IP encapsulation within IP

1. Introduction

Encapsulation and decapsulation ≈ tunneling the datagram

Encapsulator:

- Entry point of the tunnel
- Kind of intermediate destination
- Adds an IP header to an IP datagram

Decapsulator:

- End point of the tunnel
- Decapsulates, yielding the original IP datagram
- Forwarded to the original destination (original DEST addr field)

SRC ➔ Encapsulator ➔Decapsulator ➔ DEST (all separate nodes)

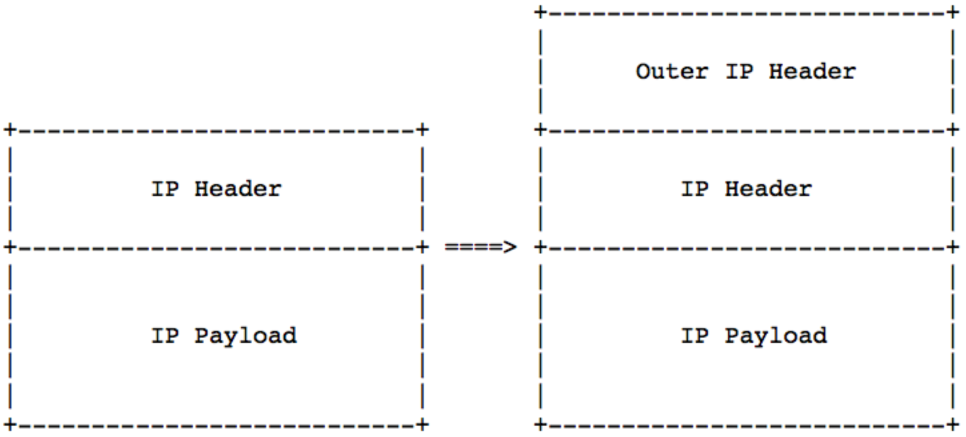
Multiple pairs (SRC-DEST) are possible for the same tunnel.

2. Motivation

Interesting information but not relevant for this project.

3. IP in IP encapsulation

General:



```
                                     +---------------------------+
                                     |                           |
                                     |      Outer IP Header      |
                                     |                           |
 +---------------------------+       +---------------------------+
 |                           |       |                           |
 |        IP Header          |       |        IP Header          |
 |                           |       |                           |
 +---------------------------+ ====> +---------------------------+
 |                           |       |                           |
 |                           |       |                           |
 |        IP Payload         |       |        IP Payload         |
 |                           |       |                           |
 |                           |       |                           |
 +---------------------------+       +---------------------------+
```

Outer IP header SRC address and DEST address ➔ tunnel-endpoints.

Inner IP header SRC address → original sender datagram.

Inner IP header DEST address → original recipient of the datagram.

Coding note:

Inner IP header is for the most part not modified by the encapsulator

- Only decrement the TTL of the inner IP header
- Remains unchanged during its delivery to the tunnel exit point.
- No changes to IP options.

    - Possible addition of IP authentication header (between inner and outer IP header)??
    - Security not needed for this project
- **<u>IP Header fields and handling</u>**
    - Version = 4
    - IHL = length of outer IP header (measured in 32 bit words)
    - TOS (TYPE OF SERVICE) = copied from inner header
    - Total length = length (Outer IP header + Inner IP header + payload)
    - Identification, flags and fragment offset
        - See rfc 791
        - Don’t fragment bit = 1 in inner IP header ➔ don’t fragment bit = 1 in outer IP header.
        - Don’t fragment bit = 0 (inner IP header) ➔ don’t fragment bit = 0 or 1 in outer IP header (cfr section 5.1).
    - TTL = value appropriate for delivery of the encapsulated datagram to the tunnel exit point
    - Protocol = 4
    - Header checksum = checksum of outer IP header
    - SRC = Address of the tunnel entry point (encapsulator)
    - DEST = Address of the tunnel exit point (decapsulator)
    - Options = options present in inner IP header →not copied to outer. New options specific to the tunnel maybe added. (security options of the inner may affect security options of the outer)

    <u>Coding note:</u>

    1) TTL inner IP header is decremented at the encapsulator and remains unchanged during forwarding in the tunnel. If resulting TTL is 0 ➔ discard datagram and send ICMP time exceeded message to sender. DON’T ENCAPSULATE A DATAGRAM WITH TTL 0.

    2) TTL inner header is not changed when decapsulating. If after decapsulation the TTL is 0 ➔ discard the datagram.

       If, after decapsulation, the decapsulator forwards the datagram to one of its network interfaces ➔ decrement TTL with 1.

       Encapsulator may use any existing IP mechanism for the datagram forwarding within the tunnel. (if use of IP option is allowed and don’t fragment bit = 0, when this is 1 ➔ fragmentation is not allowed).

- **Routing failures (→ datagrams arrive back at the encapsulator)**
  if IP SRC == router's own IP on any of its network interfaces ➔ don't tunnel the datagram but instead discard it
  if IP SRC == IP tunnel destination ➔ don't tunnel the datagram but instead discard it

4. ICMP messages from within the tunnel

After sending the encapsulated datagram, encapsulator may receive ICMP message from intermediate routers (in the tunnel).
→ take action depending the type of received message

if it contains enough information, encapsulator may create a similar ICMP message to send to the original sender of the datagram ➔ "relaying the message"
ICMP messages indicating an error carry a copy/portion of the original datagram which caused the error
Relaying = strip off the outer IP header and forward it to the sender

- **Destination unreachable (type 3)**
  Encapsulator receives these ICMP messages and they are handled according their code field:
  - Network unreachable (code 0)
    → should be returned to sender
    → if orig DEST is on same network as encapsulator, the new generated destination unreachable message may have code 1
    otherwise, send ICMP message with code 0.
  - Host unreachable (code 1)
    → encapsulator should relay this message to the original sender
  - Protocol unreachable (code 2)
    → encapsulator should send a Destination unreachable with code 0 or 1 (see above) to the sender. (sender didn't use protocol 4 in the original datagram so don't return code 2 to that sender)
  - Port unreachable (code 3)
    → should NEVER be received by the encapsulator (outer IP header doesn't refer to a port number)
    → DON'T relay it to the sender
  - Datagram too big (code 4)
    → relay message to the sender

    - Source route failed (code 5)
      → should be handled by the encapsulator itself
      → DON'T relay message to the sender
- **Source quench (type 4)**
  → should not relay the message to the sender, instead activate congestion control mechanisms to deal with the congestion in the tunnel.
- **Redirect (type 5)**
  → DON'T relay message to the sender
  → encapsulator may handle these messages itself
- **Time exceeded (type 11)**
  → routing loops in tunnel are detected
  → report these using a Host unreachable message (type 3, code 1) to the original sender
- **Parameter problem (type 12)**
  if it points to a field copied from the original unencapsulated message ➔ encapsulator may relay it to the sender
  otherwise, if it occurs due to an IP option inserted by the encapsulator ➔ don't relay it to the sender (unlikely to happen because encapsulator will never insert an option except for security which is not required for the project).
- **Other ICMP messages**
  Not related to this specification,for more information see rfc 792.

5. **Tunnel management**
   Encapsulator should maintain a soft state of the tunnel in order to provide accurate ICMP messages to the original sender:
   -) MTU of the tunnel
   -) TTL/ path length of the tunnel
   -) Reachability of the tunnel end
   Encapsulator uses the received ICMP messages to update this soft state. The possible incoming ICMP error messages are:
   -) Datagram too big
   -) Time exceeded
   -) Destination unreachable
   -) Source quench
   Subsequent datagram arrive @ encapsulator → check soft state of tunnel → in case of violation ➔ encapsulator sends an ICMP error message to the sender and ALSO encapsulate and send the datagram through the tunnel.

5.1 Tunnel MTU discovery

No messages are larger than the MTU in the project

5.2 Congestion (not needed for the project??)

Encapsulator receives source quench message or messages with congestion experienced bit 1→ probably congestion issue.

→ encapsulator should reflect these problems in the tunnels soft state and should use appropriate means (when subsequent datagrams are sent)

→ DON'T relay ICMP source quench message to the sender

6. Security considerations

Not needed for the project.

7. Acknowledgments

Not relevant for the project.