CSE 561 Lecture 4

Neil Spring and David Wetherall

12 Apr 2002

Today

Stop and wait (10 min)

Sliding window (20 mins)

TCP today vs. Cerf/Kahn (10 mins)

Digital fountain (10 mins)

IP fragementation (20 mins)

Reviews

Very Good!

Especially with so much background to learn!

#Strengths $\geq$ #Weaknesses.

Readable paper $=$ $\frac{1}{2}$ strength.

It works $=$ $\frac{1}{2}$ strength.

IP $\rightarrow$ TCP

IP is

- Unreliable
- Datagram
- Unordered (delay)
- Duplication possible
- Corruption possible
- Connectionless

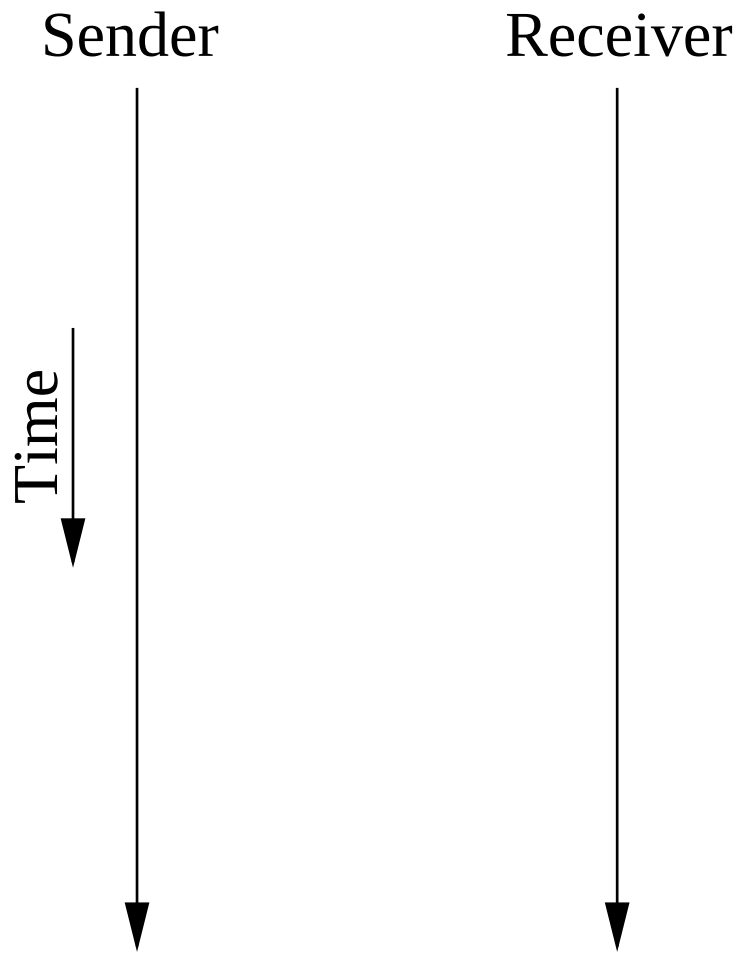
TCP is

- Reliable
- Bytestream
- Ordered
- Single delivery
- Checksummed
- Full duplex connections

TCP adds:

- Flow control
- Congestion control

Basic Reliability: Stop and Wait



(aka. Alternating-bit protocol)

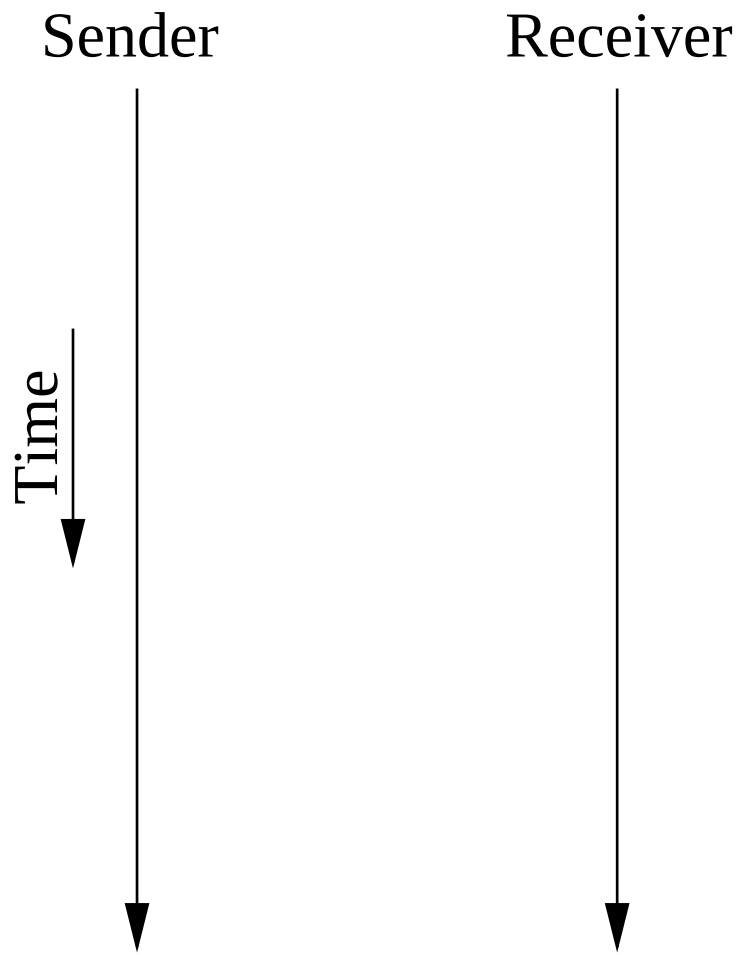
- Send a packet
- Wait for ack
- On timeout, retransmit.

Interesting cases:

- ACK loss
- Packet loss
- Spurious timeout

After sending an ack, what packets could a receiver get?

SAW Sequence Numbers



One bit sequence space:

send #1 after ACK of #0 received.

When does this go bad?

Prevent late delivery

TTL: upper bound on the lifetime of a packet.

Don’t reuse sequence numbers for duration of MSL

MSL = maximum segment lifetime.

Sequence space $\geq$ transmission rate $\times$ MSL

$$\log_2 \frac{100 \times 10^6 bit}{second} \times \frac{1 byte}{8 bits} \times 60s \leq 30$$

SAW Performance

lat - one way latency

thr - link throughput

sz - packet size

transmission time = $\frac{sz}{thr} + lat$

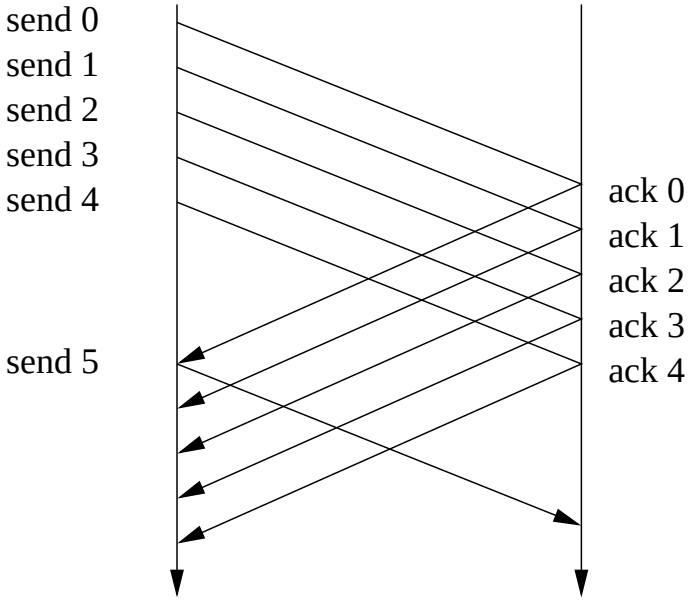
ack time = lat assuming $sz = 0$

throughput with no losses

= bytes / (transmission + ack time)

= $\frac{sz}{\frac{sz}{thr} + 2 \times lat}$

Better performance



Early Arpanet: 4 SAW streams.

Sliding window: on ack of k, send up to $k + w$.

Performance: up to $\frac{w}{2 \times lat}$

*yes, TCP uses bytes and I'm numbering packets.

Sliding window reliability

Cumulative ACKs:

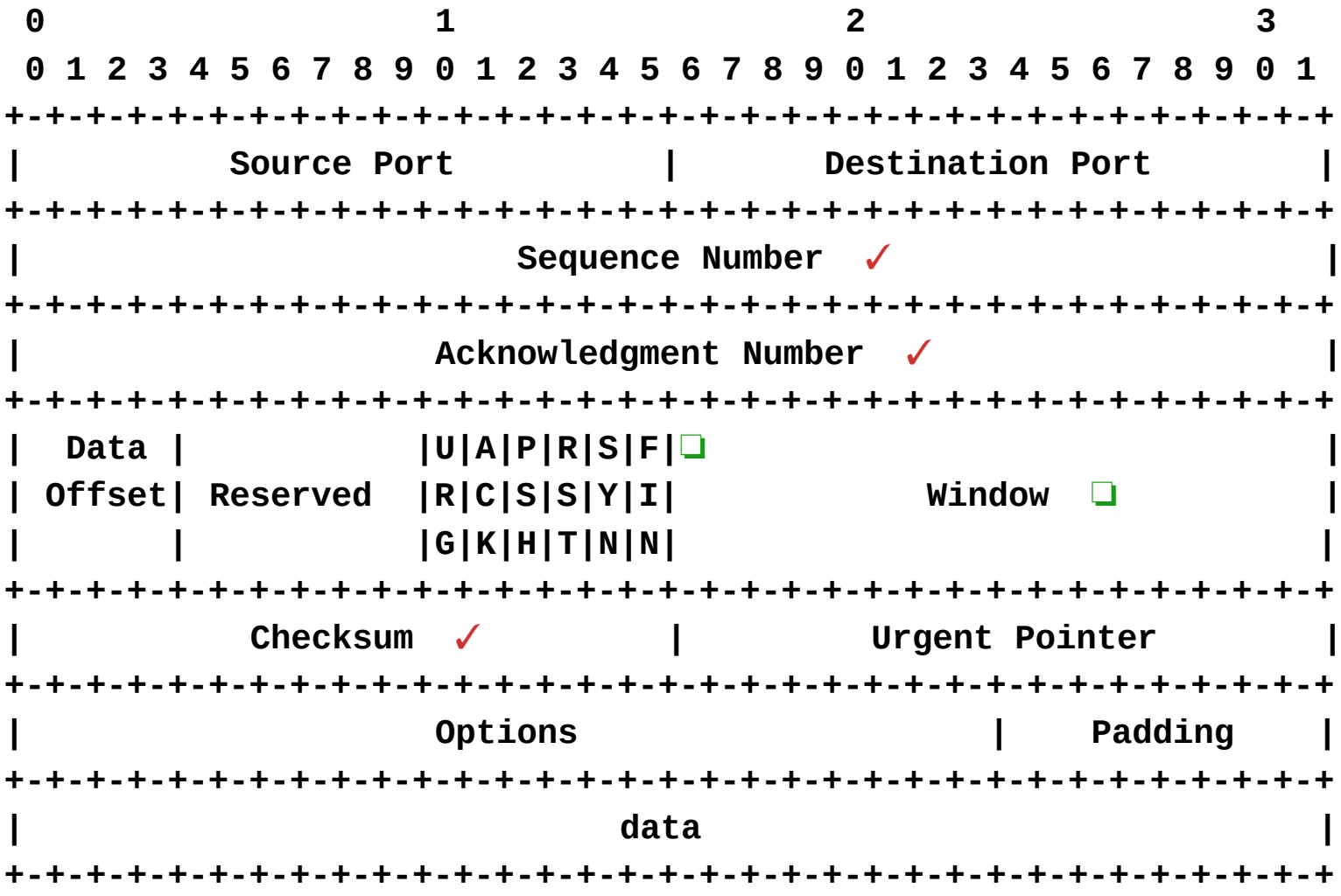
> "I got everything up to and including x."
>
> If you have x and $x + 2$, can't express $x + 2$.
>
> Standard TCP behavior.

Selective ACKs (SACK):

> "I got a and b and d and f."
>
> Optional TCP behavior.

ACKs are redundant (advantage over concurrent SAW)

TCP Header (an outline slide)



Sliding window (static)

Window w = 3.

Sender tracks “last (cumulative) ack received”:

1	2	3	4	5	6	7	8	9	10	11

Receiver tracks “next segment expected”:

1	2	3	4	5	6	7	8	9	10	11

Don’t forget to ack unexpected packets.

ACK pacing.

Sliding window with flow control

Receiving app may be slow (especially when TCP developed).

Kernel buffer space is finite (especially when TCP developed).

Receiving TCP may “close” the window if the app is stalled:

> “ACK up to and including packet 4; I have space for 2 more.”
>
> “ACK up to and including packet 5; I have space for 1 more.”
>
> “ACK up to and including packet 6; I have space for 0 more.

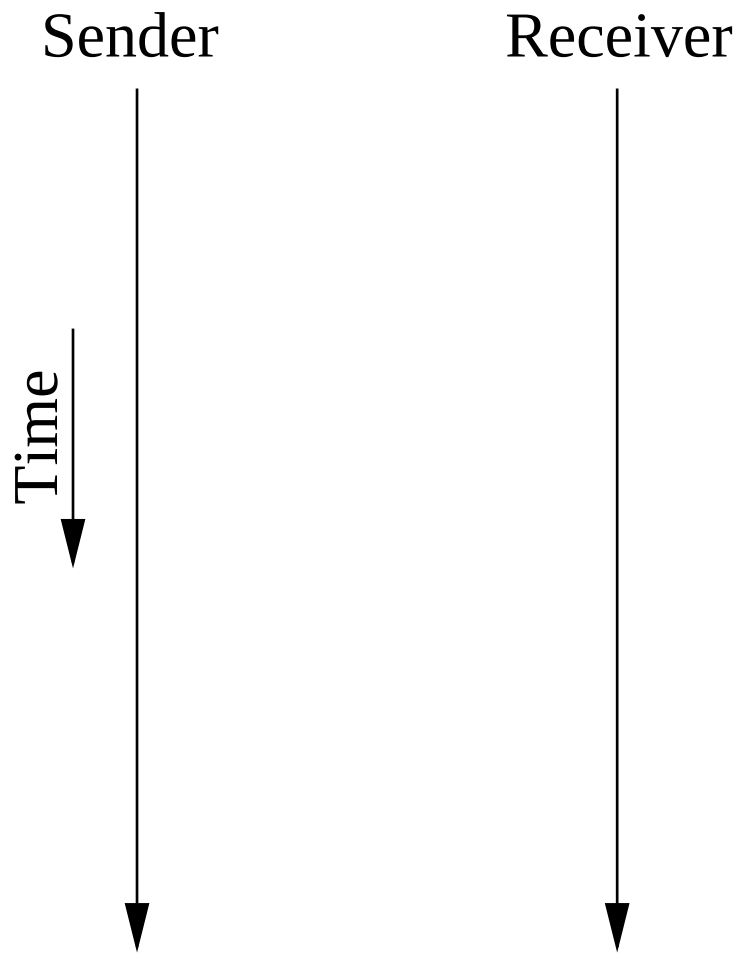
Are we stuck in Advertised Window Deadlock?

Also silly window syndrome avoidance, Nagel’s algorithm to stop tinygrams.

5.9b

$$\log_2 \frac{100 \times 10^6 bit}{second} \times \frac{1byte}{8bits} \times 0.1s = 20.25$$

TCP Connection Setup



Three way handshake.

Why the last ACK?

Cerf/Kahn missed this.

Sliding Window Summary

Keep w outstanding packets in the network.

Reliability through cumulative acks.

Ordering through sequence numbers.

Flow control by adjusting window size.

Reliability through FEC

Parity-based Error Correcting Codes (ECC):

- 2-dimensional parity
- Hamming codes

Tornado codes described in Digital Fountain paper.

Hamming codes (for fun)

a 7,4 hamming (error correcting) code

0110 is encoded as:

bit #	1	2	3	4	5	6	7
data	-	-	0	-	1	1	0
parity			-		-	-	-
index	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1

Interleaving

Tolerate burst errors List codewords, transmit column-wise.

1	1	0	0	1	1	0
0	1	1	1	1	0	0
1	0	1	0	1	0	1
1	0	0	1	1	0	0
0	1	0	0	1	0	1
0	1	0	1	0	1	0

Digital Fountain

Multicast Tornado encoded stream continuously.

Receivers decode what packets they receive.

Very little state, very little backward communication

When?

FEC

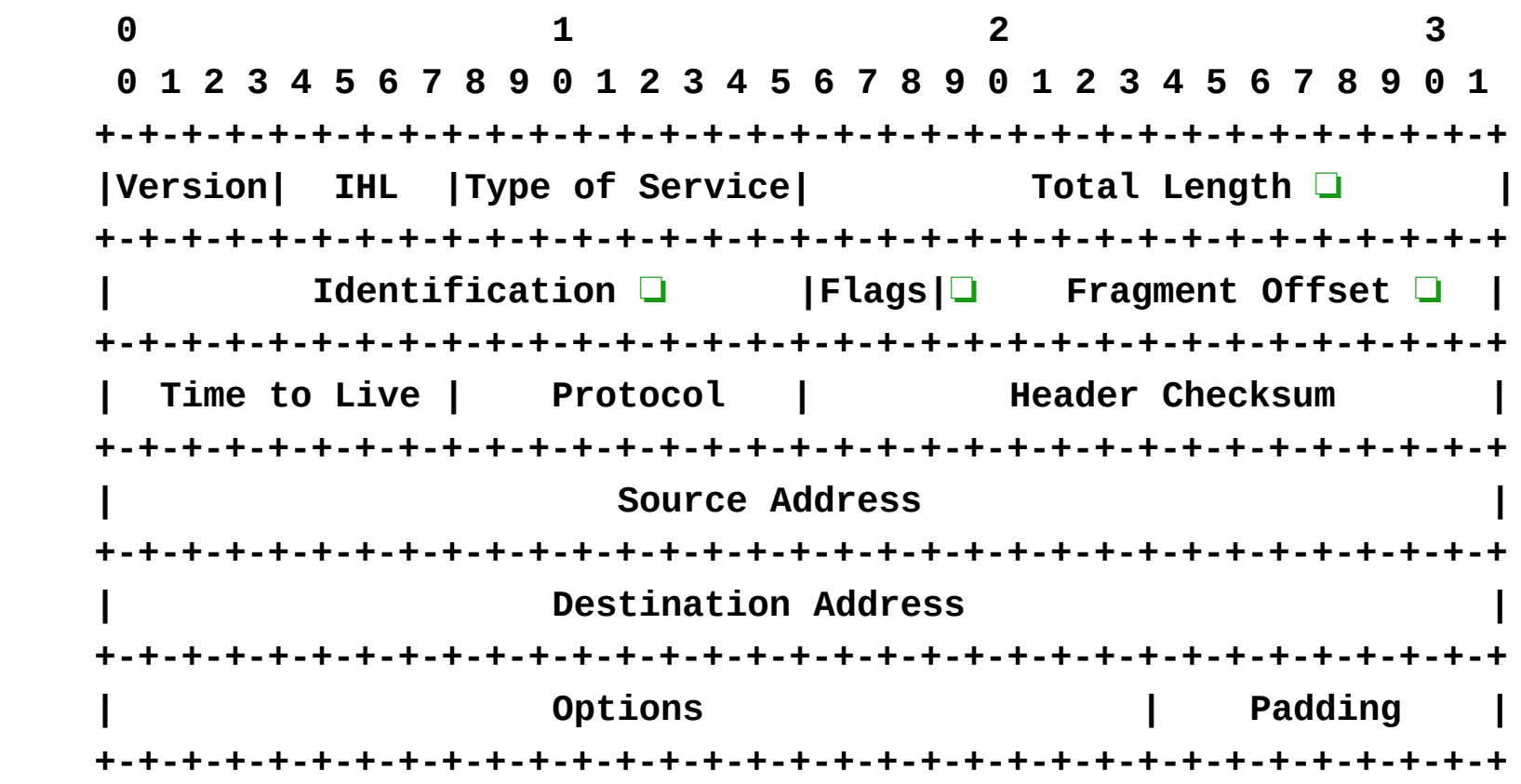
Digital Fountain

Digital Fountain

What were they worried about?

- Decoding efficiency
- Encoding / decoding time (server load)
- Receivers can “drink” at any time.

IP Fragmentation



Flag Bit 0: reserved, must be zero

Flag Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.

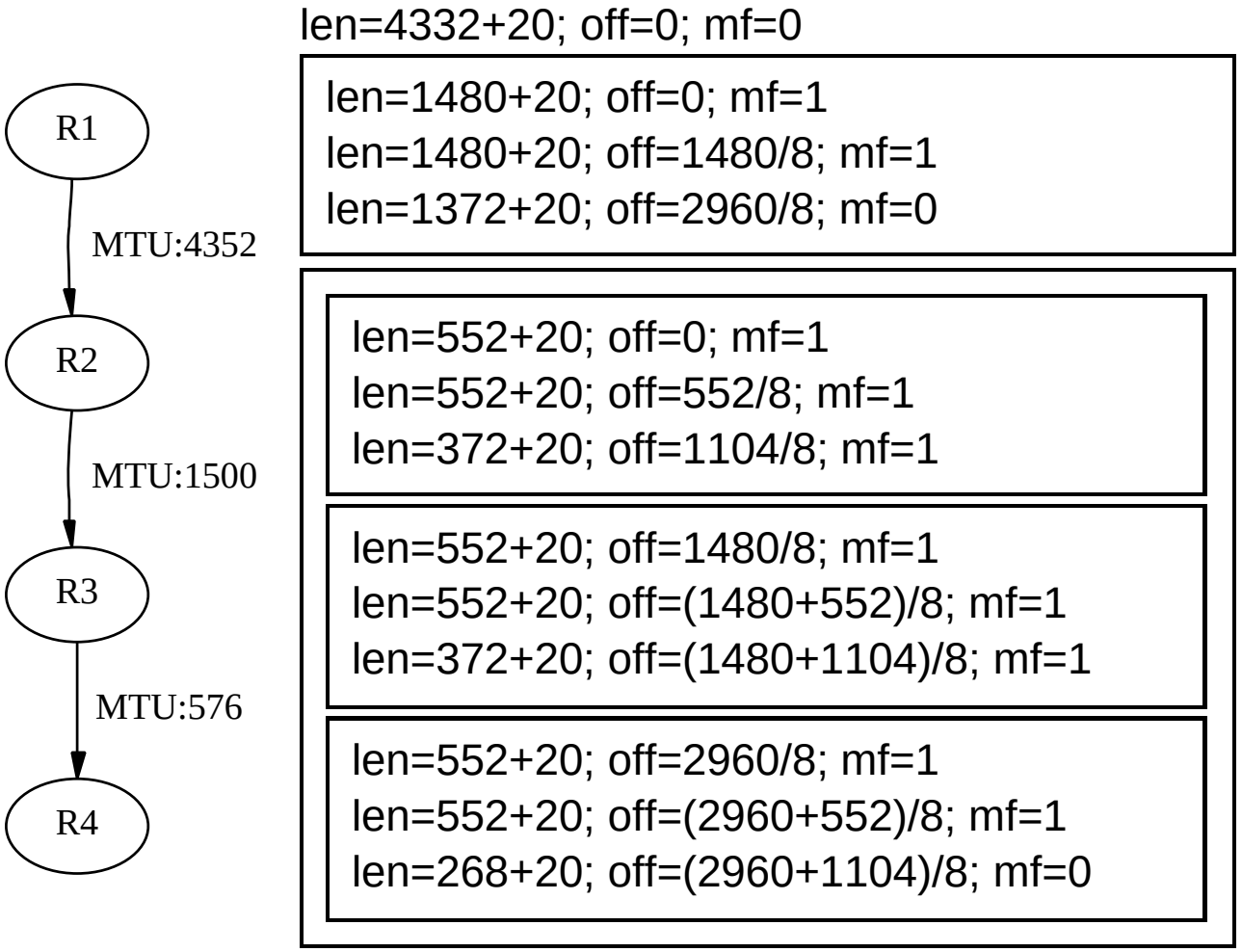
Flag Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

Why heterogenous MTU?

Why smaller MTU?

Why larger MTU?

Fragmentation Example



The rest of the header is copied, except some options.

What’s good and bad?

Common MTUs (rfc1191)

65535	Official maximum MTU	RFC 791
65535	Hyperchannel	RFC 1044
17914	16Mb IBM Token Ring	ref. [6]
8166	IEEE 802.4	RFC 1042
4464	IEEE 802.5 (4Mb max)	RFC 1042
4352	FDDI (Revised)	RFC 1188
2048	Wideband Network	RFC 907
2002	IEEE 802.5 (4Mb recommended)	RFC 1042
1536	Exp. Ethernet Nets	RFC 895
1500	Ethernet Networks	RFC 894
1500	Point-to-Point (default)	RFC 1134
1492	IEEE 802.3	RFC 1042
1006	SLIP	RFC 1055
1006	ARPANET	BBN 1822
576	X.25 Networks	RFC 877
544	DEC IP Portal	ref. [10]
512	NETBIOS	RFC 1088
508	IEEE 802/Source-Rt Bridge	RFC 1042
508	ARCNET	RFC 1051
296	Point-to-Point (low delay)	RFC 1144
68	Official minimum MTU	RFC 791

Path MTU discovery