```yaml
id: 1705788134-LOLW
aliases:
  - CN-Unit2
tags:
  - notes
  - computer_networks
author: vortex
```

DNS

- Hostname : mnemonics easy for human understanding.
- Since routers can't process variable length alphanumeric hostnames, we standardize it using a 32 bit `IP Address`
- Its divided by periods into 4 bytes each representing 0-255.

> To Convert human readable mnemonics into fixed length machine understandable byte code, we use DNS servers.

- DNS - Application layer protocol
- Apart from IP-hostname translations, DNS also provides :
  - Host Aliasing
  - Mail server aliasing
  - Load distribution
- DNS uses port 53
- Reasons why centralizing DNS servers is a bad idea:
  - Single point of failure
  - Maintenance issues and inablility to scalability
  - overloading
  - Distant servers: meaning requests from geographically farther areas takes longer
- Hence DNS is usually a distributed heirarchical database

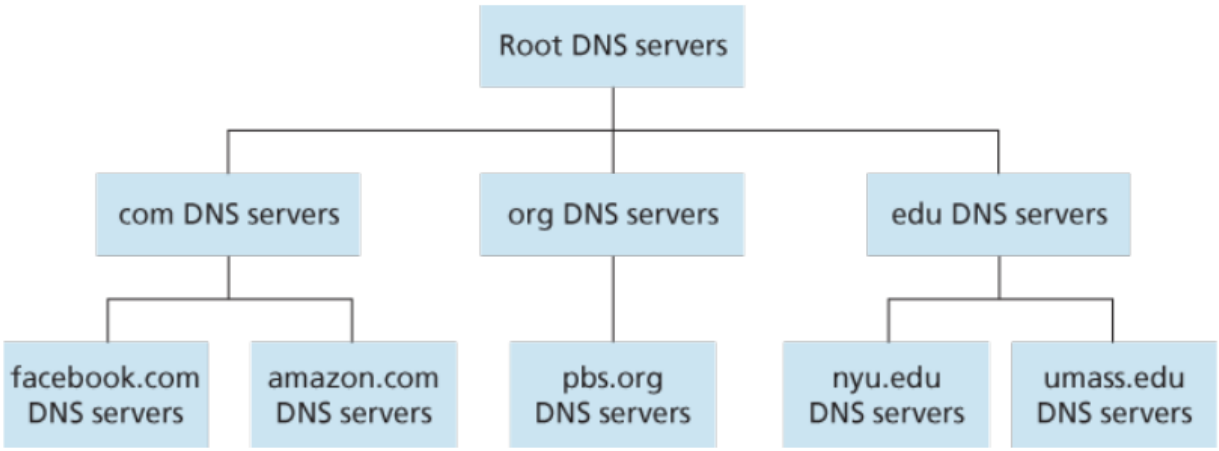


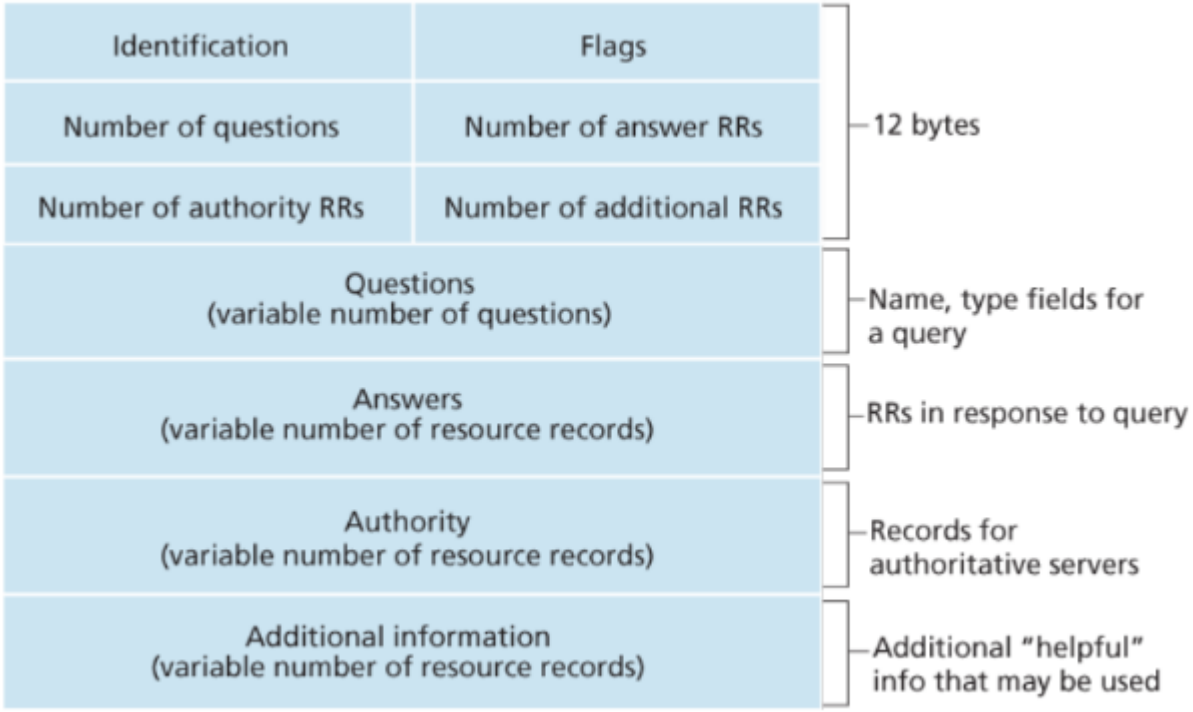
Figure 2.17 Portion of the hierarchy of DNS servers

- Root Name servers: over 400 over the world managed by 13 organizations
- Top level domain servers: `.com`, `.org`, `.edu` , `.gov` and country level domains like `.in`, `.au`
- Authoritative domain servers: maintained by the specific organization or a service provider.
- Local DNS server: Not part of the heirarchy. Usually hosted by the ISP.
  - When host connects to a network the ISP provides the host with the ISP of the local DNS servers closeby.
- DNS Queries can be of 2 types :
  - Iterative : each server forwards the host to lookup successive servers on the next DNS server.
  - Recursive: hands off responsibility to find the mapping on behalf of the host.
- DNS Caching :
  - DNS queries are cached on the DNS servers for a small amount of time (Time To Live(TTL)) to reduce the latency/lag.
  - Due to this on subsequent requests to the same domain, the server won't need to request to the root level DNS servers.

DNS Response Records(RR)

- DNS servers store data as Response Records
- 4 tuple value:
  - `(Name, Value, type, TTL)`
    - Type: A
    - Type: CNAME
    - Type: NS
    - Type: MX

DNS Message Format



- The `question` section contains information about the query that is being made. This section includes:
  - a name field that contains the name that is being queried, and
  - a type field that indicates thetype of question being asked about the name
    - for example, a host address associated with a name (Type A) or the mail server for a name (Type MX).
- In a `reply` from a DNS server, the answer section contains the resource records for the name that was originally queried.
  - A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses.
- The `authority` section contains records of other authoritative servers.
- The `additional` section contains other helpful records
- To insert your new domain into the DNS database, you need to register your domain with a DNS registrar.
- The registrar verifies the uniqueness of the domain name.

P2P Architecture

- Minimal dependence on servers.
- Clients or hosts intercommunicate with each other.

- Client server architectures depend on using the server as an intermediary and consume a huge amount of server bandwidth.
- In P2P the clients send the data to other clients as and when they receive the data.
- $D_{C-S} > max(NF/u_s, F/d_{min})$
- $D_{P2P} > max(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s+\sum u_i})$

BitTorrent

- P2P file sharing solution.
- `target` and `torrent`
- torrent: interconnected peers.
- tracker: infrastructure node
- When host first joins the torrent network it registers itself with the tracker.
- The tracker keeps a track of which clients are still in the torrent network and the clients periodically keep updating the tracker with with the status.
- The tracker gives the client a list of all the peers in the torrent.
- The client then connects to all of them simultaneously.
- At first the client has no chunks. Periodically it receives it from the other clients.
- As it receives chunks it also sends this to other peers.
- The client requests chunks it doesn't have from the clients connected to it.
- It does so in the `rarest first` manner as in it requests the chunks the it doesn't have that are rarest in the network so that it gets uniformly distributed in the network.
- The client also sends data to the top clients that send data to it at the highest rate.
- The other clients are said to be choked by this client.
- Every 10 seconds, the client randomly chooses another client and starts sharing chunks with that. This client is now said to be `optimistically unchoked`
- `churn` when the clients leave or join the server.

Sockets

- Types of network applications:
  - `Open`: based on a protocol and defined in the standard documentation.
  - `Proprietary`: made by independent developers and can't be used by others

> Each process is analogous to a house and the process's socket is analogous to a door.

Sockets in UDP

- In UDP there is no reliable communication and there is susceptibility to packet loss.

- Each packet of data that we send needs to be attached with a destination address
- This destination address will contain:
  - IP address of destination
  - Port number of destination
  - in addition (not done by UDP but by the OS), source IP and port

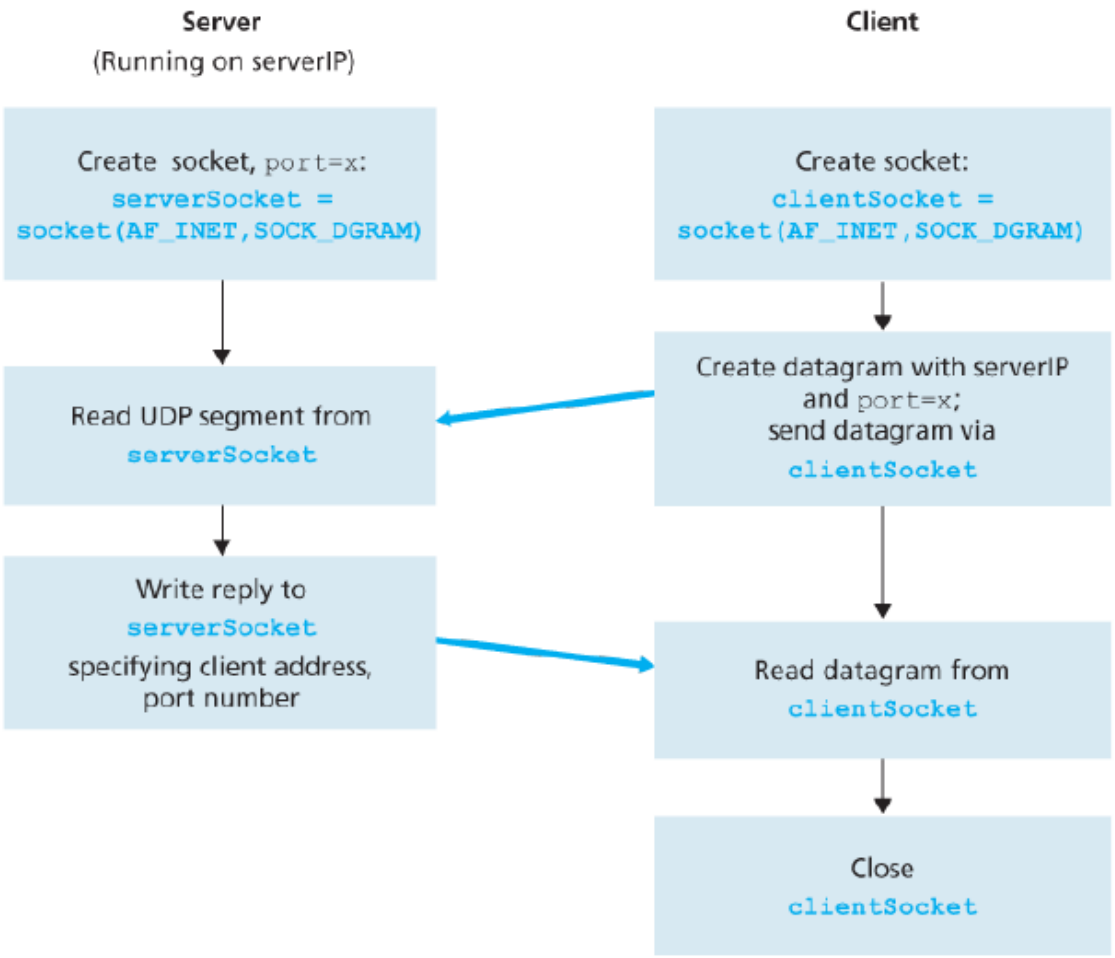


Figure 2.27 The client-server application using UDP

```
# Client Code
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

```
# Server Code
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((’’, serverPort))
print(”The server is ready to receive”)
while True:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.decode().upper()
        serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

- Client port is assigned by the OS and doesn't need to be specified.

Sockets in TCP

- Connection oriented, performs handshake
- When data needs to be send it is just dropped across the TCP connection. No requirement to attach a destination address as in UDP as the connection is already established.
- Server must be running well before the client and must have a dedicated socket to accept the connection request.
- After the client socket is created it initiates a 3-way handshake that takes place in the transport layer.
- During this process the server duplicates this socket and creates one just dedicated to this one client connection called the connection socket.
- The 2 processes are (from the perspective of the application layer) connected by a direct pipe.

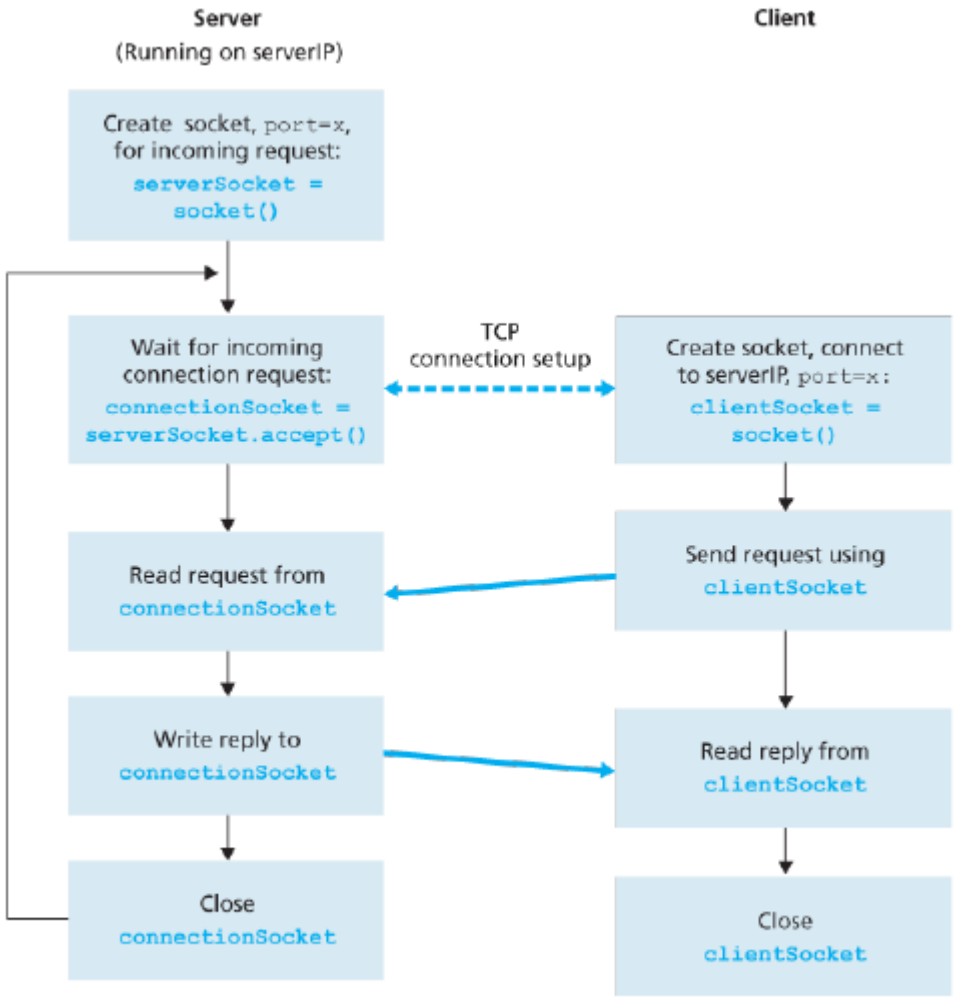


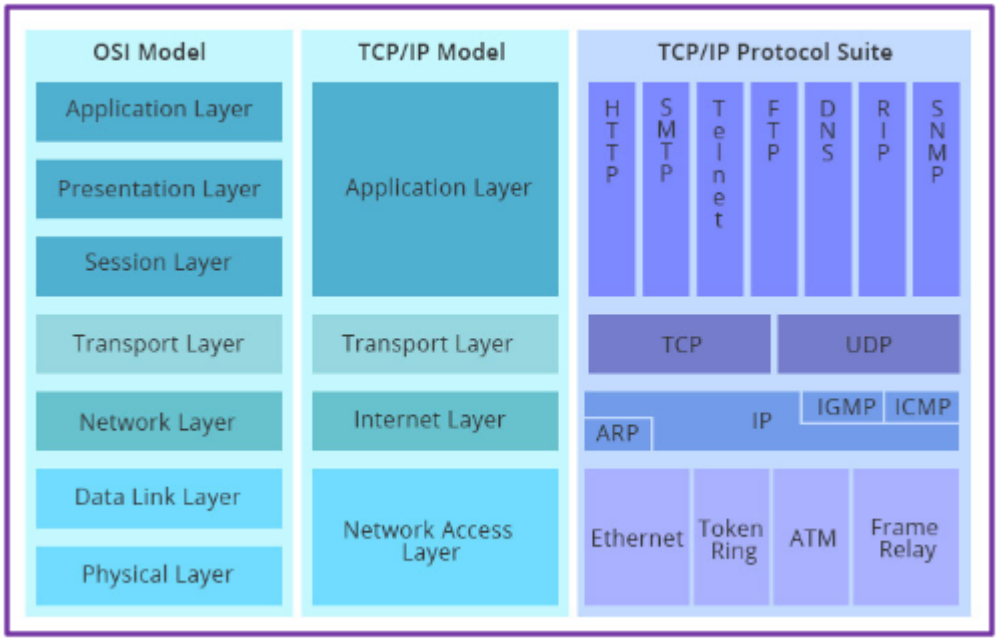
Figure 2.29 The client-server application using TCP

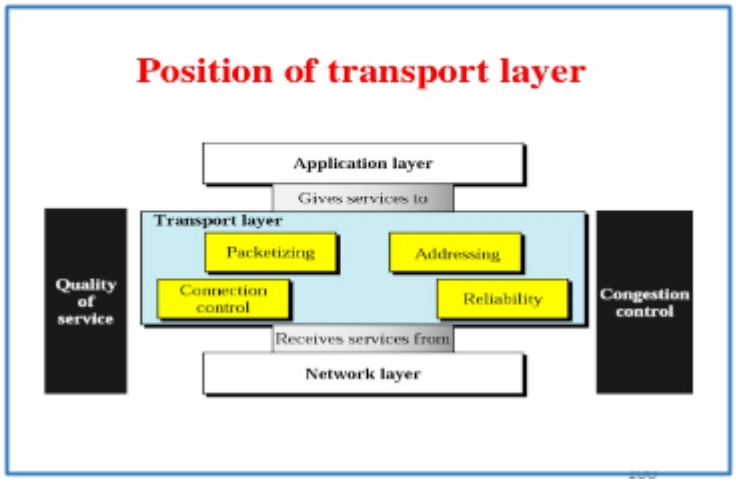
```
# Client Code
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server: ', modifiedSentence.decode())
clientSocket.close()
```

```
# Server Code
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
```

```
print('The server is ready to receive')
while True:
        connectionSocket, addr = serverSocket.accept()
        sentence = connectionSocket.recv(1024).decode()
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence.encode())
        connectionSocket.close()
```

Transport Layer





- Layer between the application and network layers.
- Logical communication : provides the illusion that the hosts are directly connected even though they may not be.
- On the sender's side the data stream from the application is split into `segment`

- Segment: data chunks + header
- Transport layer then passes on this segment to the network layer where it gets encapsulated and sent to the end system.
- The Network Layer devices don't examine the fields of the transport layer of the datagram.
- At the end system the segment from this datagram is send up to the transport layer where it is processed and the data becomes available to the application layer.
- Transport layer provides a logical communication between processes while the network layer provides a logical communication between hosts(works on finding the best path).
- The Internet Protocol is Unreliable by default.
- The transport layer protocols extend functionality of IP by connecting 2 processes on 2 hosts.
- This can be achieved either using TCP or UDP.
- UDP is similar to IP in that it is also unreliable.
    - process to process data delivery
    - error checking
- TCP on the other hand extends this:
    - Flow control
    - reliable connection
    - congestion control

Multiplexing and Demultiplexing

- At the sender the process of receiving the data from the sockets from the application, attaching a header and packaging into a segment and passes it into the network layer. This is called `Multiplexing`.
- At the receiver, the process of accepting the datagram from the network layer, detaching the headers and passing the data to the sockets of the process is called `Demultiplexing`
- Multiplexing:
    - Required unique socket identifiers
    - special fields that dictate where to deliver the data

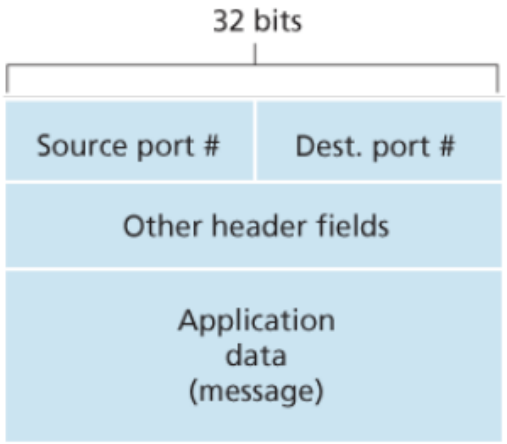


Figure 3.3 Source and destination port-number fields in a transport-layer segment

- ports : 0-65535(0-1023 are restricted/well known)
- Connectionless Mux and DeMux: Requires only destination port and IP(2 tuple). Uses UDP
  - Drawback: if 2 segments containing same destination address and port are send together, they go on the same socket.
- Connection Oriented : Requires source and destination port and IP(4 tuple). Uses TCP
  - Overcomes the drawback of UDP as for every 4-tuple a new socket is established.
- Webservers: listen on port 80 for connections from clients.
  - When a connection request is accepted a new socket is opened for every client.
  - Modern webservers open the new socket on a new thread(subprocess)
  - If connection is persistent, Client and server use the same socket throughout the lifetime of the connection.
  - If connection is non-persistent, the client and server open and close a new socket connection each time which can severely impact performance

Connectionless

- Called connectionless as there is no handshaking.
- Reasons why we may choose to use UDP over TCP
  - Finer control over how and when data is sent
  - No Handshake
  - No connection state
  - Lower buffer overhead
- DNS uses This mode to avoid the delay of handshake
- Multimedia is one sector where UDP cannot be used.

- If multiple users stream at high bitrates without congestion control, there would be unprecedented overflow and this would even adversely affect the TCP senders.
- Reliability can be built into the application do it can be Reliable UDP like QUIC.

UDP Segment Structure

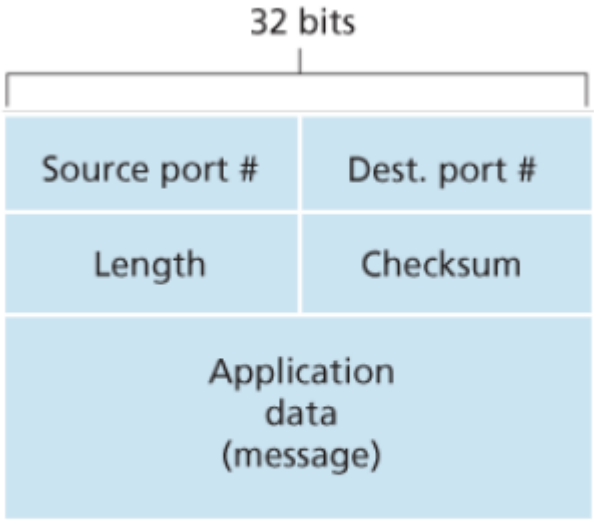


Figure 3.7 UDP segment structure

- Length = Data+header
- Checksum to verify introduction of any errors

UDP Checksum

- The 16 bit words are added and their 1's complement is taken at the source. Any overflow is wrapped around.
- This is placed in the Checksum field.
- At the destination this checksum is added with the other words. If no errors then we get only 1s. if there is a 0, then error has been introduced.