Internet Engineering Task Force
INTERNET DRAFT

C. Perkins
IBM
31 May 1996

IP Encapsulation within IP
draft-ietf-mobileip-ip4inip4-03.txt

Status of This Memo

Abstract

This document specifies a method by which an IP datagram may be encapsulated (carried as payload) within an IP datagram. Encapsulation is suggested as a means to alter the normal IP routing for datagrams, by delivering them to an intermediate destination that would otherwise not be selected by the (network part of the) IP Destination Address field in the original IP header. Encapsulation may serve a variety of purposes, such as delivery of a datagram to a mobile node using Mobile IP.

1. Introduction

This document specifies a method by which an IP datagram may be encapsulated (carried as payload) within an IP datagram. Encapsulation is suggested as a means to alter the normal IP routing for datagrams, by delivering them to an intermediate destination that would otherwise not be selected based on the (network part of the) IP Destination Address field in the original IP header. Once the encapsulated datagram arrives at this intermediate destination node, it is decapsulated, yielding the original IP datagram, which is then delivered to the destination indicated by the original Destination Address field. This use of encapsulation and decapsulation of a datagram is frequently referred to as "tunneling" the datagram, and the encapsulator and decapsulator are then considered to be the "endpoints" of the tunnel.

In the most general tunneling case we have

```
source ---> encapsulator --------> decapsulator ---> destination
```

with the source, encapsulator, decapsulator, and destination being separate nodes. The encapsulator node is considered the "entry point" of the tunnel, and the decapsulator node is considered the "exit point" of the tunnel. There in general may be multiple source-destination pairs using the same tunnel between the encapsulator and decapsulator.

2. Motivation

The Mobile IP working group has specified the use of encapsulation as a way to deliver datagrams from a mobile node's "home network" to an agent that can deliver datagrams locally by conventional means to the mobile node at its current location away from home [8]. The use of encapsulation may also be desirable whenever the source (or an intermediate router) of an IP datagram must influence the route by which a datagram is to be delivered to its ultimate destination. Other possible applications of encapsulation include multicasting, preferential billing, choice of routes with selected security attributes, and general policy routing.

It is generally true that encapsulation and the IP loose source routing option [10] can be used in similar ways to affect the routing of a datagram, but there are several technical reasons to prefer encapsulation:

- There are unsolved security problems associated with the use of the IP source routing options.

- Current Internet routers exhibit performance problems when forwarding datagrams that contain IP options, including the IP source routing options.

- Many current Internet nodes process IP source routing options incorrectly.

- Firewalls may exclude IP source-routed datagrams.

- Insertion of an IP source route option may complicate the processing of authentication information by the source and/or destination of a datagram, depending on how the authentication is specified to be performed.

- It is considered impolite for intermediate routers to make modifications to datagrams which they did not originate.

These technical advantages must be weighed against the disadvantages posed by the use of encapsulation:

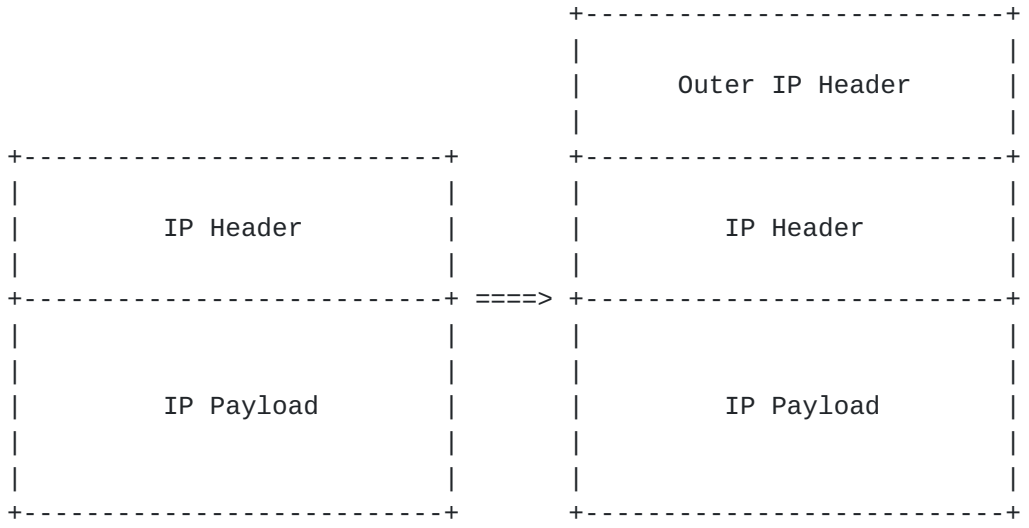
- Encapsulated datagrams typically are larger than source routed datagrams.

- Encapsulation cannot be used unless it is known in advance that the node at the tunnel exit point can decapsulate the datagram.

Since the majority of Internet nodes today do not perform well when IP loose source route options are used, the second technical disadvantage of encapsulation is not as serious as it might seem at first.

3. IP in IP Encapsulation

To encapsulate an IP datagram using IP in IP encapsulation, an outer IP header [10] is inserted before the datagram's existing IP header, as follows:



The outer IP header Source Address and Destination Address identify the "endpoints" of the tunnel. The inner IP header Source Address and Destination Addresses identify the original sender and recipient of the datagram, respectively. The inner IP header is not changed by the encapsulator, except to decrement the TTL as noted below, and remains unchanged during its delivery to the tunnel exit point. No change to IP options in the inner header occurs during delivery of the encapsulated datagram through the tunnel. If need be, other protocol headers such as the IP Authentication header [1] may be inserted between the outer IP header and the inner IP header. Note that the security options of the inner IP header MAY affect the choice of security options for the encapsulating (outer) IP header.

3.1. IP Header Fields and Handling

The fields in the outer IP header are set by the encapsulator as follows:

Version

IHL

The Internet Header Length (IHL) is the length of the outer IP header measured in 32-bit words [10].

TOS

The Type of Service (TOS) is copied from the inner IP header.

Total Length

The Total Length measures the length of the entire encapsulated IP datagram, including the outer IP header, the inner IP header, and its payload.

Identification, Flags, Fragment Offset

These three fields are set as specified in [10]. However, if the "Don't Fragment" bit is set in the inner IP header, it MUST be set in the outer IP header; if the "Don't Fragment" bit is not set in the inner IP header, it MAY be set in the outer IP header, as described in Section 5.1.

Time to Live

The Time To Live (TTL) field in the outer IP header is set to a value appropriate for delivery of the encapsulated datagram to the tunnel exit point.

Protocol

4

Header Checksum

The Internet Header checksum [10] of the outer IP header.

Source Address

The IP address of the encapsulator, that is, the tunnel entry point.

Destination Address

The IP address of the decapsulator, that is, the tunnel exit point.

Options

Any options present in the inner IP header are in general NOT copied to the outer IP header. However, new options specific to the tunnel path MAY be added. In particular, any supported types of security options of the inner IP header MAY affect the choice of security options for the outer header. It is not expected that there be a one-to-one mapping of such options to the options or security headers selected for the tunnel.

When encapsulating a datagram, the TTL in the inner IP header is decremented by one if the tunneling is being done as part of forwarding the datagram; otherwise, the inner header TTL is not changed during encapsulation. If the resulting TTL in the inner IP header is 0, the datagram is discarded and an ICMP Time Exceeded message SHOULD be returned to the sender. An encapsulator MUST NOT encapsulate a datagram with TTL = 0.

The TTL in the inner IP header is not changed when decapsulating. If, after decapsulation, the inner datagram has TTL = 0, the decapsulator MUST discard the datagram. If, after decapsulation, the decapsulator forwards the datagram to one of its network interfaces, it will decrement the TTL as a result of doing normal IP forwarding. See also Section 4.4.

The encapsulator may use any existing IP mechanisms appropriate for delivery of the encapsulated payload to the tunnel exit point. In particular, use of IP options is allowed, and use of fragmentation is allowed unless the "Don't Fragment" bit is set in the inner IP header. This restriction on fragmentation is required so that nodes employing Path MTU Discovery [7] can obtain the information they seek.

3.2. Routing Failures

Routing loops within a tunnel are particularly dangerous when they cause datagrams to arrive again at the encapsulator. Suppose a datagram arrives at a router for forwarding, and the router determines that the datagram has to be encapsulated before further delivery. Then:

- If the IP Source Address of the datagram matches the router's own IP address on any of its network interfaces, the router MUST NOT tunnel the datagram; instead, the datagram SHOULD be discarded.

- If the IP Source Address of the datagram matches the IP address of the tunnel destination (the tunnel exit point is typically

chosen by the router based on the Destination Address in the datagram's IP header), the router MUST NOT tunnel the datagram; instead, the datagram SHOULD be discarded.

See also Section 4.4.

4. ICMP Messages from within the Tunnel

After an encapsulated datagram has been sent, the encapsulator may receive an ICMP [9] message from any intermediate router within the tunnel other than the tunnel exit point. The action taken by the encapsulator depends on the type of ICMP message received. When the received message contains enough information, the encapsulator MAY use the incoming message to create a similar ICMP message, to be sent to the originator of the original unencapsulated IP datagram (the original sender). This process will be referred to as "relaying" the ICMP message from the tunnel.

ICMP messages indicating an error in processing a datagram include a copy of (a portion of) the datagram causing the error. Relaying an ICMP message requires that the encapsulator strip off the outer IP header from this returned copy of the original datagram. For cases in which the received ICMP message does not contain enough data to relay the message, see Section 5.

4.1. Destination Unreachable (Type 3)

ICMP Destination Unreachable messages are handled by the encapsulator depending upon their Code field. The model suggested here allows the tunnel to "extend" a network to include non-local (e.g., mobile) nodes. Thus, if the original destination in the unencapsulated datagram is on the same network as the encapsulator, certain Destination Unreachable Code values may be modified to conform to the suggested model.

Network Unreachable (Code 0)

An ICMP Destination Unreachable message SHOULD be returned to the original sender. If the original destination in the unencapsulated datagram is on the same network as the encapsulator, the newly generated Destination Unreachable message sent by the encapsulator MAY have Code 1 (Host Unreachable), since presumably the datagram arrived at the correct network and the encapsulator is trying to create the appearance that the original destination is local to that network even if it is not. Otherwise, if the encapsulator

returns a Destination Unreachable message, the Code field MUST be set to 0 (Network Unreachable).

Host Unreachable (Code 1)

The encapsulator SHOULD relay Host Unreachable messages to the sender of the original unencapsulated datagram, if possible.

Protocol Unreachable (Code 2)

When the encapsulator receives an ICMP Protocol Unreachable message, it SHOULD send a Destination Unreachable message with Code 0 or 1 (see the discussion for Code 0) to the sender of the original unencapsulated datagram. Since the original sender did not use protocol 4 in sending the datagram, it would be meaningless to return Code 2 to that sender.

Port Unreachable (Code 3)

This Code should never be received by the encapsulator, since the outer IP header does not refer to any port number. It MUST NOT be relayed to the sender of the original unencapsulated datagram.

Datagram Too Big (Code 4)

The encapsulator MUST relay ICMP Datagram Too Big messages to the sender of the original unencapsulated datagram.

Source Route Failed (Code 5)

This Code SHOULD be handled by the encapsulator itself. It MUST NOT be relayed to the sender of the original unencapsulated datagram.

4.2. Source Quench (Type 4)

The encapsulator SHOULD NOT relay ICMP Source Quench messages to the sender of the original unencapsulated datagram, but instead SHOULD activate whatever congestion control mechanisms it implements to help alleviate the congestion detected within the tunnel.

4.3. Redirect (Type 5)

The encapsulator MAY handle the ICMP Redirect messages itself. It MUST NOT not relay the Redirect to the sender of the original unencapsulated datagram.

4.4. Time Exceeded (Type 11)

ICMP Time Exceeded messages report (presumed) routing loops within the tunnel itself. Reception of Time Exceeded messages by the encapsulator MUST be reported to the sender of the original unencapsulated datagram as Host Unreachable (Type 3, Code 1). Host Unreachable is preferable to Network Unreachable; since the datagram was handled by the encapsulator, and the encapsulator is often considered to be on the same network as the destination address in the original unencapsulated datagram, then the datagram is considered to have reached the correct network, but not the correct destination node within that network.

4.5. Parameter Problem (Type 12)

If the Parameter Problem message points to a field copied from the original unencapsulated datagram, the encapsulator MAY relay the ICMP message to the sender of the original unencapsulated datagram; otherwise, if the problem occurs with an IP option inserted by the encapsulator, then the encapsulator MUST NOT relay the ICMP message to the original sender. Note that an encapsulator following prevalent current practice will never insert any IP options into the encapsulated datagram, except possibly for security reasons.

4.6. Other ICMP Messages

Other ICMP messages are not related to the encapsulation operations described within this protocol specification, and should be acted on by the encapsulator as specified in [9].

5. Tunnel Management

Unfortunately, ICMP only requires IP routers to return 8 octets (64 bits) of the datagram beyond the IP header. This is not enough to include a copy of the encapsulated (inner) IP header, so it is not always possible for the encapsulator to relay the ICMP message from the interior of a tunnel back to the original sender.

However, by carefully maintaining "soft state" about tunnels into which it sends, the encapsulator can return accurate ICMP messages to the original sender in most cases. The encapsulator SHOULD maintain at least the following soft state information about each tunnel:

- MTU of the tunnel (Section 5.1)
- TTL (path length) of the tunnel
- Reachability of the end of the tunnel

The encapsulator uses the ICMP messages it receives from the interior of a tunnel to update the soft state information for that tunnel. ICMP errors that could be received from one of the routers along the tunnel interior include:

- Datagram Too Big
- Time Exceeded
- Destination Unreachable
- Source Quench

When subsequent datagrams arrive that would transit the tunnel, the encapsulator checks the soft state for the tunnel. If the datagram would violate the state of the tunnel (for example, the TTL of the new datagram is less than the tunnel "soft state" TTL) the encapsulator sends an ICMP error message back to the sender of the original datagram, but also encapsulates the datagram and forwards it into the tunnel.

Using this technique, the ICMP error messages sent by the encapsulator will not always match up one-to-one with errors encountered within the tunnel, but they will accurately reflect the state of the network.

Tunnel soft state was originally developed for the IP Address Encapsulation (IPAE) specification [4].

5.1. Tunnel MTU Discovery

When the Don't Fragment bit is set by the originator and copied into the outer IP header, the proper MTU of the tunnel will be learned from ICMP Datagram Too Big (Type 3, Code 4) messages reported to the encapsulator. To support sending nodes which use Path MTU Discovery, all encapsulator implementations MUST support Path MTU Discovery [5, 7] soft state within their tunnels. In this particular application, there are several advantages:

- As a benefit of Path MTU Discovery within the tunnel, any fragmentation which occurs because of the size of the

encapsulation header is performed only once after encapsulation. This prevents multiple fragmentation of a single datagram, which improves processing efficiency of the decapsulator and the routers within the tunnel.

- If the source of the unencapsulated datagram is doing Path MTU Discovery, then it is desirable for the encapsulator to know the MTU of the tunnel. Any ICMP Datagram Too Big messages from within the tunnel are returned to the encapsulator, and as noted in Section 5, it is not always possible for the encapsulator to relay ICMP messages to the source of the original unencapsulated datagram. By maintaining "soft state" about the MTU of the tunnel, the encapsulator can return correct ICMP Datagram Too Big messages to the original sender of the unencapsulated datagram to support its own Path MTU Discovery. In this case, the MTU that is conveyed to the original sender by the encapsulator SHOULD be the MTU of the tunnel minus the size of the encapsulating IP header. This will avoid fragmentation of the original IP datagram by the encapsulator.

- If the source of the original unencapsulated datagram is not doing Path MTU Discovery, it is still desirable for the encapsulator to know the MTU of the tunnel. In particular, it is much better to fragment the original datagram when encapsulating, than to allow the encapsulated datagram to be fragmented. Fragmenting the original datagram can be done by the encapsulator without special buffer requirements and without the need to keep reassembly state in the decapsulator. By contrast, if the encapsulated datagram is fragmented, then the decapsulator must reassemble the fragmented (encapsulated) datagram before decapsulating it, requiring reassembly state and buffer space within the decapsulator.

Thus, the encapsulator SHOULD normally do Path MTU Discovery, requiring it to send all datagrams into the tunnel with the "Don't Fragment" bit set in the outer IP header. However there are problems with this approach. When the original sender sets the "Don't Fragment" bit, the sender can react quickly to any returned ICMP Datagram Too Big error message by retransmitting the original datagram. On the other hand, suppose that the encapsulator receives an ICMP Datagram Too Big message from within the tunnel. In that case, if the original sender of the unencapsulated datagram had not set the "Don't Fragment" bit, there is nothing sensible that the encapsulator can do to let the original sender know of the error. The encapsulator MAY keep a copy of the sent datagram whenever it tries increasing the tunnel MTU, in order to allow it to fragment and resend the datagram if it gets a Datagram Too Big response. Alternatively the encapsulator MAY be configured for

certain types of datagrams not to set the "Don't Fragment" bit when the original sender of the unencapsulated datagram has not set the "Don't Fragment" bit.

5.2. Congestion

An encapsulator might receive indications of congestion from the tunnel, for example, by receiving ICMP Source Quench messages from nodes within the tunnel. In addition, certain link layers and various protocols not related to the Internet suite of protocols might provide such indications in the form of a Congestion Experienced [6] flag. The encapsulator SHOULD reflect conditions of congestion in its "soft state" for the tunnel, and when subsequently forwarding datagrams into the tunnel, the encapsulator SHOULD use appropriate means for controlling congestion [3]; However, the encapsulator SHOULD NOT send ICMP Source Quench messages to the original sender of the unencapsulated datagram.

6. Security Considerations

IP encapsulation potentially reduces the security of the Internet, and care needs to be taken in the implementation and deployment of IP encapsulation. For example, IP encapsulation makes it difficult for border routers to filter datagrams based on header fields. In particular, the original values of the Source Address, Destination Address, and Protocol fields in the IP header, and the port numbers used in any transport header within the datagram, are not located in their normal positions within the datagram after encapsulation. Since any IP datagram can be encapsulated and passed through a tunnel, such filtering border routers need to carefully examine all datagrams.

6.1. Router Considerations

Routers need to be aware of IP encapsulation protocols in order to correctly filter incoming datagrams. It is desirable that such filtering be integrated with IP authentication [1]. Where IP authentication is used, encapsulated packets might be allowed to enter an organization when the encapsulating (outer) packet or the encapsulated (inner) packet is sent by an authenticated, trusted source. Encapuslated packets containing no such authentication represent a potentially large security risk.

IP datagrams which are encapsulated and encrypted [2] might also pose a problem for filtering routers. In this case, the router can

filter the datagram only if it shares the security association used for the encryption. To allow this sort of encryption in environments in which all packets need to be filtered (or at least accounted for), a mechanism must be in place for the receiving node to securely communicate the security association to the border router. This might, more rarely, also apply to the security association used for outgoing datagrams.

6.2. Host Considerations

Host implementations that are capable of receiving encapsulated IP datagrams SHOULD admit only those datagrams fitting into one or more of the following categories:

- The protocol is harmless: source address-based authentication is not needed.

- The encapsulating (outer) datagram comes from an authentically identified, trusted source. The authenticity of the source could be established by relying on physical security in addition to border router configuration, but is more likely to come from use of the IP Authentication header [1].

- The encapuslated (inner) datagram includes an IP Authentication header.

- The encapsulated (inner) datagram is addressed to a network interface belonging to the decapsulator, or to a node with which the decapsulator has entered into a special relationship for delivering such encapsulated datagrams.

Some or all of this checking could be done in border routers rather than the receiving node, but it is better if border router checks are used as backup, rather than being the only check.

7. Acknowledgements

Parts of Sections 3 and 5 of this document were taken from portions (authored by Bill Simpson) of earlier versions of the Mobile IP Internet Draft [8]. The original text for section 6 (Security Considerations) was contributed by Bob Smart. Good ideas have also been included from RFC 1853 [11], also authored by Bill Simpson. Thanks also to Anders Klemets for finding mistakes and suggesting improvements to the draft. Finally, thanks to David Johnson for going over the draft with a fine-toothed comb, finding mistakes, improving consistency, and making many other improvements to the draft.

References

[1] R. Atkinson. IP Authentication Header. RFC 1826, August 1995.

[2] R. Atkinson. IP Encapsulating Security Payload. RFC 1827, August 1995.

[3] F. Baker, Editor. Requirements for IP Version 4 Routers. RFC 1812, June 1995.

[4] R. Gilligan, E. Nordmark, and B. Hinden. IPAE: The SIPP Interoperability and Transition Mechanism. Internet Draft -- work in progress, March 1994.

[5] S. Knowles. IESG Advice from Experience with Path MTU Discovery. RFC 1435, March 1993.

[6] A. Mankin and K. Ramakrishnan. Gateway Congestion Control Survey. RFC 1254, August 1991.

[7] J. Mogul and S. Deering. Path MTU Discovery. RFC 1191, November 1990.

[8] C. Perkins, Editor. IPv4 Mobility Support. ietf-draft-mobileip-protocol-17.txt (work in progress), May 1996.

[9] J. B. Postel, Editor. Internet Control Message Protocol. RFC 792, September 1981.

[10] J. B. Postel, Editor. Internet Protocol. RFC 791, September 1981.

[11] W. Simpson. IP in IP Tunneling. RFC 1853, October 1995.

Author's Address

Questions about this memo can be directed to:

Charles Perkins
Room H3-D34
T. J. Watson Research Center
IBM Corporation
30 Saw Mill River Rd.
Hawthorne, NY 10532

Work: +1-914-784-7350
Fax: +1-914-784-6205
E-mail: perk@watson.ibm.com

The working group can be contacted via the current chair:

Jim Solomon
Motorola, Inc.
1301 E. Algonquin Rd.
Schaumburg, IL 60196

Work: +1-847-576-2753
E-mail: solomon@comm.mot.com