GATE CSE NOTES

by

Joyoshish Saha

Collision domain

A collision domain is, as the name implies, a part of a network where packet collisions can occur. A collision occurs when two devices send a packet at the same time on the shared network segment. The packets collide and both devices must send the packets again, which reduces network efficiency. Collisions are often in a hub environment, because each port on a hub is in the same collision domain. By contrast, each port on a bridge, a switch or a router is in a separate collision domain.

It is an area in a network where two ethernet frames can collide on one link between two devices. If one emits and if the other emits data at the same time, it creates the collision.

It only happened in early version of ethernets when devices where half duplex, way back! At that time the switch were called HUB. If i remember well, they used to repeat the electric signal & forward it to all other ports. They didn't have an ARP table with the list of MAC addresses from every devices connected to the hub.

Hubs were Layer 1 devices.

It was really slow because if you had 4 computers connected to the hub, only one could emit at a time.

half duplex devices can only receive or emit but not receive & emit at the same time. Our actual devices switch, routers, Wifi card, lan cards are all full duplex, meaning they can receive while they emit.

The following example illustrates collision domains.
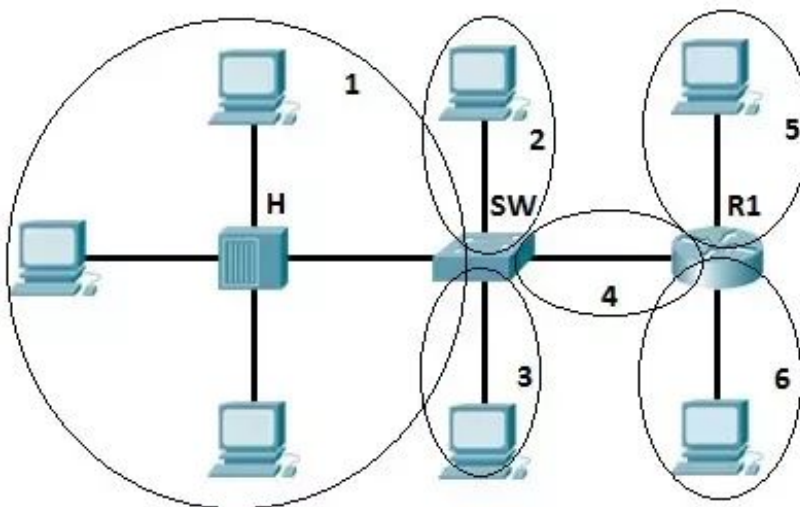
As you can see, we have 6 collision domains.

TIP – remember, each port on a hub is in the same collision domain. Each port on a bridge, a switch or router is in a separate collision domain.

Broadcast domain

A broadcast domain is a domain in which a broadcast is forwarded. A broadcast domain contains all devices that can reach each other at the data link layer (OSI layer 2) by using broadcast. All ports on a hub or a switch are by default in the same broadcast domain. All ports on a router are in the different broadcast domains and routers don't forward broadcasts from one broadcast domain to another.

Sometimes devices send ethernet frames to a mac address destination: FF:FF:FF:FF:FF:FF, it means that it is for all devices that are directly connected to the emitting device. FF:FF:FF:FF:FF:FF is what i call the broadcast mac address. All the devices who receive it are connected by a SWITCH or switches.

Nevertheless, a broadcast domain stops at the router because your router job is to remove the header of the frame including the destination MAC address with another destination MAC address to send it to another broadcast domain (another network which can be a public network).

Broadcast domain is on layer 2 (data link).

The broadcast domain is relevant for almost ALL of us. When your computer connects to the network for the first time & when it is a DHCP client, it will send an ethernet frame with FF:FF:FF:FF:FF:FF because it does not know the IP of the router. So it sends a packet called DHCP discover to all the devices on the network because it is looking for a DHCP server in order to get an IP, DNS server IP, router IP. Only, The DHCP server will catch the request and send an IP address in return

The following example clarifies the concept.
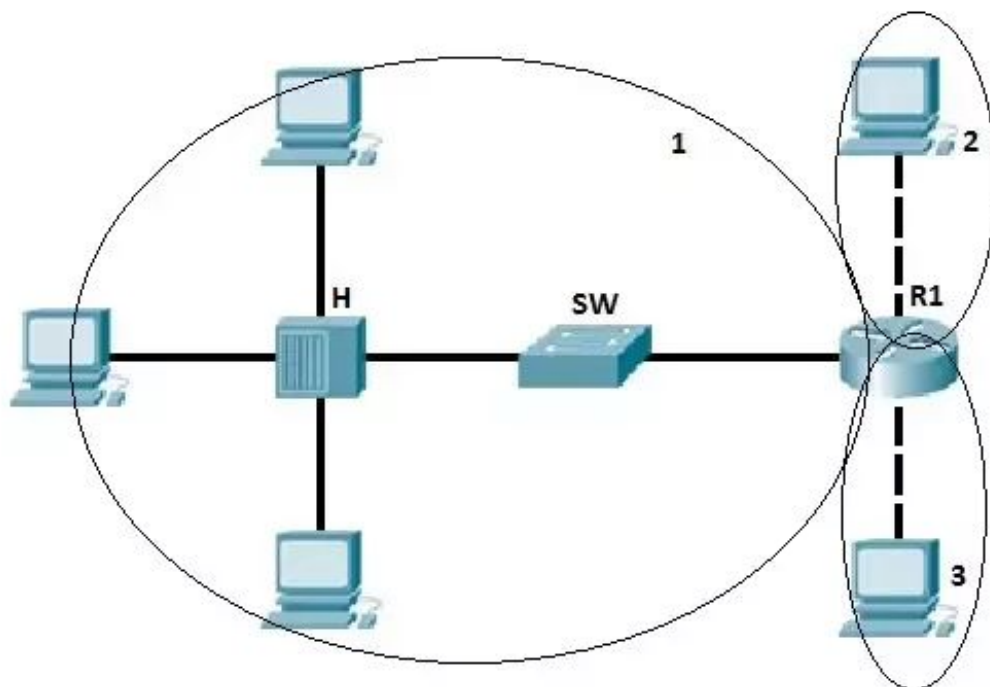
In the picture above we have three broadcast domains, since all ports on a hub or a switch are in the same broadcast domain, and all ports on a router are in a different broadcast domain.

Important Points

Hub has a single broadcast domain and collision Domain
By default, each port of switch carries a single broadcast domain and per port collision Domain.
Router has per port Collision and Broadcast Domain.

Resolve IPv4 Fragmentation, MTU, MSS, and PMTUD Issues with GRE and IPsec

Updated: January 8, 2019   **Document ID:** 25885

Contents

Introduction

This document describes how IPv4 Fragmentation and Path Maximum Transmission Unit Discovery (PMTUD) work and also discusses some scenarios that involves the behavior of PMTUD when combined with different combinations of IPv4 tunnels. The current widespread use of IPv4 tunnels in the Internet has brought the problems that involve IPv4 Fragmentation and PMTUD to the forefront.

IPv4 Fragmentation and Reassembly

The IPv4 protocol was designed for use on a wide variety of transmission links. Although the maximum length of an IPv4 datagram is 65535, most transmission links enforce a smaller maximum packet length limit, called an MTU. The value of the MTU depends on the type of the transmission link. The design of IPv4 accommodates MTU differences since it allows routers to fragment IPv4 datagrams as necessary. The receiving station is responsible for the reassembly of the fragments back into the original full size IPv4 datagram.

IPv4 fragmentation involves to break a datagram into a number of pieces that can be reassembled later. The IPv4 source, destination, identification, total length, and fragment offset fields, along with the "more fragments" and "don't fragment" flags in the IPv4 header, are used for IPv4 fragmentation and reassembly. For more information about the mechanics of IPv4 fragmentation and reassembly, see RFC 791.

This image depicts the layout of an IPv4 header.
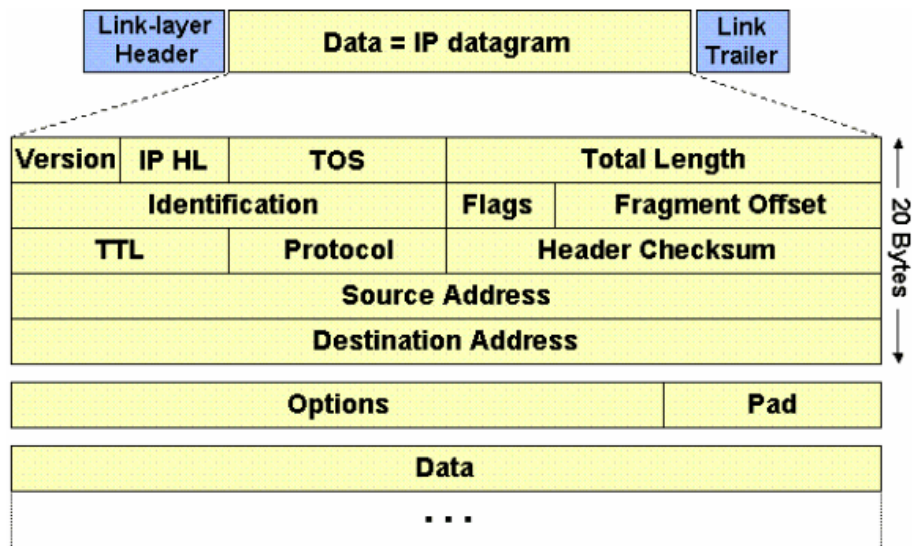
The identification is 16 bits and is a value assigned by the sender of an IPv4 datagram in order to aid in the reassembly of the fragments of a datagram.

The fragment offset is 13 bits and indicates where a fragment belongs in the original IPv4 datagram. This value is a multiple of eight bytes.

In the flags field of the IPv4 header, there are three bits for control flags. It is important to note that the "don't fragment" (DF) bit plays a central role in PMTUD because it determines whether or not a packet is allowed to be fragmented.

Bit 0 is reserved, and is always set to 0. Bit 1 is the DF bit (0 = "may fragment", 1 = "do not fragment"). Bit 2 is the MF bit (0 = "last fragment," 1 = "more fragments").

Value	Bit 0 Reserved	Bit 1 DF	Bit 2 MF
0	0	May	Last
1	0	Do not	More

The next graphic shows an example of fragmentation. If you add up all the lengths of the IPv4 fragments, the value exceeds the original IPv4 datagram length by 60. The reason that the overall length is increased by 60 is because three additional IPv4 headers were created, one for each fragment after the first fragment.

The first fragment has an offset of 0, the length of this fragment is 1500; this includes 20 bytes for the slightly modified original IPv4 header.

The second fragment has an offset of 185 (185 x 8 = 1480), which means that the data portion of this fragment starts 1480 bytes into the original IPv4 datagram. The length of this fragment is 1500; this includes the additional IPv4 header created for this fragment.

The third fragment has an offset of 370 (370 x 8 = 2960), which means that the data portion of this fragment starts 2960 bytes into the original IPv4 datagram. The length of this fragment is 1500; this includes the additional IPv4 header created for this fragment.

The fourth fragment has an offset of 555 (555 x 8 = 4440), which means that the data portion of this fragment starts 4440 bytes into the original IPv4 datagram. The length of this fragment is 700 bytes; this includes the additional IPv4 header created for this fragment.

It is only when the last fragment is received that the size of the original IPv4 datagram can be determined.

The fragment offset in the last fragment (555) gives a data offset of 4440 bytes into the original IPv4 datagram. If you then add the data bytes from the last fragment (680 = 700 – 20), that gives you 5120 bytes, which is the data portion of the original IPv4 datagram. Then, addition of 20 bytes for an IPv4 header equals the size of the original IPv4 datagram (4440 + 680 + 20 = 5140) as shown in the images.

Original IP Datagram

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0	345	5140	0	0	0

IP Fragments (Ethernet)

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0-0	345	1500	0	1	0
0-1	345	1500	0	1	185
0-2	345	1500	0	1	370
0-3	345	700	0	0	555

Issues with IPv4 Fragmentation

There are several issues that make IPv4 fragmentation undesirable. There is a small increase in CPU and memory overhead in order to fragment an IPv4 datagram. This holds true for the sender as well as for a router in the path between a sender and a receiver. The creation of fragments simply involves to create fragment headers and copy the original datagram into the fragments. This can be done fairly efficiently because all the information needed in order to create the fragments is immediately available.

Fragmentation causes more overhead for the receiver when reassembling the fragments because the receiver must allocate memory for the arriving fragments and coalesce them back into one datagram after all of the fragments are received. Reassembly on a host is not considered a problem because the host has the time and memory resources in order to devote to this task.

But, reassembly is very inefficient on a router whose primary job is to forward packets as quickly as possible. A router is not designed to hold on to packets for any length of time. Also, a router that does reassembly chooses the largest buffer available (18K) with which to work because it has no way to know the size of the original IPv4 packet until the last fragment is received.

Another fragmentation issue involves how dropped fragments are handled. If one fragment of an IPv4 datagram is dropped, then the entire original IPv4 datagram must be resent, and it is also fragmented. You see an example of this with Network File System (NFS). NFS, by default, has a read and write block size of 8192, so a NFS IPv4/UDP datagram is approximately 8500 bytes (which includes NFS, UDP, and IPv4 headers). A sending station connected to an Ethernet (MTU 1500) has to fragment the 8500 byte datagram into six pieces; five 1500 byte fragments and one 1100 byte fragment. If any of the six fragments are dropped because of a congested link, the complete original datagram has to be retransmitted, which means that six more fragments will have to be created. If this link drops one in six packets, then the odds are low that any NFS data can be transferred over this link, since at least one IPv4 fragment would be dropped from each NFS 8500 byte original IPv4 datagram.

Firewalls that filter or manipulate packets based on Layer 4 (L4) through Layer 7 (L7) information in the packet might have trouble processing IPv4 fragments correctly. If the IPv4 fragments are out of order, a firewall might block the non-initial fragments because they do not carry the information that would match the packet filter. This would mean that the original IPv4 datagram could not be reassembled by the receiving host. If the firewall is configured to allow non-initial fragments with insufficient information to properly match the filter, then a non-initial fragment attack through the firewall could occur. Also, some network devices (such as Content Switch Engines) direct packets based on L4 through L7 information, and if a packet spans multiple fragments, then the device might have trouble enforcing its policies.

## Avoid IPv4 Fragmentation: What TCP MSS Does and How It Works

The TCP Maximum Segment Size (MSS) defines the maximum amount of data that a host is willing to accept in a single TCP/IPv4 datagram. This TCP/IPv4 datagram might be fragmented at the IPv4 layer. The MSS value is sent as a TCP header option only in TCP SYN segments. Each side of a TCP connection reports its MSS value to the other side. Contrary to popular belief, the MSS value is not negotiated between hosts. The sending host is required to limit the size of data in a single TCP segment to a value less than or equal to the MSS reported by the receiving host.

Originally, MSS meant how big a buffer (greater than or equal to 65496 bytes) was allocated on a receiving station to be able to store the TCP data contained within a single IPv4 datagram. MSS was the maximum segment (chunk) of data that the TCP receiver was willing to accept. This TCP segment could be as large as 64K (the maximum IPv4 datagram size) and it could be fragmented at the IPv4 layer in order to be transmitted across the network to the receiving host. The receiving host would reassemble the IPv4 datagram before it handed the complete TCP segment to the TCP layer.

Here are a couple of scenarios that show how MSS values are set and used to limit TCP segment sizes, and therefore, IPv4 datagram sizes.

Scenario 1 illustrates the way MSS was first implemented. Host A has a buffer of 16K and Host B a buffer of 8K. They send and receive their MSS values and adjust their send MSS for sending data to each other. Notice that Host A and Host B will have to fragment the IPv4 datagrams that are larger than the interface MTU, but still less than the send MSS because the TCP stack could pass 16K or 8K bytes of data down the stack to IPv4. In Host B's case, packets could be fragmented twice, once to get onto the Token Ring LAN and again to get onto the Ethernet LAN.

Scenario 1
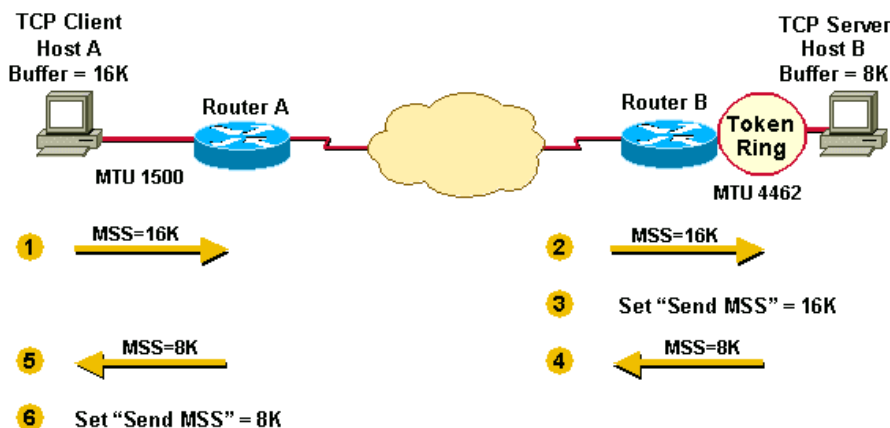
1. Host A sends its MSS value of 16K to Host B.
2. Host B receives the 16K MSS value from Host A.
3. Host B sets its send MSS value to 16K.
4. Host B sends its MSS value of 8K to Host A.
5. Host A receives the 8K MSS value from Host B.
6. Host A sets its send MSS value to 8K.

In order to assist in avoiding IPv4 fragmentation at the endpoints of the TCP connection, the selection of the MSS value was changed to the minimum buffer size and the MTU of the outgoing interface (– 40). MSS numbers are 40 bytes smaller than MTU numbers because MSS is just the TCP data size, which does not include the 20 byte IPv4 header and the 20 byte TCP header. MSS is based on default header sizes; the sender stack must subtract the appropriate values for the IPv4 header and the TCP header dependent on what TCP or IPv4 options are used.

The way MSS now works is that each host will first compare its outgoing interface MTU with its own buffer and choose the lowest value as the MSS to send. The hosts will then compare the MSS size received against their own interface MTU and again choose the lower of the two values.

Scenario 2 illustrates this additional step taken by the sender in order to avoid fragmentation on the local and remote wires. Notice how the MTU of the outgoing interface is taken into account by each host (before the hosts send each other their MSS values) and how this helps to avoid fragmentation.

Scenario 2
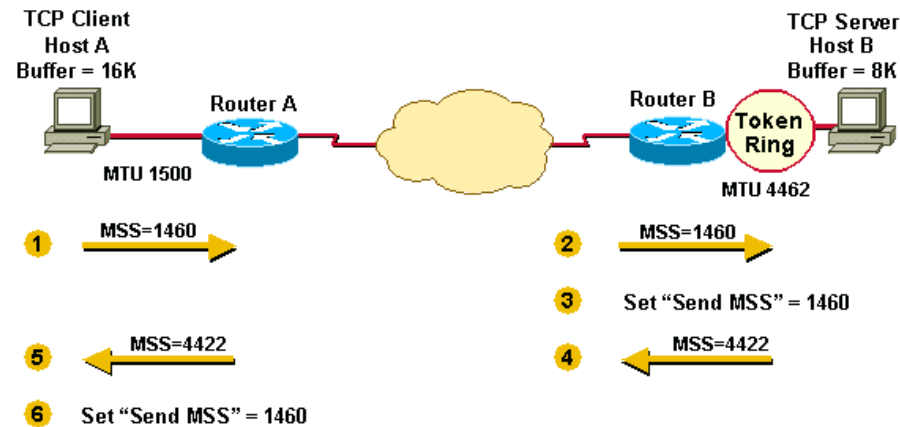
1. Host A compares its MSS buffer (16K) and its MTU (1500 – 40 = 1460) and uses the lower value as the MSS (1460) to send to Host B.
2. Host B receives Host A's send MSS (1460) and compares it to the value of its outbound interface MTU – 40 (4422).
3. Host B sets the lower value (1460) as the MSS in order to send IPv4 datagrams to Host A.
4. Host B compares its MSS buffer (8K) and its MTU (4462–40 = 4422) and uses 4422 as the MSS to send to Host A.
5. Host A receives Host B's send MSS (4422) and compares it to the value of its outbound interface MTU –40 (1460).
6. Host A sets the lower value (1460) as the MSS for sending IPv4 datagrams to Host B.

1460 is the value chosen by both hosts as the send MSS for each other. Often the send MSS value will be the same on each end of a TCP connection.

In Scenario 2, fragmentation does not occur at the endpoints of a TCP connection because both outgoing interface MTUs are taken into account by the hosts. Packets can still become fragmented in the network between Router A and Router B if they encounter a link with a lower MTU than that of either hosts' outbound interface.

What is PMTUD?

TCP MSS as described earlier takes care of fragmentation at the two endpoints of a TCP connection, but it does not handle the case where there is a smaller MTU link in the middle between these two endpoints. PMTUD was developed in order to avoid fragmentation in the path between the endpoints. It is used to dynamically determine the lowest MTU along the path from a packet's source to its destination.

✎ **Note**: PMTUD is only supported by TCP and UDP. Other protocols do not support it. If PMTUD is enabled on a host, and it almost always is, all TCP and UDP packets from the host will have the DF bit set.

When a host sends a full MSS data packet with the DF bit set, PMTUD reduces the send MSS value for the connection if it receives information that the packet would require fragmentation. A host usually "remembers" the MTU value for a destination since it creates a "host" (/32) entry in its routing table with this MTU value.

If a router tries to forward an IPv4 datagram, with the DF bit set, onto a link that has a lower MTU than the size of the packet, the router drops the packet and return an Internet Control Message Protocol (ICMP) "Destination Unreachable" message to the source of this IPv4 datagram, with the code that indicates "fragmentation needed and DF set" (type 3, code 4). When the source station receives the ICMP message, it will lower the send MSS, and when TCP retransmits the segment, it will use the smaller segment size.

Here is an example of an ICMP "fragmentation needed and DF set" message that you might see on a router after the **debug ip icmp** command is turned on:

```
ICMP: dst (10.10.10.10) frag. needed and DF set
unreachable sent to 10.1.1.1
```

This diagram shows the format of ICMP header of a "fragmentation needed and DF set" "Destination Unreachable" message.

IP header	ICMP datagram

Type = 3	Code = 4	Checksum	Reserved	Next Hop MTU
1	1	2	4	

Data = IP header + 8 bytes of packet causing error

Data
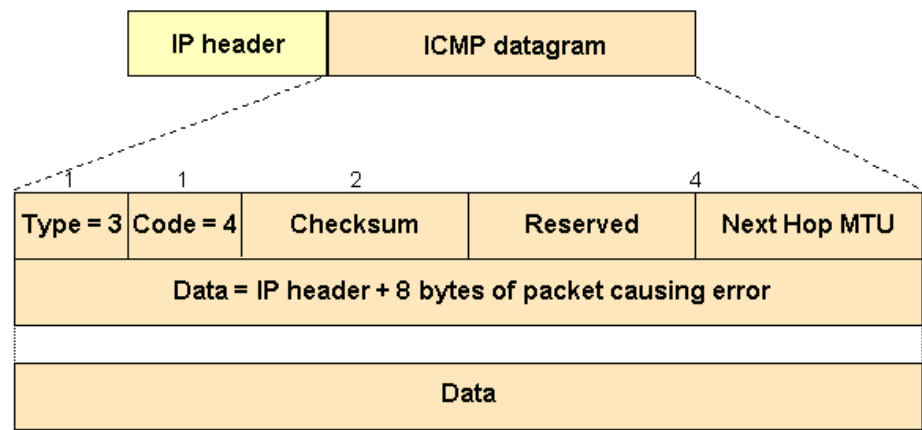
Per RFC 1191, a router that returns an ICMP message which indicates "fragmentation needed and DF set" must include the MTU of that next-hop network in the low-order 16 bits of the ICMP additional header field that is labeled "unused" in the ICMP specification RFC 792.

Early implementations of RFC 1191 did not supply the next hop MTU information. Even when this information was supplied, some hosts ignore it. For this case, RFC 1191 also contains a table that lists the suggested values by which the MTU should be lowered during PMTUD. It is used by hosts in order to arrive more quickly at a reasonable value for the send MSS and as shown in the image.

```
Plateau     MTU    Comments                       Reference
------      ---    --------                       ---------
            65535  Official maximum MTU           RFC 791
            65535  Hyperchannel                   RFC 1044
65535
32000              Just in case
            17914  16Mb IBM Token Ring            ref. [6]
17914
            8166   IEEE 802.4                     RFC 1042
8166
            4464   IEEE 802.5 (4Mb max)           RFC 1042
            4352   FDDI (Revised)                 RFC 1188
4352 (1%)
            2048   Wideband Network               RFC 907
            2002   IEEE 802.5 (4Mb recommended)   RFC 1042
2002 (2%)
            1536   Exp. Ethernet Nets             RFC 895
            1500   Ethernet Networks              RFC 894
            1500   Point-to-Point (default)       RFC 1134
            1492   IEEE 802.3                     RFC 1042
1492 (3%)
            1006   SLIP                           RFC 1055
            1006   ARPANET                        BBN 1822
1006
            576    X.25 Networks                  RFC 877
            544    DEC IP Portal                  ref. [10]
            512    NETBIOS                        RFC 1088
            508    IEEE 802/Source-Rt Bridge      RFC 1042
            508    ARCNET                         RFC 1051
508 (13%)
            296    Point-to-Point (low delay)     RFC 1144
296
68                 Official minimum MTU           RFC 791
```

PMTUD is done continually on all packets because the path between sender and receiver can change dynamically. Each time a sender receives a "Can't Fragment" ICMP messages it will update the routing information (where it stores the PMTUD).

Two possible things can happen during PMTUD:

1. The packet can get all the way to the receiver without being fragmented.

✎ **Note**: In order for a router to protect the CPU against DoS attacks, it throttles the number of ICMP unreachable messages that it would send, to two per second. Therefore, in this context, if you have a network scenario in which you expect that the router would need to respond with more than two ICMP messages (type = 3, code = 4) per second (can be different hosts), you would want to disable the throttling of ICMP messages with the **no ip icmp rate-limit unreachable [df] interface** command.

---

2. The sender can get ICMP "Can't Fragment" messages from any (or every) hop along the path to the receiver.

PMTUD is done independently for both directions of a TCP flow. There might be cases where PMTUD in one direction of a flow triggers one of the end stations to lower the send MSS and the other end station keeps the original send MSS because it never sent an IPv4 datagram large enough to trigger PMTUD.

A good example of this is the HTTP connection depicted below in Scenario 3. The TCP client sends small packets and the server sends large packets. In this case, only the server's large packets (greater than 576 bytes) will trigger PMTUD. The client's packets are small (less than 576 bytes) and will not trigger PMTUD because they do not require fragmentation to get across the 576 MTU link.

Scenario 3

Scenario 4 shows an asymmetric routing example where one of the paths has a smaller minimum MTU than the other. Asymmetric routing occurs when different paths are taken to send and receive data between two endpoints. In this scenario, PMTUD will trigger the lowering of the send MSS only in one direction of a TCP flow. The traffic from the TCP client to the server flows through Router A and Router B, whereas the return traffic that comes from the server to the client flows through Router D and Router C. When the TCP server sends packets to the client, PMTUD will trigger the server to lower the send MSS because Router D must fragment the 4092 byte packets before it can send them to Router C.

The client, on the other hand, will never receive an ICMP "Destination Unreachable" message with the code that indicates "fragmentation needed and DF set" because Router A does not have to fragment packets when it sends them to the server through Router B.

Scenario 4
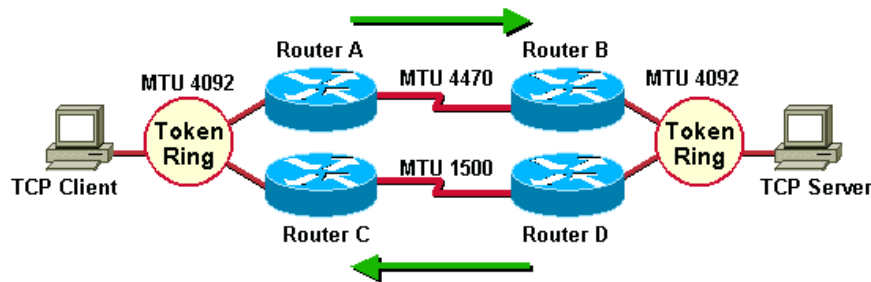
✏️ **Note**: The **ip tcp path-mtu-discovery** command is used in order to enable TCP MTU path discovery for TCP connections initiated by routers (BGP and Telnet for example).

Problems with PMTUD

There are three things that can break PMTUD, two of which are uncommon and one of which is common.

- A router can drop a packet and not send an ICMP message. (Uncommon)
- A router can generate and send an ICMP message, but the ICMP message gets blocked by a router or firewall between this router and the sender. (Common)
- A router can generate and send an ICMP message, but the sender ignores the message. (Uncommon)

The first and last of the three bullets here are uncommon and are usually the result of an error, but the middle bullet describes a common problem. People that implement ICMP packet filters tend to block all ICMP message types rather than only blocking certain ICMP message types. A packet filter can block all ICMP message types except those that are "unreachable" or "time-exceeded." The success or failure of PMTUD hinges upon ICMP unreachable messages getting through to the sender of a TCP/IPv4 packet. ICMP time-exceeded messages are important for other IPv4 issues. An example of such a packet filter, implemented on a router is shown here.

```
access-list 101 permit icmp any any unreachable
access-list 101 permit icmp any any time-exceeded
access-list 101 deny icmp any any
access-list 101 permit ip any any
```

There are other techniques that can be used in order to help alleviate the problem of ICMP being completely blocked.

- Clear the DF bit on the router and allow fragmentation anyway (This might not be a good idea, though. See Issues with IP Fragmentation for more information).
- Manipulate the TCP MSS option value MSS with the interface command **ip tcp adjust-mss <500-1460>**.

In the next scenario, Router A and Router B are in the same administrative domain. Router C is inaccessible and blocks ICMP, so PMTUD is broken. A workaround for this situation is to clear the DF bit in both directions on Router B in order to allow fragmentation. This can be done with policy routing. The syntax to clear the DF bit is available in Cisco IOS® Software Release 12.1(6) and later.

```
interface serial0
...
ip policy route-map clear-df-bit
route-map clear-df-bit permit 10
        match ip address 111
        set ip df 0

access-list 111 permit tcp any any
```
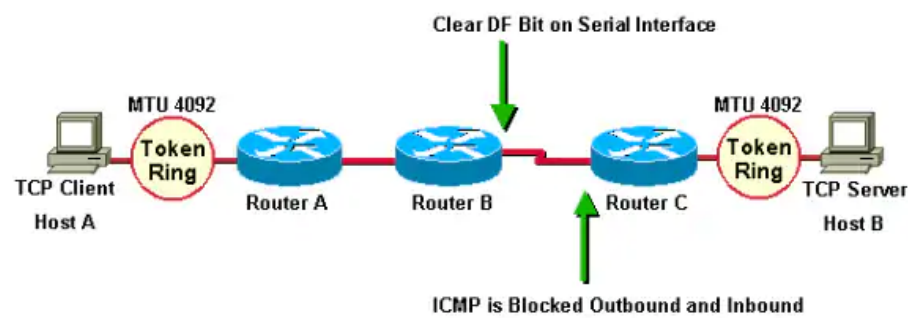
Another option is to change the TCP MSS option value on SYN packets that traverse the router (available in Cisco IOS® 12.2(4)T and later). This reduces the MSS option value in the TCP SYN packet so that it is smaller than the value (1460) in the **ip tcp adjust-mss** command. The result is that the TCP sender will send segments no larger than this value. The IPv4 packet size will be 40 bytes larger (1500) than the MSS value (1460 bytes) in order to account for the TCP header (20 bytes) and the IPv4 header (20 bytes).

You can adjust the MSS of TCP SYN packets with the **ip tcp adjust-mss** command. This syntax will reduce the MSS value on TCP segments to 1460. This command effects traffic both inbound and outbound on interface serial0.

```
int s0
ip tcp adjust-mss 1460
```

IPv4 fragmentation issues have become more widespread since IPv4 tunnels have become more widely deployed. The reason that tunnels cause more fragmentation is because the tunnel encapsulation adds "overhead" to the size of a packet. For example, the addition of Generic Router Encapsulation (GRE) adds 24 bytes to a packet, and after this increase, the packet might need to be fragmented because it is larger than the outbound MTU. In a later section of this document, you will see examples of the kinds of problems that can arise with tunnels and IPv4 fragmentation.

Common Network Topologies that Need PMTUD

PMTUD is needed in network situations where intermediate links have smaller MTUs than the MTU of the end links. Some common reasons for the existence of these smaller MTU links are:

- Token Ring (or FDDI)-connected end hosts with an Ethernet connection between them. The Token Ring (or FDDI) MTUs at the ends are greater than the Ethernet MTU in the middle.
- PPPoE (often used with ADSL) needs 8 bytes for its header. This reduces the effective MTU of the Ethernet to 1492 (1500 – 8).

Tunneling protocols like GRE, IPv4sec, and L2TP also need space for their respective headers and trailers. This also reduces the effective MTU of the outgoing interface.

In the next sections, the impact of PMTUD where a tunneling protocol is used somewhere between the two end hosts are studied. Of the three previous cases, this case is the most complex and covers all of the issues that you might see in the other cases.

Tunnel

A tunnel is a logical interface on a Cisco router that provides a way to encapsulate passenger packets inside a transport protocol. It is an architecture designed to provide services in order to implement a point-to-point encapsulation scheme. Tunneling has these three primary components:

- Passenger protocol (AppleTalk, Banyan VINES, CLNS, DECnet, IPv4, or IPX)
- Carrier protocol – One of these encapsulation protocols:
  - GRE – Cisco's multiprotocol carrier protocol. See RFC 2784 and RFC 1701 for more information.
  - IPv4 in IPv4 tunnels – See RFC 2003 for more information.
- Transport protocol – The protocol used to carry the encapsulated protocol.

The packets shown in this section illustrate the IPv4 tunneling concepts where GRE is the encapsulation protocol and IPv4 is the transport protocol. The passenger protocol is also IPv4. In this case, IPv4 is both the transport and the passenger protocol.

Normal Packet

IPv4	TCP	Telnet

Tunnel Packet

IPv4	GRE	IPv4	TCP	Telnet

- IPv4 is the transport protocol.
- GRE is the encapsulation protocol.
- IPv4 is the passenger protocol.

The next example shows the encapsulation of IPv4 and DECnet as passenger protocols with GRE as the carrier. This illustrates the fact that the carrier protocol can encapsulate multiple passenger protocols as shown in the image.
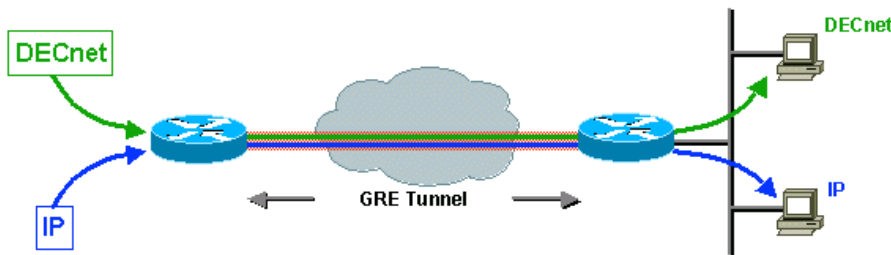
A network administrator might consider tunneling in a situation where there are two discontiguous non-IPv4 networks separated by an IPv4 backbone. If the discontiguous networks run DECnet, the administrator might not want to connect them together by configuring DECnet in the backbone. The administrator might not want to permit DECnet routing to consume backbone bandwidth because this could interfere with the performance of the IPv4 network.

A viable alternative is to tunnel DECnet over the IPv4 backbone. Tunneling encapsulates the DECnet packets inside IPv4, and sends them across the backbone to the tunnel endpoint where the encapsulation is removed and the DECnet packets can be routed to their destination via DECnet.

Encapsulating traffic inside another protocol provides these advantages:

- The endpoints use private addresses (RFC 1918) and the backbone does not support routing these addresses.
- Allow Virtual Private Networks (VPNs) across WANs or the Internet.
- Join together discontiguous multiprotocol networks over a single-protocol backbone.
- Encrypt traffic over the backbone or Internet.

For the rest of the document, IPv4 is used as the passenger protocol and IPv4 as the transport protocol.

Considerations Regarding Tunnel Interfaces

These are considerations when tunneling.

- Fast switching of GRE tunnels was introduced in Cisco IOS® Release 11.1 and CEF switching was introduced in version 12.0. CEF switching for multipoint GRE tunnels was introduced in version 12.2(8)T. Encapsulation and decapsulation at tunnel endpoints were slow operations in earlier versions of Cisco IOS® when only process switching was supported.
- There are security and topology issues when tunneling packets. Tunnels can bypass Access Control Lists (ACLs) and firewalls. If you tunnel through a firewall, you basically bypass the firewall for whatever passenger protocol you are tunneling. Therefore, it is recommended to include firewall functionality at the tunnel endpoints in order to enforce any policy on the passenger protocols.
- Tunneling might create problems with transport protocols that have limited timers (for example, DECnet) because of increased latency.
- Tunneling across environments with different speed links, like fast FDDI rings and through slow 9600-bps phone lines, might introduce packet reordering problems. Some passenger protocols function poorly in mixed media networks.
- Point-to-point tunnels can use up the bandwidth on a physical link. If you run routing protocols over multiple point-to-point tunnels, keep in mind that each tunnel interface has a bandwidth and that the physical interface over which the tunnel runs has a bandwidth. For example, you would want to set the tunnel bandwidth to 100 Kb if there were 100 tunnels running over a 10 Mb link. The default bandwidth for a tunnel is 9Kb.
- Routing protocols might prefer a tunnel over a real link because the tunnel might deceptively appear to be a one-hop link with the lowest cost path, although it actually involves more hops and is really more costly than another path. This can be mitigated with proper configuration of the routing protocol. You might want to consider running a different routing protocol over the tunnel interface than the routing protocol running on the physical interface.
- Problems with recursive routing can be avoided by configuring appropriate static routes to the tunnel destination. A recursive route is when the best path to the tunnel destination is through the tunnel itself. This situation causes the tunnel interface to bounce up and down. You will see this error when there is a recursive routing problem.

```
%TUN-RECURDOWN Interface Tunnel 0
temporarily disabled due to recursive routing
```

Router as PMTUD Participant at Endpoint of Tunnel

The router has two different PMTUD roles to play when it is the endpoint of a tunnel.

- In the first role, the router is the forwarder of a host packet. For PMTUD processing, the router needs to check the DF bit and packet size of the original data packet and take appropriate action when necessary.
- The second role comes into play after the router has encapsulated the original IPv4 packet inside the tunnel packet. At this stage, the router acts more like a host with respect to PMTUD and in regards to the tunnel IPv4 packet.

Lets have a look at what happens when the router acts in the first role, a router that forwards host IPv4 packets, with respect to PMTUD. This role comes into play before the router encapsulates the host IPv4 packet inside the tunnel packet.

If the router participates as the forwarder of a host packet it will complete these actions:

- Check whether the DF bit is set
- Check what size packet the tunnel can accommodate
- Fragment (if packet is too large and DF bit is not set), encapsulate fragments and send; or

- Drop the packet (if packet is too large and DF bit is set) and send an ICMP message to the sender

- Encapsulate (if packet is not too large) and send

Generically, there is a choice of encapsulation and then fragmentation (send two encapsulation fragments) or fragmentation and then encapsulation (send two encapsulated fragments).

Some examples that describe the mechanics of IPv4 packet encapsulation and fragmentation and two scenarios that show the interaction of PMTUD and packets that traverse example networks are detailed in this section.

The first example shows what happens to a packet when the router (at the tunnel source) acts in the role of forwarding router. Remember that to process PMTUD, the router needs to check the DF bit and packet size of the original data packet and take appropriate action. This examples uses GRE encapsulation for the tunnel. As can be seen, GRE does fragmentation before encapsulation. Later examples show scenarios in which fragmentation is done after encapsulation.

In Example 1, the DF bit is not set (DF = 0) and the GRE tunnel IPv4 MTU is 1476 (1500 – 24).

Example 1

1. The forwarding router (at the tunnel source) receives a 1500-byte datagram with the DF bit clear (DF = 0) from the sending host. This datagram is composed of a 20-byte IP header plus a 1480 byte TCP payload.

IPv4	1480 bytes TCP + data

2. Because the packet is too large for the IPv4 MTU after the GRE overhead (24 bytes) is added, the forwarding router breaks the datagram into two fragments of 1476 (20 bytes IPv4 header + 1456 bytes IPv4 payload) and 44 bytes (20 bytes of IPv4 header + 24 bytes of IPv4 payload) so after the GRE encapsulation is added, the packet will not be larger than the outgoing physical interface MTU.

IP0	1456 bytes TCP + data
IP1	24 bytes data

3. The forwarding router adds GRE encapsulation, which includes a 4-byte GRE header plus a 20-byte IPv4 header, to each fragment of the original IPv4 datagram. These two IPv4 datagrams now have a length of 1500 and 68 bytes and these datagrams are seen as individual IPv4 datagrams, not as fragments.

IPv4	GRE	IP0	1456 bytes TCP + data
IPv4	GRE	IP1	24 bytes data

4. The tunnel destination router removes the GRE encapsulation from each fragment of the original datagram, which leaves two IPv4 fragments of lengths 1476 and 24 bytes. These IPv4 datagram fragments are forwarded separately by this router to the receiving host.

IP0	1456 bytes TCP + data
IP1	24 bytes data

5. The receiving host reassembles these two fragments into the original datagram.

IPv4	1480 bytes TCP + data

Scenario 5 depicts the role of the forwarding router in the context of a network topology.

In this example, the router acts in the same role of forwarding router, but this time the DF bit is set (DF = 1).

Example 2

1. The forwarding router at the tunnel source receives a 1500-byte datagram with DF = 1 from the sending host.

IPv4	1480 bytes TCP + data

2. Since the DF bit is set, and the datagram size (1500 bytes) is greater than the GRE tunnel IPv4 MTU (1476), the router will drop the datagram and send an "ICMP fragmentation needed but DF bit set" message to the source of the datagram. The ICMP message will alert the sender that the MTU is 1476.

IPv4	ICMP MTU 1476

3. The sending host receives the ICMP message, and when it resends the original data it uses a 1476-byte IPv4 datagram.

IPv4	1456 bytes TCP + data

4. This IPv4 datagram length (1476 bytes) is now equal in value to the GRE tunnel IPv4 MTU so the router adds the GRE encapsulation to the IPv4 datagram.

IPv4	GRE	IPv4	1456 bytes TCP + data

5. The receiving router (at the tunnel destination) removes the GRE encapsulation of the IPv4 datagram and sends it to the receiving host.

IPv4	1456 bytes TCP + data

Now, you can look at what happens when the router acts in the second role as a sending host with respect to PMTUD and in regards to the tunnel IPv4 packet. Recall that this role comes into play after the router has encapsulated the original IPv4 packet inside the tunnel packet.

✎ **Note**: By default, a router does not do PMTUD on the GRE tunnel packets that it generates. The **tunnel path-mtu-discovery** command can be used to turn on PMTUD for GRE-IPv4 tunnel packets.

Example 3 shows what happens when the host sends IPv4 datagrams that are small enough to fit within the IPv4 MTU on the GRE Tunnel interface. The DF bit in this case can be either set or clear (1 or 0). The GRE tunnel interface does not have the **tunnel path-mtu-discovery** command configured so the router will not do PMTUD on the GRE-IPv4 packet.

Example 3

1. The forwarding router at the tunnel source receives a 1476-byte datagram from the sending host.

IPv4	1456 bytes TCP + data

2. This router encapsulates the 1476-byte IPv4 datagram inside GRE to get a 1500-byte GRE IPv4 datagram. The DF bit in the GRE IPv4 header will be clear (DF = 0). This router then forwards this packet to the tunnel destination.

IPv4	GRE	IPv4	1456 bytes TCP + data

3. Assume there is a router between the tunnel source and destination with a link MTU of 1400. This router will fragment the tunnel packet since the DF bit is clear (DF = 0). Remember that this example fragments the outermost IPv4, so the GRE, inner IPv4, and TCP headers will only show up in the first fragment.

IP0	GRE	IP	1352 bytes TCP + data
IP1		104 bytes data	

4. The tunnel destination router must reassemble the GRE tunnel packet.

IP	GRE	IP	1456 bytes TCP + data

5. After the GRE tunnel packet is reassembled, the router removes the GRE IPv4 header and sends the original IPv4 datagram on its way.

IPv4	1456 Bytes TCP + data

The next example shows what happens when the router acts in the role of a sending host with respect to PMTUD and in regards to the tunnel IPv4 packet. This time the DF bit is set (DF = 1) in the original IPv4 header and the **tunnel path-mtu-discovery** command has been configured so that the DF bit will be copied from the inner IPv4 header to the outer (GRE + IPv4) header.

Example 4

1. The forwarding router at the tunnel source receives a 1476-byte datagram with DF = 1 from the sending host.

IPv4	1456 bytes TCP + data

2. This router encapsulates the 1476-byte IPv4 datagram inside GRE to get a 1500-byte GRE IPv4 datagram. This GRE IPv4 header will have the DF bit set (DF = 1) since the original IPv4 datagram had the DF bit set. This router then forwards this packet to the tunnel destination.

| IPv4	GRE	IPv4	1456 bytes TCP

3. Again, assume there is a router between the tunnel source and destination with a link MTU of 1400. This router will not fragment the tunnel packet since the DF bit is set (DF=1). This router must drop the packet and send an ICMP error message to the tunnel source router, since that is the source IPv4 address on the packet.

IPv4	ICMP MTU 1400

4. The forwarding router at the tunnel source receives this "ICMP" error message and it will lower the GRE tunnel IPv4 MTU to 1376 (1400 - 24). The next time the sending host retransmits the data in a 1476-byte IPv4 packet, this packet can be too large and this router will send an "ICMP" error message to the sender with a MTU value of 1376. When the sending host retransmits the data, it will send it in a 1376-byte IPv4 packet and this packet will make it through the GRE tunnel to the receiving host.

Scenario 5

This scenario illustrates GRE fragmentation. Remember that you fragment before encapsulation for GRE, then do PMTUD for the data packet, and the DF bit is not copied when the IPv4 packet is encapsulated by GRE. In this scenario, the DF bit is not set. The GRE tunnel interface IPv4 MTU is, by default, 24 bytes less than the physical interface IPv4 MTU, so the GRE interface IPv4 MTU is 1476 as shown in the image.
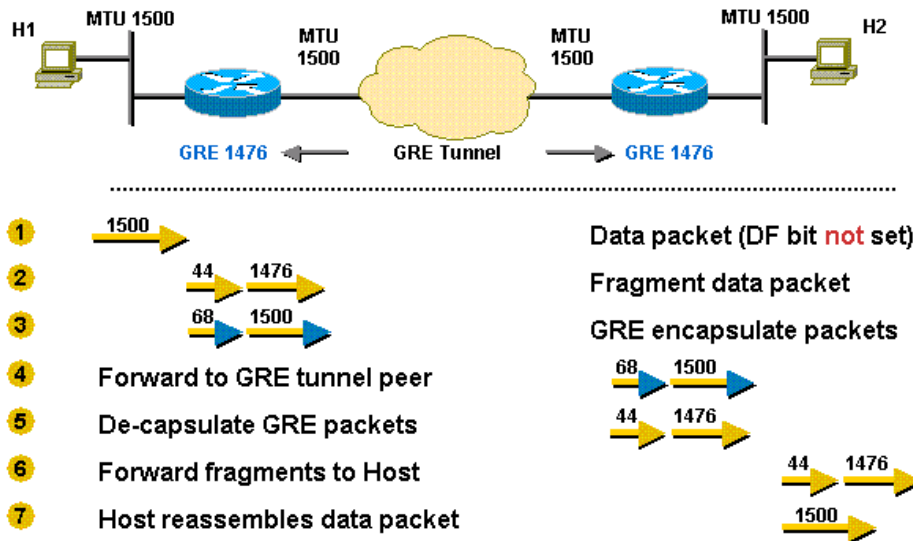
1. The sender sends a 1500-byte packet (20 byte IPv4 header + 1480 bytes of TCP payload).

2. Since the MTU of the GRE tunnel is 1476, the 1500-byte packet is broken into two IPv4 fragments of 1476 and 44 bytes, each in anticipation of the additional 24 byes of GRE header.

3. The 24 bytes of GRE header is added to each IPv4 fragment. Now the fragments are 1500 (1476 + 24) and 68 (44 + 24) bytes each.

4. The GRE + IPv4 packets that contain the two IPv4 fragments are forwarded to the GRE tunnel peer router.

5. The GRE tunnel peer router removes the GRE headers from the two packets.

6. This router forwards the two packets to the destination host.

7. The destination host reassembles the IPv4 fragments back into the original IPv4 datagram.

Scenario 6

This scenario is similar to Scenario 5, but this time the DF bit is set. In Scenario 6, the router is configured to do PMTUD on GRE + IPv4 tunnel packets with the **tunnel path-mtu-discovery** command, and the DF bit is copied from the original IPv4 header to the GRE IPv4 header. If the router receives an ICMP error for the GRE + IPv4 packet, it reduces the IPv4 MTU on the GRE tunnel interface. Again, remember that the GRE Tunnel IPv4 MTU is set to 24 bytes less than the physical interface MTU by default, so the GRE IPv4 MTU here is 1476. Also notice that there is a 1400 MTU link in the GRE tunnel path as shown in the image.
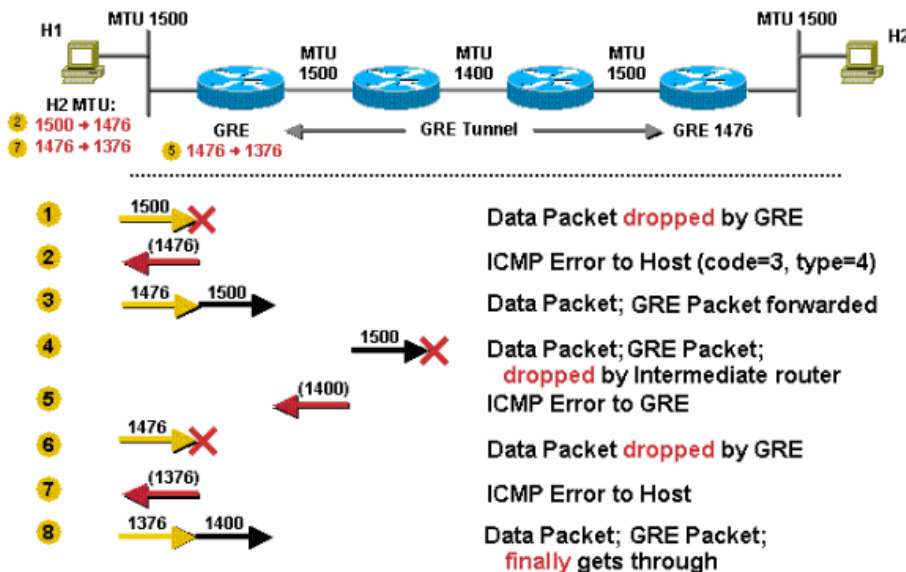
1. The router receives a 1500-byte packet (20 byte IPv4 header + 1480 TCP payload), and it drops the packet. The router drops the packet because it is larger than the IPv4 MTU (1476) on the GRE tunnel interface.

2. The router sends an ICMP error to the sender telling it that the next-hop MTU is 1476. The host will record this information, usually as a host route for the destination in its routing table.

3. The sending host uses a 1476-byte packet size when it resends the data. The GRE router adds 24 bytes of GRE encapsulation and ships out a 1500-byte packet.

4. The 1500-byte packet cannot traverse the 1400-byte link, so it is dropped by the intermediate router.

5. The intermediate router sends an ICMP (type = 3, code = 4) to the GRE router with a next-hop MTU of 1400. The GRE router reduces this to 1376 (1400 - 24) and sets an internal IPv4 MTU value on the GRE interface. This change can only be seen when using the **debug tunnel command**; it cannot be seen in the output from the **show ip interface tunnel<#>** command.

6. The next time the host resends the 1476-byte packet, the GRE router will drop the packet, since it is larger than the current IPv4 MTU (1376) on the GRE tunnel interface.

7. The GRE router will send another ICMP (type = 3, code = 4) to the sender with a next-hop MTU of 1376 and the host will update its current information with new value.

8. The host again resends the data, but now in a smaller 1376-byte packet, GRE will add 24 bytes of encapsulation and forward it on. This time the packet will make it to the GRE tunnel peer, where the packet will be decapsulated and sent to the destination host.

> ✎ **Note**: If the **tunnel path-mtu-discovery** command was not configured on the forwarding router in this scenario, and the DF bit was set in the packets forwarded through the GRE tunnel, Host 1 would still succeed in sending TCP/IPv4 packets to Host 2, but they would get fragmented in the middle at the 1400 MTU link. Also the GRE tunnel peer would have to reassemble them before it could decapsulate and forward them on.

Pure IPsec Tunnel Mode

The IPv4 Security (IPv4sec) Protocol is a standards-based method that provides privacy, integrity, and authenticity to information transferred across IPv4 networks. IPv4sec provides IPv4 network-layer encryption. IPv4sec lengthens the IPv4 packet by adding at least one IPv4 header (tunnel mode). The added header(s) varies in length dependent on the IPv4sec configuration mode but they do not exceed ~58 bytes (Encapsulating Security Payload (ESP) and ESP authentication (ESPauth)) per packet.

IPv4sec has two modes, tunnel mode and transport mode.

1. Tunnel mode is the default mode. With tunnel mode, the entire original IPv4 packet is protected (encrypted, authenticated, or both) and encapsulated by the IPv4sec headers and trailers. Then a new IPv4 header is prepended to the packet, which specifies the IPv4sec endpoints (peers) as the source and destination. Tunnel mode can be used with any unicast IPv4 traffic and must be used if IPv4sec protects traffic from hosts behind the IPv4sec peers. For example, tunnel mode is used with Virtual Private Networks (VPNs) where hosts on one protected network send packets to hosts on a different protected network via a pair of IPv4sec peers. With VPNs, the IPv4sec "tunnel" protects the IPv4 traffic between hosts by encrypting this traffic between the IPv4sec peer routers.

2. With transport mode (configured with the subcommand, **mode transport**, on the transform definition), only the payload of the original IPv4 packet is protected (encrypted, authenticated, or both). The payload is encapsulated by the IPv4sec headers and trailers. The original IPv4 headers remain intact, except that the IPv4 protocol field is changed to be ESP (50), and the original protocol value is saved in the IPv4sec trailer to be restored when the packet is decrypted. Transport mode is used only when the IPv4 traffic to be protected is between the IPv4sec peers themselves, the source and destination IPv4 addresses on the packet are the same as the IPv4sec peer addresses. Normally IPv4sec transport mode is only used when another tunneling protocol (like GRE) is used to first encapsulate the IPv4 data packet, then IPv4sec is used to protect the GRE tunnel packets.

IPv4sec always does PMTUD for data packets and for its own packets. There are IPv4sec configuration commands to modify PMTUD processing for the IPv4sec IPv4 packet, IPv4sec can clear, set, or copy the DF bit from the data packet IPv4 header to the IPv4sec IPv4 header. This is called the "DF Bit Override Functionality" feature.

> ✎ **Note**: You really want to avoid fragmentation after encapsulation when you do hardware encryption with IPv4sec. Hardware encryption can give you throughput of about 50 Mbs which depends on the hardware, but if the IPv4sec packet is fragmented you loose 50 to 90 percent of the throughput. This loss is because the fragmented IPv4sec packets are process-switched for reassembly and then handed to the Hardware encryption engine for decryption. This loss of throughput can bring hardware encryption throughput down to the performance level of software encryption (2-10 Mbs).

Scenario 7

This scenario depicts IPv4sec fragmentation in action. In this scenario, the MTU along the entire path is 1500. In this scenario, the DF bit is not set.
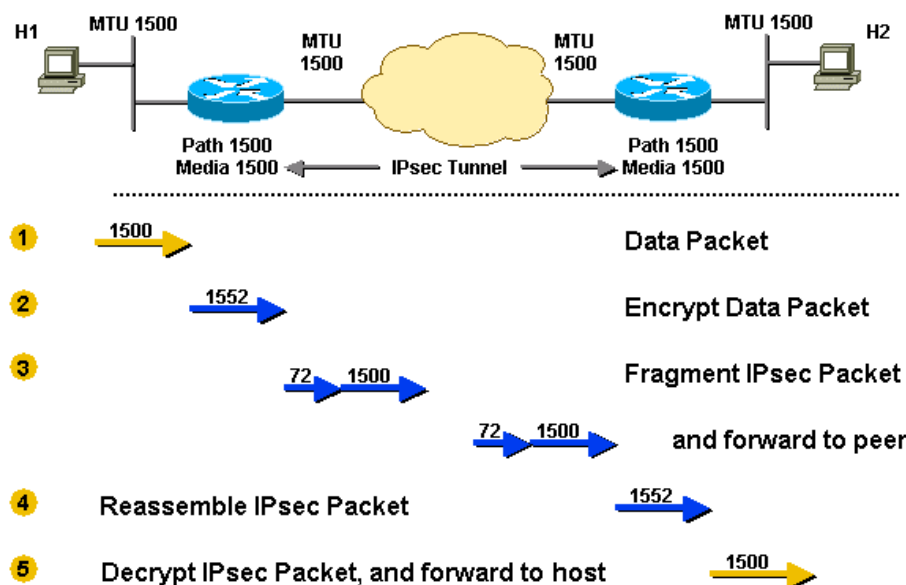
1. The router receives a 1500-byte packet (20-byte IPv4 header + 1480 bytes TCP payload) destined for Host 2.

2. The 1500-byte packet is encrypted by IPv4sec and 52 bytes of overhead are added (IPv4sec header, trailer, and additional IPv4 header). Now IPv4sec needs to send a 1552-byte packet. Since the outbound MTU is 1500, this packet will have to be fragmented.

3. Two fragments are created out of the IPv4sec packet. During fragmentation, an additional 20-byte IPv4 header is added for the second fragment, resulting in a 1500-byte fragment and a 72-byte IPv4 fragment.

4. The IPv4sec tunnel peer router receives the fragments, strips off the additional IPv4 header and coalesces the IPv4 fragments back into the original IPv4sec packet. Then IPv4sec decrypts this packet.

5. The router then forwards the original 1500-byte data packet to Host 2.

Scenario 8

This scenario is similar to Scenario 6 except that in this case the DF bit is set in the original data packet and there is a link in the path between the IPv4sec tunnel peers that has a lower MTU than the other links. This scenario demonstrates how the IPv4sec peer router performs both PMTUD roles, as described in the The Router as a PMTUD Participant at the Endpoint of a Tunnel section.

You will see in this scenario how the IPv4sec PMTU changes to a lower value as the result of the need for fragmentation. Remember that the DF bit is copied from the inner IPv4 header to the outer IPv4 header when IPv4sec encrypts a packet. The media MTU and PMTU values are stored in the IPv4sec Security Association (SA). The media MTU is based on the MTU of the outbound router interface and the PMTU is based on the minimum MTU seen on the path between the IPv4sec peers. Remember that IPv4sec encapsulates/encrypts the packet before it attempts to fragment it as shown in the image.
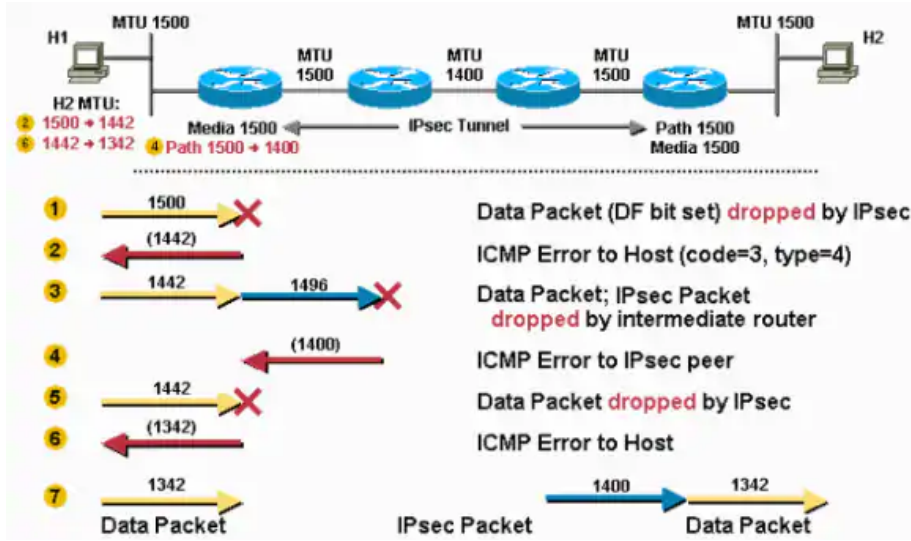
1. The router receives a 1500-byte packet and drops it because the IPv4sec overhead, when added, will make the packet larger than the PMTU (1500).

2. The router sends an ICMP message to Host 1 telling it that the next-hop MTU is 1442 (1500 - 58 = 1442). This 58 bytes is the maximum IPv4sec overhead when using IPv4sec ESP and ESPauth. The real IPv4sec overhead may be as much as 7 bytes less than this value. Host 1 records this information, usually as a host route for the destination (Host 2), in its routing table.

3. 3. Host 1 lowers its PMTU for Host 2 to 1442, so Host 1 will send smaller (1442 byte) packets when it retransmits the data to Host 2. The router receives the 1442-byte packet and IPv4sec adds 52 bytes of encryption overhead so the resulting IPv4sec packet is 1496 bytes. Because this packet has the DF bit set in its header it gets dropped by the middle router with the 1400-byte MTU link.

4. The middle router that dropped the packet sends an ICMP message to the sender of the IPv4sec packet (the first router) telling it that the next-hop MTU is 1400 bytes. This value is recorded in the IPv4sec SA PMTU.

5. The next time Host 1 retransmits the 1442-byte packet (it didn't receive an acknowledgment for it), the IPv4sec will drop the packet. Again the router will drop the packet because the IPv4sec overhead, when added to the packet, will make it larger than the PMTU (1400).

6. The router sends an ICMP message to Host 1 telling it that the next-hop MTU is now 1342. (1400 - 58 = 1342). Host 1 will again record this information.

7. When Host 1 again retransmits the data, it will use the smaller size packet (1342). This packet will not require fragmentation and will make it through the IPv4sec tunnel to Host 2.

GRE and IPv4sec Together

More complex interactions for fragmentation and PMTUD occur when IPv4sec is used in order to encrypt GRE tunnels. IPv4sec and GRE are combined in this manner because IPv4sec does not support IPv4 multicast packets, which means that you cannot run a dynamic routing protocol over the IPv4sec VPN Network. GRE tunnels do support multicast, so a GRE tunnel can be used to first encapsulate the dynamic routing protocol multicast packet in a GRE IPv4 unicast packet that can then be encrypted by IPv4sec. When doing this, IPv4sec is often deployed in transport mode on top of GRE because the IPv4sec peers and the GRE tunnel endpoints (the routers) are the same, and transport-mode will save 20 bytes of IPv4sec overhead.

One interesting case is when an IPv4 packet has been split into two fragments and encapsulated by GRE. In this case IPv4sec will see two independent GRE + IPv4 packets. Often in a default configuration one of these packets will be large enough that it will need to be fragmented after it has been encrypted. The IPv4sec peer will have to reassemble this packet before decryption. This "double fragmentation" (once before GRE and again after IPv4sec) on the sending router increases latency and lowers throughput. Also, reassembly is process-switched, so there will be a CPU hit on the receiving router whenever this happens.

This situation can be avoided by setting the "ip mtu" on the GRE tunnel interface low enough to take into account the overhead from both GRE and IPv4sec (by default the GRE tunnel interface "ip mtu" is set to the outgoing real interface MTU – GRE overhead bytes).

This table lists the suggested MTU values for each tunnel/mode combination assuming the outgoing physical interface has an MTU of 1500.

Tunnel Combination	Specific MTU Needed	Recommended MTU
GRE + IPv4sec (Transport mode)	1440 bytes	1400 bytes
GRE + IPv4sec (Tunnel mode)	1420 bytes	1400 bytes

✎ **Note**: The MTU value of 1400 is recommended because it covers the most common GRE + IPv4sec mode combinations. Also, there is no discernable downside to allowing for an extra 20 or 40 bytes overhead. It is easier to remember and set one value and this value covers almost all scenarios.

Scenario 9

IPv4sec is deployed on top of GRE. The outgoing physical MTU is 1500, the IPv4sec PMTU is 1500, and the GRE IPv4 MTU is 1476 (1500 – 24 = 1476). Because of this, TCP/IPv4 packets will be fragmented twice, once before GRE and once after IPv4sec. The packet will be fragmented before GRE encapsulation and one of these GRE packets will be fragmented again after IPv4sec encryption.

Configuring "ip mtu 1440" (IPv4sec Transport mode) or "ip mtu 1420" (IPv4sec Tunnel mode) on the GRE tunnel would remove the possibility of double fragmentation in this scenario.
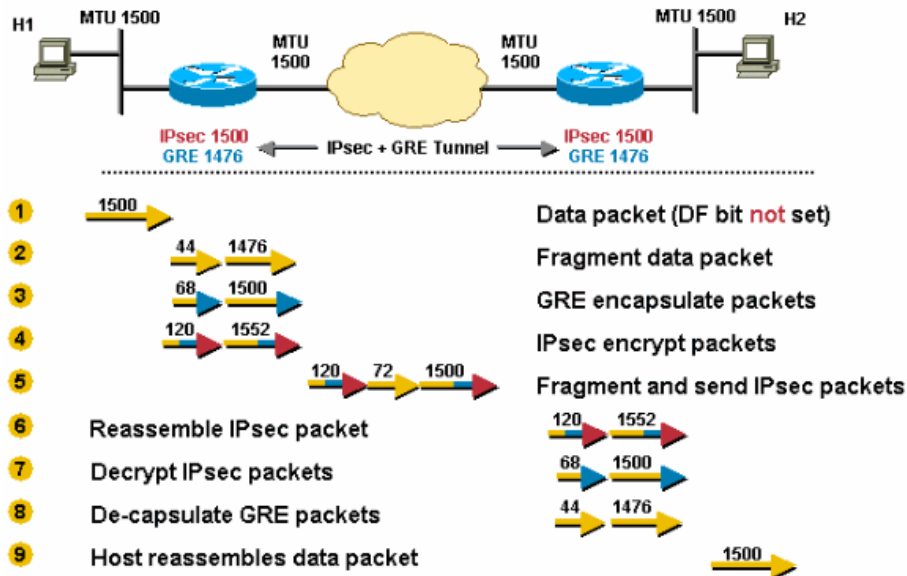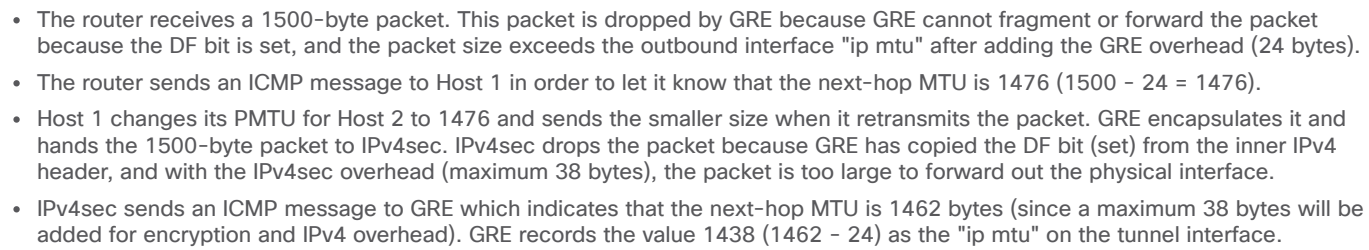

1. The router receives a 1500-byte datagram.

2. Before encapsulation, GRE fragments the 1500-byte packet into two pieces, 1476 (1500 – 24 = 1476) and 44 (24 data + 20 IPv4 header) bytes.

3. GRE encapsulates the IPv4 fragments, which adds 24 bytes to each packet. This results in two GRE + IPv4sec packets of 1500 (1476 + 24 = 1500) and 68 (44 + 24) bytes each.

4. IPv4sec encrypts the two packets, adding 52 byes (IPv4sec tunnel-mode) of encapsulation overhead to each, in order to give a 1552-byte and a 120-byte packet.

5. The 1552-byte IPv4sec packet is fragmented by the router because it is larger than the outbound MTU (1500). The 1552-byte packet is split into pieces, a 1500-byte packet and a 72-byte packet (52 bytes "payload" plus an additional 20-byte IPv4 header for the second fragment). The three packets 1500-byte, 72-byte, and 120-byte packets are forwarded to the IPv4sec + GRE peer.

6. The receiving router reassembles the two IPv4sec fragments (1500 bytes and 72 bytes) in order to get the original 1552-byte IPv4sec + GRE packet. Nothing needs to be done to the 120-byte IPv4sec + GRE packet.

7. IPv4sec decrypts both 1552-byte and 120-byte IPv4sec + GRE packets in order to get 1500-byte and 68-byte GRE packets.

8. GRE decapsulates the 1500-byte and 68-byte GRE packets in order to get 1476-byte and 44-byte IPv4 packet fragments. These IPv4 packet fragments are forwarded to the destination host.

9. Host 2 reassembles these IPv4 fragments in order to get the original 1500-byte IPv4 datagram.

Scenario 10 is similar to Scenario 8 except there is a lower MTU link in the tunnel path. This is a worst case scenario for the first packet sent from Host 1 to Host 2. After the last step in this scenario, Host 1 sets the correct PMTU for Host 2 and all is well for the TCP connections between Host 1 and Host 2. TCP flows between Host 1 and other hosts (reachable via the IPv4sec + GRE tunnel) will only have to go through the last three steps of Scenario 10.

In this scenario, the **tunnel path-mtu-discovery** command is configured on the GRE tunnel and the DF bit is set on TCP/IPv4 packets that originate from Host 1.

Scenario 10

- The router receives a 1500-byte packet. This packet is dropped by GRE because GRE cannot fragment or forward the packet because the DF bit is set, and the packet size exceeds the outbound interface "ip mtu" after adding the GRE overhead (24 bytes).
- The router sends an ICMP message to Host 1 in order to let it know that the next-hop MTU is 1476 (1500 - 24 = 1476).
- Host 1 changes its PMTU for Host 2 to 1476 and sends the smaller size when it retransmits the packet. GRE encapsulates it and hands the 1500-byte packet to IPv4sec. IPv4sec drops the packet because GRE has copied the DF bit (set) from the inner IPv4 header, and with the IPv4sec overhead (maximum 38 bytes), the packet is too large to forward out the physical interface.
- IPv4sec sends an ICMP message to GRE which indicates that the next-hop MTU is 1462 bytes (since a maximum 38 bytes will be added for encryption and IPv4 overhead). GRE records the value 1438 (1462 - 24) as the "ip mtu" on the tunnel interface.

✎ **Note**: This change in value is stored internally and cannot be seen in the output of the **show ip interface tunnel<#>** command. You will only see this change if you turn use the **debug tunnel** command.

- The next time Host 1 retransmits the 1476-byte packet, GRE drops it.
- The router sends an ICMP message to Host 1 which indicates that 1438 is the next-hop MTU.
- Host 1 lowers the PMTU for Host 2 and retransmits a 1438-byte packet. This time, GRE accepts the packet, encapsulates it, and hands it off to IPv4sec for encryption. The IPv4sec packet is forwarded to the intermediate router and dropped because it has an outbound interface MTU of 1400.
- The intermediate router sends an ICMP message to IPv4sec which tells it that the next-hop MTU is 1400. This value is recorded by IPv4sec in the PMTU value of the associated IPv4sec SA.
- When Host 1 retransmits the 1438-byte packet, GRE encapsulates it and hands it to IPv4sec. IPv4sec drops the packet because it has changed its own PMTU to 1400.
- IPv4sec sends an ICMP error to GRE which indicates that the next-hop MTU is 1362, and GRE records the value 1338 internally.
- When Host 1 retransmits the original packet (because it did not receive an acknowledgment), GRE drops it.
- The router sends an ICMP message to Host 1 which indicates the next-hop MTU is 1338 (1362 - 24 bytes). Host 1 lowers its PMTU for Host 2 to 1338.
- Host 1 retransmits a 1338-byte packet and this time it can finally get all the way through to Host 2.

More Recommendations

Configuring the **tunnel path-mtu-discovery** command on a tunnel interface can help GRE and IPv4sec interaction when they are configured on the same router. Remember that without the **tunnel path-mtu-discovery** command configured, the DF bit would always be cleared in the GRE IPv4 header. This allows the GRE IPv4 packet to be fragmented even though the encapsulated data IPv4 header had the DF bit set, which normally would not allow the packet to be fragmented.

If the **tunnel path-mtu-discovery** command is configured on the GRE tunnel interface, this will happen.

1. GRE will copy the DF bit from the data IPv4 header to the GRE IPv4 header.

2. If the DF bit is set in the GRE IPv4 header and the packet will be "too large" after IPv4sec encryption for the IPv4 MTU on the physical outgoing interface, then IPv4sec will drop the packet and notify the GRE tunnel to reduce its IPv4 MTU size.

3. IPv4sec does PMTUD for its own packets and if the IPv4sec PMTU changes (if it is reduced), then IPv4sec does not immediately notify GRE, but when another "too large" packet comes thorough, then the process in step 2 occurs.

4. GRE's IPv4 MTU is now smaller, so it will drop any data IPv4 packets with the DF bit set that are now too large and send an ICMP message to the sending host.

The **tunnel path-mtu-discovery** command helps the GRE interface set its IPv4 MTU dynamically, rather than statically with the **ip mtu** command. It is actually recommended that both commands are used. The **ip mtu** command is used to provide room for the GRE and IPv4sec overhead relative to the local physical outgoing interface IPv4 MTU. The **tunnel path-mtu-discovery** command allows the GRE tunnel IPv4 MTU to be further reduced if there is a lower IPv4 MTU link in the path between the IPv4sec peers.

Here are some of the things you can do if you have problems with PMTUD in a network where there are GRE + IPv4sec tunnels configured.

This list begins with the most desirable solution.

1. Fix the problem with PMTUD not working, which is usually caused by a router or firewall that blocks ICMP.

2. Use the **ip tcp adjust-mss** command on the tunnel interfaces so that the router will reduce the TCP MSS value in the TCP SYN packet. This will help the two end hosts (the TCP sender and receiver) to use packets small enough so that PMTUD is not needed.

3. Use policy routing on the ingress interface of the router and configure a route map to clear the DF bit in the data IPv4 header before it gets to the GRE tunnel interface. This will allow the data IPv4 packet to be fragmented before GRE encapsulation.

4. Increase the "ip mtu" on the GRE tunnel interface to be equal to the outbound interface MTU. This will allow the data IPv4 packet to be GRE encapsulated without fragmenting it first. The GRE packet will then be IPv4sec encrypted and then fragmented to go out the physical outbound interface. In this case you would not configure **tunnel path-mtu-discovery** command on the GRE tunnel interface. This can dramatically reduce the throughput because IPv4 packet reassembly on the IPv4sec peer is done in process-switching mode.

Related Information

- IP Routing Support Page
- IPSec (IP Security Protocol) Support Page
- IPSec Overhead Calculator (Calculate Packet Size with IPSec Encapsulation Protocols)
- RFC 1191 Path MTU Discovery
- RFC 1063 IP MTU Discovery Options
- RFC 791 Internet Protocol
- RFC 793 Transmission Control Protocol
- RFC 879 The TCP Maximum Segment Size and Related Topics
- RFC 1701 Generic Routing Encapsulation (GRE)
- RFC 1241 A Scheme for an Internet Encapsulation Protocol
- RFC 2003 IP Encapsulation within IP
- Technical Support & Documentation – Cisco Systems

PDU SDU Data Units at different OSI layers

In the context of packet switching data networks, a protocol data unit (PDU) is best understood in relation to a service data unit (SDU).

The features or services of the network are implemented in distinct "layers". For example, sending ones and zeros across a wire, fiber, etc. is done by the physical layer, organizing the ones and zeros into chunks of data and getting them safely to the right place on the wire is done by the data link layer, passing data chunks over multiple connected networks is done by the network layer and delivery of the data to the right software application at the destination is done by the transport layer.

Between the layers (and between the application and the top-most layer), the layers pass service data units across the interfaces. The higher layer understands the structure of the data in the SDU, but the lower layer at the interface does not; the lower layer treats the SDU as payload, undertaking to get it to the same interface at the destination. In order to do this, the protocol (lower) layer will add to the SDU certain data it needs to perform its function; which is called encapsulation. For example, it might add a port number to identify the application, a network address to help with routing, a code to identify the type of data in the packet and error-checking information. All this additional information, plus the original service data unit from the higher layer, constitutes the protocol data unit at this layer.

The SDU and metadata added by the lower layer can be larger than the maximum size of that layer's PDU (known as the maximum transmission unit; MTU). When this is the case, the SDU must be split into multiple payloads of size suitable for transmission or processing by the lower later; a process known as IP fragmentation.

The significance of this is that the PDU is the structured information that is passed to a matching protocol layer further along on the data's journey that allows the layer to deliver its intended function or service. The matching layer, or "peer", decodes the data to extract the original service data unit, decide if it is error-free and where to send it next, etc. Unless we have already arrived at the lowest (physical) layer, the PDU is passed to the peer using services of the next lower layer in the protocol "stack". When the PDU passes over the interface from the layer that constructed it to the layer that merely delivers it (and therefore does not understand its internal structure), it becomes a service data unit to that layer. The addition of addressing and control information (encapsulation) to an SDU to form a PDU and the passing of that PDU to the next lower layer as an SDU repeats until the lowest layer is reached and the data passes over some medium as a physical signal.

The above process can be likened to the mail system in which a letter (SDU) is placed in an envelope on which is written an address (addressing and control information) making it a PDU. The sending post office might look only at the post code and place the letter in a mail bag so that the address on the envelope can no longer be seen, making it now an SDU. The mail bag is labelled with the destination post code and so becomes a PDU, until it is combined with other bags in a crate, when it is now an SDU, and the crate is labelled with the region to which all the bags are to be sent, making the crate a PDU. When the crate reaches the destination matching its label, it is opened and the bags (SDUs) removed only to become PDUs when someone reads the code of the destination post office. The letters themselves are SDUs when the bags are opened but become PDUs when the address is read for final delivery. When the addressee finally opens the envelope, the top-level SDU, the letter itself, emerges.

Protocol data units of the OSI model are:
The Layer 4: transport layer PDU is the segment or the datagram.
The Layer 3: network layer PDU is the packet.
The Layer 2: data link layer PDU is the frame.
The Layer 1: physical layer PDU is the bit or, more generally, symbol.
Given a context pertaining to a specific OSI layer, PDU is sometimes used as a synonym for its representation at that layer.

When the PDU of one layer (n), sent as SDU to the layer one lower than it (n-1), plus any metadata (headers/trailers) layer n-1 adds to the SDU would exceed the largest size layer n-1's PDU can be, it must be broken up into multiple payloads to fit in the layer-n-1 PDU's. This process is called IP fragmentation.

A media access control protocol data unit (MPDU) is a message that is exchanged between media access control (MAC) entities in a communication system based on the layered OSI model. In systems where the MPDU may be larger than the MAC service data unit (MSDU), the MPDU may include multiple MSDUs as a result of packet aggregation. In systems where the MPDU is smaller than the MSDU, then one MSDU may generate multiple MPDUs as a result of packet segmentation.

Different layer protocols

Layer	Name	Function	Protocols
Layer 7	Application	To allow access to network resources.	SMTP, HTTP, FTP, POP3, SNMP
Layer 6	Presentation	To translate, encrypt and compress data.	MPEG, ASCH, SSL, TLS
Layer 5	Session	To establish, manage, and terminate the session	NetBIOS, SAP
Layer 4	Transport	The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine.	TCP, UDP
Layer 3	Network	To provide internetworking To move packets from source to destination	IPV5, IPV6, ICMP, IPSEC, ARP, MPLS.
Layer 2	Data Link	To organize bits into frames To provide hop-to-hop delivery	RAPA, PPP, Frame Relay, ATM, Fiber Cable, etc.
Layer 1	Physical	To transmit bits over a medium To provide mechanical and electrical specifications	RS232, 100BaseTX, ISDN, 11.

OSI LAYERS	DEVICES	DATA UNIT	FUNCTIONS
PHYSICAL LAYER	HUB & REPEATER	BIT	Bit synchronization , Bit rate control , Physical topologies , Transmission mode
DATA LINK LAYER	BRIDGE & SWITCH	FRAME	Framing , Physical Addressing , Error Control , Flow Control & Access Control
NETWORK LAYER	ROUTER & GATEWAY	PACKET/ DATAGRAM	Routing , Logical Addressing **H/W LAYER**
TRANSPORT LAYER ❤	GATEWAY &PROXY SERVER	SEGMENTS	Sender : Segmentation & Flow and error control Receiver : Segmentation & Reassembly , Service Point Addressing
SESSION LAYER	GATEWAY &PROXY SERVER	DATA	Session Establishment , Maintenance & Termination , Synchronization & Dialog Controller **S/W LAYER**
PRESENTATION LAYER	GATEWAY &PROXY SERVER	DATA	Translation , Encryption / Decryption & Compression
APPLICATION LAYER	GATEWAY &PROXY SERVER	DATA	Network Virtual Terminal , FTAM-File transfer access and management Mail Services , Directory Services