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 informa-

tion 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 $L_1L_2L_3L_4L_5L_6L_7S_1L_8S_2$, reassembly of L cannot succeed, despite adequate buffer space. Upon reception of S_1, the reassembly process could discard L_1 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 MINMTUs, 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 MINMTUs, 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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.