Network Working Group
Request For Comments 1241

R.A. Woodburn
SAIC
D.L. Mills
University of Delaware

July 1991

A Scheme for an Internet Encapsulation Protocol: Version 1

1. Status of this Memo

This memo defines an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

2. Glossary

Clear Datagram
The unmodified IP datagram in the User Space before Encapsulation.

Clear Header
The header portion of the Clear Datagram before Encapsulation. This header includes the IP header and possibly part or all of the next layer protocol header, i.e. the TCP header.

Decapsulation
The stripping of the Encapsulation Header and forwarding of the Clear Datagram by the Decapsulator.

Decapsulator
The entity responsible for receiving an Encapsulated Datagram, decapsulating it, and delivering it to the destination User Space. Delivery may be direct, or via Encapsulation. A Decapsulator may be a host or a gateway.

Encapsulated Datagram
The datagram consisting of a Clear Datagram prepended with an Encapsulation Header.

Encapsulation
The process of mapping a Clear Datagram to the Encapsulation Space, prepending an Encapsulation Header to the Clear Datagram and routing the Encapsulated Datagram to a Decapsulator.

Encapsulation Header
The header for the Encapsulation Protocol prepended to the Clear Datagram during Encapsulation. This header consists of an IP header followed by an Encapsulation Protocol Header.

Encapsulation Protocol Header
The Encapsulation Protocol specific portion of the Encapsulation Header.

Encapsulation Space
The address and routing space within which the Encapsulators and Decapsulators reside. Routing within this space is accomplished via Flows. Encapsulation Spaces do not overlap, that is, the address of any Encapsulator or Decapsulator is unique for all Encapsulation Spaces.

Encapsulator
The entity responsible for mapping a given User Space datagram to the Encapsulation Space, encapsulating the datagram, and forwarding the Encapsulated Datagram to a Decapsulator. An Encapsulator may be a host or a gateway.

Flow
Also called a "tunnel." A flow is the end-to-end path in the Encapsulation Space over which Encapsulated Datagrams travel. There may be several Encapsulator/Decapsulator pairs along a given flow. Note that a Flow does not denote what User Space gateways are traversed along the path.

Flow ID
A 32-bit identifier which uniquely distinguishes a flow in a given Encapsulator or Decapsulator. Flow IDs are specific to a single Encapsulator/Decapsulator Entity and are not global quantities.

Mapping Function
This is the function of mapping a Clear Header to a particular Flow. All encapsulators along a given Flow are required to map a given Clear Header to the same Flow.

User Address
The address or identifier uniquely identifying an entity within a User Space.

Source Route
A complete end-to-end route which is computed at the source and enumerates transit gateways.

User Space
The address and routing space within which the users reside. Routing within this space provides reachability between all address pairs within the space. User Spaces do not overlap, that is, a given User Address is unique in all User Spaces

3. Background

For several years researchers in the Internet community have needed a means of "tunneling" between networks. A tunnel is essentially a Source Route that circumvents conventional routing mechanisms. Tunnels provide the means to bypass routing failures, avoid broken gateways and routing domains, or establish deterministic paths for experimentation.

There are several means of accomplishing tunneling. In the past, tunneling has been accomplished through source routing options in the IP header which allow gateways along a given path to be enumerated. The disadvantage of source routing in the IP header is that it requires the source to know something about the networks traversed to reach the destination. The source must then modify outgoing packets to reflect the source route. Current routing implementations generally don't support source routes in their routing tables as a means of reaching an IP address, nor do current routing protocols.

Another means of tunneling would be to develop a new IP option. This option field would be part of a separate IP header that could be prepended to an IP datagram. The IP option would indicate information about the original datagram. This tunneling option has the disadvantage of significantly modifying existing IP implementations to handle a new IP option. It also would be less flexible in permitting the tunneling of other protocols, such as ISO protocols, through an IP environment. An even less palatable alternative would be to replace IP with a new networking protocol or a new version of IP with tunneling built in as part of its functionality.

A final alternative is to create a new IP encapsulation protocol which uses the current IP header format. By using encapsulation, a destination can be reached transparently without the source having to know topology specifics. Virtual networks can be created by tying otherwise unconnected machines together with flows through an encapsulation space.

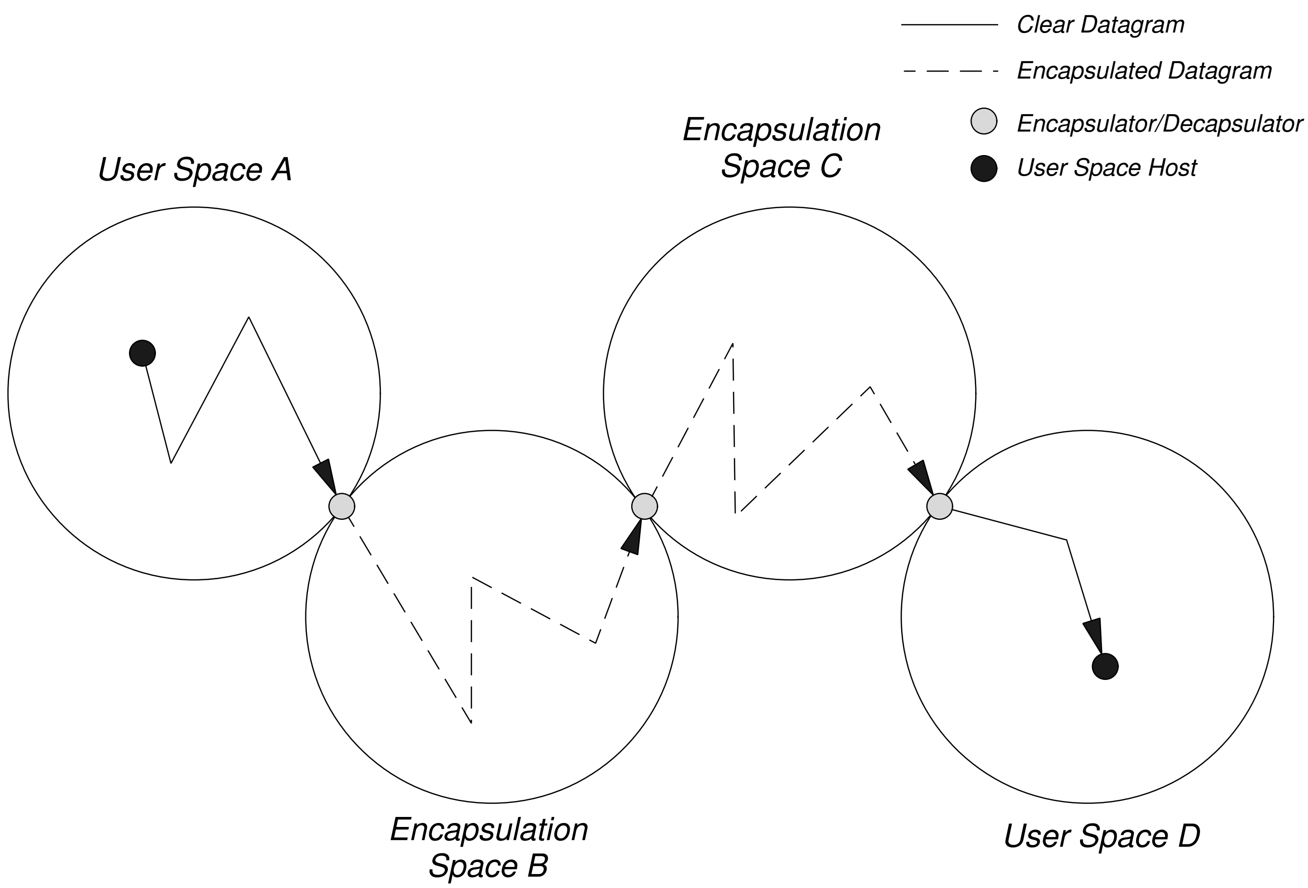

Fig. 1. Encapsulation Architectural Model

Up until now, there has been no standard for an encapsulation protocol. This RFC provides a means of performing encapsulation in the Internet environment.

4. Architecture and Approach

The architecture for encapsulation is based on two entities -- an Encapsulator and a Decapsulator. These entities and the associated spaces are shown in Fig. 1.

Encapsulators and Decapsulators have addresses in the User Spaces to which they belong, as well as addresses in the Encapsulation Spaces to which they belong. An encapsulator will receive a Clear Datagram from its User Space, and after determining that encapsulation should be used, perform a mapping function which translates the User Space information in the Clear Header to an Encapsulation Header. This Encapsulation Header is then prepended to the Clear Datagram to form the Encapsulated Datagram, as in Fig 2. It is desirable that the encapsulation process be transparent to entities in the User Space. Only the Encapsulator need know that encapsulation is occurring.

Encapsulating IP Header	Encapsulation Proto. Header	Clear Header	Remainder of Clear Datagram

Encapsulation Header	Clear Datagram

Fig. 2. Example of an Encapsulated Datagram

The Encapsulator forwards the datagram to a Decapsulator whose identity is determined at the time of encapsulation. The Decapsulator receives the Encapsulated Datagram and removes the Encapsulation Header and treats the Clear Datagram as if it were received locally. The requirement for the address of the Decapsulator is that it be reachable from the Encapsulator's Encapsulation Space address.

5. Generation of the Encapsulation Header

The contents of the Encapsulation Header are generated by performing a mapping function from the Clear Header to the contents of the Encapsulation Header. This mapping function could take many forms, but the end result should be the same. The following paragraphs describe one method of performing the mapping. The process is illustrated in Fig. 3.

In the first part of the mapping function, the Clear Header is matched with stored headers and masks to determine a Flow ID. This is essentially a "mask-and-match" table look up, where the lookup table holds three entries, a Clear Header, a header mask, and a corresponding Flow ID. The mask can be used for allowing a range of source and destination addresses to map to a given flow. Other fields, such as the IP TOS bits or even the TCP source or destination port addresses could also be used to discriminate between Flows. This flexibility allows many possibilities for using the mapping function. Not only can a given network be associated with a particular flow, but even a particular TCP protocol or connection could be distinguished from another.

How the lookup table is built and maintained is not part of this protocol. It is assumed that it is managed by some higher layer entity. It would be sufficient to configure the tables from ascii text files if necessary.

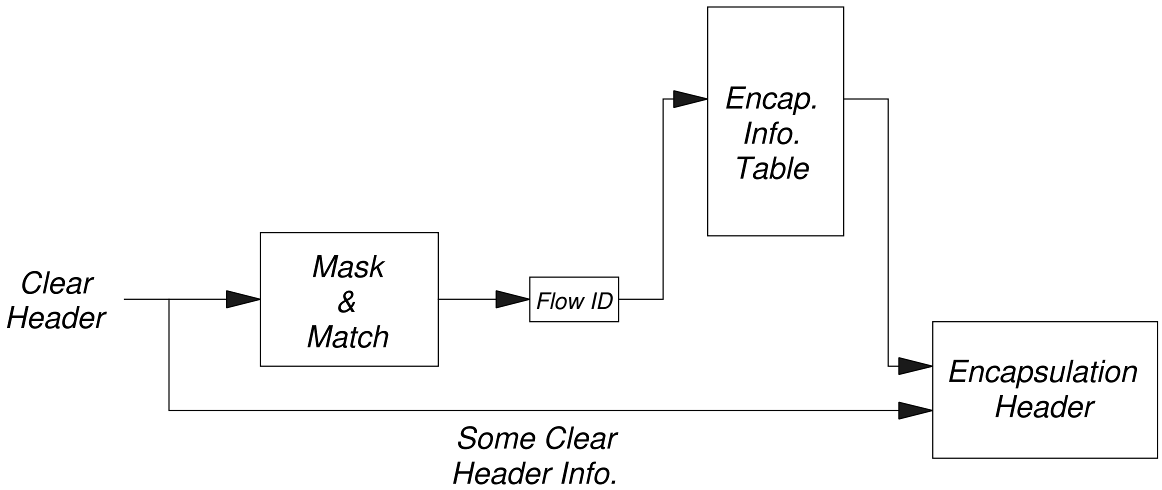

Fig. 3. Generation of the Encapsulation Header

The Flow IDs are managed at a higher layer as well. An example of how Flow IDs can be managed is found in the Setup protocol of the Inter-Domain Policy Sensitive Routing Protocol (IDPR). [4] The upper layer protocol would be responsible for maintaining information not carried in the encapsulation protocol related to the flow. This could include the information necessary to construct the Encapsulation Header (described below) as well as information such as the type of data being encapsulated (currently only IP is defined), and the type of authentication used if any. Note that IDPR Setup requires the use of a longer Flow ID which is unique for the entire universe of Encapsulators and is the same at every Encapsulator.

The Flow ID that results from the mapping of a Clear Header is a 32 bit quantity and identifies the Flow as it is seen by the Encapsulator. If a Clear Datagram must be encapsulated and decapsulated several times in order reach the destination, the Flow ID may be different at each Encapsulator, but need not be. The Flow ID acts as an index into a table of Encapsulation Header information that is used to build the Encapsulation Header. Note that the decision to make the Flow ID local to the Encapsulator is due to the difficulty in choosing and maintaining globally unique identifiers.

The intermediate step of using a Flow ID entirely optional. The important requirement is that all Encapsulators along a Flow map the same Clear Header to the same Flow (which could be identified by different identifiers along the way). However, by allowing for a Flow ID in the protocol, a more efficient implementation of the mapping function becomes possible. This is discussed in more detail when we consider the Decapsulator.

The following information is required to construct the Encapsulation Header:

Flow ID
This is the key for this table of information and represents the Flow ID relative to the current Encapsulator.

Decapsulator Address
The IP address of the Decapsulator in the Encapsulation Space must be known to build the IP portion of the Encapsulation Header.

Decapsulator's Flow ID
The Flow ID, if any, for the Flow as seen by the Decapsulator must be known.

Previous Encapsulator's Address
If this is not the first Encapsulator along the Flow, the previous Encapsulator's address must be known for error reporting.

Previous Encapsulator's Flow ID
In addition to the previous Encapsulator's address, the Flow ID of the Flow relative to the previous Encapsulator must be known.

The Encapsulation Header consists of an IP Header as well as an Encapsulation Protocol Header. The two pieces of information required for the Encapsulation Protocol Header which must be determined at the time of encapsulation are the protocol which is being encapsulated and the Flow ID to send to the Decapsulator. The generation of the IP header is more complicated.

There are two possible ways each field in the Clear Header could related to the new IP header.

Copy
Copy the existing field from the Clear Header to the IP header in the Encapsulation Header.

Ignore
The field may or may not have existed in the Clear Header, but does not apply to the new IP header.

The IP header has a fixed portion and a variable portion, the options list. A summary of all possible IP fields and the relation to the Clear Header follows in Table 1. [2]

Note that most of the fields in the Clear Header are simply ignored. Fields such as the Header Length in the Clear Header have no effect on the Header Length of the new IP header. The fields which are more interesting and require some thought are now discussed.

The Quality of Service bits should be copied from the Clear Header to the new IP header. This is in keeping with the transparency principle that if the User Space was providing a given service, then the Encapsulation Space must provide the same service..

The More Fragments bit and Fragment Offset should not be copied, since the datagram being built is a complete datagram, regardless of the status of the encapsulated datagram. If the completed datagram is too large for the interface, it will be fragmented for transmission to the decapsulator by the normal IP fragmentation mechanism.

The Don't Fragment bit should not be copied into the Encapsulation Header. The transparency principle would again be violated. It should be up to the Encapsulator to decide whether fragmentation should be allowed across the Encapsulation Space. If it is decided that the DF bit should be used, then ICMP message would be returned if the Encapsulated Datagram required fragmentation across the Encapsulation Space The mechanism for returning an ICMP message to the source in the User space will have to be modified, however, and this is discussed in the Appendix B.

Regarding the Time To Live (TTL) field, the easiest thing to do is to ignore the TTL from the Clear Header. If this field were copied from the Clear Header to the new IP header, the packet life might be prematurely exceeded during transit in the Encapsulation Space. This breaks the transparency rule of encapsulation as seen from the User Space. The TTL of the Clear Header is decremented before encapsulation by the IP forwarding function, so there is no chance of a packet looping forever if the links of a Flow form a loop.

Field	Mapping
Version	Ignore
Header Length	Ignore
Precedence	Copy
QoS bits	Copy
Total Length	Ignore
Identification	Ignore
Don't Fragment Bit	Ignore
More Fragments Bit	Ignore
Fragment Offset	Ignore
Time to Live	Ignore
Protocol	Ignore
Header Checksum	Ignore
Source Address	Ignore
Destination Address	Ignore
End of Option List	Ignore
NOP Option	Ignore
Security Option	Copy
LSR Option	Ignore
SSR Option	Ignore
RR Option	Ignore
Stream ID Option	Ignore
Timestamp Option	Ignore

Table 1. Summary of IP Header Mappings

The protocol field for the new IP header should be filled with the protocol number of the encapsulation protocol.

The source address in the new IP header becomes the IP address of the Encapsulator in the Encapsulation Domain. The destination address becomes the IP address of the Decapsulator as found in the encapsulation table.

IP Options are generally not copied because most don't make sense in the context of the Encapsulation Space, as the transparency principle would indicate. The security option is probably the one option that should get copied for the same reason QOS and precedence fields are copied, the Encapsulation Space must provide the expected service. Timestamp, Loose Source Route, Strict Source Route, and Record Route are not copied during encapsulation.

6. Decapsulation

In the ideal situation, a Decapsulator receives an Encapsulated Datagram, strips off the Encapsulation Header and sends the Clear Datagram back into IP so that it is forwarded from that point. However, if the Clear Datagram has not reached the destination User Space, it must again be encapsulated to move it close to the destination User Space. In this latter case the Decapsulator would become an Encapsulator and would perform the same calculation to generate the Encapsulation Header as did the previous Encapsulator. In order to make this process more efficient, the use of Flow IDs have been incorporated into the protocol.

When Flow IDs are used, the Flow ID received in the Encapsulation Header corresponds to a stored Flow ID in the Decapsulator. At this point the Decapsulator has the option of bypassing the mask and match operation on the Clear Header. The received Flow ID can be used to point directly into the local Encapsulator tables for the construction of the next Encapsulation Header. If the Flow ID is unknown, an error message is sent back to the previous Encapsulator to that effect and a signal is sent to upper layer entity managing the encapsulation tables.

Because the normal IP forwarding mechanism is being bypassed when Flow IDs are used, certain mechanisms normally handled by IP must be taken care of by the Decapsulator before encapsulation. The Decapsulator must decrement the TTL before the next encapsulation occurs. If a Time Exceeded error occurs, then an ICMP message is sent to the source indicated in the Clear Header.

7. Error Messages

There are two kinds of error message built into the encapsulation protocol. The first is used to report unknown flow identifiers seen by a Decapsulator and the second is for the forwarding of ICMP messages.

When a Decapsulator is using the received Flow ID in an Encapsulation Header to forward a datagram to the next Decapsulator in a Flow, it is possible that the Flow ID may not be known. For this case the Decapsulator will notify the previous Encapsulator that the Flow was not known so that the problem may be reported to the layer responsible for the programming of the Flow tables. This is accomplished through an encapsulation error message.

If an Encapsulator receives an ICMP messages regarding a given flow, this message should be forwarded backwards along the flow to the source Encapsulator. This is accomplished by the second kind of error message. The ICMP message will contain the Flow ID of the message which caused the error. This Flow ID must be translated to the Flow ID relative to the Encapsulator to which the error message is sent.

If an error occurs while sending any error message, no further error message are generated.

8. References

[1] J. Postel, *Internet Control Message Protocol*, RFC 792, September 1981.

[2] J. Postel, *Internet Protocol*, RFC 791, September 1981.

[3] J. Postel, *Transmission Control Protocol*, RFC 793, September 1981.

[4] ORWG, *Inter-Domain Policy Routing Protocol Specification and Usage*, Draft, August 1990

A. Packet Formats

This section describes the packet formats for the encapsulation protocol.

0	4	8	12	16 24 31
Vers	HL	MT	RC	Checksum
Flow ID				

Fig. A.1. Encapsulation Protocol Header Example

Vers	4 bits	The version number of the encapsulation protocol. The version of the protocol described by this document is 1.
HL	4 bits	The header length of the Encapsulation Protocol Header in octets.
MT	4 bits	The message type of the Encapsulation Protocol message. A data message has a message type of 1. An error message has a message type of 2.
RC	4 bits	The reason code. This field is unused in the Data Message and must have a value of 0. In the Error Message it contains the reason code for the Error Message. Defined reason code values are: 1 Unknown Flow ID 2 ICMP returned
Checksum	16 bits	A one's complement checksum for the Encapsulation Protocol Header. This field is set to 0 upon calculation of the checksum and is filled with the checksum calculation result before the data message is sent.
Flow ID	32 bits	The Flow ID as seen by the Decapsulator or Encapsulator to which this message is being sent. In the case of an Unknown Flow ID error, the Flow ID causing the error is used.

For Data Messages, the Encapsulation Protocol Header is followed by the Clear Datagram. For Error Messages, the header is followed by the ICMP message being forwarded along a flow.

B. Encapsulation and Existing IP Mechanisms

This section discusses in detail the effect of this encapsulation protocol upon the existing mechanisms available with IP and some the possible effects of IP mechanisms upon this protocol. Specifically these are Fragmentation and ICMP messages.

B.1 Fragmentation and Maximum Transmission Unit

An immediate concern of using an encapsulation mechanism is that of restrictions based upon MTU size. The source of a Clear Datagram is going to generate packets consistent with MTU of the interface over which datagram is transmitted. If these packets reach an Encapsulator and are encapsulated, they may be fragmented if they are larger than the MTU of the Encapsulator, even though the physical interfaces of the source and Encapsulator may have the same MTU. Because the Encapsulated Datagram is sent to the Decapsulator using IP, there is no problem in allowing IP to perform fragmentation and reassembly. However, fragmentation is known to be inefficient and is generally avoided. Because a new header is being prepended to the Clear Datagram by the encapsulation process, the likelihood of fragmentation occurring is increased. If the Encapsulator decides to disallow fragmentation through the Encapsulation Space, it must send an ICMP message back to the source. This means that the MTU of the interface in the encapsulation space is effectively smaller than that of the physical MTU of the interface.

Fragmentation by intermediate User Space Gateways introduces another problem. Fragmentation occurs at the IP level. If a TCP protocol is in use and fragmentation occurs, the TCP header is contained in the first fragment, but not the following fragments. [3] If these fragments are forwarded by an Encapsulator, discrimination of the Clear Header for a given flow will only be able to occur on the IP header portion of the Clear Header. If discrimination is attempted on the TCP portion of the header, then only the first fragment will be matched, while remaining fragments will not.

B.2 ICMP Messages

The most controversial aspect of encapsulation is the handling of ICMP messages. [1] Because the Encapsulation Header contains the source address of the Encapsulator in the Encapsulation Space, ICMP messages which occur within the Encapsulation Space will be sent back to the Encapsulator. Once the Encapsulator receives the ICMP message, the question is what should the next action be. Since the original source of the Clear Datagram knows nothing about the Encapsulation Space, it does not make sense to forward an ICMP message on to it and ICMP message are not supposed to beget ICMP messages. Yet not sending the original source something may break some important mechanisms.

In addition to deciding what to forward to the source of the Clear Datagram, there is the problem of possibly not having enough information to send anything at all back to the source. An ICMP message returns the header of the offending message and the first eight octets of the data after the header. For the case of the encapsulation protocol, this translates to the IP portion of the Encapsulation Header, the first eight octets of the Encapsulation Protocol Header, and nothing else. The contents of the Clear Datagram are completely lost. Therefore, for the Encapsulator to send an ICMP message back to the source it has to reconstruct the Clear Header. However, it is essentially impossible to reproduce the exact header.

For the purpose of this specification, the Flow ID has been assumed to be a unique one way mapping from a Clear Header. There is no guarantee that the Flow ID could be used to map back to the Clear Header, since several headers potentially map to the same flow. With there being no effective way to regenerate the original datagram, some compromises must be examined.

For each of the possible ICMP messages, the alternatives and impact will be assessed. There are three categories of ICMP message involved. The first is those ICMP messages which are not applicable in the context of Encapsulation. These are: Echo/Echo Reply and Timestamp/Timestamp Reply.

The second category are those ICMP messages which concern mechanisms local to the encapsulation domain. These are messages which would not make sense to the original source if it did receive them. In these cases the encapsulator will have to decide what to do, but no ICMP message need be sent back to the original source. The datagram will simply be lost, IP is not meant to be a reliable protocol. Subsequent messages received for encapsulation may cause the encapsulator to generate ICMP Destination Unreachable messages back to the original source if the encapsulator can no longer send messages to the destination decapsulator. This requires that ICMP messages inside the encapsulation domain affect the mapping from the Flow ID. ICMP messages in the second category are: Parameter Problem, Redirect, Destination Unreachable, Time Exceeded.

Finally there is one ICMP message which has direct bearing on the operation of the original source of datagrams destined for encapsulation, the ICMP Source Quench message. The only possible mechanism available to the Encapsulator to handle this message is for the source quench message set a flag for the offending Flow ID such that subsequent messages that map the Flow cause the generation of a source quench back to the original source before the datagram is encapsulated.

This last mechanism may be a solution for the more general problem. The rule of thumb could be that when an ICMP message is received for a given flow, then flag the Flow so that then next message encapsulated will cause the next message encapsulated on that flow to force an ICMP message to the source. After the ICMP message is sent to the source, the mechanism could be reset. This would effectively cause every other packet to receive an ICMP message if there were a persistent problem. This mechanism is probably only safe for Unreachable messages and Source Quench.

C. Reception of Clear Datagrams

In order to use the encapsulation protocol a modification is required to IP forwarding. There must be some way for the IP module in a system to pass Clear Datagrams to the encapsulation protocol. A suggested means of doing this is to make an addition to a system's routing table structures. A flag could be added to a route that tells the forwarding function to use encapsulation. Note that the default route could also be set to use encapsulation.

With this mechanism in place, a system's IP forwarding mechanism would examine its routing tables to try and match the IP destination to a specific route. If a route was found, it would be then checked to see if encapsulation should be used. If not the packet would be handled normally. If encapsulation was turned on for the route, then the datagram would be sent to encapsulation for forwarding.

In addition to snagging packets as they are forwarded, something must be done at the last Decapsulator on a given flow so that packets that are decapsulated are properly dumped into the IP module for delivery. Because the packets are encapsulated just before forwarding, it should be a simple matter for decapsulated datagrams to be injected into the output portion of IP. However, the source address in the Clear Header must not change. The address must remain the address of the source in the source User Space and not be overwritten with that of the Decapsulator.

D. Construction of Virtual Networks with Encapsulation

Because of the modification to the routing table to permit encapsulation, it becomes possible to specify a virtual interface whose sole purpose is encapsulation. Using this mechanism, it would become possible to link topologically distant entities with Flows. This would allow the construction of a Virtual Network which would overlay the actual routing topology. An example of such a virtual network is shown in Fig. 4.

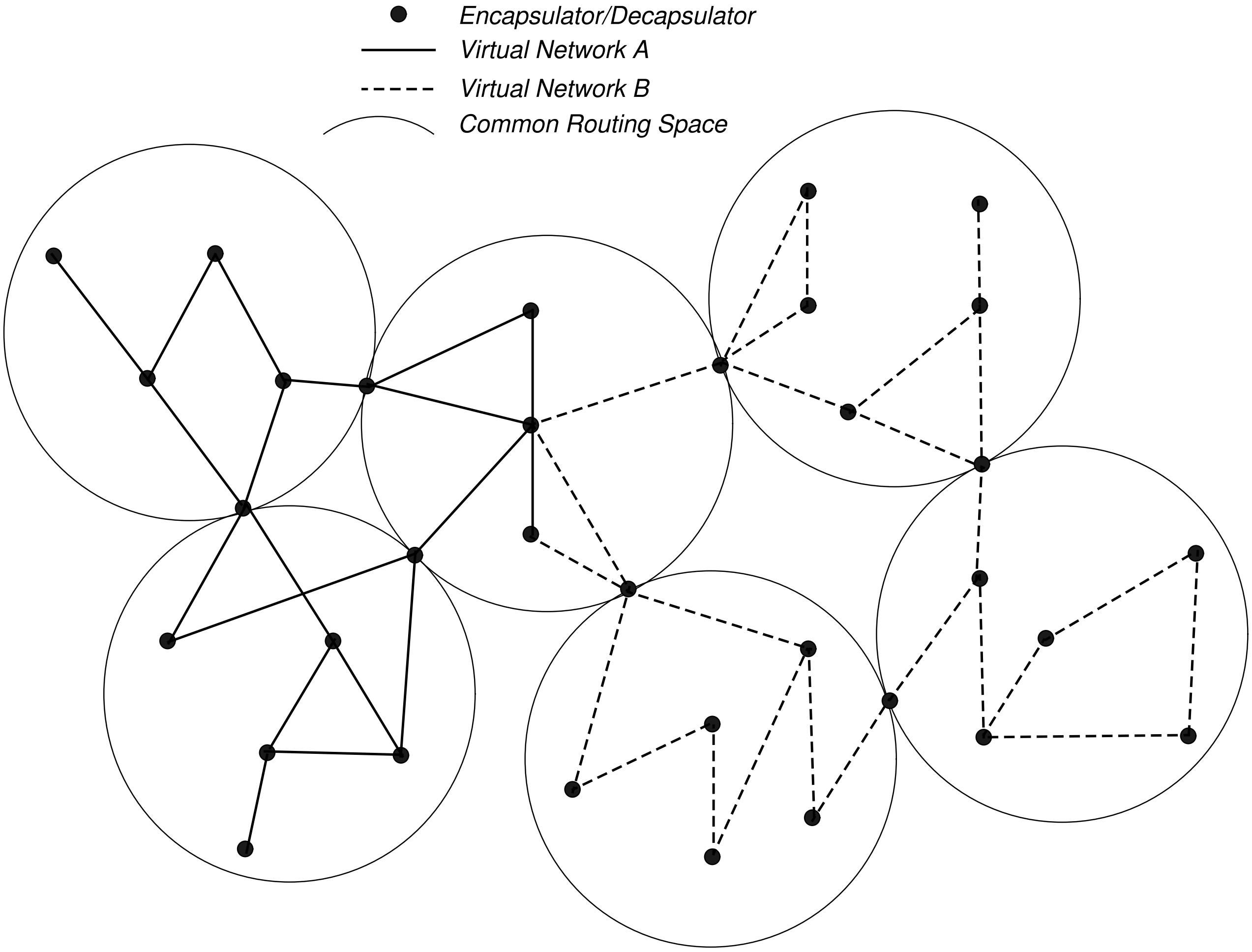

Fig. 4. Virtual Networks Example

Each Encapsulator shown has an virtual interface on one of the virtual networks. The lines represent individual links in the flows that connect each member of the virtual network. Note that new links could be added between any points as long as the two entities are visible to each other in a common Encapsulation Space. The routing within the virtual network would be handled by the encapsulation mechanism. The programming of the routing tables could be a variant of any of the currently existing routing protocols, an encapsulated OSPF for example.

With this in mind, it would be possible to have special encapsulation gateways with virtual interfaces on two virtual networks to form an entire virtual internet. This is the role of the Encapsulators joining Virtual Network A and Virtual Network B.

E. Encapsulation and OSI

It is intended that the encapsulation mechanism described in the memo be extensible to other environments outside of the Internet. It should be possible to encapsulate many different protocols within IP and IP within many other protocols.

The key concepts defined in this memo are the mapping of a header to a Flow ID and the mapping of fields in the original header to the encapsulating header. Special mappings between protocols would have to be defined, i.e. for the QoS bits, and some sort of translation of meanings carefully crafted, but it would be possible, none the less.

F. Security Considerations

No means of authentication or integrity checking is specifically defined for this protocol apart from the checksum for the header information. However for authentication or integrity checking to be used with this protocol, it is suggested that the authentication information be appended to the Encapsulated Datagram. Information regarding the type of authentication or integrity check in use would have to be included in the flow management protocol which is used to distribute the flow information.

G. Authors' Addresses

Robert A. Woodburn
SAIC
8619 Westwood Center Drive
Vienna, VA 22182

Phone: (703) 734-9000 or (703) 448-0210
EMail: woody@cseic.saic.com

David L. Mills
Electrical Engineering Department
University of Delaware
Newark, DE 19716

Phone: (302) 451-8247
EMail: mills@udel.edu