Fragmentation Considered Harmful

Christopher A. Kent
Jeffrey C. Mogul

Digital Equipment Corporation
Western Research Lab

(Originally published in Proc. SIGCOMM '87, vol. 17, No. 5, October 1987)

Abstract

Internetworks can be built from many different kinds of networks, with varying limits on maximum packet size. Throughput is usually maximized when the largest possible packet is sent; unfortunately, some routes can carry only very small packets. The IP protocol allows a gateway to *fragment* a packet if it is too large to be transmitted. Fragmentation is at best a necessary evil; it can lead to poor performance or complete communication failure. There are a variety of ways to reduce the likelihood of fragmentation; some can be incorporated into existing IP implementations without changes in protocol specifications. Others require new protocols, or modifications to existing protocols.

1. Introduction

Internetworks built of heterogeneous networks are valuable because they insulate higher-level protocols from changes in network technology, because they allow universal communication without the expense of constructing a homogeneous universal infrastructure, and because they allow the use of different network technologies as appropriate to both local-area and long-haul links. Most datagram networks set a maximum limit on tile size of packets they carry, to simplify packet buffering in the nodes and to limit how long one packet can lie up the link. In a heterogeneous interned such as the DARPA IP Internet, these packet-size limits, known as MTUs (for *maximum transmission unit*) vary widely from 254 bytes for Packet Radio networks to 2000 bytes for the Wideband Satellite Network [22]; since nobody knows exactly what is connected to the Internet, the range in MTUs may be even broader.

In general, it is better to use a few large packets instead of many small packets to carry a given amount of data, because much of the cost of packetized communication is per-packet rather than per-byte. On a high-speed LAN, throughput can increase almost linearly with packet size over a wide range of sizes. Therefore, we prefer to make our packets as large as possible.

This desire for large packets conflicts with the variation in MTUs across an internet. We want to send large packets but some network along the packets' path may not be able to carry them. One approach to this dilemma is *fragmentation* when a node must transmit a packet that is larger than the MTU of the network. it breaks the packet into several smaller *fragments* and sends them instead. If the fragments are all sent along the same data link and are immediately reassembled at the next node, this is called *transparent* or *intra-network* fragmentation. If the fragments are allowed to follow independent routes, and are reassembled only upon reaching their ultimate destination this is called *inter-network* fragmentation. A good discussion of both methods, in more detail, may be found in Shoch [23].

In this paper, drawing on experience with a large heterogeneous internetwork, we examine fragmentation in the context of the IP protocol [18]. IP supports the use of inter-network fragmentation. (Transparent fragmentation may be also be used as long as it is invisible to the IP layer.) Fragmentation appears at first to be an elegant solution to the problem, but subtle complications arise in real networks that can result in poor performance or even total communication failure.

Experience with inter-network fragmentation in the IP Internet has convinced us that it is something to avoid. In section 2 we compare the advantages and disadvantages of fragmentation, in order to justify this assertion. We then discuss, in section 3, a variety of schemes for avoiding or recovering from fragmentation

2. What is wrong with fragmentation?

The arguments in favor of fragmentation are straightforward. Fragmentation allows higher level protocols to be unconcerned with the characteristics of the transmission channel, and to send data in conveniently sized pieces. Sending larger quantities of data in each IP datagram minimizes the bookkeeping overhead associated with managing the data. (See section 3.5 fowl specific example.)

Fragmentation allows the source host to deal with routes having different MTUs without having to know what path packet are taking. The safest strategy is for the source to send very small datagrams, at a great loss of efficiency. Fragmentation allows the source to choose a size that is "reasonable" and, when that size proves to be too large, provides a mechanism that allows data to continue to get through.

Finally, fragmentation allows protocols to optimize performance for high bandwidth connections. Emerging network technologies have larger and larger MTUs. Most local networks have MTUs large enough to send 1024 bytes of user data plus associated overhead in a single packet; new technologies will allow ten times that. Fragmentation provides a mechanism for deciding the actual packet size as late as possible. It especially allows protocols to avoid choosing to send small datagrams until absolutely necessary. Protocols can choose large segment sizes to take advantage of the large MTU in a local network, and rely on fragmentation at gateways to send the segments through networks with small MTUs when needed. If datagrams must traverse a route consisting of several high-MTU links followed by a low-MTU link, by delaying the use of small packets until the low-MTU link is reached, fragmentation allows the use of large packets on the initial high MTU links, and thus uses those links more efficiently.

The arguments against fragmentation fall into three categories

- **Fragmentation causes inefficient use of resources:** Poor choice of fragment sizes can greatly increase the cost of delivering a datagram. Additional bandwidth is used for the additional header information, intermediate gateways must expend computational resources to make additional routing decisions, and the receiving host must reassemble the fragments.
- **Loss of fragments leads to degraded performance:** Reassembly of IP fragments is not very robust. Loss of a single fragment requires the higher level protocol to retransmit all of the data in the original datagram, even if most of the fragments were received correctly.
- **Efficient reassembly is hard:** Given the likelihood of lost fragments and the information present in the IP header, there are many situations in which the reassembly process, though straightforward, yields lower than desired performance.

2.1. An overview of fragmentation in IP

IP is a protocol providing unreliable delivery of *datagrams*. IP datagrams are encapsulated in network-specific *packets*. Gateways may fragment an incoming packet if it will not fit in a single outgoing packet; in this case, each *fragment* is sent as a separate packet. The [P header contains several fields that are used to manage fragmentation [18]:

- **Identification:** A 16-bit field assigned by the sender to aid in assembling the fragments of a datagram. The tuple *(source, destination, protocol, identification*) for a given datagram must be unique over all existing datagrams. When a packet is fragmented, the value of the Identification field of the original packet is copied into each fragment.
- **Time to live (TTL):** An 8-bit field that specifies the maximum time, measured in seconds, that the packet may remain in the Internet system. If TTL contains the value zero, the packet must be discarded. The TTL must be decreased by at least one every time the packet passes through a gateway, even if the time required to process the packet is less than a second. Thus, the TTL field is an upper bound on packet lifetime.
- **Fragment offset:** A 13-bit field that identifies the fragment location, relative to the beginning of the original, unfragmented datagram. Fragment offsets are in units of 8 bytes.
- **More fragments:** A l-bit field that indicates whether or not this is the last fragment of the datagram.

The reassembly process consists of matching the protocol and identification fields of incoming fragments with those of fragments already held, and coalescing the data into complete datagrams. Fragments must be discarded if their TTL expires while they are held for reassembly. (For more details of the reassembly algorithm, see [5].)

Higher level protocols such as TCP (Transmission Control Protocol) [19] use IP as a basis to implement a reliable connection between two client processes. Portions of the data stream known as *segments* are sent in individual IP datagrams, along with control information used by the cooperating TCP processes to ensure reliable communication. In particular, TCP uses a sequence number that covers individual bytes in the data stream, and an acknowledgment mechanism that allows the receiving process to tell the sender “I have correctly received all data up to and including sequence number *n*.”

2.2. Fragmentation causes inefficient resource usage

Consider the costs associated with sending a packet. Each time it passes through a gateway, there is some constant computational overhead to make routing decisions, modify the packet header, compute the new checksum, and move the packet between the appropriate incoming and outgoing queues. In addition, a portion of the available bandwidth on the incoming and outgoing interfaces is consumed. In many cases, the constant computational overhead dominates the cost. Input and output may be overlapped using DMA devices; in a typical uniprocessor gateway, there is no way to parallelize the computational overhead.

Fragmenting at an IP gateway, rather than having the host choose the appropriate segment size to avoid fragmentation, may lead to suboptimal use of gateway resources and network bandwidth. Consider a TCP process that tries to send 1024 data bytes across a route that includes the ARPAnet, which has an MTU of 1006 bytes. The IP and TCP headers are at least 40 bytes long, leading to a total unfragmented IP datagram 1064 bytes in length. To cross the ARPAnet, this will be broken into a 1006 byte fragment, followed by a 78 byte fragment. These short fragments amortize the fixed overhead per ARPAnet packet over very few bytes of data, and the total packet count is much higher than needed. If the sending TCP instead chooses segments that fit in a 1006 byte ARPAnet packet, the total packet count is minimized, and the total overhead is as low as possible.

For example, consider sending 10 Kbytes of data. Sending 1024-byte TCP segments generates 10 IP datagrams, each 1064 bytes long. Each datagram is fragmented into two ARPAnet packets, one 1006 bytes long and the other 78 bytes, for a total of 20 packets. If the originating TCP instead sends 966 byte segments (the largest that will fit in a single ARPAnet packet), only 11 packets are sent.

Another limit to utilizing available bandwidth lies in the interaction of the TTL and Identification fields. Assume that a reasonable initial value for the TTL field is 32 (the maximum hop count from edge to edge of the DARPA Internet is currently estimated to be between 15 and 20). If we allow fragmentation, we must ensure that all datagrams in flight have unique values for the Identification field. Thus, the maximum datagram rate is $2^{16}/32$, or 2048 datagrams per second. Current gateways can forward nearly 1000 packets per second; high performance workstations and interfaces can generate packets much more rapidly, and can probably forward 4000 packets per second. We are certainly within five years of having commonly available processor and network technology that pushes against the limit imposed by the 16-bit Identification field.

This limit implies that, to increase bandwidth in the presence of fragmentation, hosts should send larger datagrams, so as to carry more data per value of the Identification field. This is a bad idea, because large datagrams lead to more fragments, and we shall show that this increases the likelihood of a severe decrease in performance. If we simply avoid fragmented datagrams, values of the Identification field need not be unique, and there is no bandwidth limit imposed by its size.

2.3. Poor performance when fragments are lost

When segments are sent that are large enough to require fragmentation, the loss of any fragment requires the entire segment to be retransmitted. This can lead to poorer performance than would have been achieved by originally sending segments that didn't require fragmentation.

Gateways in the Internet must drop packets when congested. If the gateways are congested, dropping fragments only makes the situation worse. Dropped fragments mean increased retransmissions, which leads to more fragments. As the loss rate goes up due to heavy congestion, the total throughput drops dramatically, since the loss of any one fragment means that the resources expended in sending the other fragments of that datagram are entirely wasted.

Even when congestion is not the problem, retransmission does not necessarily increase the likelihood that all the fragments that make up the segment will arrive unscathed. In particular, network idiosyncrasies may conspire to cause the same fragment or fragments to be lost on successive retransmission. We call this *deterministic fragment loss*.

An example of deterministic fragment loss occurs in the 4.2BSD Unix implementation of TCP when datagrams pass between a local network (typically an Ethernet or a Proteon ring, with MTUs of 1500 or 2046 bytes, respectively) and the ARPAnet. The TCP prefers to send 1024 byte data segments, which are transmitted in 1064 byte IP datagrams. As seen earlier, this results in two fragments, 1006 and 78 bytes long.

The receiving gateway receives both fragments and sends them out over the local Proteon ring. The Proteon ring interface does not have sufficient buffering to receive back-to-back packets, so it consistently drops the second fragment. The sending TCP times out, and retransmits the 1024 byte segment, which will again be fragmented. The second fragment is again lost, the segment times out, and eventually the connection is broken.

In addition, many of the gateways in the Internet today are derived from 4.2BSD Unix. This implementation of IP does not properly fragment a previously fragmented packet, preventing some fragments from ever reaching their destination, which might better be called *guaranteed* fragment loss.

2.4. Efficient reassembly is difficult

Reassembling fragments into datagrams at the IP layer is considerably less robust than constructing a reliable stream at the TCP layer. The window mechanism in TCP allows the reassembly process to accurately gauge how much buffer space to allocate for the current stream of unacknowledged data bytes. Also, because in TCP the data stream is covered by a sequence number for each data byte, once a contiguous sequence of bytes at the beginning of the outstanding data stream has been reassembled, it can be acknowledged and handed up to the next layer. Thus, progress can always be made, even if in small amounts.

At the IP layer, there is no indication in the header of a fragmented packet of how many other fragments follow, or of the length of the entire datagram. The More Fragments bit tells only if this the last fragment of the datagram, and the Fragment Offset field tells only the position of this fragment in the complete datagram. If the total size of the incoming datagram is too large to fit available buffer space, no progress can be made. The IP specification requires hosts to be able to reassemble datagrams at least 576 bytes in length; larger segment sizes must be explicitly negotiated by higher level protocols.

Even if there is sufficient buffer space to reassemble a very large datagram, conflicts can occur. In the Internet, it is possible for fragments of the same datagram to take different routes to their ultimate destination. Depending on queue management strategies at gateways along the way, a fragment of a small datagram may arrive intermixed with the fragments of a large datagram. More concretely, assume two datagrams, L (large) and S (small), are fragmented as $L_1L_2L_3L_4L_5L_6L_7L_8$ and S_1S_2. If there are only eight buffers available, and the reception order is $L_lL_2L_3L_4L_5L_6L_7S_1L_8S_2$, reassembly of L cannot succeed, despite adequate buffer space. Upon reception of S_l, the reassembly process could discard L_l through L_7, which would leave six free buffers and allow S to be reassembled when S_2 arrives. Or, it could discard L_8 (and subsequently S_2), blocking reassembly of both L and S; the buffers would be kept full until the fragments expire. In either case, the work done to transport all the fragments of L is entirely wasted. It is not possible to coalesce a complete initial string of fragments and partially acknowledge receipt of the datagram in order to free some of the buffer space. (Dave Mills first pointed out this behavior in [13].)

It is difficult to decide how long to hold on to received fragments. The only firm limit is the TTL field; the reassembly process must discard fragments as their TTLs expire. Since each gateway decrements the TTL field, it must be set high enough to traverse the longest possible route, and thus may still be quite high when the packet arrives at its destination. Naive use of the received TTL as a reassembly timeout will cause some fragments to occupy buffer space for a much longer time than necessary. Use of too short a reassembly timeout will cause fragments to be dropped too quickly, leading to unnecessary retransmissions.

Because IP is a datagram protocol, there is no guarantee that a given fragment will ever arrive. A higher level protocol may retransmit a lost IP datagram. If a retransmitted datagram does not have the same value for the IP Identification field, its data will not be recognized as being the same as that in previously received fragments. The old fragments will occupy buffer space until timed out or forced out by incoming packets, and cannot fill holes left by fragments dropped from the second datagram. This suggests that higher level protocols should attempt to use the same value for the IP Identification on both the original and retransmitted data. (This idea was proposed by John Shriver [24].)

3. Avoiding fragmentation

We believe that, in most circumstances, the potential disadvantages of fragmentation far outweigh the expected advantages. Thus, hosts should avoid sending datagrams that are so large that they will be fragmented. The length limit can be determined by a variety of general approaches:

- **Always send small datagrams:** There is some datagram size that is small enough to fit without fragmentation on any network; we could simply send no datagrams larger than this limit.
- **Guess minimum MTU of path:** Use a heuristic to guess the minimum MTU along the path the datagram will follow.
- **Discover actual minimum MTU of path:** Use a protocol to determine the actual minimum MTU along the path the datagram will follow.
- **Guess or discover MTU and backtrack if wrong:** Since an estimate might be wrong, and a discovered MTU may change if a route changes, sometimes we may have to adjust the length limit.

This requires both a mechanism for detecting errors, and a mechanism for correcting them.

Later in this section we will discuss more specific fragmentation avoidance schemes.

All these strategies assume that the route the datagrams will follow is independently determined. If multiple routes are available between source and destination, one might instead try to avoid fragmentation by using source-routing to avoid data links with small MTUs. Suitable alternate routes seldom exist, however, and even when they do we see no efficient way for an IP host to obtain enough information to choose a good source-route.

IP is a layered protocol architecture, and fragmentation avoidance must be done at the right layer. It makes little sense to build redundant mechanisms into several layers if it is possible to do it once. This implies that the right place for fragmentation avoidance is the layer common to all IP communication, the IP datagram layer itself (and its partner, the ICMP protocol). It would be a poor idea to put the entire fragmentation avoidance mechanism in, say, the TCP layer, because both the mechanism and any additional protocol would have to be duplicated in parallel layers, such as UDP[17], NETBLT[6], and VMTP[3], and because it would be awkward for a TCP-based mechanism to share knowledge with other layers and across connections.

This is not to say that layers above IP should be uninvolved in fragmentation avoidance. Architectural layering does not mean that higher layers must be kept ignorant of fragmentation issues. Optimal performance depends upon cooperation between layers for example, the TCP layer should not send huge segments if the IP layer knows that they will be fragmented.

Most of the fragmentation-avoidance schemes we will propose depend on keeping some knowledge about the minimum MTU (MINMTU) on the path a datagram will follow. A MINMTU value could be associated with a specific destination network, a specific destination host, a specific route (there may be several routes to one destination, with differing MINMTUs), or a specific connection (since for different applications, we may want to choose between optimizing for maximum bandwidth versus minimum delay, and thus might want to accept different risks of fragmentation for different connections to the same host). The MINMTU values could be kept in the IP routing database, or in a separate database, especially if per-connection MINMTUs are wanted. To support per-connection MlNMTUs, the IP layer must obtain a connection identifier from connection-oriented higher layers.

It is our belief that a per-connection scheme (degenerating to a per-route-to-specific-host scheme for connectionless protocols) is the most flexible one. While it is true that by keeping per-destination-network information one might be able to pool information about several hosts, this is not necessarily safe. Because many networks are subnetted [15], because MTUs may vary among the subnets of a given network, and because one cannot tell whether a remote network is subnetted or not, it is not true that knowing the MINMTU for one host reliably gives you the MINMTU for all other hosts on the same network.

Routes in a datagram network are not necessarily symmetric; the route a packet takes may not be the reverse of the route taken by a packet traveling in the opposite direction. Because of this, it is not safe for a host to assume that it can send a datagram as large as the one it has received from its peer. An independent MINMTU determination must be made for each direction, although the peer hosts may assist each other in doing so.

When the IP layer has determined the MINMTU for a connection or destination, it can make this information available to higher layers, such as TCP, that are generating segments to be sent as IP datagrams Segment-generating layers should ask the IP layer for a MINMTU before sending a segment; connection-based layers should either check periodically that the MINMTU has not changed, or should be able to handle asynchronous notification of a change.

3.1. Fragmentation avoidance without protocol changes

In this section we describe several fragmentation avoidance schemes that can be implemented without changing existing protocol specifications or creating new protocols. There are obvious advantages to such approaches, since they can be taken immediately by individual sites or vendors; further, we have sufficient experience with one of them to believe that it works fairly well. On the other hand, none of these schemes can make use of exact knowledge of MlNMTUs, and so may not provide optimal performance.

3.1.1. Always send tiny datagrams

If a host always sent datagrams no larger than the minimum MTU over the entire internet, these datagrams would never be fragmented. In the IP Internet the limit is no higher than 254 bytes, and might be lower. Since almost all of the Internet supports larger MTUs, and since performance depends so strongly on packet size, this approach can't provide reasonable performance. It is worth invoking only as a temporary diagnostic measure if performance actually increases when the datagram size is decreased, this is a clear indication that inappropriate fragmentation is taking place for larger datagrams.

Alternatively, one might assume that using a 576-byte limit is small enough to avoid fragmentation in virtually all cases (we hope that in the future, all new LP network links would be capable of handling packets of this size). 576 bytes is set forth in the IP specification [18] as the maximum size a host can send without explicit permission from the receiving host, so it is reasonable as an arbitrary value.

3.1.2. Send 576-byte datagrams if the route goes via a gateway

The IP layer can determine if the route for a connection or destination goes via a gateway. If it does, then the size limit is set to 576 (our favorite arbitrary value); otherwise, any size up to the MTU of the data-link layer may be used.

This approach provides maximum performance for local connections, and reasonable assurance that on most non-local connections, datagrams will not be fragmented. It is not perfect, since

1. It does not avoid fragmentation on every path
2. It may unnecessarily limit packet size, especially on subnetted collections of high-speed LANs that all support large packets.
3. If proxy ARP is used [14] then the IP layer may be fooled into believing that a non-local path is local, and thus use large datagrams when they are not necessarily safe.

However, it is quite easy to implement and in general provides good performance. A variant of this scheme, implemented in the TCP layer, has been used for several years at many sites and is now incorporated in 4.3BSD Unix [12]. This is the method we recommend in the absence of protocol changes.

3.13. Send 576-byte datagrams if the route goes off-net

Instead of checking whether a destination is behind a gateway, the IP layer can examine the destination's network number to decide if it is local or non-local. In a subnetted environment, this trades a higher risk of guessing too high a MINMTU for higher performance within the local collection of subnets.

3.2. Fragmentation avoidance with protocol changes

In this section we describe several fragmentation avoidance schemes that require changes to existing protocol specifications or the creation of new protocols. Mostly, these involve changes to gateways and some minor changes to IP-layer software; all are designed so as to coexist with unmodified gateways and hosts.

3.2.1. Probe mechanisms

Ideally, for a host to be able to send the largest possible datagrams that will not be fragmented, it must have perfect information as to the MINMTU along the path the datagrams will follow. Since most IP hosts do not even know what that route is, much less what the MTUs are along the route, we need a mechanism for discovering MINMTU.

The most straightforward kind of mechanism is to send a packet along the route, collecting MTU information as it goes; we call these probe mechanisms. Probe mechanisms require support from gateways each gateway along the route must update the probe according to the MTU of the hop it is about to take. Probe mechanisms also require support from peer hosts, since paths are asymmetric, once a probe reaches the end of its route, the information it has collected must be returned to the source host.

A probe may either gather a list of all the MTUs along the path (somewhat analogous to the IP “Record Route” option), with which the host can determine the MINMTU, or the probe may simply carry only the lowest MTU value seen along the route. The former method provides a little more information; the latter method is easier to implement and results in shorter packets.

A probe may be made only once, at the beginning of a connection or the use of a route, or it may be made periodically. Periodic probes are preferable if the MINMTU is kept per-destination or per-connection, since the route may change. If MINMTU information is kept per-route, then it will not change and consequently probes need not be repeated.

Probe mechanisms are useful for discovering other path characteristics besides MINMTU. As long as one is processing a probe, it makes sense to collect a variety of information, since it comes at little additional cost. This information could include:

Minimum bandwidth
Useful for determining appropriate transmission rates; if a host knows that a 9600-baud link is part of the path, it should behave differently than if the path is entirely via 100 Mbit fiber networks.

Maximum delay
Useful for determining realistic round-trip times; if a satellite channel is in use, with a delay of several hundred milliseconds, a host should not retransmit as quickly as if the end-to-end delay were several milliseconds.

Maximum queue length
: A high value implies congestion; if measured using the "fair-queueing" algorithm [16] it indicates to a host whether it is sending too much. Alternatively, a "congestion-encountered" flag could be set if any gateway along the path is experiencing congestion.

Maximum error rate
: When a link along the path is experiencing a high error rate, a host might choose to send shorter packets (so as to reduce the likelihood that an entire datagram is dropped because of a single error) or use error-correcting codes.

Hop Count
: The total number of links traversed along the route may be of interest, for example, in choosing a value for the "Time To Live" field. (Collection of hop counts was proposed by Mike Karels [10].)

It is not necessary for every gateway along the path to support probing, providing they all forward the probe. Gaps in the probe information are not fatal; at worst, host behavior is the same as if no probing is done. A gateway that does support probing can cover up for an occasional uncooperative gateway by looking at the incoming link as well as the outgoing link when determining the MINMTU.

Since route choices may depend on the IP "Type of Service" and perhaps the IP "Security" option, probes should carry the same Type of Service and Security as the data packets will [4]; gateways should observe Type of Service and Security when updating values in probes.

3.2.2. Probing with ICMP messages

A probe can be done using a separate packet; in the IP architecture, we would do this using a new ICMP "Probe Path" message. This is described in detail in appendix I.

Briefly, a host wishing to probe a path sets initial values for the fields of the Probe Path message, then sends it to the destination host. Each gateway along the route updates various fields of the message. When the destination host receives the message, it copies the recorded information into a different area of the message, reinitializes the recording fields, and returns the message to the original host. If the second host requests, the message may make one more trip, after which both hosts will have the path information, including MINMTU.

3.23. Probes piggybacked on IP headers

It is not necessary to send a separate packet to probe the path. Instead, the probe information can be piggybacked on the actual data packets, as part of the IP header. In appendix II we describe new IP header options for recording and returning MINMTU information. (Additional options could be defined for recording other path characteristics.)

In this case, a host wishing to probe a path sets initial values for the "Probe MTU" option in the IP header of a datagram it is sending. Each gateway along the route may update the value carried in this option. When the destination host receives the datagram, it copies the recorded information into a "MTU Reply" option and attaches it to the next datagram going back to the source host. When this reply is received, the first host knows the MINMTU; the second host may pass its own Probe MTU option along with the MTU Reply, so after one more datagram both hosts can know the MINMTU.

Because the piggyback method does not involve additional packets, it may be cheaper than the ICMP method; it should be cheap enough that one can send it frequently, to react to changes in the route. The drawback is that it adds overhead to the processing of data packets, and may be harder to implement, since when a host wants to return a MINMTU value it must find an outgoing datagram to which it can attach the MTU Reply option.

3.3. Recovery from fragmentation

Instead of avoiding fragmentation by trying to predict when it will occur, a host could instead detect when it occurs and recover by shrinking the datagram size. Detection can be accomplished with or without the use of new protocols.

All methods start by assuming a large datagram size, and adjusting the datagram size based on indications of fragmentation. After a few iterations of this process using a binary search on the datagram size, and after receiving acknowledgments from the remote host to verify that the datagrams are actually arriving, the source host can have arbitrarily precise knowledge of the MINMTU for the path.

Detection methods have the advantage that they cause no additional overhead unless fragmentation occurs. They have the disadvantage that they can create lots of useless traffic if not carefully implemented.

3.3.1. Use of "Don't Fragment" flag

One such approach is to set the "Don't Fragment" flag on every datagram, and use the MTU of the first hop as the initial datagram size. If a datagram reaches a gateway that would have to fragment it, the gateway (obeying the "Don't Fragment" flag) must drop the datagram and return an ICMP "Destination Unreachable/Fragmentation Needed and DF Set" message [20]. The source host, upon receiving this ICMP, should try a smaller datagram size and retransmit any unacknowledged data. (A variant of this scheme was first proposed by Geof Cooper [7].)

This method requires no protocol changes, but the drawback is that until the proper MINMTU is discovered, many datagrams will be dropped, and thus it may take a long time to set up a connection. Since many connections transfer only a few data packets (mail, for example), this is a significant overhead; it would only be useful for long connections.

3.3.2. Passive detection of fragmentation

A complementary approach is to allow fragmentation of any datagram, but to detect when this happens and adjust the datagram size accordingly. While this might take longer to arrive at the proper MINMTU, it does not force gateways to drop any datagrams in the meantime; thus it improves performance on long connections without harming brief ones. One way to do this without changing protocols is to observe the retransmission rate; from a high retransmission rate one might deduce that deterministic fragment loss is occurring. The datagram size can be lowered until the retransmission rate drops noticeably.

The problem is that a high retransmission rate may also be caused by other problems, especially congestion. Cutting the datagram size is exactly the wrong approach when fragmentation is not occurring and the path is congested, since it increases the number of packets required to send the same data and thus increases the congestion. This approach should therefore be used only when an independent mechanism is used to detect or suppress congestion, such as the use of ICMP source quenches, Nagle’s fair-queueing algorithm [16], or statistical properties of the round-trip delays [9].

3.3.3. Proper use of “Time Exceeded” ICMP messages

The receiving host can tell if it is losing fragments because partial datagrams waiting on its reassembly queue will time out. The ICMP protocol currently includes a “Time Exceeded” message, including a code that can be set to indicate “fragment reassembly time exceeded.” While this message does not convey complete information on the MINMTU of the path, it is a clear indication that the source host has guessed too high and should reduce the size of the datagrams it is sending. Apparently, many IP implementations do not end this message, nor do many know what to do with it.

3.3.4. “Fragmentation Warning” ICMP messages

In this scheme, when a gateway fragments a datagram, it forwards the fragments as usual but also sends a message back to the source host. This “Fragmentation Warning” ICMP message would carry the maximum allowable datagram size, so that the source host could reduce the datagram size to fit through the link in question. The process may have to be repeated if a subsequent link has a slightly smaller MTU. (A variant of this scheme was first proposed by Art Berggreen [1].)

This scheme has a serious danger if the source host does not receive or act upon the warning message, not only will fragmentation continue to occur, but a lot of useless traffic will be created by subsequent warning messages. A gateway could avoid sending multiple warnings to the same host, at the cost of maintaining a cache of recently warned host addresses. Alternatively, we could introduce a new “Warn if Fragmented” flag in the IP header, analogous to the “Don't Fragment” flag. Only if this flag is set would a warning be issued, and a source host setting this flag should take care to heed warnings.

3.3.5. “Fragments Received” ICMP messages

A similar approach generates warnings at the destination, rather than the gateways. If a host receives a fragmented datagram, it can send a “Fragments Received” ICMP message back to the source host. This warning would carry the size of the largest fragment of the datagram; this is a lower bound on MINMTU, although it is possible that larger datagrams could be sent without fragmentation. Again, some mechanism is needed to limit unheeded warnings, so as to prevent congestion. (A variant of this scheme was first proposed by Charles Lynn [11].)

3.4. Use of Transparent Fragmentation

The need for inter-network fragmentation, and consequently its dangers, can be reduced or eliminated by the use of transparent fragmentation (sometimes called intra-network fragmentation). If all the fragments of a datagram are sent to a unique next-hop gateway for reassembly, and if the fragmenting and reassembling gateways use a low-level protocol that increases the chances of complete delivery, two benefits are obtained:

1. Deterministic fragment loss is unlikely, if the protocol between the two gateways supports acknowledgments of individual fragments. It need not be completely reliable, since the end hosts are willing to accept occasional lost or mis-sequenced datagrams.
2. If the (reassembled) datagram subsequently traverses a network with a larger MTU, it makes more efficient use of that network than a collection of smaller fragments.

On the other hand, transparent fragmentation has many drawbacks: (1) a datagram may be repeatedly reassembled and refragmented; (2) gateway implementations become more complex and require much more buffer memory; (3) the performance gains are limited because a datagram cannot be larger than the MTU of the first-hop network and because a maximum size must be enforced to provide a limit on gateway reassembly buffer space. Most important in the IP world a destination host may still have to perform reassembly, since the MTU of the last hop link may be smaller than the datagram size; this means that most of the problems associated with internetwork fragmentation would still be present, although to a lesser degree. (Transparent fragmentation was successful in the Pup architecture because almost all non-gateway Pup hosts were attached to networks with MTUs larger than the maximum allowed Pup datagram size [2].)

Since transparent fragmentation is invisible except to the gateways involved, it can be used in an IP internet wherever the benefits outweigh the drawbacks; this is specifically allowed by the IP specification [18]. We encourage designers of gateways and networks to consider the use of transparent fragmentation, especially if the natural MTU of the network is unusually small. For example, the actual IMP-to-IMP messages on the ARPAnet are only 1008 bits long; the 1007-byte MTU of the ARPAnet is an illusion created by transparent fragmentation and reassembly within the IMPs [8].

3.5. Careful use of intentional fragmentation

In certain restricted cases, and with a little luck, one can obtain significant performance improvements by sending such large datagrams that they must be fragmented immediately by the source host, before being transmitted on their first hop. This is done in Sun Microsystems' implementation of their NFS protocol [21]. Throughput is improved because fewer datagrams are handled in the layers above IP, and because end-to-end acknowledgements are done only in the RPC layer, rather than in the transport layer as well. This performance improvement is extremely unstable; because it is vulnerable to deterministic fragment loss, performance may drop radically if gateways or interfaces with insufficient buffering are in use. In some cases, the protocol fails entirely.

We do not believe that intentional fragmentation is a good idea, since careful protocol design and implementation should be able to provide similar peak performance without anywhere near the risk. It is especially irresponsible to use over an internet prone to congestion, since congestion may cause deterministic fragment loss and the resulting retransmission of long packet trains can only worsen the congestion.

Intentional fragmentation, in spite of its risk, is appealing because as long as it works it requires no implementation modifications beyond some parameter tuning. If it is used, we think the risk can be reduced by approaches analogous to those we have suggested for determining MINMTU.

For example, intentional fragmentation could be restricted to those destinations that are on the local network, or those without an intervening gateway. Alternatively, the source host could observe the retransmission rate and cease intentional fragmentation if the rate is high; since intentional fragmentation is worse for congestion than the use of undersized datagrams, this is a good idea even when one cannot distinguish retransmissions caused by congestion from those caused by deterministic fragment loss.

If accurate information, say from a probe mechanism, shows that somewhere along the path the fragments will be re-fragmented, one would clearly not want to use intentional fragmentation. Aside from the problem that many existing gateways derived from 4.2BSD code cannot properly fragment a fragment, this results in a great expansion in the total number of fragments and consequently the risks of congestion and deterministic fragment loss.

4. Summary and Recommendations

We believe that future heterogeneous internetworks will include networks with a wide range of bandwidths, because economics force long-haul networks to have lower bandwidths than local-area networks. The MTU that maximizes performance varies with bandwidth on low-bandwidth links, a small MTU is used to limit the time the link can be occupied by one packet; on high-bandwidth links, a large MTU allows per-packet overheads to be amortized over many bytes. Therefore, link-level MTUs will always vary within heterogeneous internetworks.

In this paper, we have explored the use of internetwork fragmentation as a solution to the problem of differing MTUs. Fragmentation frees higher level protocols from having to alter their behavior based on the route over which packets flow. To the designers of IP, internetwork fragmentation appeared to be the right choice. Unfortunately, as we have shown, blind reliance on fragmentation in IP can be costly in both performance and reliability.

4.1. Recommendations

In section 3 we described a broad variety of schemes for avoiding or ameliorating fragmentation. Not all of these schemes are worthwhile, and not all of the good ones should be adopted together. Here we suggest what we believe are the most appropriate steps to take.

We are proposing engineering modifications to a large, heterogeneous internetwork where there is a tremendous delay in disseminating change to all sites; some sites may never catch up. Stability is important; this means that whenever possible, a change should not disrupt those hosts that are not yet updated.

Since a robust host implementation should simply ignore packets and options that it does not understand, one might think it safe to make changes that involve sending additional packets on the off chance that the receiver knows what to do with them. In the larger context of an internetwork prone to congestion, however, we should worry about injecting useless packets. We thus favor approaches that do not repeatedly send packets that might be ignored.

4.1.1. Recommendations not involving protocol changes

Some solutions can be implemented immediately, without changes to protocol specifications. Most effective is the one described in section 3.1.2, limiting the datagram size to 576 bytes whenever the packet is routed via a gateway. This should be implemented in the IP layer, rather than in the TCP layer as in 4.3BSD.

Somewhat more difficult, but still possible without protocol changes at the IP layer or above, is the use of transparent fragmentation (that is, immediate reassembly) over networks with small MTUs. Effectively, this means increasing the MTU of such networks as viewed by hosts on other networks.

Finally, we strongly encourage implementors who use intentional fragmentation to do so only when packets are sent directly, with no gateway along the route. Also, intentional fragmentation should cease when the retransmission rate increases beyond a certain level.

We do not recommend the use of the ICMP “fragment reassembly time exceeded” message since it appears that most hosts simply ignore it; this is a shame but it may be too late to correct.

4.1.2. Recommendations for protocol changes

If protocol changes were to be considered, we would recommend the adoption of both the ICMP “Probe Path” message described in appendix I, and the IP “Probe MTU” option described in appendix II. Both of these changes require support from gateways and from host IP layers, but they can be incrementally adopted without confusing existing implementations. On the other hand, the ICMP “Probe Path” message can still cause the “useless packets” problem.

We suggest implementing both probe mechanisms because the cost of doing so is not much higher than that of implementing one, and we cannot predict which is more effective. Each may be optimal for certain applications.

4.1.3. Recommendations for new architectures

The IP Internet was intended as an experiment, and from this experiment we can extract some lessons to use in designing new architectures. Any large heterogeneous datagram internetwork is likely to require fragmentation at times; careful design can make fragmentation normally unnecessary, and can avert its most serious drawbacks. We do not believe that fragmentation should be completely hidden from hosts; to do so would be to fall into the trap of providing reliable stream protocols at too low a level. Rather, fragmentation should simply be as robust as possible, so that it does not lead to performance disasters.

We urge consideration of transparent fragmentation whenever possible. There is little value in the ability to send fragments of one datagram along different routes, and reassembly by gateways should not be prohibitively expensive. Main memory sizes and costs are improving so rapidly that buffer space should no longer be considered the limiting resource; reassembly might actually improve the switching rates of gateways by reducing the number of individually switched fragments. We suggest that a source host be able to turn off transparent fragmentation by setting a flag in the datagram header, analogous to the IP “Don't Fragment” flag.

We also believe that the ability to record path information — not only about MTU but also about congestion, bandwidth, etc. — is so valuable that it should be done on every packet. The header overhead could be reduced by encoding the numerical information, either by reducing resolution or by using a logarithmic scale. By doing recording in standard rather than optional header fields, its cost could be made negligible.

Finally, since transparent fragmentation cannot entirely obviate the use of inter-network fragmentation, there must be a way to recover from inappropriate fragmentation. The receiving host can detect the problem, through repeated reassembly timeouts, and should notify the sending host via something akin to the ICMP “fragmentation reassembly time exceeded” message. If support for this message had been mandatory in IP, it would have eliminated the problem of complete communication failure due to deterministic fragment loss.

Acknowledgements

This paper was inspired by a flurry of messages on the TCP-IP electronic mailing list, which provoked us to write down some thoughts we had been harboring for several years. Many of the ideas in this paper come from TCP-IP participants; we have tried to cite specific references. Other participants whose important contributions are not otherwise cited include Bob Braden, Hans-Wemer Braun, Vint Cerf, Doug Kingston, John Wobus, and Lixia Zhang. We would also like to thank Dave Boggs for continually reminding us that Pup got it right.

Appendix I. ICMP "Probe Path" message

The format of the proposed ICMP "Probe Path" message is shown in figure 1-1.

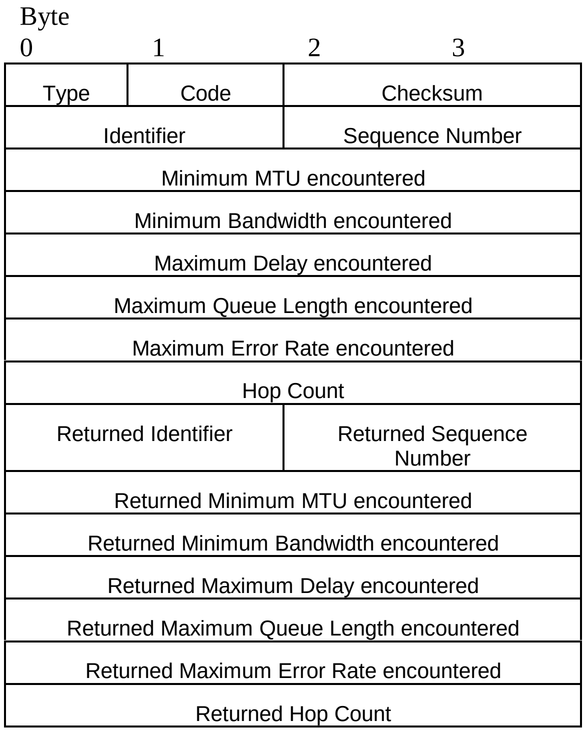
Byte
0
1
2
3
Type
Code
Checksum
Identifier
Sequence Number
Minimum MTU encountered
Minimum Bandwidth encountered
Maximum Delay encountered
Maximum Queue Length encountered
Maximum Error Rate encountered
Hop Count
Returned Identifier
Returned Sequence Number
Returned Minimum MTU encountered
Returned Minimum Bandwidth encountered
Returned Maximum Delay encountered
Returned Maximum Queue Length encountered
Returned Maximum Error Rate encountered
Returned Hop Count

Figure I-1 Format of ICMP "Probe Path" message

The fields of a Probe Path message are:

Type
To be assigned.

Code
Indicates how far through a "three-way" handshake this message is:

1 = Initial message don't believe "Returned" values; please reply.

2 = Second message believe "Returned" values; please reply.

3 = Third message believe "Returned" values; no reply expected.

For codes I and 2, gateways must update values in other fields as specified. For code 3, gateways need not update values in other fields.

Checksum
The usual ICMP checksum. It must be updated when a gateway modifies any of the other fields.

Identifier
An identifier to aid in matching probes with replies, may be zero.

Sequence Number
A sequence number to aid in matching probes with replies, may be zero.

If the Code is 1 or 2, the following six fields are updated by gateways as the message follows its route; they are all 32-bit two's complement (signed) integers. Their initial values are set by the source host, as indicated. Gateways should observe the Type of Service field and Security option in the IP header of the Probe Path message when updating these fields.

Minimum MTU encountered
Initially set to the MTU of the first hop data link, or to MAXINT (2^{3l}-l). Each gateway compares this value to the MTU of the incoming and outgoing links for the message, and reduces the recorded value, if necessary. MTU is measured in octets.

Minimum Bandwidth encountered
Initially set to the bandwidth of the first hop data link, or to MAXINT. Each gateway compares this value to the bandwidth of the incoming and outgoing links for the message, and reduces the recorded value, if necessary. Bandwidth is measured in bits per second.

Maximum Delay encountered
Initially set to the delay of the first hop data link, or to zero. Each gateway compares this value to the time it will take the packet to traverse the incoming and outgoing links, and increases the recorded value, if necessary. Delay is measured in microseconds. For networks, such as CSMA/CD, where the delay is not a simple function of the packet length, use the expected value of the delay for an average packet.

Maximum Queue Length encountered
Initially set to the length of the output queue of the source host when the packet is placed on that queue, or to zero. Each gateway compares this value to the length of the queue in which the packet is placed, and increases the recorded value, if necessary.

Maximum Error Rate encountered
Initially set to the error rate of the first hop data link,

or to zero. Each gateway compares this value to the error rate of the incoming and outgoing links, and increases the recorded value, if necessary. Error rate is measured as the reciprocal of the bit-error rate; i.e., it is the expected value of the number of bits between errors.

Hop Count
Initially set to 1. Each gateway increments this value by one.

If the Code of an incoming message is 1 or 2, the following eight ("returned") fields are used to return the previous eight ("incoming") fields to the probing host:

Returned Identifier
Returned Sequence Number
Returned Minimum MTU encountered
Returned Minimum Bandwidth encountered
Returned Maximum Delay encountered
Returned Maximum Queue Length encountered
Returned Maximum Error Rate encountered
Returned Hop Count

The destination host copies the appropriate fields (e.g., "Minimum MTU encountered" is copied to "Returned Minimum MTU encountered"), reinitializes the incoming fields, sets the code field to 2 or 3 depending on whether it needs path information, recomputes the checksum, reverses the source and destination addresses, and returns the message. When a host receives a Code 2 or Code 3 message, it can use the values in the returned fields to update its path information database. The 32-bit wide fields are interpreted as two's complement integers; a negative value means that the field value is not valid.

Appendix II. IP "Probe MTU" options

The format of the proposed IP "Probe MTU" option is shown in figure II-1. The option type code *is to be assigned*; it is not copied on fragmentation, it is of option class 2 (debugging and measurement). This option is always 8 octets long.

010xxxx0	Length	Minimum MTU encountered	Identifier
Type=yyy	Length=8		

Figure II-1 Format of IP "Probe MTU" option

The value of the Probe MTU option is the minimum MTU encountered along the route followed by the packet, measured in octets. It is initialized by the source host to the MTU of the first hop data-link, or it may be initialized to 2^{31}-l. Each gateway compares this value to the MTU of the incoming and outgoing links for the packet, and reduces the recorded value, if necessary. Gateways should observe the Type of Service field and Security option in the IP header when updating this value. The source host should set the Identifier field to allow it to match the reply to this option with the appropriate connection or route.

When a destination host receives a packet with the Probe MTU option, it creates an "MTU Reply" option, whose format is shown in figure II-2. The option type code is *to be assigned*; it is not copied on fragmentation, it is of option class 2 (debugging and measurement). This option is always 8 octets long. The "Returned Minimum MTU" and "Identifier" fields are copies of the corresponding fields in the received Probe MTU option.

010xxxx1	Length	Returned Minimum MTU	Identifier
Type=zzz	Length=8		

Figure 11-2 Format of IP "MTU Reply" option

The destination host returns this option attached to the next packet sent to the originating host. Because several Probe MTU options may arrive before one is sent, the MTU Reply option may appear more than once in a packet. A packet carrying an MTU Reply option may also carry a Probe MTU option.

When a host receives an MTU Reply option, it uses the Identifier field to associate the Minimum MTU value with a particular connection or destination.

References

1. Art Berggreen. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, no MessageID.

2. David R. Boggs, John F. Shoch, Edward A. Taft, and Robert M. Metcalfe. "Pup An intemetwork architecture." *IEEE Transactions on Communications COM-28*, 4 (April 1980), 612-624.

3. David R. Cheriton. VMTP: A transport protocol for next generation communication systems. SIGCOMM '86, ACM SIGCOMM, August, 1986.

4. J. Noel Chiappa. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessageID <12304502322.33.JNC@XX.LCS.MlT.EDU>.

5. David D. Clark. IP Datagram Reassembly Algorithms. RFC 815, Network Information Center, SRI International, July, 1982.

6. David D. Clark, Mark L. Lambert, and Lixia Zhang. NETBLT A Bulk Data Transfer Protocol. RFC 998, Network Information Center, SRI International, March, 1987.

7. Geof Cooper. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessagelD <8705230517.AA01407@apolling.imagen.uucp>.

8. F. E. Heart, R. E. Kahn. S. M. Ornstein, W. R. Crowther, and D. C. Walden. The interface message processor for the ARPA computer network Proc. AFIPS Spring Joint Computer Conference, May, 1970, pp. 551-567.

9. Van Jacobson. Retransmit Timers Theory and Practice. In preparation.

10. Mike Karels. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessagelD <8705262316.AA08021@okeefe.Berkeley.EDU>.

11. Charles Lynn. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessagelD <[G.BBN.COM]22-May-87 20:ll:55.CLYNN>.

12. M. Kirk McKusick, Mike Karels, and Sam Leffler. Performance Improvements and Functional Enhancements in 4.3BSD. Proc. Summer USENIX Conference, June, 1985, pp. 519-531.

13. Dave Mills. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessagelD <8705261315.aO29301@Huey.UDEL.EDU>.

14. Jeffrey Mogul. Internet Subnets. RFC 917, Network Information Center, SRI International, October, 1984.

15. Jeffrey Mogul and Jon Postel. Internet Standard Subnetting Procedure. RFC 950, Network Information Center, SRI International, August, 1985.

16. John B. Nagle. "On Packet Switches with Infinite Storage.” *IEEE Transactions on Communications COM-35*, 4 (April 1987), 435–438.

17. Jon Postel. User Datagram Protocol. RFC 768, Network Information Center, SRI International, August, 1980.

18. Jon Postel. Internet Protocol. RFC 791, Network Information Center, SRI International, September, 1981.

19. Jon Postel. Transmission Control Protocol. RFC 793, Network Information Center, SRI International, September, 1981.

20. Jon Postel. Internet Control Message Protocol. RFC 792, Network Information Center, SRI International, September, 1981.

21. Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. Proc. Summer USENIX Conference, June, 1985, pp. 119-130.

22. Alan Sheltzer, Robert Hinden, and Mike Brescia. “Connecting different types of networks with gateways.” Data Communications (August 1982), 119126.

23. John Shoch. “Packet Fragmentation in Inter-network Protocols.” *Computer Networks 3*, 1 (February 1979), 3-8.

24. John A. Shriver. IP Datagram Sizes. Electronic distribution of the TCP-IP Discussion Group, MessageID < 8705261623.AA12946@monk.proteon.com>.