CSE 3231
Computer Networks

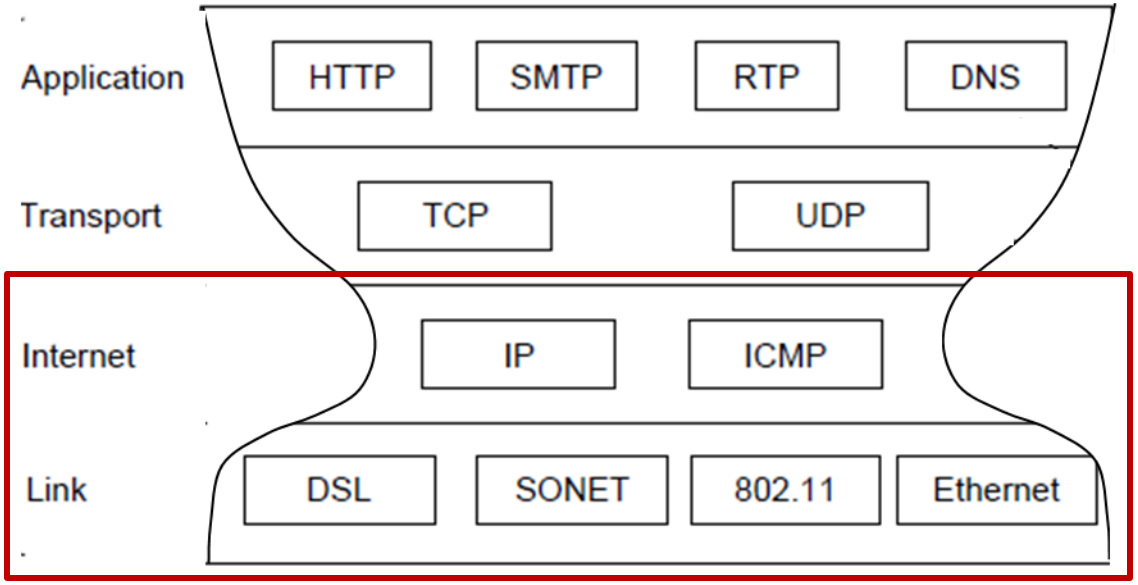
Chapter 6
The Transport Layer
part 1

William Allen, PhD

Spring 2022

Review: the TCP/IP Model

The four layers of the TCP/IP reference model provide modularity, flexibility and reliability to support many different network applications



Protocols are shown in their respective layers

So far, we have looked at the lower two layers of the TCP/IP model used in the Internet

Link-Level Connections

When one node (computer) sends a frame to another node over a *direct link*, only the Data Link-Level protocols are used

- Encoding, using NRZ, Manchester, 4B5B, etc.
- Framing, using BISYNC, HDLC, PPP, etc.
- Error detection, using CRC or similar methods
- Reliable delivery, using Stop and Wait, Sliding Window, etc.

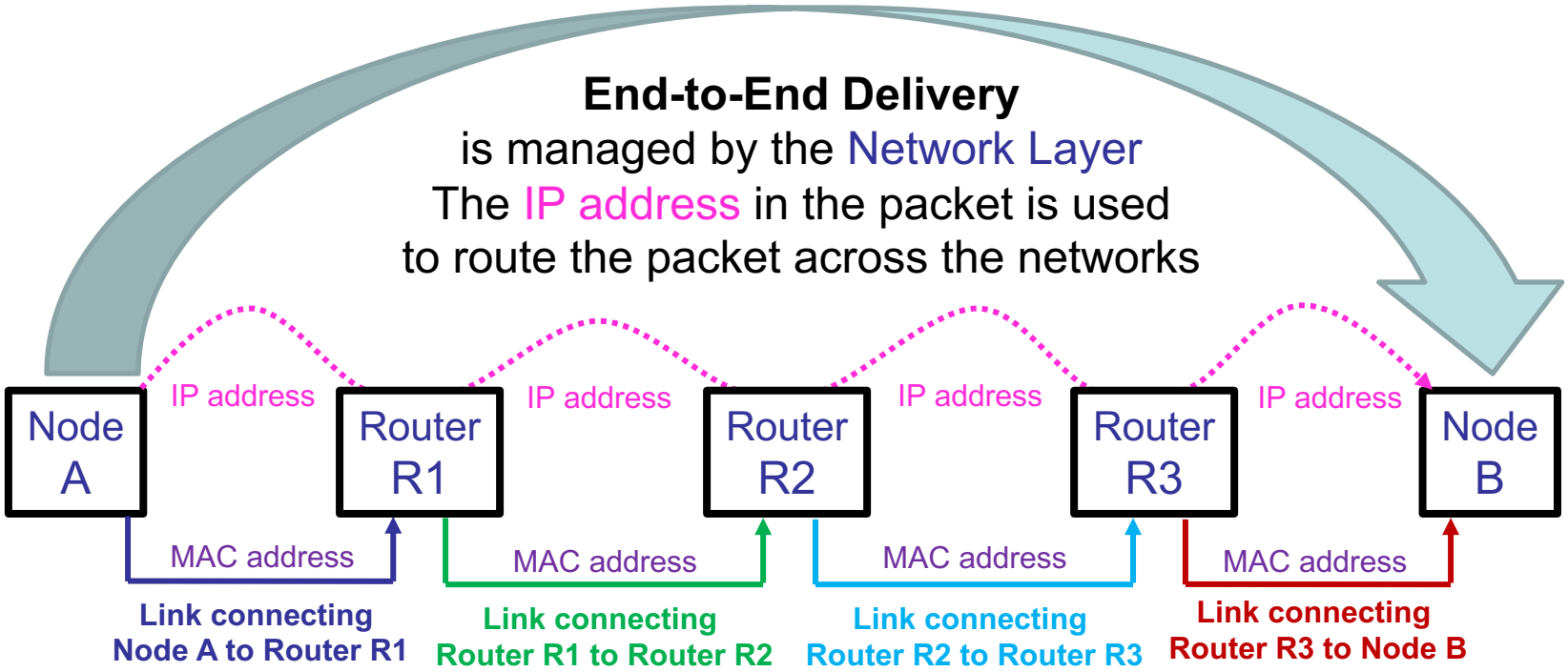
At the Data Link Level, Frames Are Delivered by Using the MAC Address

- Node A has to "know" node B's MAC address to send frames directly to node B
  - Node B will only receive frames that have its MAC address in the destination field
  - ARP can help node A get B's MAC address
- However, we *cannot* use MAC addresses to deliver messages outside of the local network because we don't know where they are located
  - nodes can't store the network locations of all MAC addresses and we can't ARP the entire Internet
  - IP addressing supports Internet-wide delivery

Routing from One Local Network to Another

- We route packets from one network to another to get them to their final destination
- This requires that we use a combination of *both MAC addresses and IP addresses*
  - In a local network, the MAC address is used to deliver frames across links from router to router
  - But to relay messages from one network to another, the router located between those networks uses the IP address to determine which router to send it to
  - Therefore, the Data Link and Network layers work together to deliver packets across the Internet

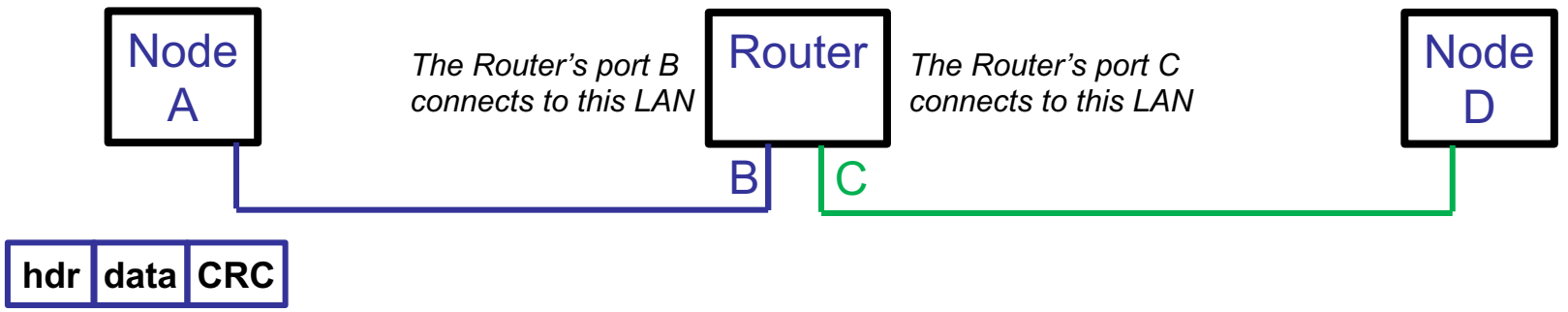
End to End vs Link-Level



The Data Link-Layer wraps each packet in a frame and reliably delivers those frames across each of the links that connect the routers along the path from node A to node B.

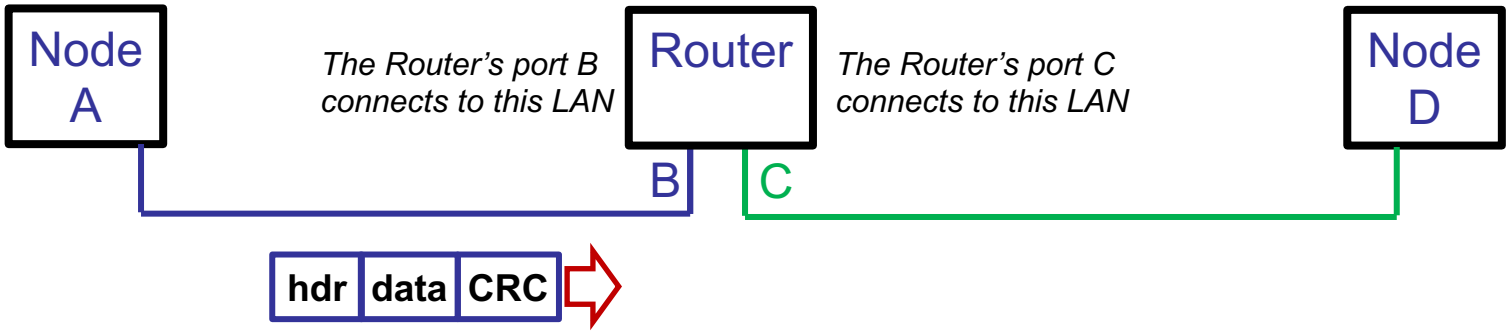
It uses the MAC address in the frame's header to deliver the frames to the next router and to acknowledge that each frame arrived.

Node-to-Node Delivery



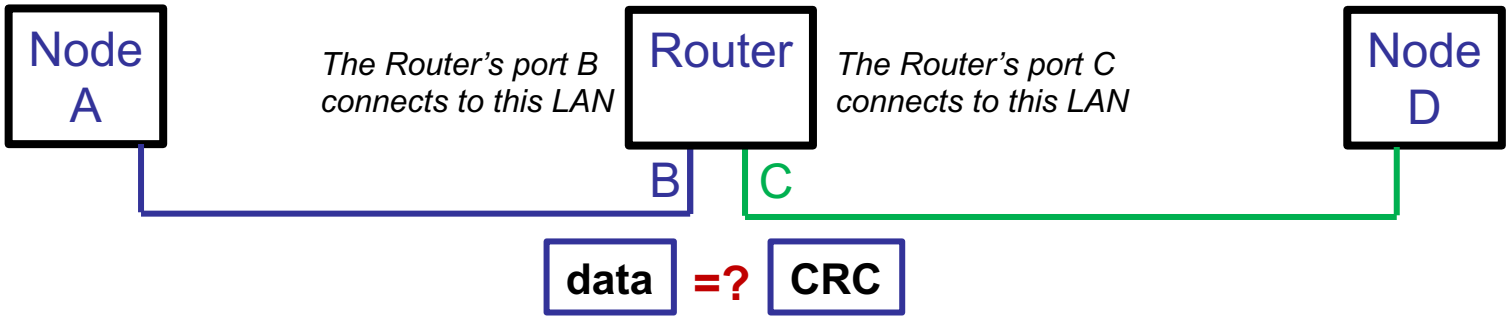
1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.

Node-to-Node Delivery



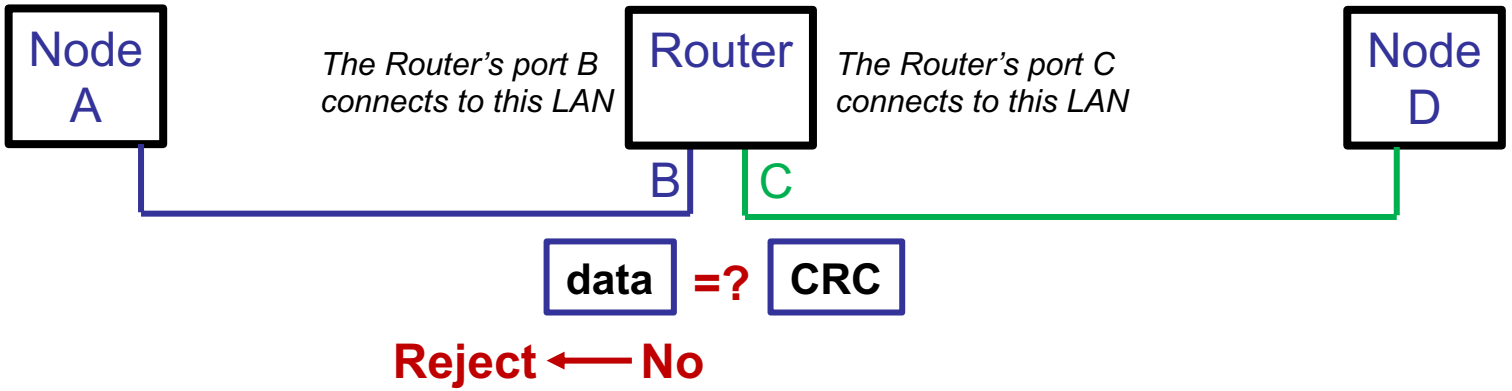
1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.

Node-to-Node Delivery



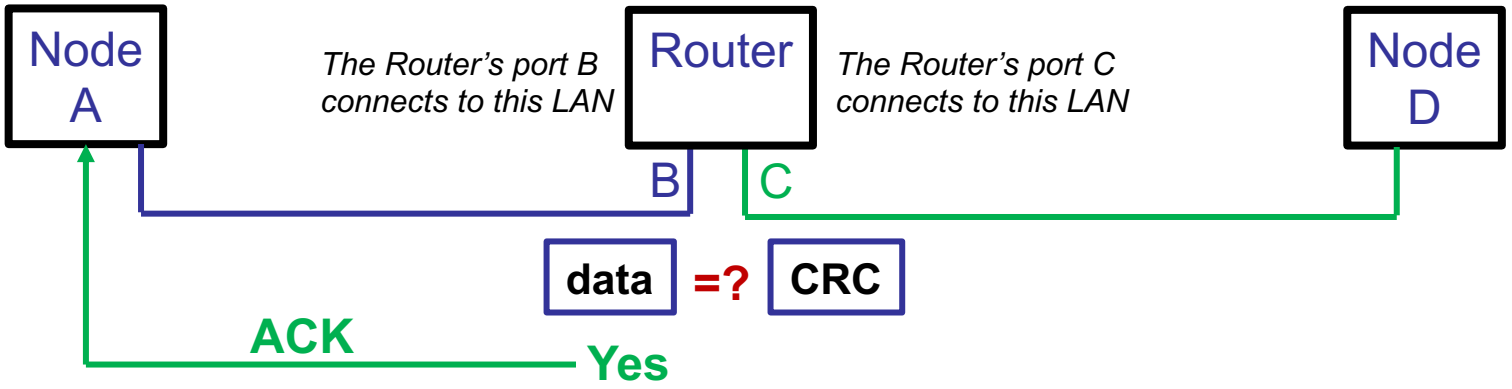
1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.
3. The Router calculates the CRC remainder to see if an error occurred.

Node-to-Node Delivery



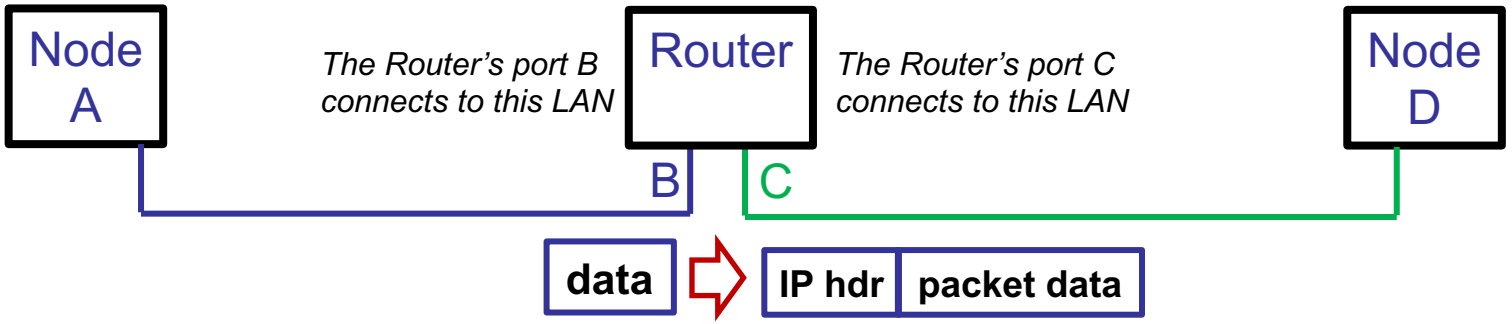
1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.
3. The Router calculates the CRC remainder to see if an error occurred.
4. If it is not valid, then the Router rejects the frame and it must be retransmitted.

Node-to-Node Delivery



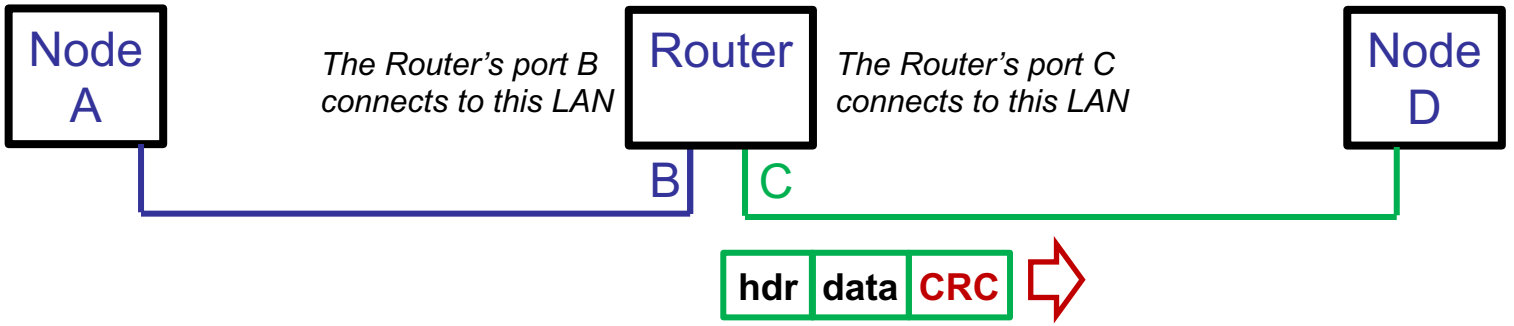
1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.
3. The Router calculates the CRC remainder to see if an error occurred.
4. If it is not valid, then the Router rejects the frame and it must be retransmitted. If it is valid then the frame is accepted and the Router *sends an ACK* to Node A.

Node-to-Node Delivery



1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.
3. The Router calculates the CRC remainder to see if an error occurred.
4. If it is not valid, then the Router rejects the frame and it must be retransmitted. If it is valid then the frame is accepted and the Router *sends an ACK* to Node A. Then, the Router *extracts the IP packet* and looks at its IP address to see where it should be routed to.

Node-to-Node Delivery



1. Node A creates a frame in memory with the Router's MAC address as the destination, calculates the CRC and adds it to the frame.
2. Then it encodes the binary data in some format (like 4B5B) and sends it over the delivery medium to input B on the Router.
3. The Router calculates the CRC remainder to see if an error occurred.
4. If it is not valid, then the Router rejects the frame and it must be retransmitted. If it is valid then the frame is accepted and the Router *sends an ACK* to Node A. Then, the Router *extracts the IP packet* and looks at its IP address to see where it should be routed to.
5. It then follows the same steps to send the frame along the next link towards the destination. It must *recalculate the CRC* because the destination address in the header is now Node D's MAC address.

End-to-end Protocols

- There are limitations in an Internet Protocol (IP) network because IP does not guarantee the following properties:
  - May drop messages
  - May deliver parts of a message out of order
  - May deliver duplicate copies of a given message
  - May limit messages to some finite size
  - May deliver messages after an arbitrarily long delay
- Therefore, we need an end-to-end protocol to provide the missing features

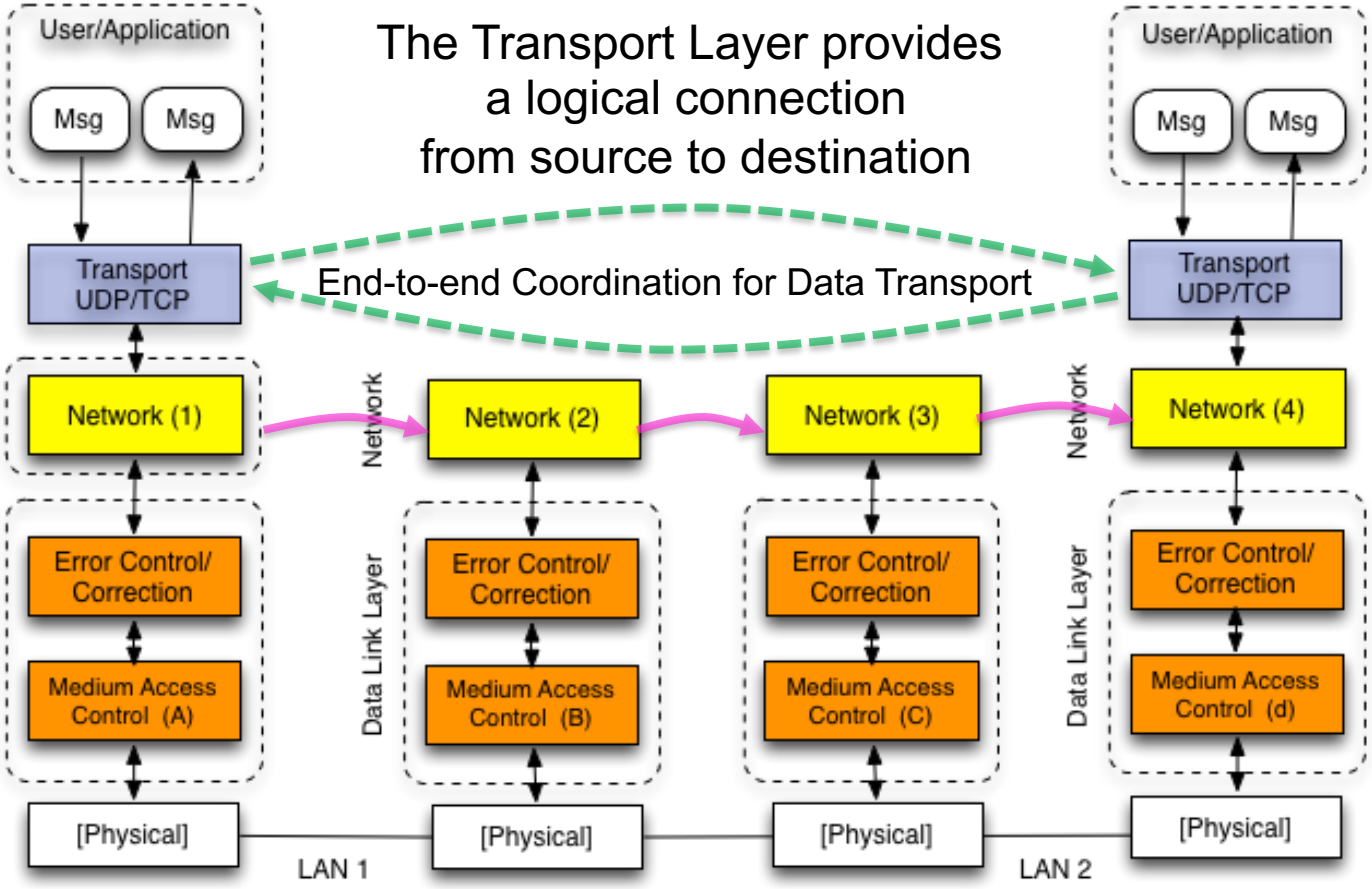
End-to-end Protocols

- Common properties that a transport protocol can be expected to provide:
  - Guarantees message delivery
  - Delivers messages in the same order they were sent
  - Delivers at most one copy of each message
  - Supports arbitrarily large messages
  - Supports synchronization between the sender and the receiver
  - Allows the receiver to apply flow control to the sender
    - i.e., control the rate of packet transmission
  - Supports multiple application processes on each host

The Transport Layer

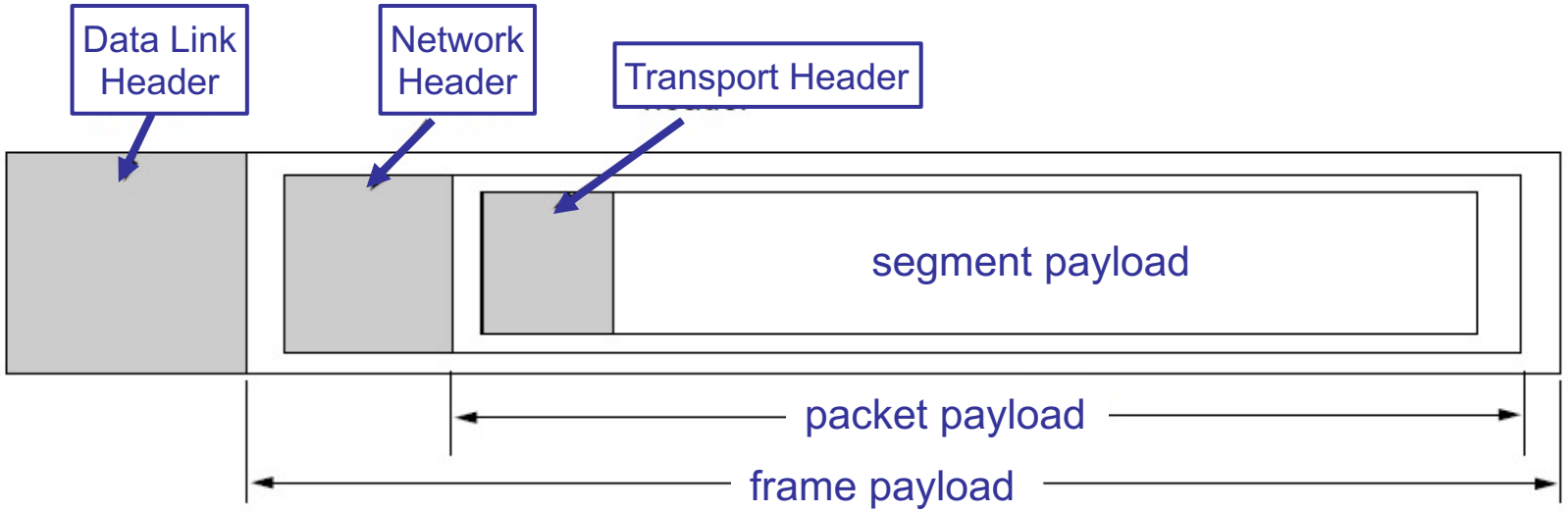
The Network Layer moves packets from node to node

The Transport Layer provides a logical connection from source to destination



Services Provided by Transport Layer Protocols

1. The transport layer delivers blocks of data called *segments* to the network layer
2. The segments are then wrapped in *packets* by the Network Layer
3. And in *frames* by the Data Link Layer

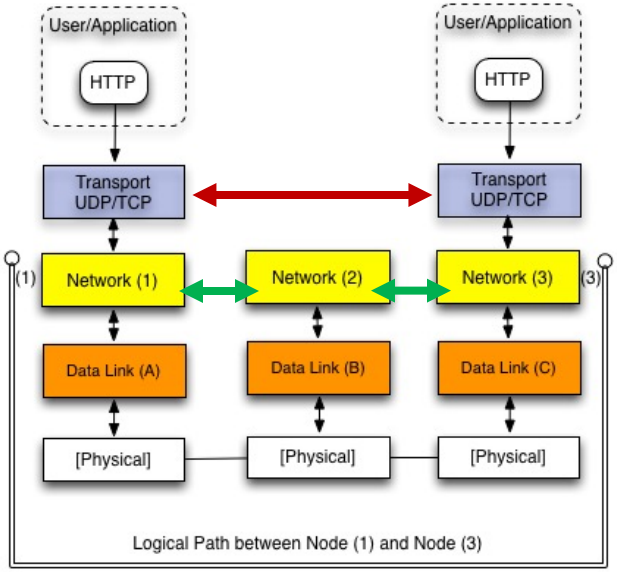


Common Transport Layers

- Several transport-layer protocols have been proposed, developed and applied to different applications and network settings
- Two of the most common transport protocols used in the Internet are:
  - **UDP** (User Datagram Protocol)
  - **TCP** (Transmission Control Protocol)
- Together, these two protocols cover the major portion of the end-to-end transport requirements to applications and services.

Interactions with the Network Layer

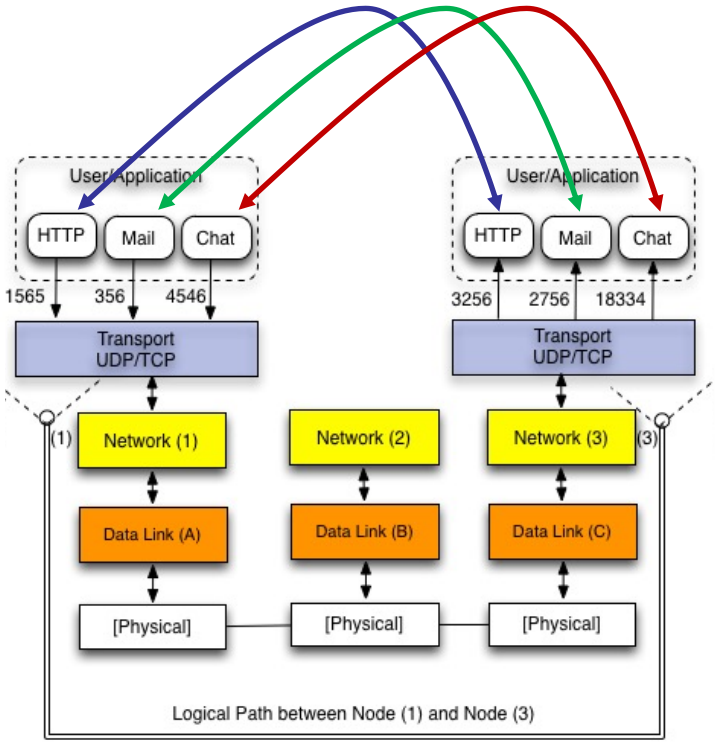
- The Network Layer provides an end to end path between two machines
  - From a network perspective every packet has a source host (and its internet address), and a destination host (and its Internet address)
  - Network protocols handle packet delivery between the end points by routing packets from node to node



Transport protocols can establish a connection directly between the end-point nodes and exchange information directly between those end-points to guarantee delivery of segments.

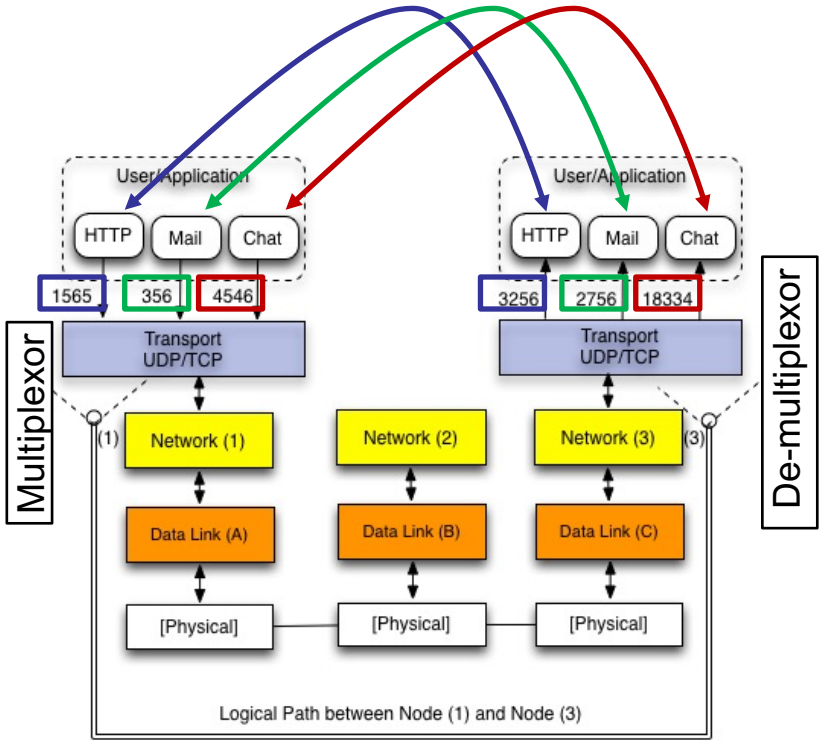
Multiplexing and De-multiplexing

- The transport layer must support multiple processes interacting between two hosts along the same network path provided by the network layer.
- After network packets are reassembled into segments by the receiver, the transport layer must deliver the segments to the appropriate process



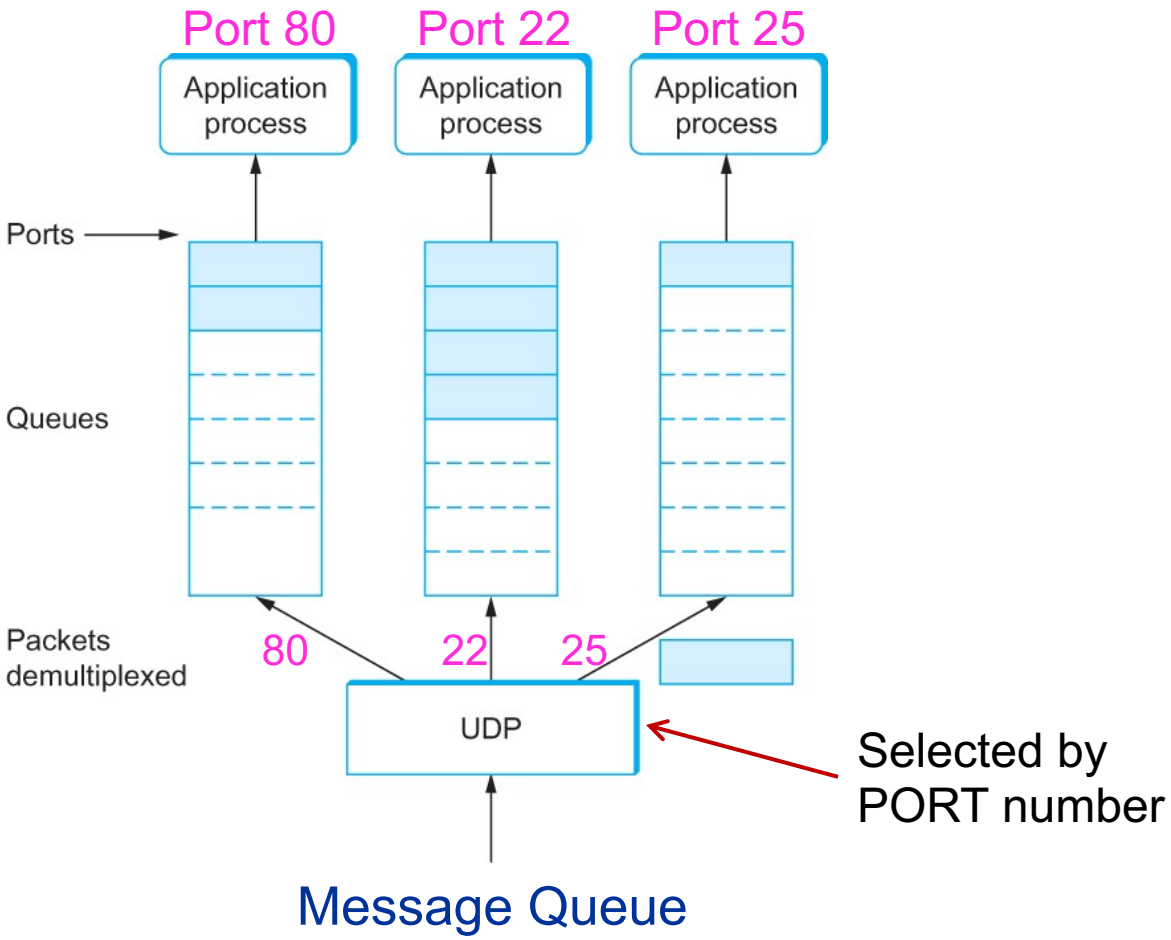
Multiplexing and De-multiplexing

- The transport layer creates an interface identifier that connects sessions between corresponding applications.
- These identifiers are the source and destination *PORT* numbers and are are established when the transport layer negotiates the end-to-end connection



- Ports are used to *multiplex* the segments at the sender side and *demultiplex* the segments at the receiver side

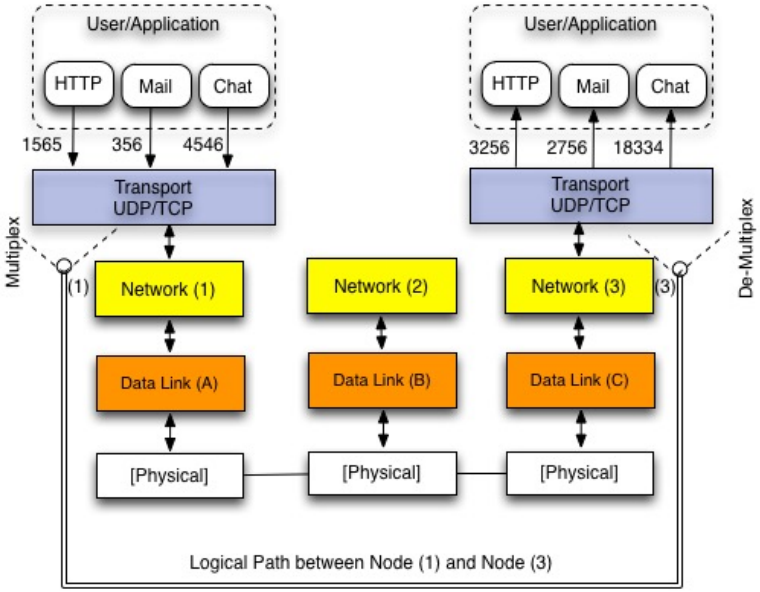
Simple Demultiplexer



Multiplexing and De-multiplexing

The most common interfaces to the transport layer are the Socket APIs (for UDP and TCP sockets)

- When a client application (HTTP client, for example) creates a new socket connection, source and destination ports are created for that connection to allow the multiplexing of messages



Java Example:

No port specified

Port specified

```
DatagramSocket s = new Datagramsocket()
DatagramSocket s = new DatagramSocket (27884)
```

Well-Known Port Numbers

- A range of UDP and TCP ports are assigned to specific protocols, others are not reserved
  - Ports numbered 0-1023 are reserved by the Internet Assigned Numbers Authority (IANA)
  - Servers listen on specific ports to provide services
    - SSH - port 22
    - HTTP - port 80
    - NTP (Network Time Protocol) - port 123
    - IMAP (email) - port 143 or port 220
    - LDAP (authentication protocol) - port 289
    - HTTPS (using TLS/SSL) - port 443

Connectionless Transport (UDP)

- The User Datagram Protocol (UDP) is a *connectionless* protocol and does not create a *connection* between sender and receiver
  - The destination address and port number are added to the transport segment's header and the segment is sent to the destination.
    - When received, the segment is delivered to the process associated with the identified port
  - No confirmation of error or of delivery (ACK) is returned to the sender
    - Because there is no confirmation of delivery – the protocol is considered to be ***unreliable***

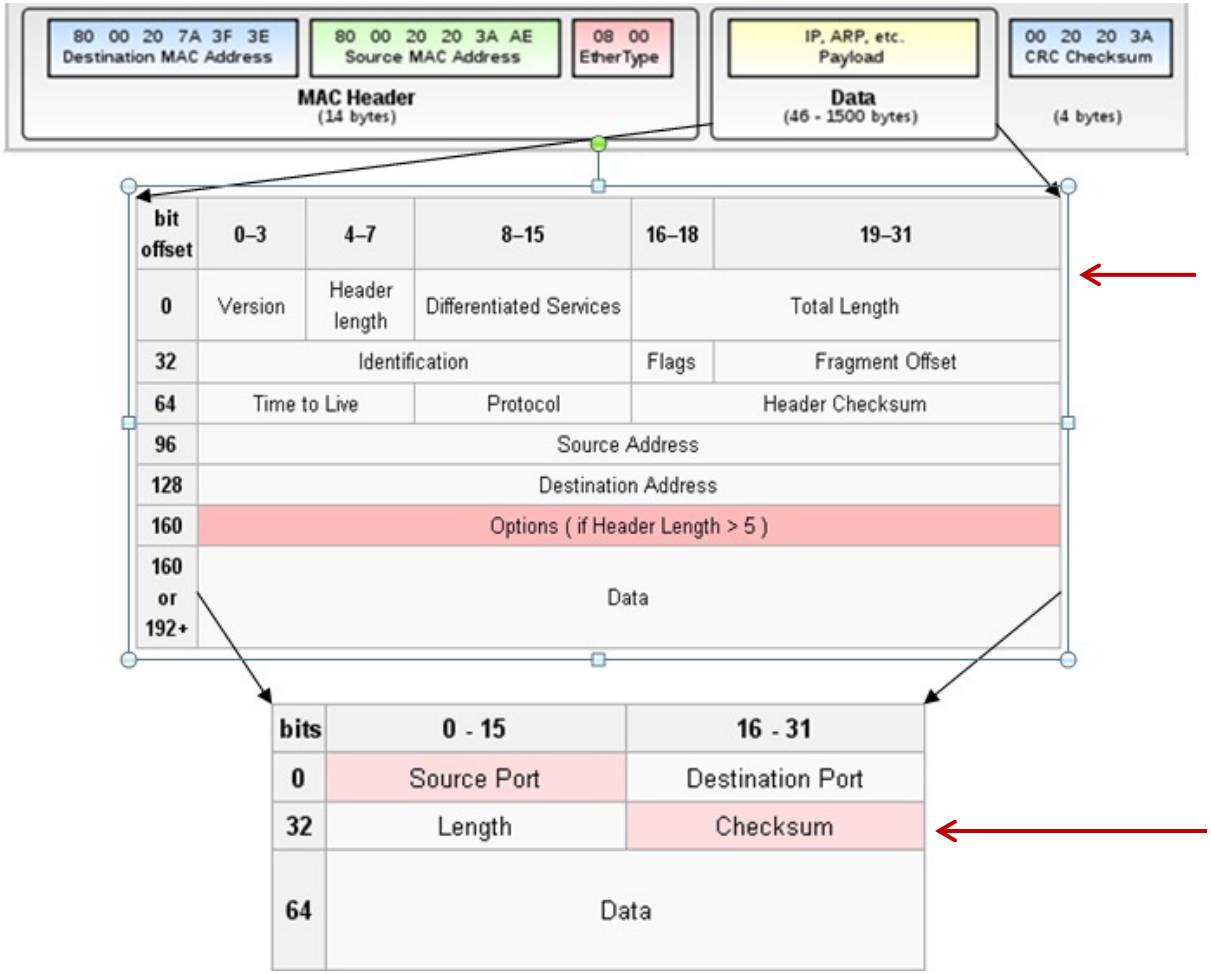
Advantages of UDP

- Applications can have direct control over data transmission
  - Applications pass data packets directly to the transport layer, where they are encapsulated and transmitted.
  - As an unreliable protocol, there are no congestion-control or retransmission mechanisms taking place.
  - Data is transmitted immediately and will either reach the receiver or not.

Advantages of UDP

- No Connection Establishment
  - There are no additional messages, or negotiations that take place outside the transmitted data packet
  - There are no delays caused by connection establishment and negotiation
- No connection state
  - UDP does not maintain any state information, which allows for more client connections per server
- Small Packet Header
  - Adds less overhead to each packet
- Messages can be sent in broadcast or multicast mode
  - one message can be sent to multiple receivers (multicast) or all nodes on the network (broadcast)

Ethernet Frame, IP & UDP Headers

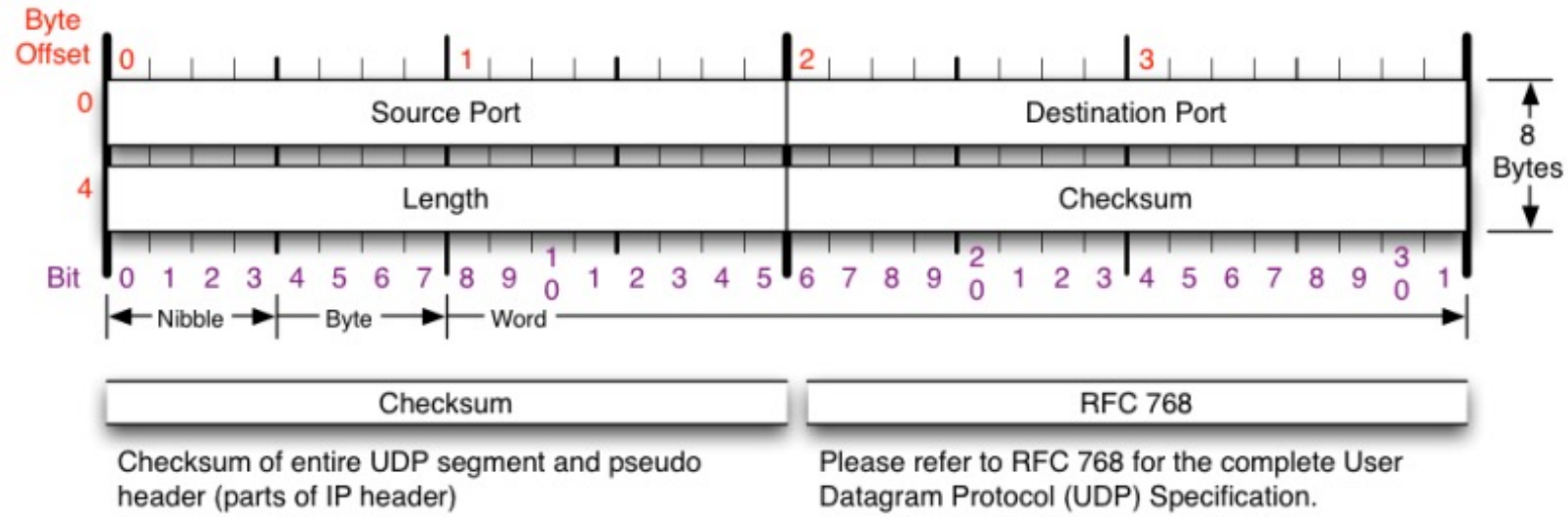


Ethernet (Data Link Layer) Frame

Internet Protocol (IP) Header

UDP Header

UDP Header

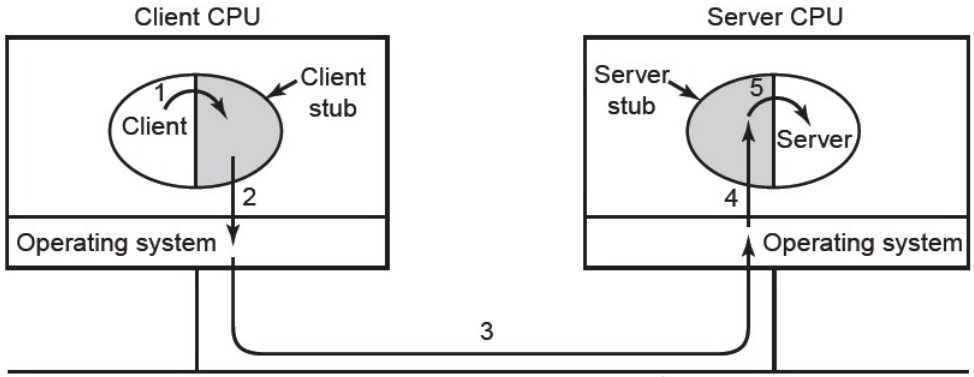


Note: The checksum field is initially set to all 0's and the checksum value is calculated over both the header fields and the data portion of the packet.

Some Uses for UDP

A Remote Procedure Call (RPC) lets users execute code on a remote server in a way that resembles procedure calls across a network

- Client code calls a "stub" which links the call to the Operating System and passes parameters by creating packets containing the parameter values
- The server program is also linked to the OS and accepts parameters through stub code at the server



Some Uses for UDP

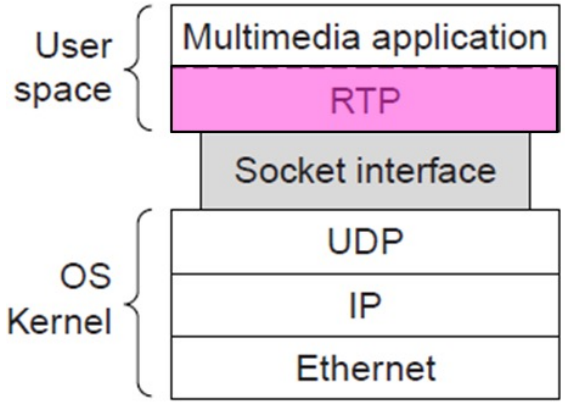
Other features/issues of RPC:

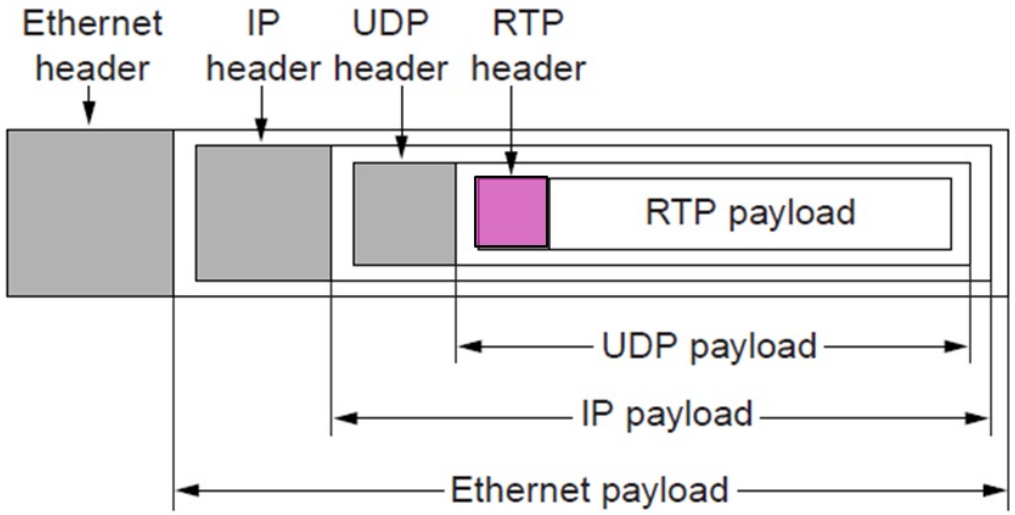
- The Operating Systems at each endpoint manage connections and retransmission of lost packets
- Details of how the RPC actually occurs are hidden from programmers creating client and server code
- However, there are a number of issues with RPC
  - pass by reference must either be converted to two-way pass by value or may be restricted to certain data types
  - weakly-typed languages have issues with bounds checking and calling functions with var-args (like printf)
  - global variables are problematic, no memory is shared
  - RPC can be useful with the correct combination of language and program design

Real-Time Transport

RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP

- RTP can multiplex streams of audio and video
- RTP would sit between UDP and the application, thus it is often implemented as part of the application

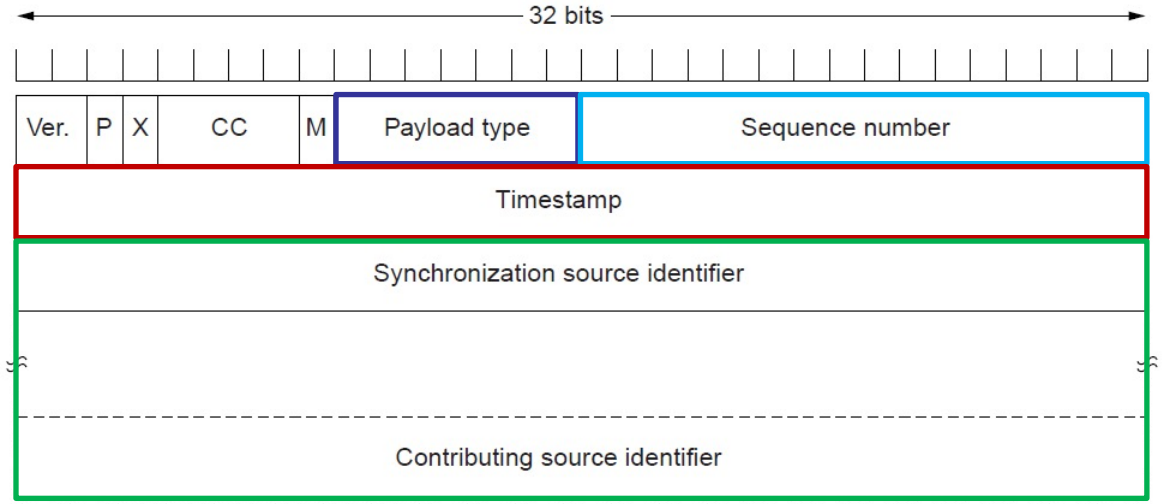




Real-Time Transport

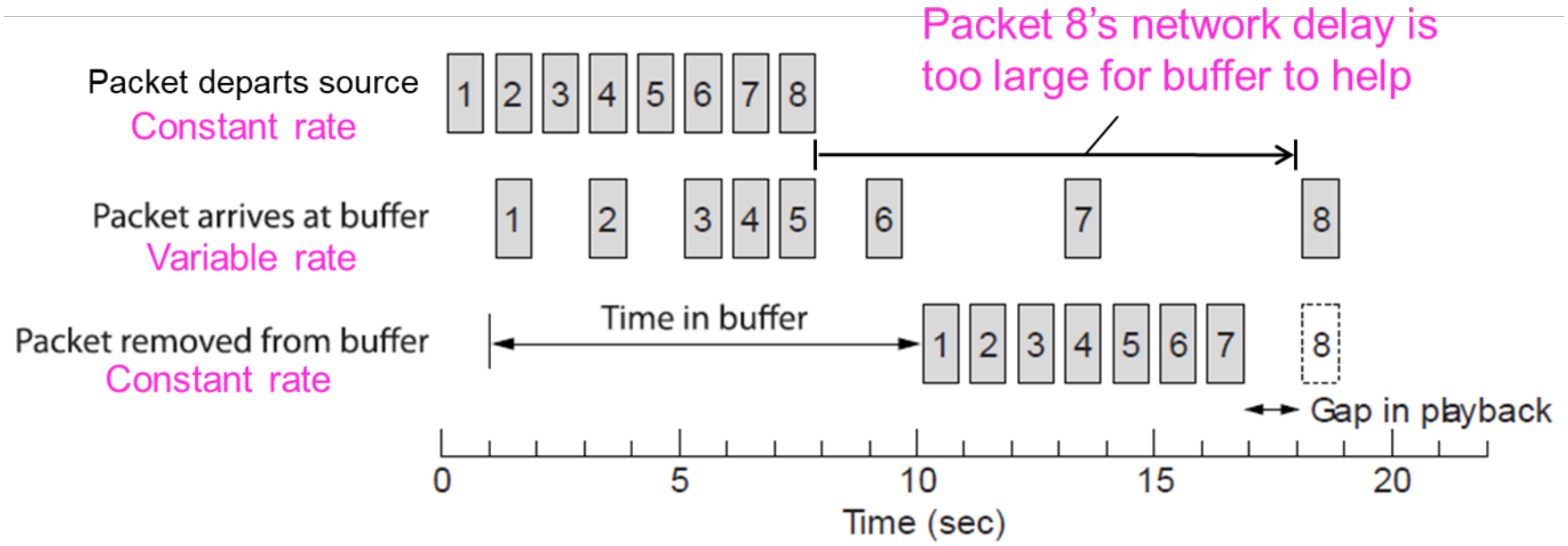
RTP header fields describe the type of media and synchronize it across multiple streams

- Payload type field indicated media encoding method
- Sequence numbers are used to detect when packets are lost
- Timestamps indicate expected time interval between packets
- Source identifiers are used to multiplex media streams



Real-Time Transport

Buffer at receiver is used to delay packets and absorb jitter so that streaming media is played out smoothly without losing data samples



Real-Time Transport

Higher rates of jitter, or more variation in delay, requires a larger playback buffer to avoid missing samples in playback