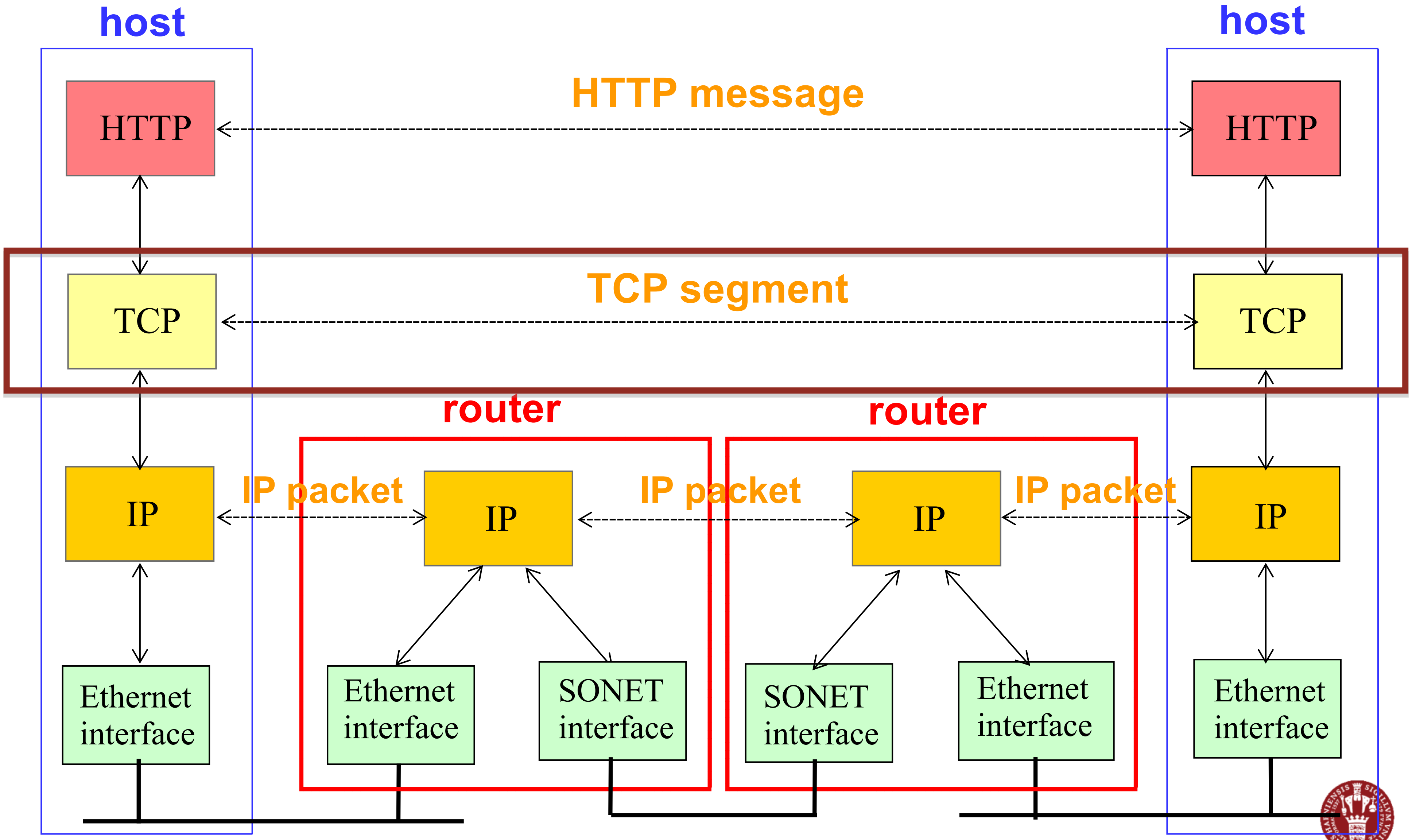
Transport Layer: UDP

Michael Kirkedal Thomsen

Based on slides compiled by Marcos Vaz Salles, adaptions by Vivek Shah

Recap: Internet Layering Model



Source: Freedman

Transport Layer

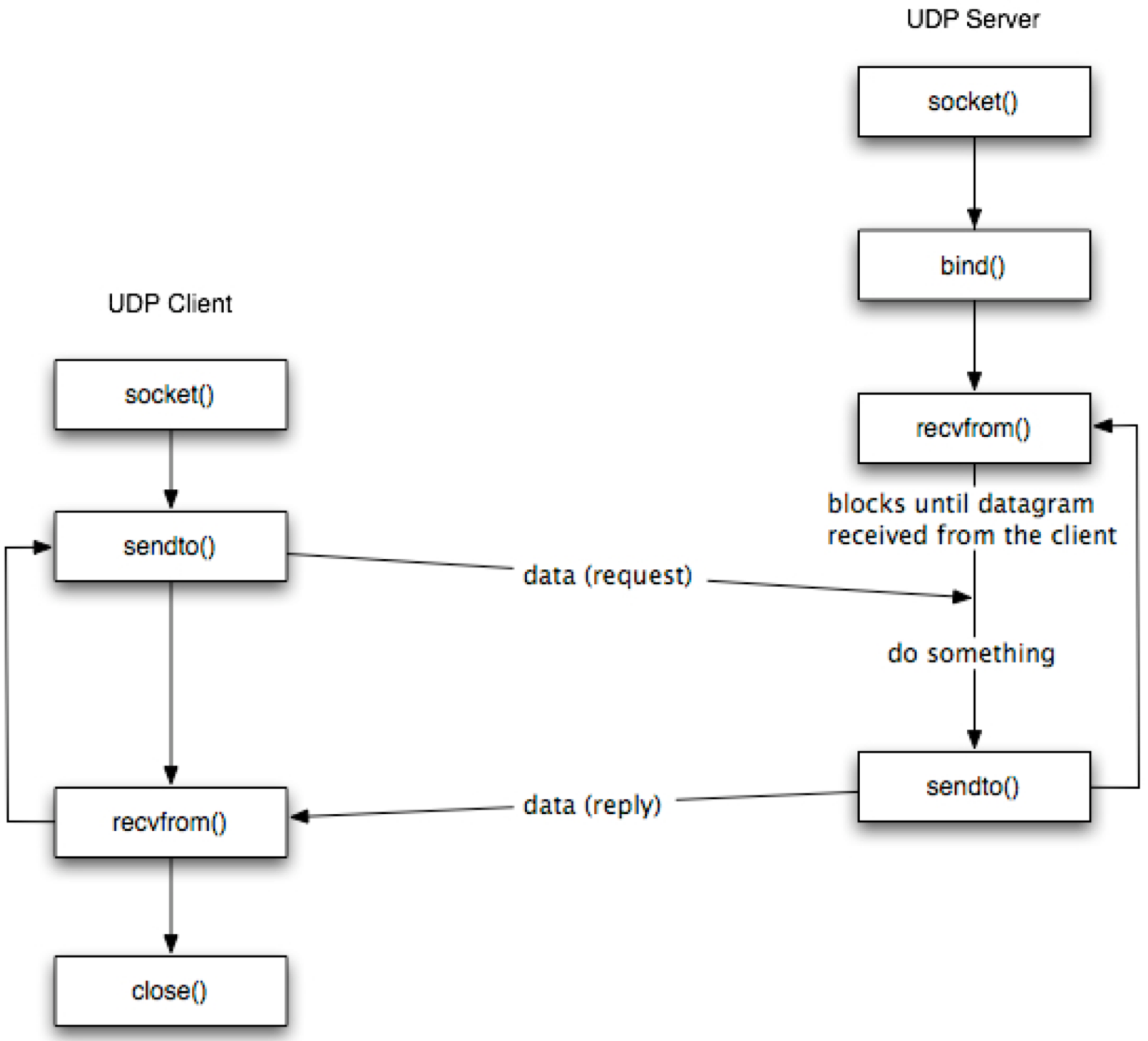
- Logical Communication between processes
  - Sender divides messages into segments.
  - Receiver re-assembles messages into segments.

- Principles underlying transport-layer services
  - (De)multiplexing
  - Detecting corruption
  - Optional: Reliable delivery, Flow control, Congestion control

- Transport-layer protocols in the Internet
  - User Datagram Protocol (UDP)
    - Simple (unreliable) message delivery
  - Transmission Control Protocol (TCP)
    - Reliable bidirectional stream of bytes

Source: Freedman

Socket Programming Using UDP



Source: Campbell

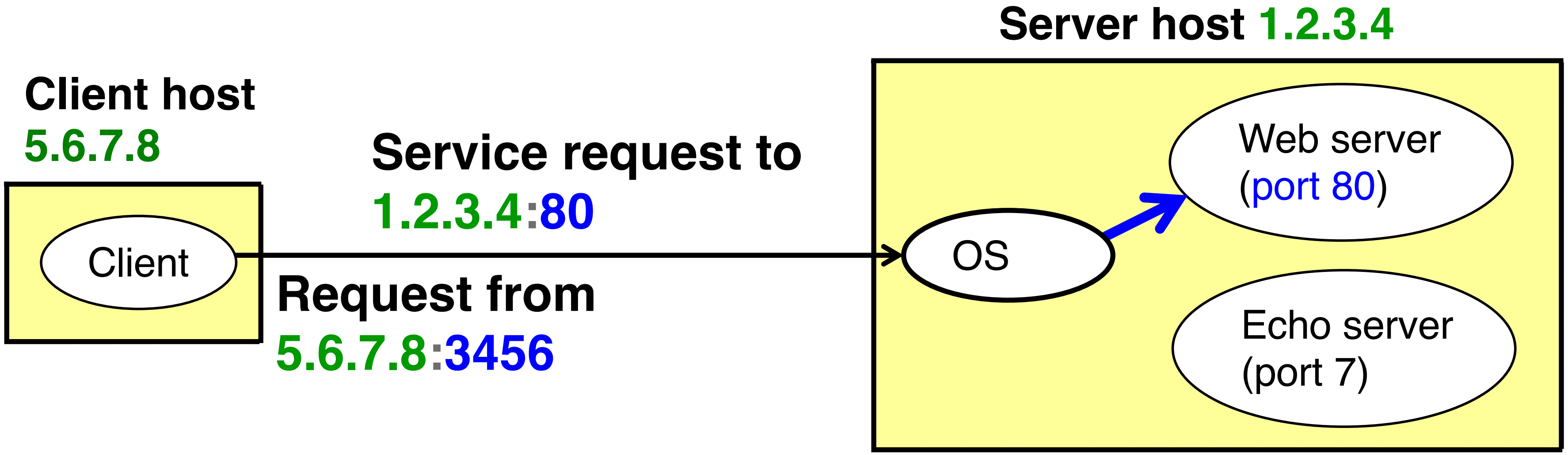
Socket Programming Using UDP

```
ssize_t recvfrom(int sockfd, void* buff,
    size_t nbytes, int flags, struct sockaddr* from,
    socklen_t *addrlen);
```

```
ssize_t sendto(int sockfd, const void *buff,
    size_t nbytes, int flags,
    const struct sockaddr *to, socklen_t addrlen);
```

Two Basic Transport Features

- **Demultiplexing:** port numbers



Server host 1.2.3.4

Client host 5.6.7.8

Client

Service request to 1.2.3.4:80

Request from 5.6.7.8:3456

OS

Web server (port 80)

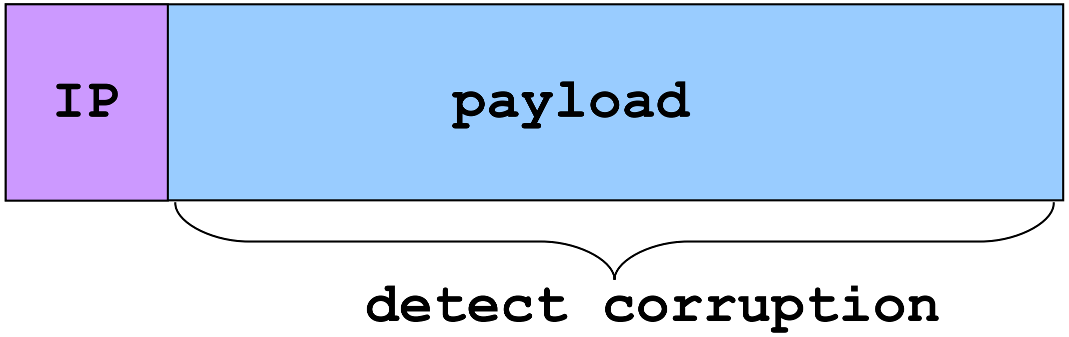
Echo server (port 7)

Demux table (“5 tuple”)	Socket
<*, *, 1.2.3.4, 80, TCP>	5
<5.6.7.8, 3456, 1.2.3.4, 80, TCP>	6

Source: Freedman

Two Basic Transport Features

- **Error detection:** checksums

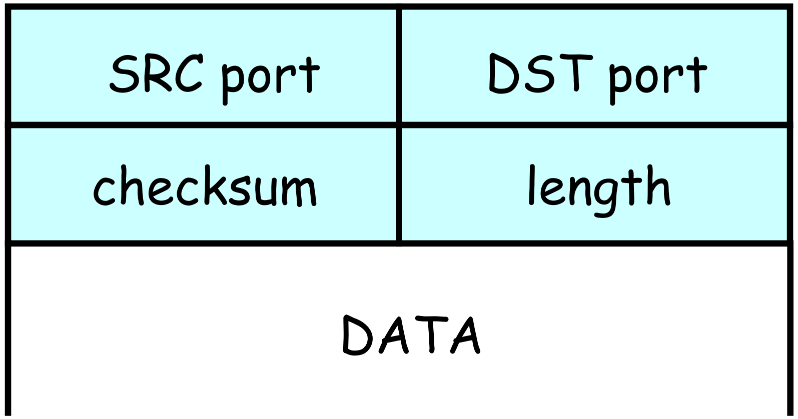


- Optional for IPv4
  - 16-bit one's complement of the sum of a pseudo header of information from the IP header, the UDP header, and the data

Source: Freedman

User Datagram Protocol (UDP)

- Datagram messaging service
  - Demultiplexing of messages: port numbers
  - Detecting corrupted messages: checksum
- Lightweight communication between processes
  - Send messages to and receive them from a socket
- Avoid overhead and delays of ordered, reliable delivery



SRC port	DST port
checksum	length
DATA	

Source: Freedman

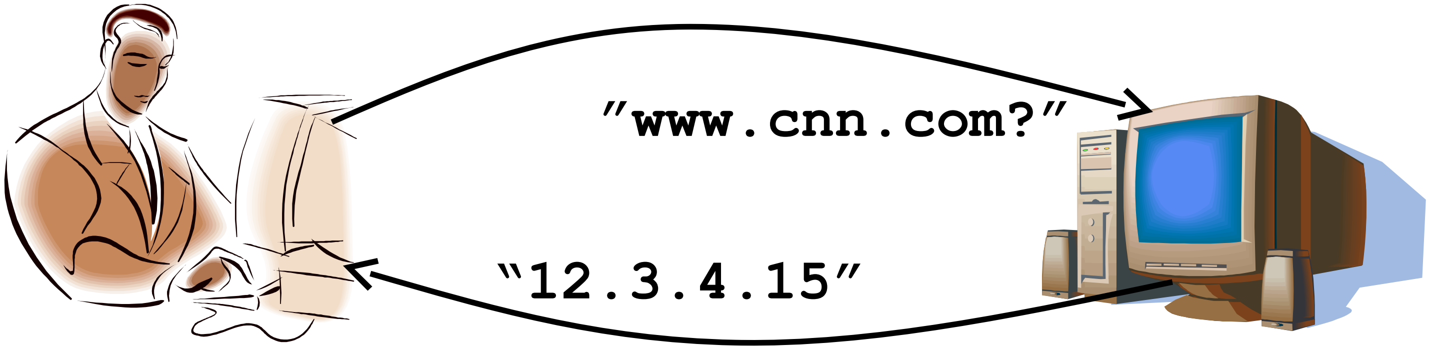
Why Would Anyone Use UDP?

- Fine control over what data is sent and when
  - As soon as app process writes into socket
  - … UDP will package data and send packet
- No delay for connection establishment
  - UDP blasts away without any formal preliminaries
  - … avoids introducing unnecessary delays
- No connection state (no buffers, sequence #’s, etc.)
  - Can scale to more active clients at once
- Small packet header overhead (header only 8B long)

Source: Freedman

Popular Applications That Use UDP

- Simple query protocols like DNS
  - Overhead of connection establishment is overkill
  - Easier to have the application retransmit if needed



- Multimedia streaming (VoIP, video conferencing, ...)
  - Retransmitting lost/corrupted packets is not worthwhile
  - By time packet is retransmitted, it's too late

Source: Freedman

Summary

- UDP
  - basic multiplexing, checksums

- QUIC (now named HTTP/3) gives (some) TCP guaranteed over UDP

What’s next ? Reliable Data Transfer & TCP