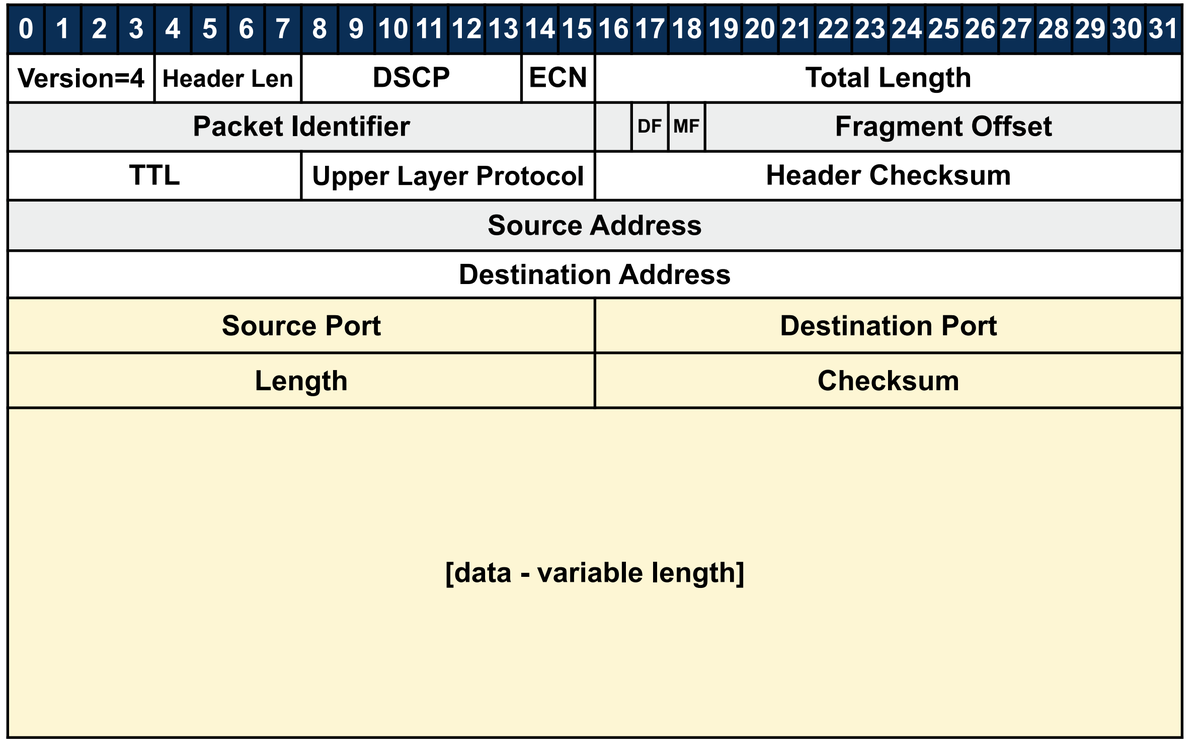
Unreliable Data Using UDP

- UDP service model
- Sending and receiving data
- Higher-layer protocols

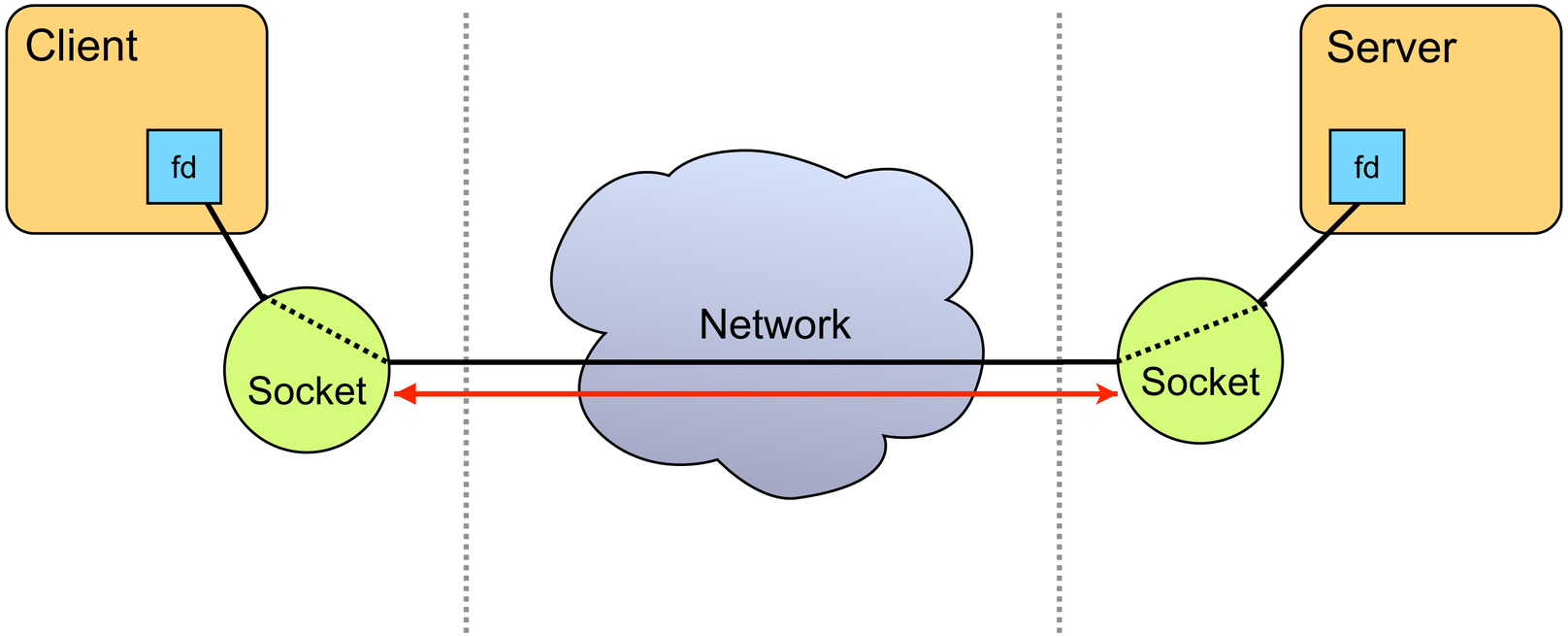
Packet Loss and UDP



- UDP → **unreliable**, connectionless, datagram service
  - Adds port numbers and a checksum to IP
  - Does not provide reliability or congestion control – best effort packet delivery
- A **substrate for building new transport protocols**
  - QUIC → Lecture 4
  - RTP → Lecture 7
  - DNS → Lecture 8

Well-known UDP port numbers: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Using UDP Datagrams




```
int fd = socket(...)

bind(fd, ..., ...)

sendto(fd, data, datalen, addr, addrlen)

recvfrom(fd, buffer, buflen, flags, addr, addrlen)

close(fd)
```

Sending UDP Datagrams

- The **`sendto()`** call sends a single datagram
- Each call to **`sendto()`** can send to a different address, even though they use the same socket.

```
int                 fd;
char                buffer[...];
int                 buflen = sizeof(buffer);
struct sockaddr_in  addr;
...
if (sendto(fd, buffer, buflen, (struct sockaddr *) addr, sizeof(addr)) < 0) {
    // Error...
}
```

- Alternatively, **`connect()`** to an address, then use **`send()`** to send the data
  - There is no connection made by the UDP layer, a **`connect()`** call only sets the destination address for future packets.

Receiving UDP Datagrams

- The **`recv()`** call may be used to read a single datagram, but doesn't provide the source address of the datagram.
- Most code uses **`recvfrom()`** instead – this fills in the source address of the received datagram:

```
int                fd;
char               buffer[...];
int                buflen = sizeof(buffer);
struct sockaddr    addr;
socklen_t          addr_len = sizeof(addr);
int                rlen;
...
rlen = recvfrom(fd, buffer, buflen, 0, &addr, &addrlen);
if (rlen < 0) {
    // Error...
}
```

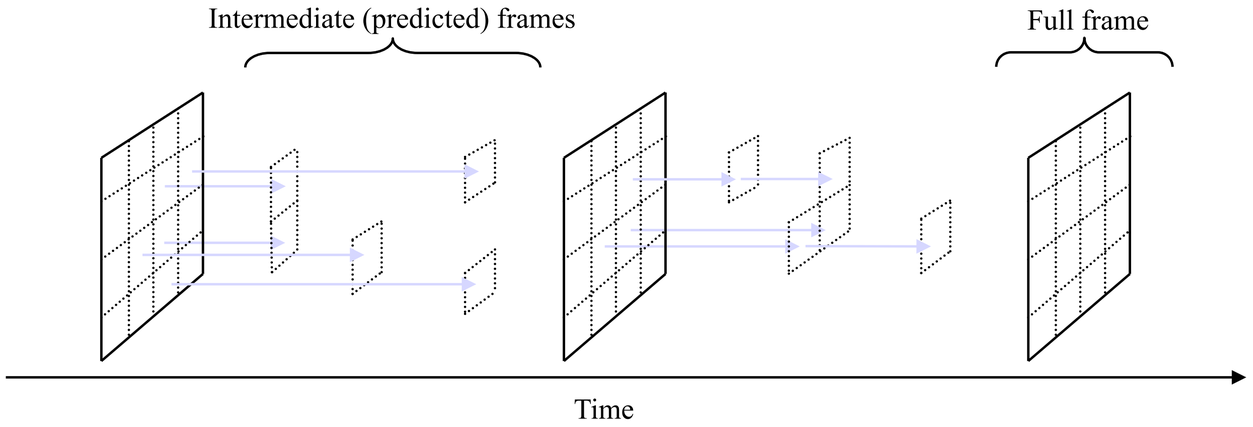
- Packets can be lost, delayed, or reordered in transit – UDP **does not** attempt to recover from this

UDP Framing and Reliability (1/2)

- Each **`sendto()`** calls generates exactly one UDP datagram
  - Each **`recvfrom()`** corresponds to a single **`sendto()`**
- But, UDP is unreliable:
  - Datagrams may be lost, delayed, reordered, or duplicated in transit
  - Data sent with **`sendto()`** might never arrive
    - Data sent with **`sendto()`** might arrive more than once
    - Data sent in consecutive calls to **`sendto()`** might arrive out-of-order
    - UDP **does not** attempt to correct this
- The **protocol built on UDP** is responsible for correcting the order, detecting duplicates, and repairing loss – if necessary
  - The protocol running within UDP generally includes a sequence number to support this
  - e.g., RTP and QUIC both include a packet sequence number inside UDP packets

UDP Framing and Reliability (2/2)

- UDP provides framing – data is delivered a packet at a time – but is unreliable
- Application must organise the data so it's useful if some datagrams lost
  - e.g. video conferencing with I- and P-frames



See also: D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols", ACM SIGCOMM Conference, Philadelphia, PA, USA, September 1990.http://dx.doi.org/10.1145/99517.99553

UDP Sequencing and Reliability

- UDP does not attempt to provide sequencing, reliability, or timing recovery
  - It provides a best-effort datagram delivery service, and identifies applications according to port numbers – substrate on which application protocols can be implemented
  - The syntax and semantics of the datagrams depends on the application layer protocol running within UDP
    - This might provide sequence numbers and acknowledgements
    - This might provide retransmission and/or forward error correction
    - This might provide timestamps to recover timing
    - This might provide payload identification, or other information

Guidelines for writing UDP applications

- IETF provides guidelines for writing UDP-based applications
  - RFC 8085 – https://tools.ietf.org/html/rfc8085
  - **Read this before trying to write UDP-based applications**

Unreliable Data Using UDP

- UDP service model
- Sending and receiving data
- Higher-layer protocols