly. *Java(TM) Network Programming and Distributed Computing*. First edition, Addison Wesley, 2002





Unit 1. Core Network Protocols
Lecture 1. Introduction and
Revision



Basics of Communication

- All methods of communication (between devices as well) have the following things in common
 - There is **source** of a message or a **sender**
 - There is a **destination** of the message or a **receiver**
 - There is a **channel** that consists of the media that provides the pathway for the message

- A **computer network** is an infrastructure that allows computing devices to communicate with each other



Communicating Devices

□ End devices (or hosts)

- Any devices that allows us to interface with the network
- Are either the source or destination of a message
- Work stations, servers, laptops, printers, security cameras, etc
- Can be a client or server, or both, depending on the software installed

□ Intermediary devices

- Any device that provides connectivity to the network, connectivity to other network, or links between network segments
- Routers, switches, hubs, wireless access points, security devices



Intermediary Devices

- Manage data as it flows through the network
- Find the best path through the network
- Regenerate and retransmit data signals
- Maintain information about what pathways exist through the network and internetwork
- Notify other devices of errors and communication failures
- Direct data along alternate pathways when there is a link failure
- Permit or deny the flow of data, based on security settings



Media

- Provides the channel over which messages travel from source to destination
- The types are
 - Copper – metallic wires within cables
 - Fiber optics – glass or plastic fibers
 - Wireless – wireless transmission
- Signal encoding
 - Copper – electrical impulses with specific patterns
 - Fiber optics – pulses of light in the infrared or visible ranges
 - Wireless – patterns of electromagnetic waves



LANs, WANs

- LAN – Local Area Network – an individual network usually spans a single geographical area, providing services and applications to people within a common organizational structure, such as a single business, campus or region
- WAN – Wide Area Network – networks that connect LANs in geographically separated locations. Usually implemented with leased connections through a telecommunications service provider (TSP) network
- Internet Service Providers (ISPs) connect their customers to the Internet through their network infrastructure



Communicating the Messages

- Segmentation
- The data stream is divided into smaller, more manageable segments
- Benefits
 - Multiplexing – different transmissions can be interleaved on the network
 - Reliability
- Separate pieces of each message can travel across different paths to destination
- If a part of the message fails to make it to the destination, only the missing part need to be retransmitted



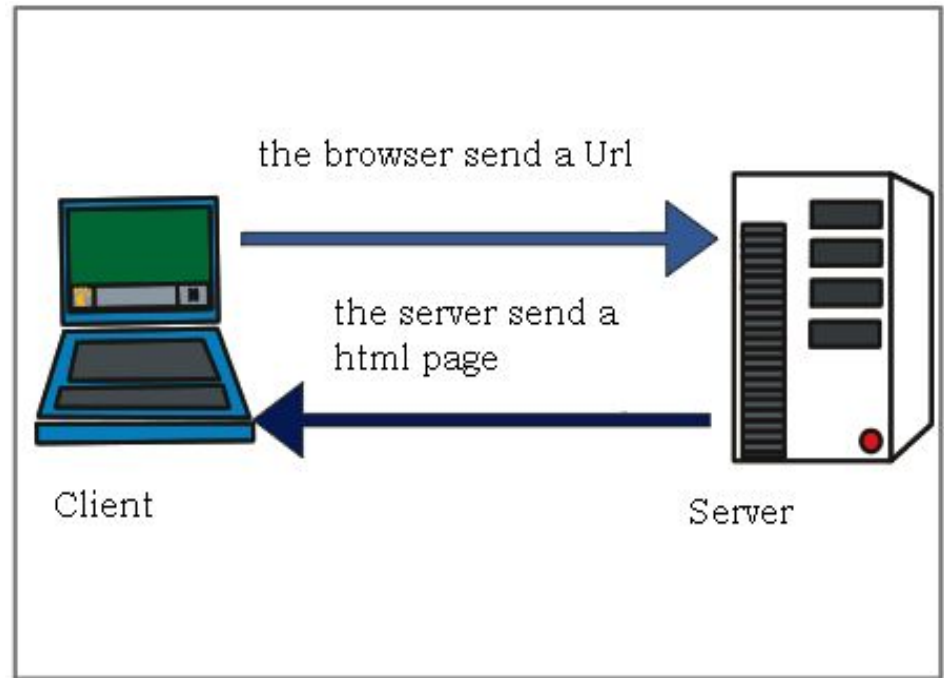
Uses of Computer Networks

- People-computer interaction (Web)
- People-people interaction (Email, video conferencing)
- Computer-computer interaction



Web Browsers and Servers

- Client-server application interaction
- Web browser and server are able to “talk” to each other
- Web browser and server understand each other
- This is achieved by means of a **protocol**



Protocol

- A protocol is a set of rules that allows network applications to talk to each other
- Defines the message format
- A typical message:
 - Header
 - Body
- Defines the behavior of the sender and the receiver
- Defines the rules for establishing and terminating communication sessions
- Protocol suite – a group of inter-related protocols that are necessary to perform a communication function



Layered Models

OSI Model	TCP/IP Protocol Suite						TCP/IP Model
Application	File Transfer	Web Browser	Email	Remote Login	Name Resolution	IP Address	Application
Presentation	FTP TFTP	HTTP	SMTP IMAP POP3	Telnet Rlogin	DNS	DHCP	
Session							
Transport	Transaction Control Protocol TCP			User Datagram Protocol UDP			Transport
Network	Internet Protocol IP			ARP, RARP ICMP			Internet
Data Link	Ethernet	Token Ring	FDDI	WAN Protocols			Network Access
Physical	Copper Twisted Pair Fiber Optic Wireless						



Layered Models

□ Reference (OSI) model

- Provides a common reference for maintaining consistency with

- No

- He

□ Proto

- Clo

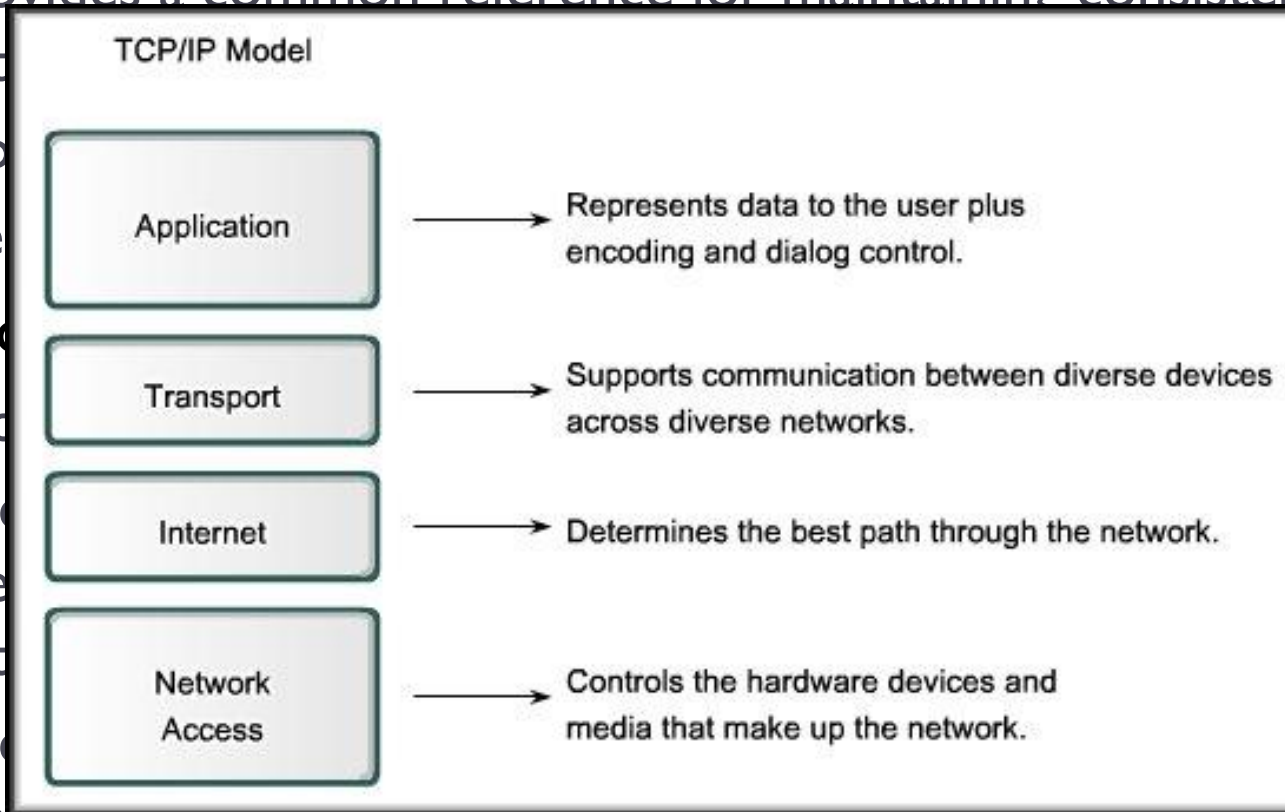
- The

the

wit

- The

functions that occur at each layer or protocols only within the TCP/IP suite



ess

suite

nts all

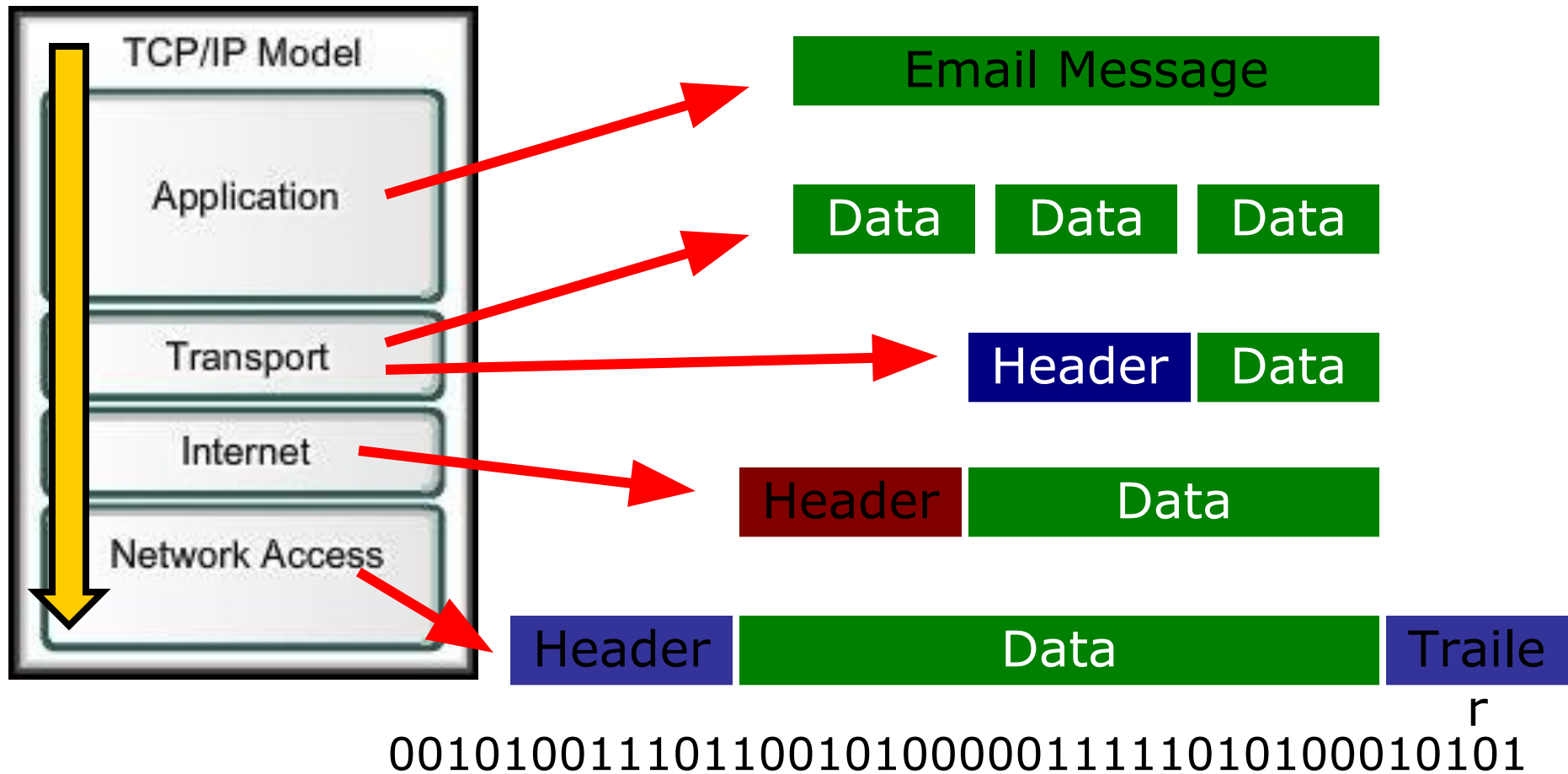
ork

cribes the



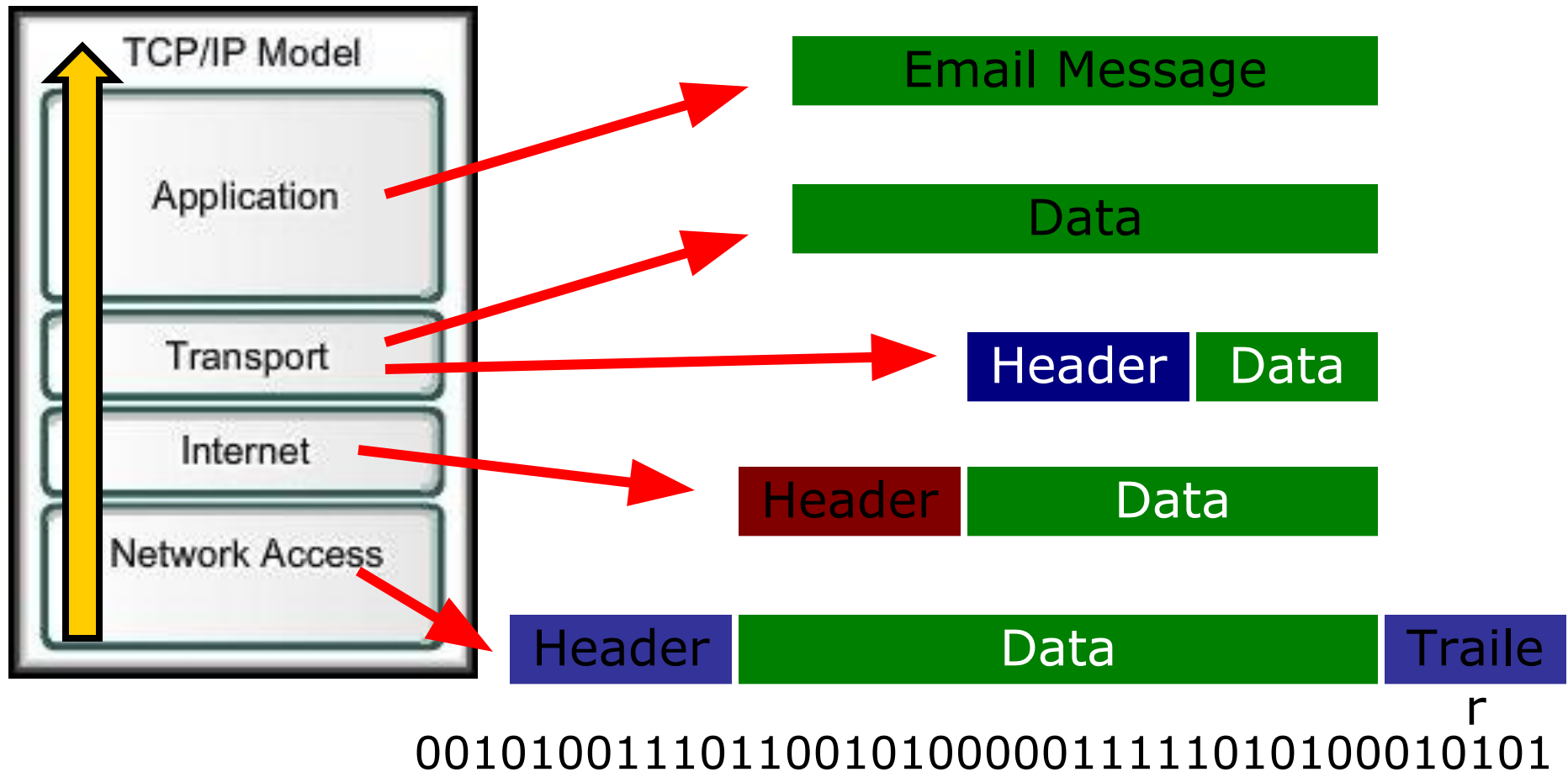
Protocol Data Units and Encapsulation

Segmentation and Encapsulation



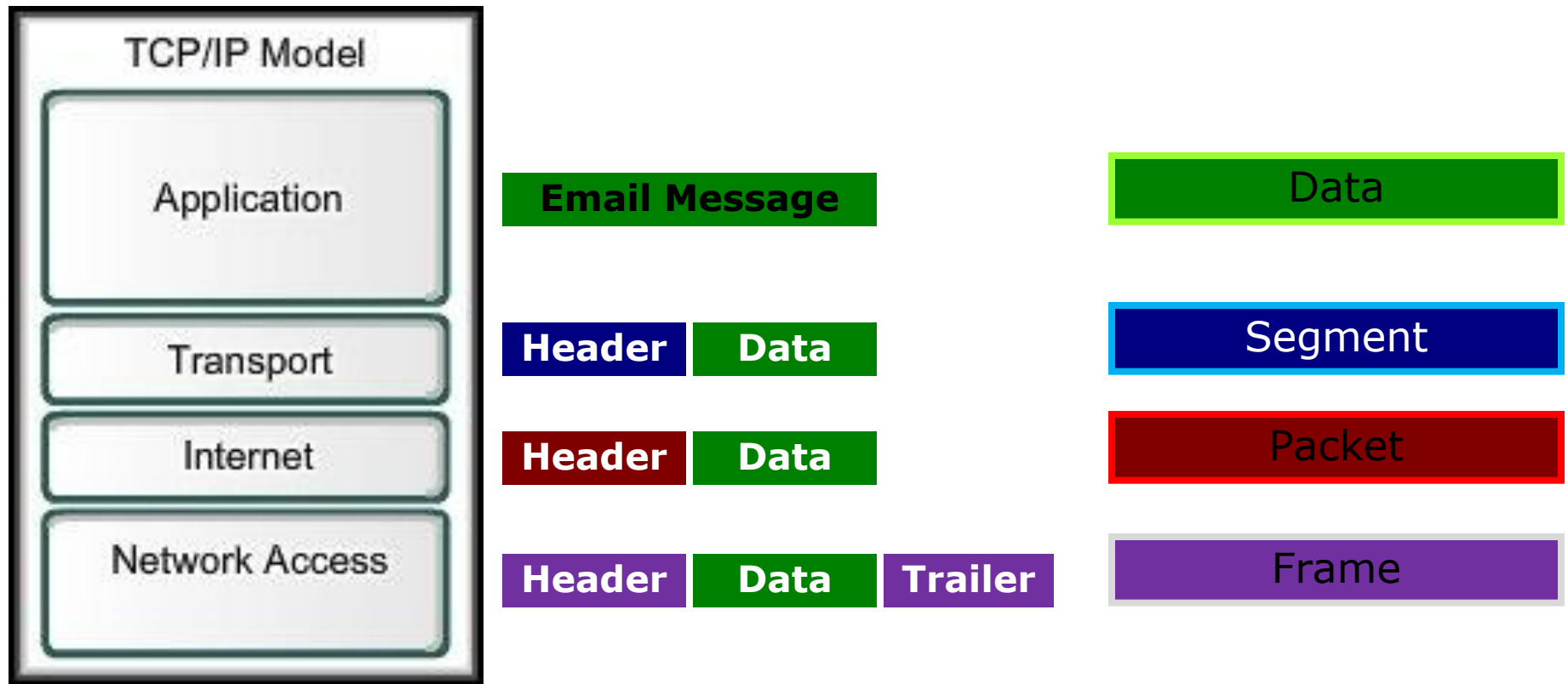
Protocol Data Units and Decapsulation

Decapsulation and Reassembly



Protocol Data Units and Encapsulation

Protocol Data Units



HTTP

- HyperText Transfer Protocol
- Application-level protocol between browsers and servers to deliver *resources* on the World Wide Web
- A browser is an *HTTP client*
- A Web server is an *HTTP server* (Web server)
- The standard (and default) port for HTTP servers is 80
- A resource is some chunk of information that can be identified by a URL (file, dynamically-generated query result, the output of a CGI script, a document that is available in several languages, etc)



Structure of HTTP Transactions

- HTTP uses the client-server model: An *HTTP client* sends a *request message* to an *HTTP server*; the server returns a *response message* with the resource
- Both kinds of messages consists of:
 - an initial line
 - zero or more header lines
 - a blank line (i.e. a CRLF by itself)
 - an optional message body (resource)



Format of an HTTP Message

<initial line, different for request vs. response>

Header1: value1

Header2: value2

Header3: value3

<optional message body goes here, like file contents
or query data; it can be many lines long, or even
binary data \$&*%@!^\$@>



Initial Line for Request

- A request line has three parts, separated by spaces:
 - a *method* name
 - the local path of the requested resource
 - the version of HTTP being used

- A typical request line is

```
GET /path/to/file/index.html HTTP/1.0
```



Initial Line for Response (Status Line)

- A response line has three parts separated by spaces:
 - the HTTP version
 - a *response status code* that gives the result of the request
 - an English *reason phrase* describing the status code. Typical status lines are:
- Typical status lines are
`HTTP/1.0 200 OK` or `HTTP/1.0 404 Not Found`
- The status code is a three-digit integer:
 - **1xx** indicates an informational message only
 - **2xx** indicates success of some kind
 - **3xx** redirects the client to another URL
 - **4xx** indicates an error on the client's part
 - **5xx** indicates an error on the server's part



Header Lines

- Provide information about the request or response, or about the object sent in the message body
- The format is "**Header-Name: value**", ending with CRLF
 - The header name is not case-sensitive (the value may be)
 - Any number of spaces or tabs may be between the ":" and the value
 - Header lines beginning with space or tab are actually part of the previous header line, folded into multiple lines for easy reading
- Thus, the following two headers are equivalent:
 - Header I: some-long-value- I a, some-long-value- I b
 - HEADER I: some-long-value- I a,
 some-long-value- I b



The Message Body

- In a response, the message body contains the requested by the client resource or perhaps explanatory text if there's an error
- In a request, this is where user-entered data or uploaded files are sent to the server
- If an HTTP message includes a body, there are usually header lines in the message that describe the body.
 - The **Content-Type:** header gives the MIME-type of the data in the body, such as **text/html** or **image/gif**
 - The **Content-Length:** header gives the number of bytes in the body



Sample HTTP Exchange (Request)

- The task is to retrieve the file at the URL

`http://www.somehost.com/path/file.html`

- First open a socket to the host **www.somehost.com**, port 80
- Then, send something like the following through the socket:

```
GET /path/file.html HTTP/1.0
```

```
From: someuser@jmarshall.com
```

```
User-Agent: HTTPTool/1.0
```

```
[blank line here]
```



Sample HTTP Exchange (Response)

HTTP/1.0 200 OK

Date: Mon, 29 Sep 2016 01:01:01 GMT

Content-Type: text/html

Content-Length: 1354

<html>

<body>

<h1>Happy New Millennium!</h1>

(more file contents)

.

.

.

</body>

</html>

□ **After sending the response, the server closes the socket**



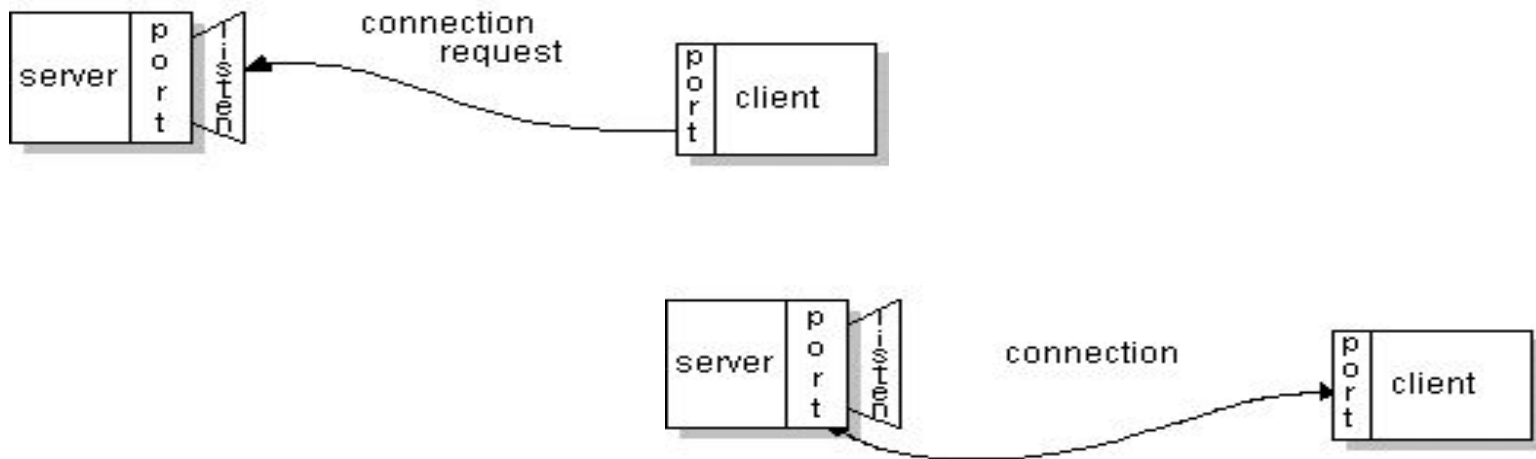
Socket-Level Programming

- Two styles of communication
- First, by establishing a reliable, two-way, byte stream channel. Any data sent by a host will be received correctly by the destination host. Supported by the Transmission Control Protocol (TCP)
- Second, by simply sending messages – datagrams, without establishing a connection or channel. Supported by the User Datagram Protocol (UDP). UDP is more efficient than TCP, but messages might not reach their destination



Sockets

- A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to



Sockets

- An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints
- The `java.net` package in the Java platform provides a class, `Socket`, that implements one side of a two-way connection between your Java program and another program on the network. By using the `Socket` class programs can communicate over the network in a platform-independent fashion
- The `ServerSocket` class implements a socket that servers can use to listen for and accept connections to clients



Sockets – EchoClient Class (1)

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);
```



Sockets – EchoClient Class (2)

```
try (  
    Socket echoSocket = new Socket(hostName, portNumber);  
    PrintWriter out =  
        new PrintWriter(echoSocket.getOutputStream(), true);  
    BufferedReader in = new BufferedReader(  
        new InputStreamReader(echoSocket.getInputStream()));  
    BufferedReader stdIn = new BufferedReader(  
        new InputStreamReader(System.in))  
    )
```



Sockets – EchoClient Class (3)

```
{
    String userInput;
    while ((userInput = stdin.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
    }
} catch (UnknownHostException e) {
    System.err.println("Don't know about host " + hostName);
    System.exit(1);
} catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection
to " + hostName);
    System.exit(1);
}
}
```



Sockets – EchoServer (1)

```
import java.net.*;
import java.io.*;

public class EchoServer {
    public static void main(String[] args) throws IOException {

        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);
```



Sockets – EchoServer (2)

```
try (  
    ServerSocket serverSocket =  
        new ServerSocket(Integer.parseInt(args[0]));  
    Socket clientSocket = serverSocket.accept();  
    PrintWriter out =  
        new PrintWriter(clientSocket.getOutputStream(),  
true);  
    BufferedReader in = new BufferedReader(  
        new InputStreamReader(clientSocket.getInputStream()));  
    )
```



Sockets – EchoServer (3)

```
    {
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            out.println(inputLine);
        }
    } catch (IOException e) {
        System.out.println("Exception caught when trying to listen
on port "
            + portNumber + " or listening for a connection");
        System.out.println(e.getMessage());
    }
}
```

