# Transport layer Protocols (layer 4)

**TCP**

**TCP Acknowledgement**

**UDP**

# Transport layer Protocols (level 4)

- Layer 3 downwards is responsible for moving data across the Internet, and is largely hidden from application processes.

- Transport layer protocols provide services for higher level software applications to work over the internet

- The two key protocols are
- Transport Control Protocol (TCP)
- User Datagram Protocol (UDP)

# TCP and UDP

- Recall that the layer 3 IP protocol is *connectionless*, *unreliable* and *unacknowledged*.
  - The IP protocol makes a *best effort* to deliver the data, but there is no guarantee.
- TCP
  - Is a connection oriented, reliable protocol that allows multiple software applications to communicate simultaneously using a single IP address.
  - It is fully featured but has higher overheads
- UDP provides addressing but little else
  - No connections are established and data can be lost
  - Very efficient

# Ports and Sockets

- The IP protocol allows two devices on the internet to exchange information.
- However there are many different processes on computers, so it is not sufficient to just specify an IP number.

- In order to identify which process the data should go to, a **port number** is used in combination with the IP address

# Clients and Servers

- Most internet communication follows the client server model.
- A client computer sends a message to a server computer and the server responds

- For example, when you use the world wide web, your browser (the client) sends a Hypertext Transfer Protocol (HTTP) message to a web server
- For this to work, the client needs to know the port number of the web server process on the remote machine, as well as the IP address

# Poera

- Since clients initiate contact with servers, and not the other way around, servers for common applications need to have “well known” port numbers
- The use of port numbers (range 0-65535) is controlled by the Internet Assigned Numbers Authority (IANA)
- Port numbers 0-1023 are reserved for common TCP/IP applications and controlled by IANA. Some common ones include

| | | | |
|---|---|---|---|
| FTP | 20 | HTTP | 80 |
| SSH | 22 | HTTPS | 443 |
| TELNET | 23 | DHCP client | 546 |
| SMTP | 25 | DNS | 53 |
| BGP | 179 | | |

- When a client opens a connection to a server it is allocated a socket number (> 1023) by the TCP/IP software on the client

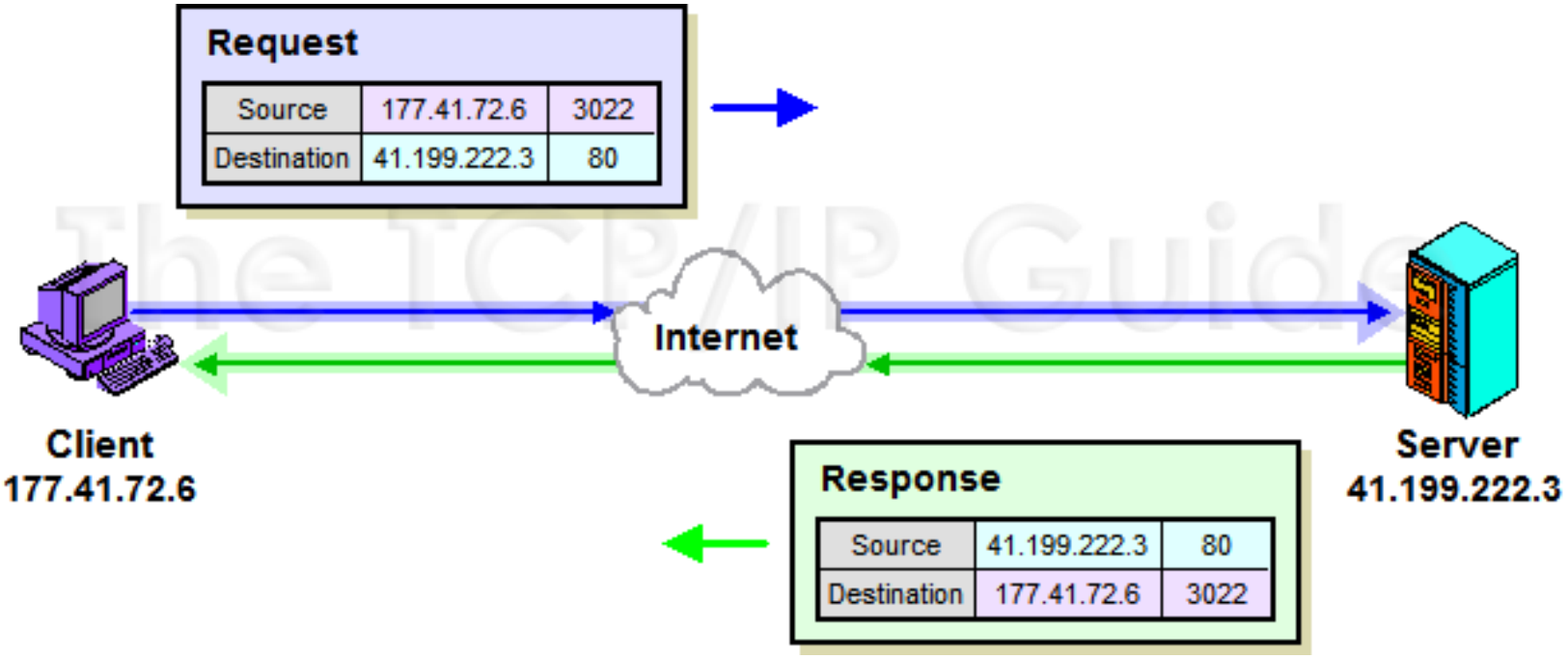
- These are known as ***Ephemeral ports***, as they only exist for the duration of the connect and get re-cycled.

# Example

- Suppose your computer has an IP address of 177.41.72.6 and the web browser wants to send an HTTP request to a website at address 41.199.222.3
- Your web browser will get a source port from the local TCP/IP process, say 3022.
- The destination port (HTTP) is 80
- The web browser will send a request to IP 41.199.222.3:port 80, with source address 177.41.72.6:port 3022
- The web server will generate a reply and send it back to IP 177.41.72.6:port 3022

# Example

- In this way the two processes have established a **connection** and can communicate with each other using a pair of <IP:port> addresses



# Socker Pair

- A pair of <IP:port> addresses like this is referred to as a ***socket*** *or* ***socket-pair***.

  – This also neatly solves the problem of having several different processes on the one client machine connecting the same process on a server machine. Each different client process will have a different client port number.

| | |
|---|---|
| Client1 : 3021 → Server : 80 | Server : 80 → Client1 : 3021 |
| Client2 : 3022 → Server : 80 | Server : 80 → Client2 : 3022 |
| Client3 : 3023 → Server : 80 | Server : 80 → Client3 : 3023 |

- The TCP/IP application program interface (API) that handles this is called ***socket***. In windows it is called windows socket or ***WinSock***

# Socker Pair

- A socket pair may also be specified as

  <client name:port> <host name : post>

- This typically happens when you type a web address into a browser, eg rmit.edu.au

- In this case the web browser on the client must first contact a DNS server to resolve rmit.edu.au into an IP address.

# TCP message acknowledgement

- Unlike IP protocols where delivery is not guaranteed, TCP provides confirmation of delivery and also guarantees that messages will be delivered to the application in the same order that they were sent.

- To see how this happens in TCP, lets first look at some simple protocols

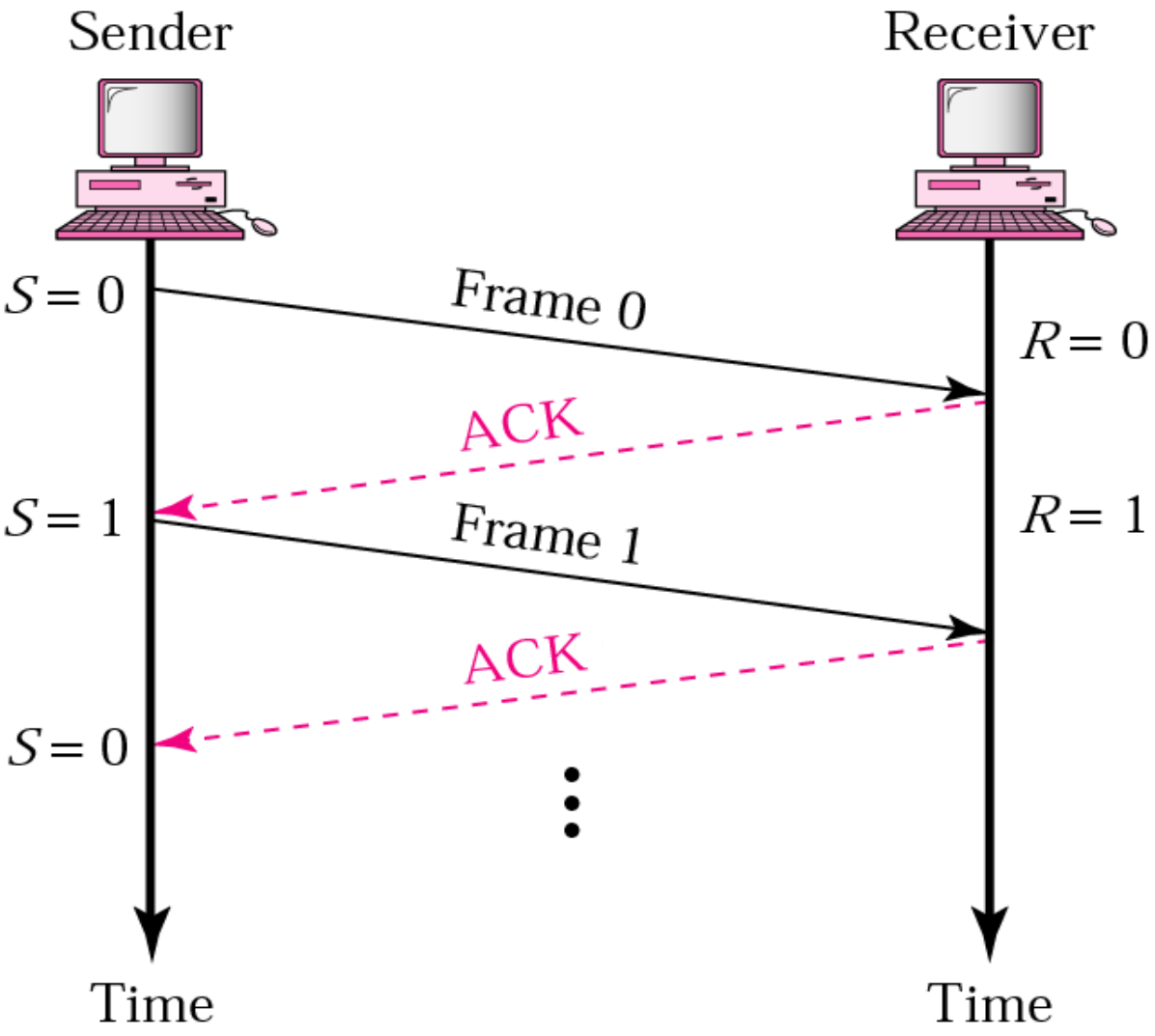
# Stop and wait protocol

1. Source transmits frame
2. Destination receives frame and replies with ACK
3. Source waits for ACK before sending next frame
4. If there is no acknowledgement after a certain time, the sender retransmits the frame

Works well for a few large frames

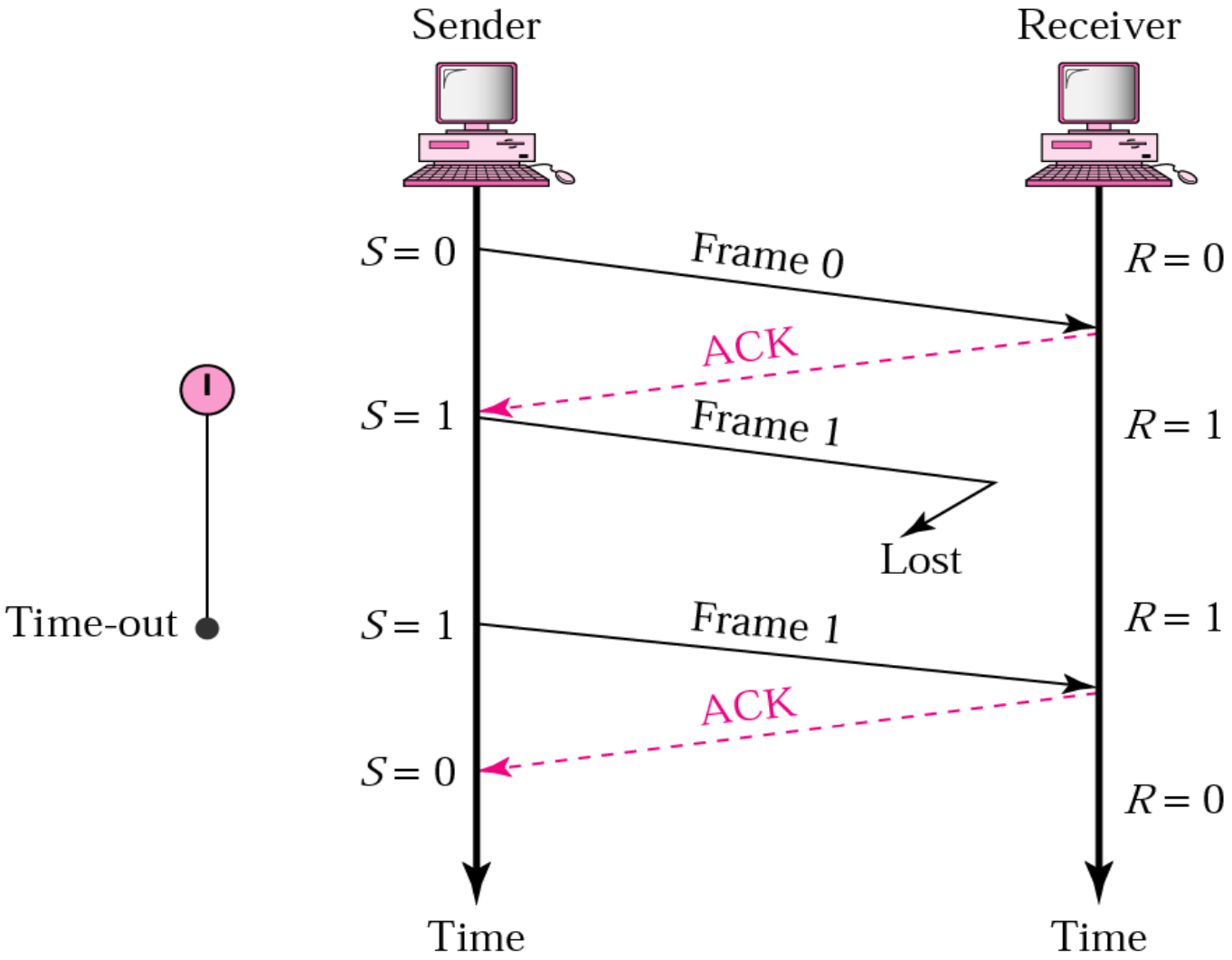
# Stop and wait

**Normal operation**



# Stop and wait

**Retransmission of lost data frame**



# Limitations – Stop and Wait

Inefficient as sender can’t send another frame until either ACK received or timeout – so channel is often empty

To improve efficiency, multiple frames should be in transition while waiting for acknowledgement (solution: sliding window protocols)

# Sliding window protocol

- The basic idea is that there can be some number ***n*** of frames waiting to be acknowledged, rather than just one.
- Each packet now needs to have sequence number
- Throughput can be maximized if the number n is large enough so that the acknowledgement for the first frame arrives back at the sender as the nth frame is being sent.
- In this way, data can be sent continuously provided no errors occur.

- In the basic form, an acknowledgement is sent for every packet received in the correct order

- If a packet is lost or has an error, no acknowledgement will be sent for that packet, or any subsequent packet, even if the subsequent packets are ok

- See https://www.youtube.com/watch?v=9BuaeEjIeQI&list=PLVJDTaS7rj9MsS3Rb1reGi4AOTh3gpoP8
- For and example of this

- Note that in the basic implementation, all unacknowledged packets are re-transmitted, even though only one packet was in error

- You can see in this example that the “sliding window” is of size 5. This means that the sender can have at most 5 unacknowledged packets at any one time. As each packet is acknowledged (in sequence) the window can move along and more packets can be transmitted.

- If a packet is not acknowledged with a certain time, a timeout occurs and that packet, and all others in the current window are re-transmitted.

# Sequence and Acknowledgement number

- The sequence number is used to guarantee that segments are delivered in sequence and to facilitate the re-transmission of any lost or erroneous segments
- In previous examples (stop and wait, and sliding window) we have assumed a single sender and single receiver, with the receiver acknowledging receipt of the senders data
- In practice, each party is both a sender and receiver, so as well as each segment having a sequence number, it will contain an acknowledgement number of the last segment it received. This is why the segment header has both a sequence and acknowledgement number.

# Improvements

- One obvious improvement is to add a selective repeat message, sometimes called a “negative acknowledgement” or NACK

- Eg if packet 3 is not received after a specified time, but packets 4 and 5 arrive, the receiver can send “NACK 3” meaning “please re-send packet 3”

# HTTP Client Server Interaction: Detailed Example

- To illustrate how a client, such as Firefox, retrieves a document usingHTTP/1.1, consider the following fragment /index.html:

```
<HTML><HEAD>
  <TITLE>The title</TITLE>
</HEAD><BODY>
  <h1>3 Images</h1>
  <img src="1.gif">
  <img src="2.gif">
  <img src="3.gif">
</BODY></HTML>
```

- Images included in a web page in this way are known as inline images.

Q: How many things are downloaded?

# Requesting the Document: Signalling

- Not surprisingly, the first request from the browser will be
  **`GET /index.html`**
  to retrieve the document.

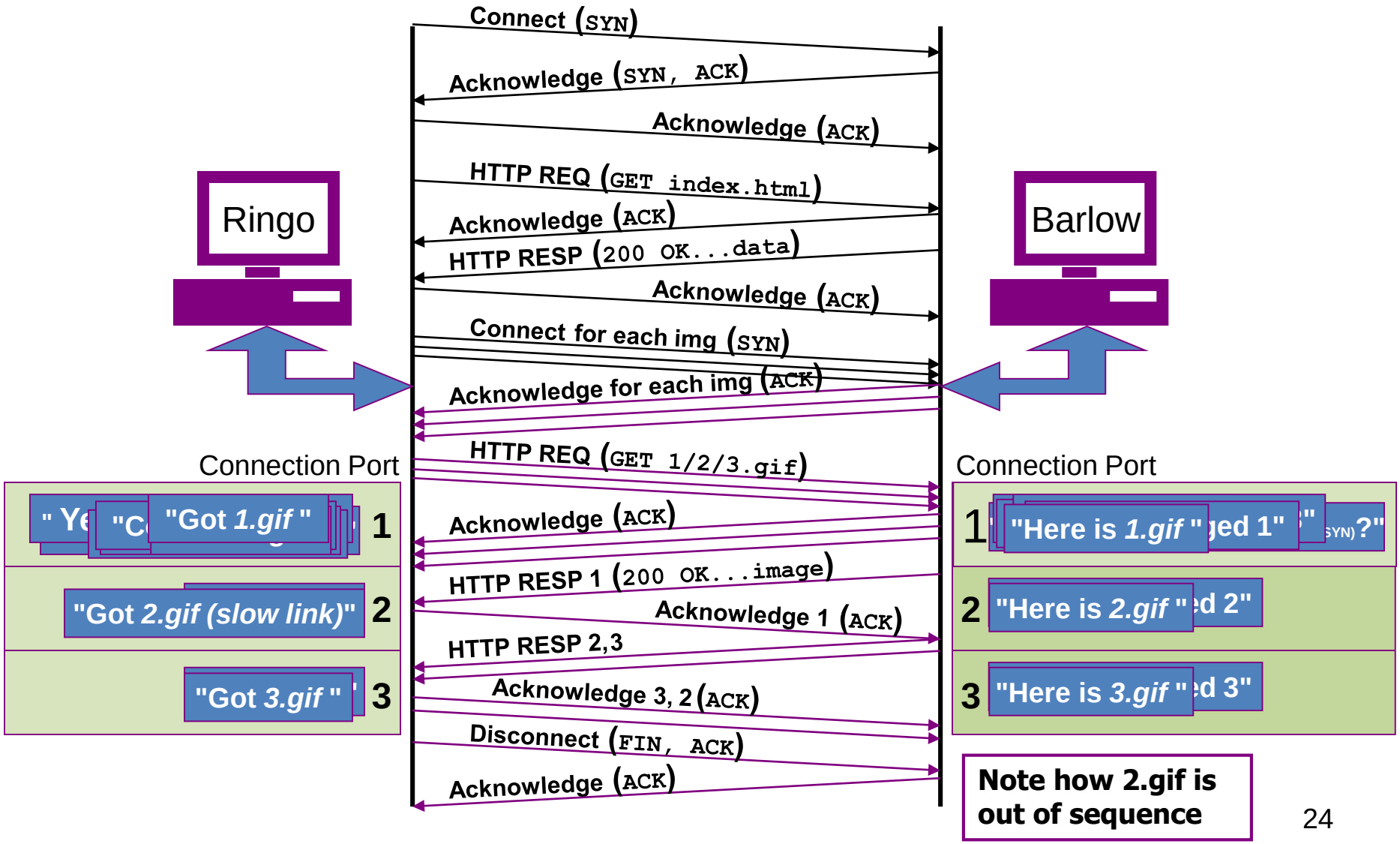
- This retrieves the body of the document, including the three image tags but not the images themselves.

- After this the browser will issue three simultaneous requests for the three image files 1.gif, 2.gif, and 3.gif.

- Firefox can be configured for the maximum number of simultaneous connections to each web server, which is 8 by default. For fast internet connections this should be increased.

- These four connections - one for the document and three for the images - each use different client ports, say, ports 1,024 to 1,027.

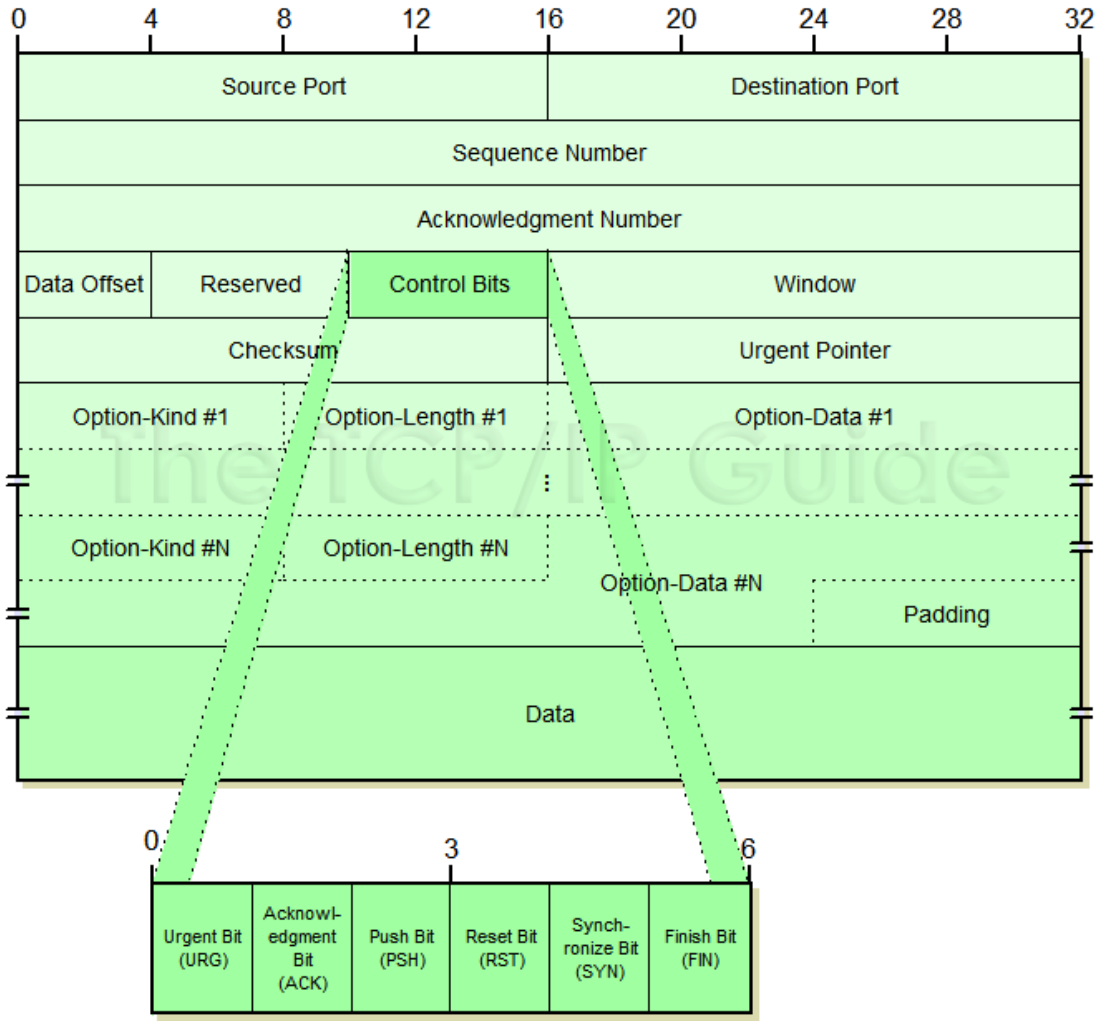
# Requesting the Document: Signalling (Full example)



# TCP Format

- Each TCP message is called a segment
- It is part of a *connection* that has been established between two processes on internet hosts, normally a *client* and a *server*.

- Although there is a lot of information and options available in a TCP segment header, the ones that are most important are:
- Source and Destination port
- Sequence and Acknowledgement number
- Checksum


0
4
8
12
16
20
24
28
32
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
Control Bits
Window
Checksum
Urgent Pointer
Option-Kind #1
Option-Length #1
Option-Data #1
Option-Kind #N
Option-Length #N
Option-Data #N
Padding
Data
0
3
6
Urgent Bit (URG)
Acknowl-edgment Bit (ACK)
Push Bit (PSH)
Reset Bit (RST)
Synch-ronize Bit (SYN)
Finish Bit (FIN)

# UDP

- UDP is connectionless (no opening of connections or sequencing of messages) and no guarantee of delivery

- There are two types of situations where the UDP protocol has advantages over TCP

1. When performance is more important then completeness
   - eg streaming a video clip
2. For very short data exchanges
   - When only a single request and reply is require
   - Loss of request or reply must be handled by the application

- See over for some common applications that use UDP

# Common UDP ports

| Port # | Keyword | Protocol | Comments |
|---|---|---|---|
| **53** | domain | Domain Name Server (DNS) | Uses a simple request/reply messaging system for most exchanges (but also uses TCP for longer ones). |
| **67 and 68** | bootps / bootpc | Bootstrap Protocol (BOOTP) and Dynamic Host Configuration Protocol (DHCP) | Host configuration protocols that consist of short request and reply exchanges. |
| **69** | tftp | Trivial File Transfer Protocol (TFTP) | TFTP is a great example of a protocol that was specifically designed for UDP, especially when it is compared to regular FTP. The latter protocol uses TCP to establish a session between two devices, and then makes use of its own large command set and TCP's features to ensure reliable transfer of possibly very large files. In contrast, TFTP is designed for the quick and easy transfer of small files. It includes simple versions of some of TCP's features, such as acknowledgments, to avoid file corruption. |
| **161 and 162** | snmp | Simple Network Management Protocol | An administrative protocol that uses relatively short messages. |
| **520 and 521** | router / ripng | Routing Information Protocol (RIP-1, RIP-2, RIPng) | Unlike more complex routing protocols like BGP, RIP uses a simple request/reply messaging system, doesn't require connections, and does require multicasts/broadcasts. This makes it a natural choice for UDP. If a routing update is sent due to a request and is lost, it can be replaced by sending a new request. Routine (unsolicited) updates that are lost are replaced in the next cycle. |
| **2049** | nfs | Network File System | NFS is an interesting case. Since it is a file sharing protocol, one would think that it would use TCP instead of UDP, but it was originally designed to use UDP for performance reasons. There were many people who felt this was not the best design decision, and later versions moved to the use of TCP. The latest version of NFS uses only TCP. |

# UDP Message Format

The UDP message format is simple, and 8 bytes long

- Source and Destination ports
- Length – 2 bytes
- Checksum – 2 bytes

- Note that once passed to the IP layer, the IP layer information is added, which includes the source and destination IP addresses.