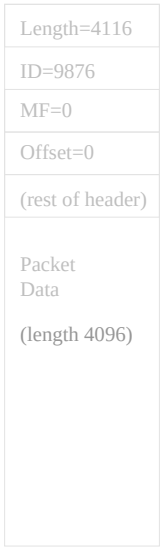
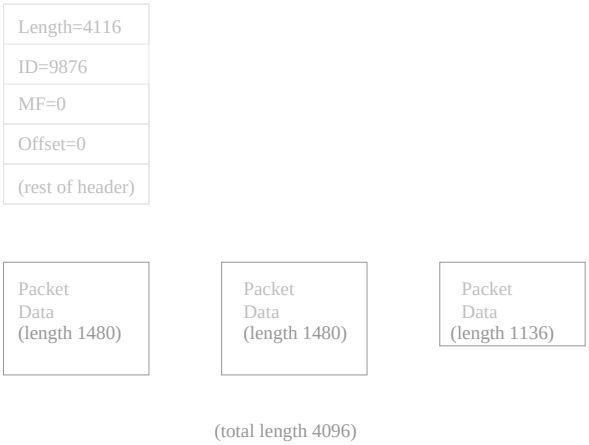
Fragmentation Example



Fragmentation Eg (2)

Fragmentation Eg (3)



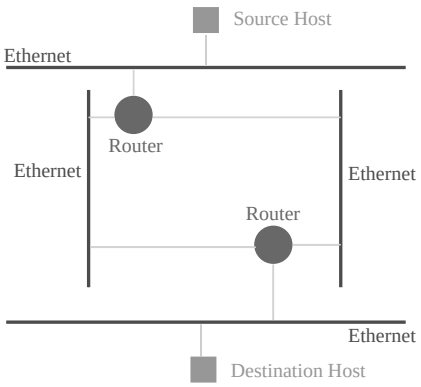
Fragmentation Eg (4)

Length=1500	Length=1500	Length=1156
ID=9876	ID=9876	ID=9876
MF=1	MF=1	MF=0
Offset=0	Offset=185	Offset=370
(rest of header)	(rest of header)	(rest of header)
Packet Data (length 1480)	Packet Data (length 1480)	Packet Data (length 1136)

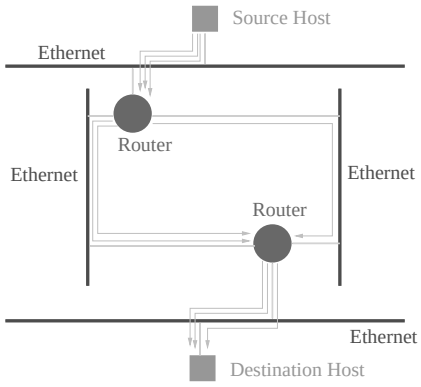
Fragmentation Eg (5)

Length=1500	Length=1500	Length=1156
ID=9876	ID=9876	ID=9876
MF=1	MF=1	MF=0
Offset=0	Offset=185	Offset=370
(rest of header)	(rest of header)	(rest of header)
Packet Data	Packet Data	Packet Data

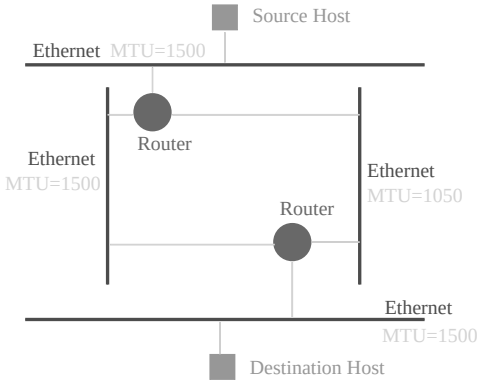
Example Network



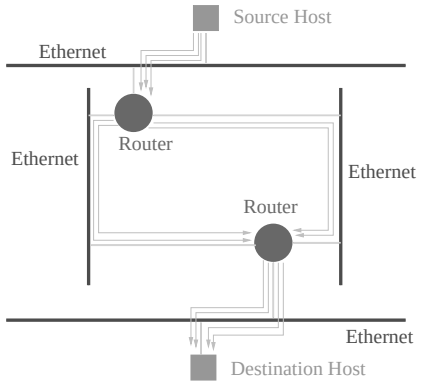
Sending Fragments



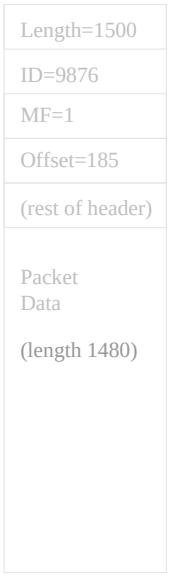
Example Network (with MTU)



Actual Packet Transmission

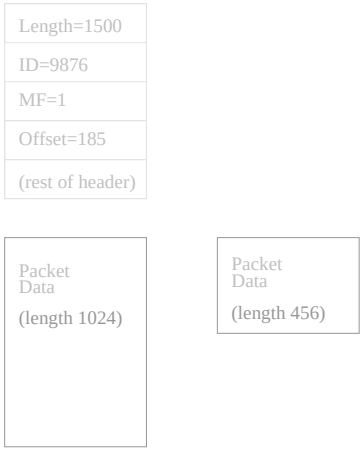


Fragmenting a Fragment

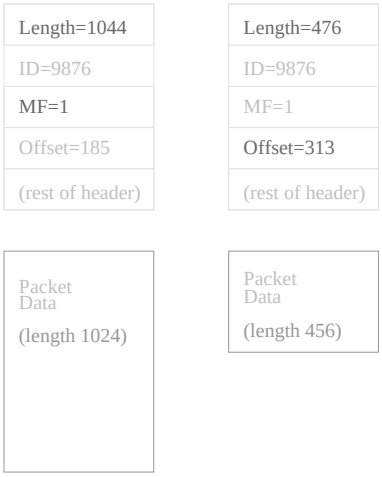


Fragmenting a Fragment (2)

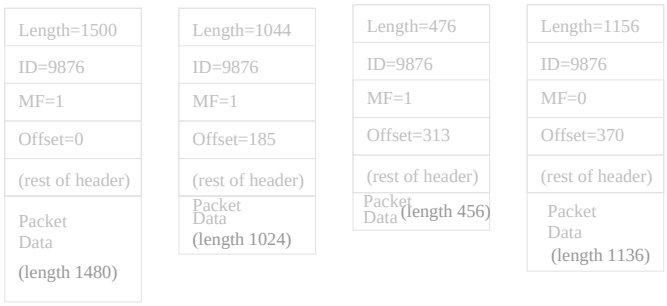
Fragmenting a Fragment (3)



Fragmenting a Fragment (4)



Fragmented Packet Sequence



Design Tradeoffs

- Why this design?
  - Why multiples of 8?
- Competing interests
  - Want header as small as possible
    - Less overhead traffic
  - Want as much info as possible
    - size of each fragment
    - where fragment fits in ordering
    - total packet size
    - which fragment is the last
    - should packet be fragmented
    - which packet is fragment part of

Design Tradeoffs

- Some information is implicit
  - need not be explicitly included
  - Eg:
    - total packet size (before fragmentation)
      - Take offset of last fragment
      - Add size of last fragment
      - Packet size now known
    - requires knowing which is last fragment
  - alternative
    - know size of packet
    - know offset and size of each fragment
    - last fragment is
      - fragment where offset + frag size == packet size
    - last fragment info implicit
  - Which is better?
    - Why?

Design Tradeoffs

- Want to include
  - fragment size
    - use ip length field
  - is this last fragment
    - needs a new field
      - yes or no answer
        so just one bit required
  - OK to fragment
    - needs a new field
      - yes or no answer
        so just one bit required
  - offset of fragment
    - needs to represent any offset within packet
      - 16 bit value required
  - which packet is this a fragment of?
    - need a packet identifier
      - N bit field
        allows 2^N fragmented packets

Design Tradeoffs

- How is all this information included?
  - We already say how it was done
  - But why was it done that way?
- Want identifier as big as possible
  - reduce chance of incorrect reassembly
  - parts of unrelated packets recombined
  - At least 16 bits desireable
- Also need offset
- And at least 2 flags
  - Maybe room for future expansion
- Totals 34 or more bits
  - Inconvenient value

Design Tradeoffs

- Examine packet offset
  - Do we need to allow fragmentation anywhere?
  - That is,
    - do we need to allow all 2^64 (-1) offsets?
  - No
    - can restrict where fragmentation happens
    - makes no significant difference
  - If we allow fragments only at 2^M byte boundary
    - only need 16-M bits for offset
      - M==0 (fragment anywhere)
        16 bits
      - M==1 (fragment only at even offset)
        15 bits
      - M==2 (offset multiple of 4)
        14 bits
      - M==3 (offset multiple of 8)
        13 bits
      - M==4 (offset multiple of 16)
        12 bits

Design Tradeoffs

- If M == 2 (or M > 2)
  - need only 14 (or smaller) bit offset field
  - Leaves space for 2 (or more) flag bits
    - "last" "don't fragment" ...
- Pick M=3
  - 3 flag bits
    - 2 assigned
    - one reserved - always 0 now
      - if 1
        perhaps change interpretation
        or who knows what
- Means offset always multiple of 8
  - take offset field
  - multiply by 8
    - gives byte offset

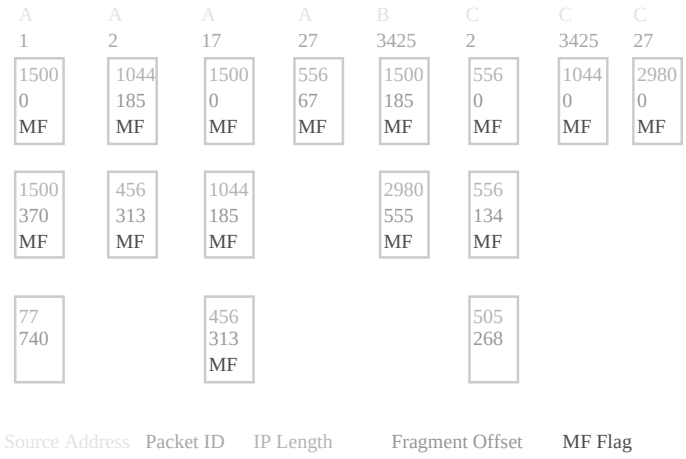
Design Tradeoffs

- Classic design issue of protocol design
  - of any design
- Cannot always have everything we want
  - and achieve reasonable cost
- Must examine each requirement
      - carefully!
  - Discover what is actually required
  - how much we really need
  - What it costs to add more
    - what we lose in other ways
- Then decide what is possible
  - and from the possible
  - what is optimal for the design

Reassembly

- Arriving fragments kept in queues until packets reassembled
- One queue for each Source-Address / Packet ID pair
- In each queue, the length/offset/MF fields control reassembly

Reassembly Queues



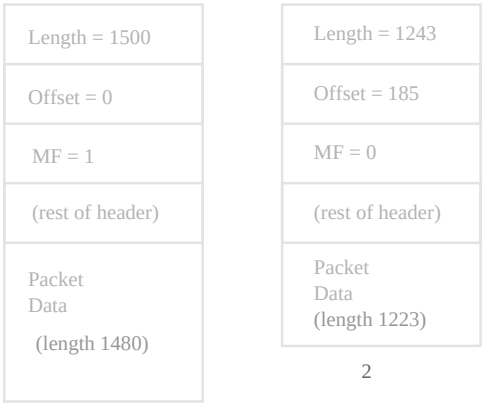
Merging Fragments

- If Old Fragment (Offset + DataLength) >= New Fragment (Offset)
  - then new fragment merged
    - onto end of old fragment
- If Old Fragment (Offset) <= New Fragment (Offset + DataLength)
  - then new fragment merged
    - onto beginning of old fragment

Merging Fragments (2)

- New fragment may merge
  - at end of one old fragment
  - and beginning of another old fragment
- If merged fragment has
  - Offset==0
- and
  - MF==0
- then whole packet has arrived, deliver it.

Fragment Reassembly Example

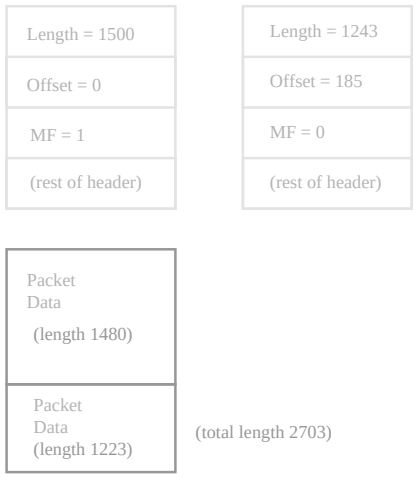


Length(1) + Offset(1) >= Offset(2) ==> Merge
1480 0 (*8) 185(*8)

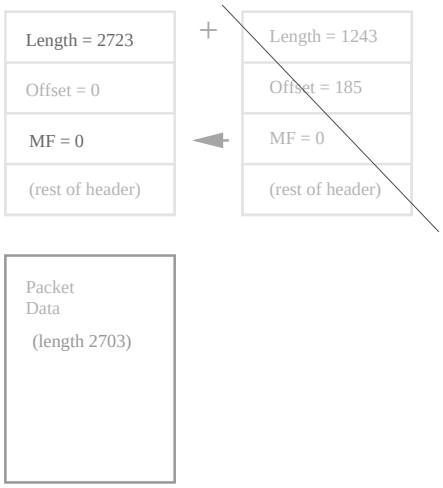
185 * 8 = 1480

Fragment Reassembly Eg (2)

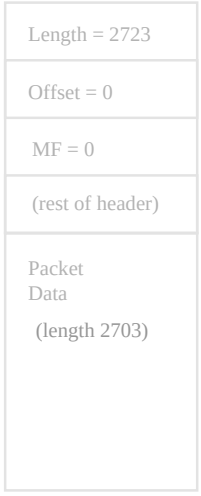
Fragment Reassembly Eg (3)



Fragment Reassembly Eg (4)



Fragment Reassembly Eg (5)



TTL and Reassembly

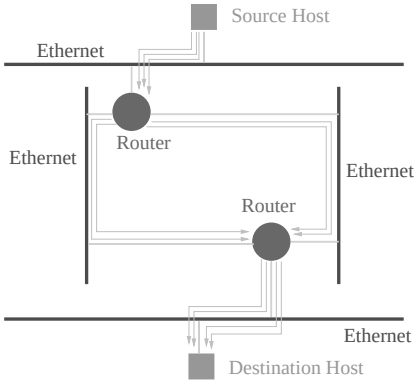
- All fragments (being IP packets) have a TTL
- Implementations may also
  - lower the TTL of arrived packets
    - not increase it
  - to set an upper bound upon
    - how long the fragment can wait
    - in the reassembly queue
- The TTL then works as normal
  - decremented once a second
  - while fragment waits in reassembly queue
- Packet must be discarded
  - when TTL reaches 0
    - unless delivered before then
    - delivered only when complete
      - reassembled

TTL and Reassembly (2)

- If TTL reaches zero, fragment is discarded
  - If it was an initial fragment (ie: offset == 0)
    - Send ICMP "time to live expired"
      - with code indicating
      - "expired waiting for reassembly"
- When merging packets
  - take the minimum of the two TTLs
    - using values at instant fragments merged
    - TTL not permitted to increase
      - even if another fragment could live longer

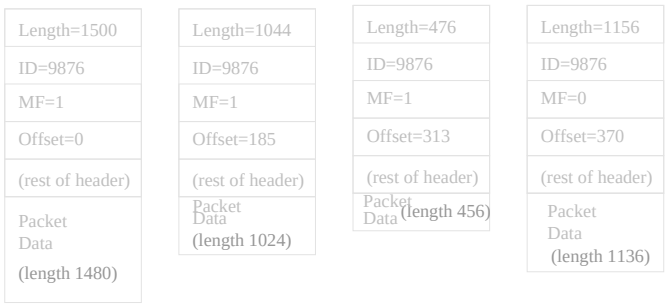
Odd Reassembly Cases

- Consider this example again ...



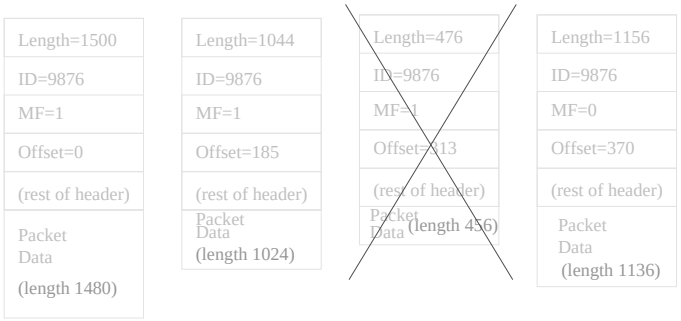
Odd Reassembly Cases (2)

- Which results in these fragments being generated and sent



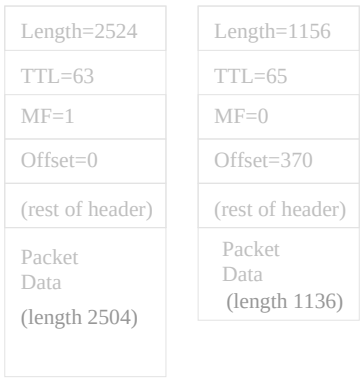
Odd Reassembly Cases (3)

- Assume that one fragment never arrives ...



Odd Reassembly Cases (4)

- Reassembly Queue for this packet at Destination Host will contain...

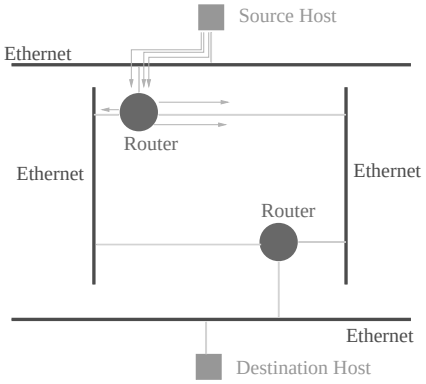


- Note that the first two fragments have been merged

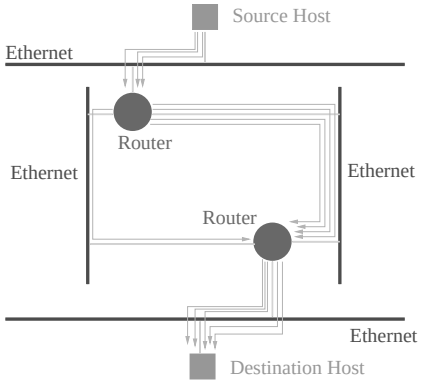
Odd Reassembly Cases (5)

- Now assume
  - a higher level protocol timer goes off
  - the packet is transmitted again.
- If everything goes as before
  - but all fragments arrive this time
  - packet will be merged, and delivered.
- But assume that the routers
  - now decide to send the packets
  - the opposite directions from first time...

Odd Reassembly Cases (6)

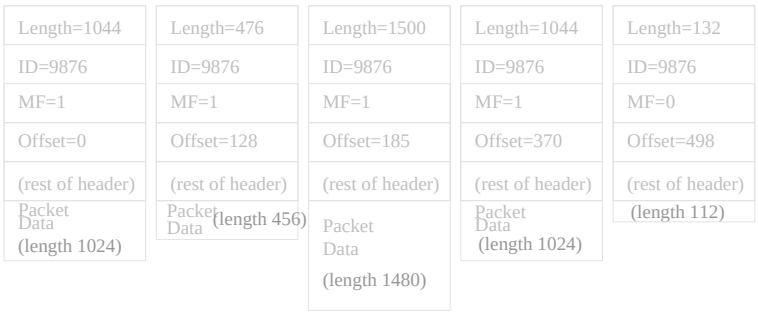


Odd Reassembly Cases (7)



Odd Reassembly Cases (8)

- This results in the following 5 packets being delivered to the Destination Host



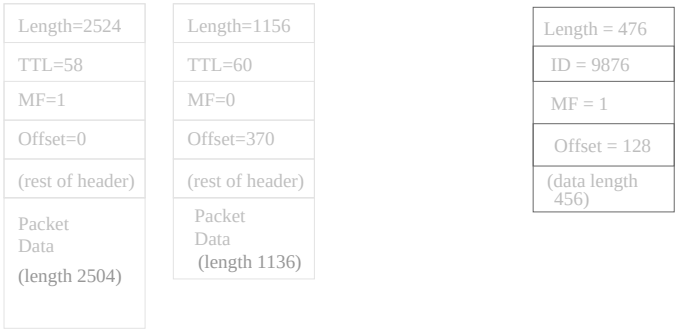
Odd Reassembly Cases (9)

- Consider what happens to the reassembly queue as each of the packets arrive
  - assume they arrive in the "natural" order

First Packet Arrives

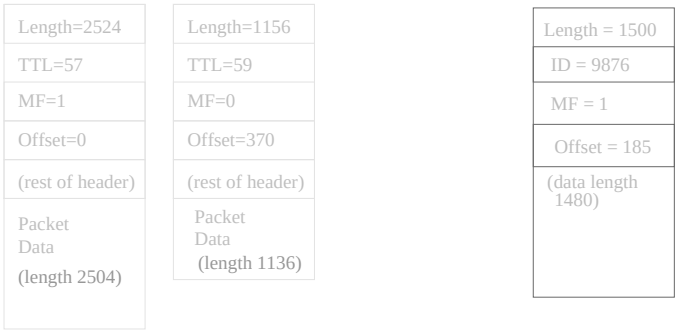
- 0*8 < 0*8 + 2504 ==> merge with first in Queue
- 1024 < 2504 ==> no new data in arriving packet, nothing to merge

Second Packet Arrives



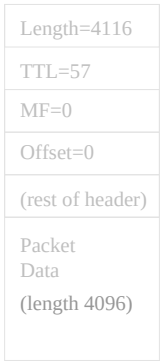
- 128*8 < 0*8 + 2504 ==> merge with first in Q
- 128*8 + 476 < 2504 ==> no new data in arriving packet, nothing to merge

Third Packet Arrives



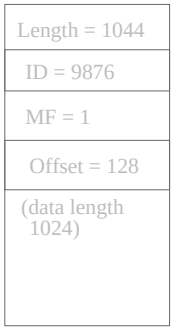
- 185*8 < 0*8 + 2504 ==>
  - merge with first in Q
- 185*8 + 1480 > 2504 ==>
  - new data in arriving packet, merge it
- 185*8 + 1480 >= 370*8 ==>
  - also merge with 2nd packet in queue

Third Packet Merged



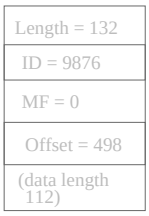
- Note overlapping data dropped
- Merged packet has
  - Offset==0
  - MF==0
  - ==> deliver it

Fourth Packet Arrives



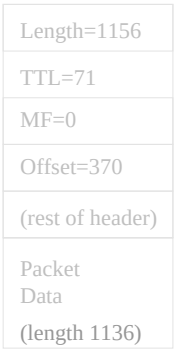
- Start a new reassembly queue
  - nothing currently there

Fifth Packet Arrives

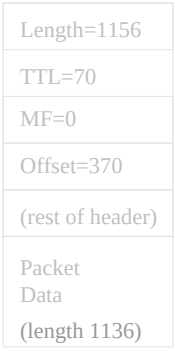


- 370*8 + 1024 <= 498*8 ==>
  - merge new packet with one in Q

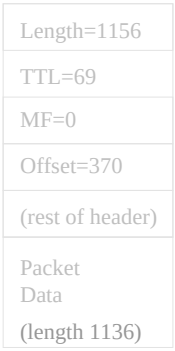
Fifth Packet Merged



Time Passes



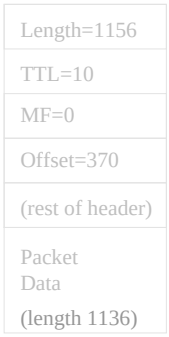
Time Passes



- Packet has been delivered to higher level
  - so this packet will (normally)
  - not be transmitted again.

Time Passes

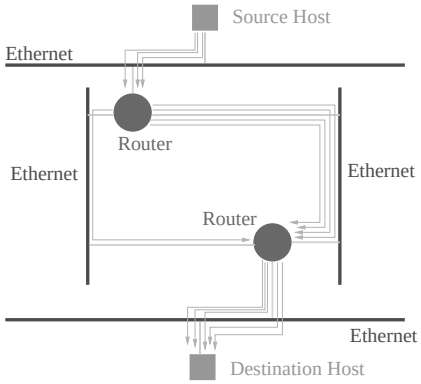
Time Passes



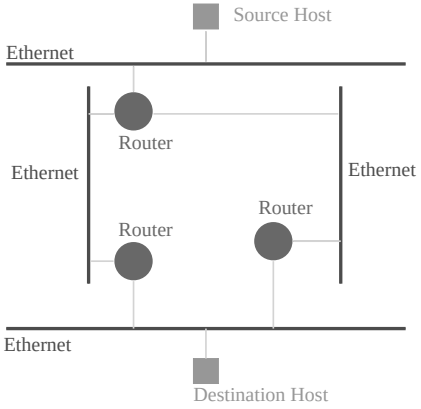
Time Passes

Time Passes

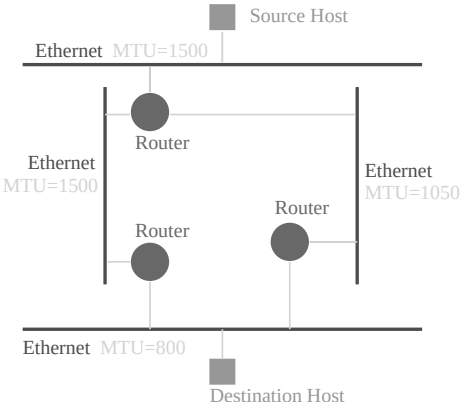
Where to Reassemble



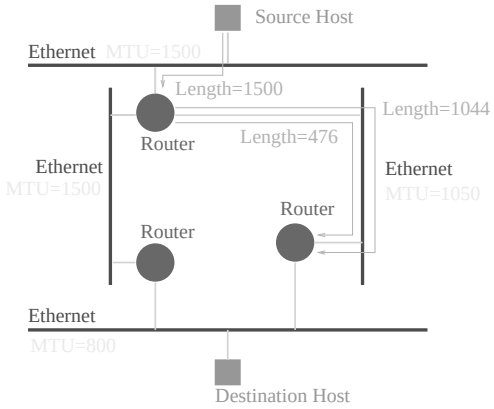
What if topology were different?



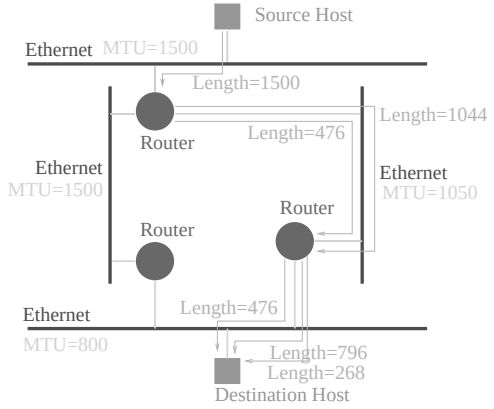
Another Fragmentation Eg



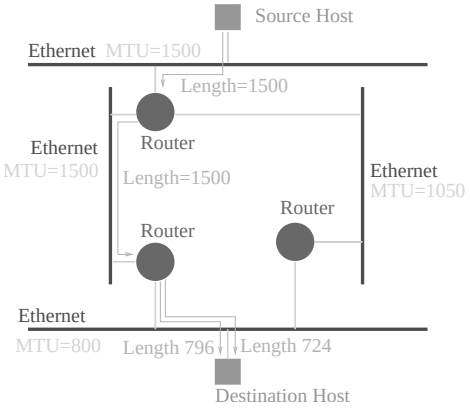
Another Fragmentation Eg (2)



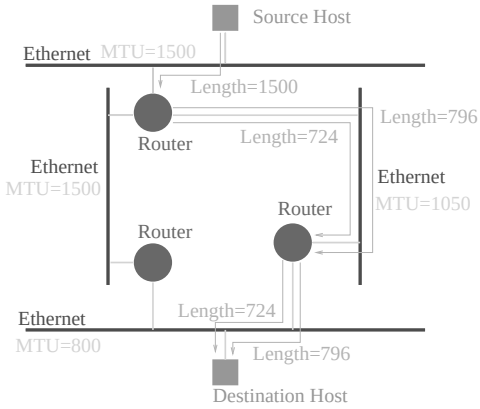
Another Fragmentation Eg (3)



Another Fragmentation Eg (4)



Another Fragmentation Eg (5)



Fragmentation Problem

- Where should packet be divided
  - when fragmentation is required
    - that is, where in the packet
    - not at which router
- Make minimum number
  - of required fragments (easy)
    - Generate biggest possible
      - plus whatever is left over
    - Equalise fragment sizes
      - as much as possible
- Exercise: Work out previous example
  - if final MTU were 700 rather than 800
  - How could router possibly know what to do?

Path MTU Discovery (PMTUD)

- Sending Host discovers the minimum MTU
  - of the Path used
  - and then sends its packets
  - no bigger than that
    - (fragmenting if needed)

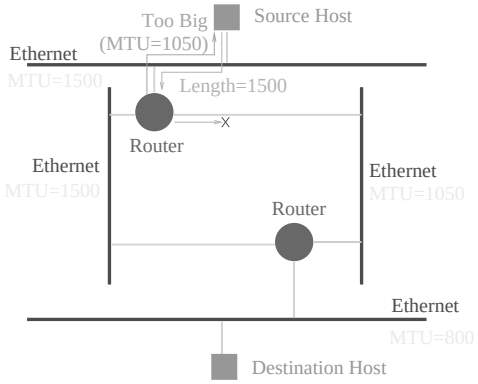
Vers	H.L.	TOS	Total Length
Packet Identifier		RDF / MF	Fragment Offset
TTL	Protocol		Header Checksum
Source Address			
Destination Address			
Options			
Data			

PMTUD (cont)

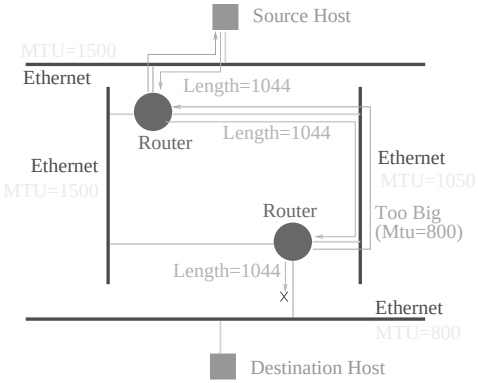
- If a packet arrives at a router
  - The DF bit is set in the header
  - Fragmentation is required to send the packet
- Router discards the packet
  - Sends ICMP "packet too big and DF set" to source
  - If new enough
    - Includes the MTU of the link to be used
    - Source host knows MTU of this link
      - smallest link so far

        WHY are we sure about that?

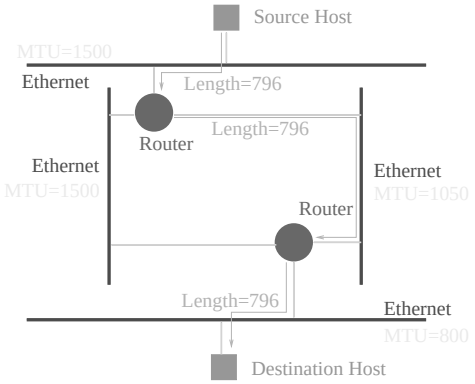
PMTUD (cont)



PMTUD (cont)



PMTUD (cont)



Fragmentation Lessons

- Fragmentation wastes bandwidth
  - Entire packets transmitted if fragment lost
- Fragmentation requires router complexity
  - More than just forward or drop decision
- Avoid fragmentation if possible
  - Use PMTUD
    - Discover max possible packet size
    - Use that as MSS for TCP
  - Keep UDP packets small
    - Avoid fragmentation
      - Probably...