CO324: Network and Web Application Design

Network Programming

Client-server Model

- Network programs are typically play two roles.
  - **Client (user agent)** end-users applications used to access various *services* such as mail and web. Typically run on smartphones and PCs.
  - **Server (daemon)** a program providing the *service* such as serving a website. Typically run on dedicated hardware or, cloud-based VMs.
- We use these terms refer to *software* in this course.

IP Addresses and Ports

- A machine commonly has a single IP address, but runs multiple network programs.
- How do we distinguish packets meant for different programs?
- **Port number** a unique integer identifier for each client/server located a particular IP.
- **Privileged ports** Ports up to 1023 are assigned by the IANA for hosting *well-known* services.
  - e.g. a web server listens on port 80. ssh (22) smtp (25)
- Privileged ports require administrative rights (“root”) for use.

Sockets

- sockets, as one of the major solutions employed by network programming for the inter-process communications.
- Sockets provide the application developer with direct basic access to transport protocols, offering data packet transport services between a sender and a receiver host over the network, while hiding the complexity and implementation details of the protocol stack below.
- The most popular transport protocols are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

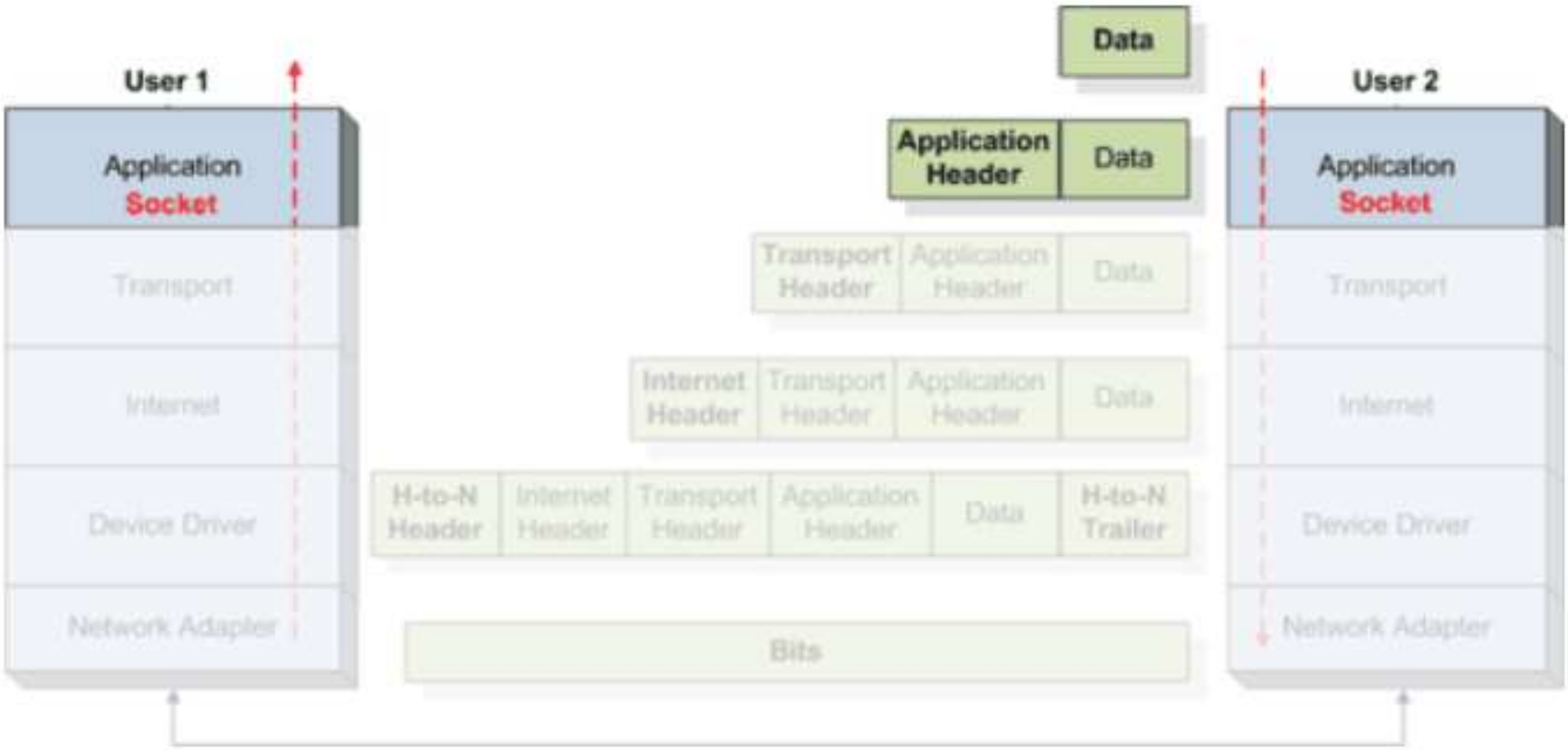

Fig. 5.1 Socket-based network data communication

Transport Layer Sockets

- *Transport layer sockets* make use of transport-layer protocols such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).
- UDP is a connectionless non-reliable transmission of datagrams protocol, similar to the postal service.
- TCP is a connection-based reliable, orderly transmission of data packets, similar to the telephone service.

Sockets in Java

- A *network socket* represents an endpoint of communication. A socket is *bound* to a port on the local machine.
- Most languages have a socket API based on the original API developed for BSD UNIX. e.g. UDP communication is set up in Java as,

```
DatagramSocket s = new DatagramSocket();
```

- This constructs a datagram socket and binds it to any available port on the local host machine

Name Resolution

- We prefer to use *hostnames* rather than IP addresses to refer to servers.
- DNS is used to *resolve* hostnames to addresses. Networking API usually provides a function to do this.

// Construct IP address from hostname given in args [0]

```
InetAddress address = InetAddress.getByName(args[0]);
```

The host name can either be a machine name, such as “aiken.ce.pdn.ac.lk", or a textual representation of its IP address.

Sending a Datagram

```
// Construct IP address from hostname given in args [0]
InetAddress address = InetAddress.getByName(args[0]);
// Construct socket.
DatagramSocket ds = new DatagramSocket();
```

connection-less socket for sending and receiving datagram packets

```
// Construct Packet with destination address , port and, data.
byte[] buf = new byte[256];
DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 12345);

//Send the packet.
ds.send(packet);
```

Note: exception handling has been omitted.

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

Error Handling

- How you handle an error depends on the severity of the error.
- **Permanent failures** the error condition is not likely to change. Only possibility is to inform user or administrator (in the case of a server.) Examples: no connectivity, incorrect peer address, name resolution failure.
- **Transient errors** the error condition is temporary. Possibility of recovering by retrying the operation. Examples: slow or unreliable network connection.

Java Exceptions

- Java uses *checked exceptions* to signify errors. They must be handled in a `try - catch` block or declared in the method's `throws` clause.
- **Permanent failures** display or log the error and abort the operation.
- **Transient errors** retry the operation a given number of times before giving up.
- Never just discard exceptions in a catch block!

Error Handling with throws Clause

```
public static void main(String[] args) throws Exception {
  InetAddress address = InetAddress.getByName(args[0]);
  DatagramSocket ds = new DatagramSocket();

  byte[] buf = new byte[256];
  DatagramPacket packet = new DatagramPacket(
    buf, buf.length, address, 12345);

  ds.send(packet);
}
```

- Simply throwing exceptions from main is bad practice!

Error handling with try-catch

```
public static void main(String[] args) {
   try {
      InetAddress address = InetAddress.getByName(args[0]);
      DatagramSocket ds = new DatagramSocket();

      byte[] buf = new byte[256];
      DatagramPacket packet = new DatagramPacket(
      buf, buf.length, address, 12345);

      ds.send(packet);
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Network Exception Types

Method / Constructor	Exception type
InetAddress.getByName	UnknownHostException
DatagramSocket	SocketException
DatagramSocket.send	IOException

UDP server

```
// Set up the socket before the server loop
while (true) {
  try {
    byte[] buf = new byte[256];
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    //Extract sender address and port.
    InetAddress address = packet.getAddress();
    int port = packet.getPort();
    //Send back response.
    packet = new DatagramPacket(buf, buf.length, address, port);
    socket.send(packet);
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Servers

- Why does the server sit in an infinite loop? In what cases (if any) should this loop exit?
- Why does a server need to bind to a well-known port, whereas a client can use any available one?

Protocol properties

- UDP provides data integrity only.
- An application can implement other properties atop UDP.
- **Reliability** Requires message acknowledgement.
- **Ordering** Requires message sequence numbering.
- **Flow control** Ensures that sender does not overwhelm receiver. Requires a windowing protocol.
- If all three are required, just use TCP!

Multicast

- Multicast lets you send a datagram to a group of recipients, with just a single transmission, e.g., webcast and IPTV.
- A group of recipients defined by a *multicast group* that has an address in the Class-D range 224.0.0.0/4. Multicast enabled routers are needed to forward traffic across subnets.
- Better supported in IPv6 than IPv4.

Multicast in Java

```
byte[] inBuf = new byte[256];
try {
  //Prepare to join multicast group
  MulticastSocket socket = new MulticastSocket(8888);
  InetAddress address = InetAddress.getByAddress("224.2.2.3");
  socket.joinGroup(address);
  while (true) {
    DatagramPacket inPacket = new DatagramPacket(inBuf, inBuf.length);
    socket.receive(inPacket);
  }
} catch (IOException ioe) {
  System.out.println(ioe);
}
```

Fragmentation

- A IP packet may be *fragmented* if its size is larger than the *maximum transfer unit* MTU of a link. Degrades performance.
- To avoid this the maximum recommended size of a UDP payload is 512 bytes.

Connectionless (UDP) socket operations