PROPOSED DCEC IP SPECIFICATION

prepared by

Mary M. Bernstein

System Development Corporation
2500 Colorado Avenue
Santa Monica, CA 90406
(213) 820-4111

Abstract

This document specifies the Internet Protocol (IP) which supports the interconnection of communication subnetworks. The document includes an introducion to IP with a model of operation, a definition of services provided to users, and a description architectural and environmental requirements. The protocol service interfaces and mechanisms are specified using an abstract state machine model.

FOREWORD

This document has been submitted to the DCEC for consideration as a standard specification of the Internet Protocol. The document incorporates the organization and specification techniques presented in the DCEC Protocol Specification Guidelines. This is a preliminary version; a revised version is to be released in December 1981. Any comments regarding completeness and consistency or suggestions for improvement of this document are welcomed.

Mary M. Bernstein

System Development Corporation
2500 Colorado Avenue
Santa Monica, CA 90406
(213) 820-4111

ARPANET: brnstein@dti

CONTENTS

1. OVERVIEW

This document specifies the Internet Protocol (IP) which supports the interconnection of communication subnetworks. The document introduces the Internet Protocol's role and purpose, defines the services provided to users, and specifies the mechanisms needed to support those services. This document also defines the services required of the lower protocol layer, describes the upper and lower interfaces, and outlines the operating system primitives needed for implementation. In addition, a glossary of terms and a set of appendices discussing certain aspects of IP are included. The reader is assumed to be familiar with the DCEC Architecture Report which presents a protocol architecture model for DoD communication services[1]. This document incorporates the organization and specification techniques presented in the DCEC Protocol Specification Guidelines[2].

The Internet Protocol is designed to interconnect packet-switched communication subnetworks to form an internetwork. IP transmits blocks of data, called internet datagrams, from sources to destinations throughout the internet. Sources and destinations are hosts located on either the same subnetwork or connected subnetworks. IP is purposely limited in scope to provide the basic functions necessary to deliver a block of data. Each internet datagram is an independent entity unrelated to any other internet datagram. IP does not create connections or logical circuits. IP has no mechanisms to promote data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

```
+------+ +-----+ +------+        +-----+
|Telnet| | FTP | | EFTP |  ...   |     |
+------+ +-----+ +------+        +-----+
     |   |         |                |
    +-----+     +-----+          +-----+
    | TCP |     | UDP |   ...    |     |
    +-----+     +-----+          +-----+
       |           |                |
    +-------------------------------+
    |     HOST INTERNET PROTOCOL    |
    +-------------------------------+
                   |
       +-----------------------+
       |  Subnetwork Protocol  |
       +-----------------------+
```

1. Example Host Protocol Hierarchy

This document specifies a host IP. In the DoD architectural model, a host IP resides between transport layer and the lower

network sublayer in each internet host. In each gateway, a site interconnecting two or more subnets, an IP resides above two or more subnetwork entities. In a gateway, IP is closely coupled to the Gateway to Gateway Protocol (GGP) to form a gateway IP. A gateway IP supports many of the same functions as a host IP, but provides additional services such as maintenance of current internet topology data. Throughout the remainder of the document, a host IP is simply referred to as IP.

```
              GATEWAY INTERNET PROTOCOL
      +-----------------------------------+
      |      Gateway-Gateway Protocol     |
      +- - - - - - - - - - - - - - - - - -+
      |         Internet Protocol         |
      +-----------------------------------+
         |            |                |
   +---------+   +---------+      +---------+
   |subnet 1 |   |subnet 2 |  ... |subnet i |
   | protocol|   | protocol|      | protocol|
   +---------+   +---------+      +---------+
        |            |                |
     ARPANET     local net        public net
```

2. Example Gateway Protocol Hierarchy

A protocol in an upper layer passes data to IP for delivery. IP packages the data as an internet datagram and passes it to the local subnetwork protocol for transmission across the local subnet. If the destination host is on the local subnet, IP sends the datagram through the subnet directly to that host. If the destination host is on a connected subnet, IP sends the datagram to an appropriate gateway. The gateway, in turn, sends the datagram through the next subnet to the destination host, or to another gateway.

Thus, datagrams move from one IP module to another through an interconnected set of subnetworks until the destination is reached. The sequence of IP modules handling the datagram in transit is called the gateway route. The gateway route is distinct from the lower level node-to-node route supplied by a particular subnetwork. The gateway route is based on the destination internet address. The IP modules share common rules for interpreting internet addresses to perform internet routing.

In transit, datagrams may traverse a subnetwork whose maximum packet size is smaller than the size of the datagram. To handle this, IP provides fragmentation and reassembly mechanisms. The gateway at the smaller-packet subnet fragments the original datagram into pieces, called datagram fragments, small enough for transmission. The IP module in the destination host reassembles

the datagram fragments to reproduce the original datagram.

IP can support a diverse set of upper layer protocols (ULPs). A transport protocol with real-time requirements, such as the Network Voice Protocol (NVP), can make use of IP's datagram service directly. A transport protocol providing ordered reliable delivery, such as TCP, can build additional mechanisms on top of IP's basic datagram service. Also, IP's delivery service can be customized in some ways to suit the special needs of an upper layer protocol. For example, a pre-defined gateway route, called a source route, can be supplied for an individual datagram. Each IP module forwards the datagram according to the source route instead of using only the independent routing mechanism.

The Internet Protocol shares a common history with the Transmission Control Protocol, first described in [3]. Later, IP and TCP were separated and specified as two distinct protocols in [4] and [5]. A wide range of technical, legal, and political issues associated with subnetwork interconnection is presented in [6]. Alternative approaches to internetting, including the CCITT Recommendation X.75 and the PUP architecture, are introduced and compared in in [7], [8], and [9]. Certain aspects of fragmentation and routing are are discussed in [10], and [11]. [12] and [13] contain current address mappings and assigned numbers used in network protocol implementations.

1.1 Scenario

The following scenario illustrates the model of operation for transmitting a datagram from one upper layer protocol to another. The scenario is purposely simple so that IP's basic operation is not obscured by the details of interface parameters or header fields.

A ULP in host A is to send data to a peer protocol in host B on another subnetwork. In this case, the source and destination hosts are on subnetworks directly connected by a gateway.

```
     HOST A                                     HOST B
 ----------------                          ------------------
 |   Sending    |                          |    Receiving    |
 | Upper Layer  |                          |    Upper Layer  |
 |  Protocol    |        GATEWAY           |     Protocol    |
 |      \       |   ------------------     |       /         |
 |    IP Module |   |   IP Module     |    |  IP Module      |
 |        \     |   |    /     \      |    |    /            |
 |         SNP-1 |  | SNP-1    SNP-2  |    | SNP-2           |
 ----------------   ------------------     ------------------
           \          /        \          /
          Local Subnet 1      Local Subnet 2

              Basic Model of Operation
```

a. The sending ULP passes its data to the IP module, along with the destination internet address and other parameters.

b. The IP module prepares an IP header and attaches the ULP's data to form an internet datagram. Then, the IP module determines a local subnetwork address from the destination internet address. In this case, it is the address of the gateway to the destination subnetwork. The internet datagram along with the local subnet address is passed to the local subnetwork protocol (abbreviated as SNP).

c. The SNP creates a local subnetwork header and attaches it to the datagram forming a subnetwork packet. The SNP then transmits the packet across the local subnet.

```
<========= subnetwork packet =========>

+------+---------+---------/ /--------+
|  *   |    *    |    *               |
+--:---+----:----+----:---/ /---------+
   :        :         :
   :        :         :
   :        :      ULP data
   :     IP header
   :
subnetwork header
```

d. The packet arrives at the gateway connecting the first and second subnetworks. The SNP of the first subnet strips off the local subnetwork header and passes the remainder to the IP module.

e. The IP module determines from the destination internet address in the IP header that the datagram is intended for a host in the second subnet. The IP module then derives a local subnetwork address for the destination host. That address is passed along with the datagram to the SNP of the second subnetwork for delivery.

f. The second subnet's SNP builds a local subnetwork header and appends the datagram to form a packet for the second subnetwork. That packet is transmitted across the second subnet to the destination host.

g. The SNP of the destination host strips off the local subnetwork header and hands the remaining datagram to the IP module.

h. The IP module determines that the datagram is bound for a ULP within this host. The data portion of the datagram, the source internet address, and other option parameters are passed to the ULP.

Delivery of data across the internet is complete.

2. DESCRIPTION OF SERVICES PROVIDED TO UPPER LAYERS

This section describes the services offered by the Internet Protocol to upper layer protocols (ULPs). The goals of this section are to provide the motivation for protocol mechanisms and a definition of the functions provided by this protocol.

The services provided by IP are:

o intact internet datagram service

o virtual network service

o error reporting service

A description of each service follows.

2.1 Datagram Service

The Internet Protocol shall provide a datagram service between homogeneous upper layer protocols in an internet environment. A datagram service is characterized by data delivery to the destination with non-zero probability; some data may possibly be lost. Also, a datagram service does not necessarily preserve the sequence in which data is supplied by the source upon delivery at the destination.

IP shall deliver received data to a destination ULP in the same form as sent by the source ULP. IP shall discard datagrams when insufficient resources are available for processing. IP does not detect datagrams lost or discarded by the subnetwork layer.

As part of the delivery service, IP insulates upper layer protocols from subnetwork specific characteristics. For example, IP maps internet addresses supplied by ULPs into local addresses used by the local subnetwork. Also, IP hides any packet-size restrictions of subnetworks along the transmission path within the internet.

2.2 Virtual Network Services

IP shall provide to upper layer protocols the ability to select virtual network service parameters. IP shall provide a standard command set for the ULPs to indicate the services desired. Thus, the ULPs can tune certain properties of IP and the underlying subnetworks to customize the transmission service according to their needs.

The virtual network parameters fall into two categories: service quality parameters and service options. Service quality

parameters influence the transmission service provided by the subnetworks; service options are additional functions provided by IP. A brief description of each follows:

o Service Quality Parameters

  - Precedence : attempts preferential treatment for high importance datagrams

  - Transmission Mode : datagram vs. stream. Stream mode attempts to minimize delay and delay dispersion through reservation of network resources

  - Reliability : attempts to minimize data loss and error rate

  - Speed : attempts prompt delivery

  - Resource Tradeoff : indicates relative importance of speed vs. reliability.

o Service Options

  - Security Labelling : identifies datagram for compartmented hosts

  - Source Routing : selects set of gateway IP modules to visit in transit

  - Route Recording : records gateway IP modules encountered in transit

  - Stream Identification : names reserved resources used for stream service

  - Time Stamping : records time information

  - Don't Fragment : marks a datagram as an indivisible unit

2.3 Error Reporting Service

IP shall provide error reports to the upper layer protocols indicating errors detected in providing the above services. In addition, certain errors detected by lower layer protocols shall be passed to the ULPs. These reports indicate several classes of errors including invalid arguments, insufficient resources, and transmission errors. The errors that IP must report to ULPs have not been defined. IP's error reporting responsibilities need further examination.

3. UPPER LAYER SERVICE/INTERFACE SPECIFICATION

This section specifies the IP services provided to upper layer protocols and the interface through which these services are accessed. The first part defines the interaction primitives and interface parameters for the upper interface. The second part contains the abstract machine specification of the upper layer services and interaction discipline.

3.1 Interaction Primitives

An interaction primitive defines the purpose and content of information exchanged between two protocol layers. Primitives are grouped into two classes based on the direction of information flow. Information passed downward, in this case from a ULP to IP, is called a service request primitive. Information passed upward, in this case from IP to a ULP, is called a service response primitive. Interaction primitives need not occur in pairs. That is, a request does not necessarily elicit a "response"; a "response" may occur independently of a request.

The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data is to be treated. The data is not examined or modified. The format of the parameters and data is implementation dependent and therefore not specified.

A given IP implementation may have slightly different interaction primitives imposed by the execution environment or system design factors. In those cases, the primitives can be modified to include more information or additional primitives can be defined to satisfy system requirements. However, all IPs must provide at least the interaction primitives specified below to guarantee that all IP implementations can support the same protocol hierarchy.

3.1.1 Service Request Primitives

A single service request primitive supports IP's datagram service, the SEND primitive.

3.1.1.1 SEND

The SEND primitive contains complete control information for each unit of data to be delivered. IP accepts in a SEND at least the following information:

o source address - internet address of ULP sending data

o destination address - internet address of ULP to receive data

o protocol - name of the recipient ULP

o type of service indicators - relative transmission quality associated with unit of data

- precedence - one of five levels : (P1, P2, P3, P4, P5) where P1 <= P2 <= P3 <= P4 <= P5

- reliability - one of four levels : (R1, R2, R3, R4) where R1 <= R2 <= R3 <= R4

- speed - one of two levels : (S1, S2) where S1 <= S2

- resource trade-off - speed or reliability : (T1, T2) where T1 = speed, T2 = reliability

- transmission mode - stream or datagram : (M1, M2) where M1 = stream, M2 = datagram

o identifier - value distinguishing this portion of data from others sent by this ULP.

o don't fragment indicator - flag showing whether IP can fragment data to accomplish delivery

o time to live - value in seconds indicating maximum lifetime of data within the internet

o data length - size of data being transmitted

o option data - options requested by ULP from following list: security, source routing, return routing, stream identification, or time stamp. (See section 6.2.14.)

o data - present when data length is greater than zero.

3.1.2 Service Response Primitives

A single service response primitive supports IP's datagram service, the DELIVER primitive.

3.1.2.1 DELIVER

The DELIVER primitive contains the data passed by a source ULP in a SEND, along with addressing, quality of service, and option information. IP passes in a DELIVER at least the following information:

o source address - internet address of sending ULP

o destination address - internet address of the recipient ULP

o protocol - name of recipient ULP as supplied by the sending ULP

o type of service indicators - relative transmission quality associated with unit of data

- precedence - one of five levels : (P1, P2, P3, P4, P5) where P1 <= P2 <= P3 <= P4 <= P5

- reliability - one of four levels : (R1, R2, R3, R4) where R1 <= R2 <= R3 <= R4

- speed - one of two levels : (S1, S2) where S1 <= S2

- resource trade-off - speed or reliability : (T1, T2) where T1 = speed, T2 = reliability

- transmission mode - stream or datagram : (M1, M2) where M1 = stream, M2 = datagram o data length - number of octets of received data (possibly zero)

o option data - options requested by source ULP from following list: security, source routing, return routing, stream identification, or time stamps. (See section 6.2.14.)

o data - present when data length is greater than zero.

In addition, a DELIVER may contain error reports from IP either together with parameters and data listed above, or, independently of that information. This area needs further examination and definition.

3.2 Abstract Machine Definition of Upper Level Services and Interaction Discipline

The abstract machine defines the behavior of the entire service machine from the perspective of the upper layer protocol. An abstract machine definition is composed of a machine identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

3.2.1 Machine Identifier

Each upper interface state machine is uniquely identified by the four values:

o source address

o destination address

o protocol

o identifier

3.2.2 State Diagram

The upper interface state machine has a single state which never changes. No diagram is needed.

3.2.3 State Vector

The upper interface state machine has a single state which never changes. No state vector is needed.

3.2.4 State Machine Data Structures

No data structures are used for the upper interface state machine. However, the following abbreviations are used to refer to parameters of the interaction primitives:

```
S = source internet address
D = destination internet address
P = protocol identifier
TOS = type of service where
          p(i) = one of the precedence levels (P1, P2, P3, P4, P5)
             (Note: read p(i) as "p sub i")
          r(i) = one of the reliability levels (R1, R2, R3, R4)
          s(i) = one of the speed levels (S1, S2)
          t(i) = one of the resource trade-offs (T1, T2)
          m(i) = one of the transmission modes (M1, M2)
ID = data identifier
TTL = time to live value
DF = don't fragment flag
Opts = the set of desired options including zero or more
       of the following:
          sr = source route
          rr = return route
          sl = security labelling
          sid = stream identifier
          its = internet timestamp
          sts = satellite timestamp
Data = data unit for delivery
L = length of data unit
t = time of event initiation
N = time elapsed during transmission
```

3.2.5 Event List

The events are drawn from the interaction primitives specified in section 3.1 above. An event is composed of a service primitive and an abstract timestamp to indicate the time of event initiation. The event list:

1. SEND(S, D, P, TOS(p(i), r(i), s(i), t(i), m(i)), ID, TTL, DF, Opts(sr, rr, sl, sid, its, sts), Data, L) at time t

2. NULL - Although no service request is issued by ULP, certain conditions within IP or lower layers produce a service response.

3.2.6 Events and Actions

The following section defines the set of possible actions elicited by each event.

```
EVENT = SEND( S, D, P, TOS(p(i), r(i), s(i), t(i), m(i)), ID, TTL, DF,
             Opt(sr, rr, sl, sid, its, sts), Data, L ) at time t

   Actions:

        1. DELIVER Data at time t+N to protocol P at destination D
           with all of the following properties:

               a. The time elapsed during data transmission satisfies
                  the time-to-live limit, i.e. N <= TTL.

               b. The quality of data transmission is at least
                  equal to the relative levels specified by
                  TOS(p(i), r(i), s(i), t(i), m(i)).

               c. if (DF = TRUE)
                  then IP fragmentation has not occurred in transit.

               d. if (Opts contains sr)
                  then Data has visited in transit at least the nodes
                        named by source route provided in SEND.

               e. if (Opts contains rr)
```

```
                  then the list of nodes visited in transit is
                       delivered with Data.

               f. if (Opts contains sl)
                  then the security label is delivered with Data.

               g. if (Opts contains sid)
```

```
                  then the stream identifier is delivered with Data

               h. if (Opts contains its)
                  then the internet timestamp is delivered with Data

               j. if (Opts contains sts)
                  then the satellite timestamp is delivered with Data

     OR,
       2. DELIVER to protocol P at source S indicating one of
          the following error conditions:

                a. destination D unreachable

                b. protocol P unreachable

                c. if (DF = TRUE)
                   then fragmentation needed but prohibited

                d. if (Opts contains any option)
                   then parameter problem with option.

     OR,
       3. no action



 EVENT = NULL

     Actions:

       1. DELIVER to protocol P at source S indicating
          the following error condition:

               a. error conditions in subnet layer
```

4. DESCRIPTION OF LOWER LAYER SERVICE REQUIREMENTS

This section describes the minimal services required of the sub-network layer.
The services required are:

o transparent data transfer between hosts within a subnetwork

o error reporting

A description of each service follows.

4.1 Data Transfer

The subnetwork layer must provide a transparent data transfer between hosts within a single subnetwork. Only the data to be delivered, and the necessary control and addressing information should be required as input from IP. Intranet routing and subnetwork operation shall be handled by the subnetwork layer itself.

The subnetwork need not be a reliable communications medium. Data should arrive with non-zero probability at a destination. Data may not necessarily arrive in the same order as it was supplied to the subnetwork layer, nor is data guaranteed to arrive error free.

4.2 Error Reporting

The subnetwork layer shall provide reports to IP indicating errors from the subnetwork and lower layers as feasible.
The specific error requirements of the subnetwork layer have not been defined. This area needs further examination.

5. LOWER LAYER SERVICE/INTERFACE SPECIFICATION

This section specifies the minimal subnetwork protocol services required by IP and the interface through which those services are accessed. The first part defines the interaction primitives and their parameters for the lower interface. The second part contains the abstract machine specification of the lower layer services and interaction discipline.

5.1 Interaction Primitives

An interaction primitive defines the purpose and contents of information exchanged between two protocol layers. Two kinds of primitives, based on the direction of information flow, are defined. Service requests pass information downward; service responses pass information upward. These primitives need not occur in pairs, nor in a synchronous manner. That is, a request does not necessarily elicit a "response"; a "response" may occur independently of a request.

The information associated with an interaction primitive falls into two categories: parameters and data. The parameters describe the data and indicate how the data is to be treated. The data is not examined or modified. The format of interaction primitive information is implementation dependent and so is not specified.

A given IP implementation may have slightly different interfaces imposed by the nature of the subnetwork or execution environment. Under such circumstances, the primitives can be modified to include more parameters or include additional primitives can be defined. However, all IPs must provide at least the interface specified below to guarantee that all IP implementations can support the same protocol hierarchy.

5.1.1 Service Request Primitives

A single service request primitive is required from the SNP, a SNP_SEND primitive.

5.1.1.1 SNP_SEND

The SNP_SEND contains an IP datagram, a destination, and parameters describing the desired transmission quality. The SNP receives in an SNP_SEND at least the following information:

o local destination address - local subnetwork address of destination host or gateway

o type of service indicators - relative transmission quality associated with the datagram

```
    - precedence - one of five levels : (P1, P2,  P3,  P4,  P5  )
                  where P1 <= P2 <= P3 <= P4 <= P5

    - reliability - one  of  four  levels  :  (R1,  R2,  R3,  R4)
                   where R1 <= R2 <= R3 <= R4

    - speed - one of two levels : (S1, S2) where S1 <= S2

    - resource trade-off  -  speed  or  reliability  :  (T1,  T2)
                          where T1 = speed, T2 = reliability

    - transmission mode - stream or datagram  :  (M1,  M2)  where
                         M1 = stream, M2 = datagram
```

o length - size of the datagram

o datagram

5.1.2 Service Response Primitives

One service request primitive is required to support IP's datagram service, the SNP_DELIVER primitive.

5.1.2.1 SNP_DELIVER

The SNP_DELIVER contains only a datagram which is an independent entity containing all the information needed by IP. An IP receives in an SNP_DELIVER at least the following information:

o datagram

In addition, a SNP_DELIVER may contain error reports from the SNP, either together with a datagram or independently of one. This area needs further examination and definition.

5.2 Abstract Machine Definition of Lower Level Services and Interaction Discipline

The abstract state machine defines the behavior of the entire service machine with respect to the lower layer protocol. An abstract machine definition is composed of a machine identifier, a state diagram, a state vector, a set of data structures, an event list, and an events and actions correspondence.

5.2.1 Machine Identifier

Each lower interface state machine is uniquely identified by the four values:

o source address

o destination address

o protocol

o identification

5.2.2 State Diagram

The lower interface state machine has a single state which never changes. No diagram is needed.

5.2.3 State Vector

No state vector is needed for the lower interface state machine.

5.2.4 State Machine Data Structures

No data structures are used for the lower interface state machine. However, the following abbreviations are used to refer to parameters of the interaction primitives:

```
     LD = local subnetwork destination address
     TOS = type of service where
               p(i) = one of the precedence levels (P1, P2, P3, P4, P5)
                  (Note: read p(i) as "p sub i")
               r(i) = one of the reliability levels (R1, R2, R3, R4)
               s(i) = one of the speed levels (S1, S2)
               t(i) = one of the resource trade-offs (T1, T2)
               m(i) = one of the transmission modes (M1, M2)
     L = datagram length
```

5.2.5 Event List

The events are drawn from the interactions primitives specified in section 5.1 above. An event is composed of a service primitive with its parameters and data.

1. SNP_SEND(LD, TOS(p(i), r(i), s(i), t(i), m(i)), L, Datagram)

2. NULL - Although IP issues no service request, certain conditions within the subnet layer elicit a service response.

5.2.6 Events and Actions

The following section defines the set of possible actions elicited by each event.

```
EVENT = SNP_SEND( LD, TOS(p(i), r(i), s(i), t(i), m(i)), L, Datagram)

     ACTIONS:

          1. SNP_DELIVER Datagram to IP at local destination LD with
             all of the following properties:

              a. The quality of data transmission is at least
                 equal to the relative levels specified by
                 TOS(p(i), r(i), s(i), t(i), m(i)).

        OR,
          2. no action


EVENT = NULL

     ACTIONS:

          1. SNP_DELIVER to IP indicating the following error condition:

               a. error conditions within the subnet layer
```

6. MECHANISM SPECIFICATION

This section defines the mechanisms supporting the services offered by IP. The first subsection motivates the specific mechanisms chosen and discusses the underlying philosophy of those choices. The second subsection defines the format and use of the IP header fields. The last subsection specifies the peer protocol interactions with a state machine model.

6.1 Overview of IP Mechanisms

The IP mechanisms are motivated by the IP services, described in section 2:

o intact datagram delivery service

o virtual network service

o error reporting service

Each service could be supported by any of a set of mechanisms. The selection of mechanisms is guided by design standards including simplicity, generality, flexibility, and efficiency. The following mechanism descriptions identify the service or services supported, discuss the design criteria used in selection, and explain how the mechanisms work.

6.1.1 Routing Mechanism

IP contains an adaptive routing mechanism to support the delivery service. The routing mechanism uses the internet addressing scheme and internet topology data to direct datagrams along the best path between source and destination. The mechanism provides routing options for ULPs needing the flexibility to select or record routes and routing information.

A distinction is made between names, addresses, and routes. A name indicates the object sought independently of physical location. An address indicates where the object is. A route indicates how to get there. It is the task of the upper layer protocols to map from names to addresses. The internet protocol maps from internet addresses to local subnet addresses to perform routing through the internet. It is the task of lower layer protocols to map from the local subnet addresses to routes through local subnetworks.

Internet addresses have a fixed length of four octets (32 bits). An internet address begins with a one octet network number and ends with a three octet REST field. The REST field usually contains a local subnetwork address. However, alternate schemes for use of this field can extend the address range to allow more subnetworks on the internet.

```
 0                   1                   2             3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1     8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/ /+-+-+-+-+
|    NETWORK    |        REST                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-/ /-+-+-+-+-+
```

The 24-bit REST field, assigned by each local subnetwork, should allow for a single physical host to act as several distinct internet hosts. That is, there should exist a mapping between internet host addresses and subnetwork/host interfaces that allows several internet addresses to correspond to one interface. A host should be allowed to have several physical interfaces (i.e. multi-homing) and to treat the datagrams from several of them as if they were all addressed to a single host.

To route a datagram, an IP module examines the NETWORK field of the internet address indicating the destination for the datagram. If the network number is the same as the IP module's subnetwork, the module uses the REST field of the internet address to derive the local subnet address of the destination host. If the network number doesn't match, the module determines a local subnet address of a gateway on the best path to the destination subnetwork. In turn, the gateway IP module derives the next local subnet address to either a host or gateway. In this way, the datagram is relayed through the internet to the destination host.

In a static environment, the routing algorithm is straightforward. However, internet topology tends to change due to hardware or software failure, host availability, or heavy traffic load conditions. So, each host and gateway IP along the gateway route also uses its current knowledge of internet topology to make routing decisions.

6.1.1.1 Routing Options

IP provides a mechanism, called source routing, to override the gateway's independent routing decisions. This mechanism allows an upper layer protocol to influence the gateway route a datagram traverses. The ULP can pass a list of internet addresses, called a source route, along with a datagram. Each address on the list is an intermediate destination. The source IP module uses its normal routing mechanism to transmit the datagram to the first address on the list. At that address, the first entry is removed from the list, and the next address becomes the datagram's destination. When the last entry of the list is removed, the normal destination address field becomes the final destination.

For testing or diagnostic purposes, a ULP can acquire a datagram's gateway route by using a mechanism called return routing. The sending ULP indicates that a record of the route is to be accumulated in transit. Then, as gateway IP module on the

gateway route relays the datagram, it adds its address to the return list. The destination ULP receives the original datagram along with the return list containing the gateway route.

6.1.2 Fragmentation and Reassembly

IP contains a fragmentation mechanism to break a large datagram into smaller datagrams. This is a more general solution for overcoming differences between subnetwork capacity than legislating a restrictive datagram size small enough for every subnetwork on the internet. This mechanism can be overriden using the "don't fragment" option to prevent fragmentation. IP also contains a reassembly mechanism which reverses the fragmentation to enable delivery of intact data portions.

When an IP module encounters a datagram that is too big to be transmitted through a subnetwork, it applies its fragmentation mechanism. First, the module divides the data portion of the datagram into two or more pieces. The data must be broken on 8-octet boundaries. For each piece, it then builds a datagram header containing the identification, addressing, and options information needed. Fragmentation data is adjusted in the new headers to correspond to the data's relative position within the original datagram. The result is a set of small datagrams, called fragments, each carrying a portion of the data from the original large datagram. Section 6.3.7.8 defines the fragmentation algorithm.

Each fragment is handled independently until the destination IP module is reached. The fragments may follow different gateway routes as internet topology and traffic conditions change. They are also subject to further fragmentation if 'smaller-packet' subnetworks are subsequently traversed.

Every IP module must be able to forward a datagram of 68 octets without further fragmentation. This size allows for a header length of up to 60 octets and the minimum data length of 8 octets.

To reassemble fragments into the original datagram, an IP module combines all those received having the same value for the identification, source address, destination address, and protocol. IP allocates reassembly resources when a "first-to-arrive" fragment is recognized. Based on the fragmentation data in the header, the fragment is placed in a reassembly area relative to its position in the original datagram. When all the fragments have been received, the IP module passes the data in its original form to the destination ULP.

All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams. The number 576 is selected to allow a reasonable amount of data to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximum internet header size is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of upper layer protocols.

Because the subnetwork may be unreliable, some fragments making up a complete datagram can be lost. IP uses the "time-to-live" data (explained in section 6.1.4 below) to set a timer on the reassembly process. If the timer expires before all the fragments have been collected, IP discards the partially reassembled datagram.

Only the destination IP module should perform reassembly. This recommendation is intended to reduce gateway overhead and minimize the chance of deadlock[3]. However, reassembly by private agreement between gateways is transparent to the rest of the internet and is allowed.

A ULP can prevent its data from being broken into smaller pieces during transmission. IP provides an override mechanism to prohibit fragmentation called "Don't Fragment." Any internet datagram marked "don't fragment" cannot be fragmented by an IP module along the gateway route under any circumstances. If an IP module cannot deliver such a datagram to its destination without fragmenting it, the module discards the datagram and returns an error to the source IP. (Please note that fragmentation, transmission, and reassembly at the subnetwork layer is transparent to IP and can be used at any time.)

6.1.3 Checksum

IP assumes the subnetwork layer to be unreliable regardless of the actual subnetwork protocol present. So, IP provides a checksum mechanism supporting the delivery service to protect the IP header from transmission errors. The data portion is not covered by the IP checksum.

An IP module recomputes the checksum each time the IP header is changed. Changes occur during time-to-live reductions, option updates (both explained below), and fragmentation. The checksum is currently a simple one's complement algorithm, and experimental evidence indicates its adequacy. However, the algorithm is provisional and may be replaced by a CRC procedure, depending on future experience.

6.1.4 Time To Live

As mentioned in the routing discussion above, a datagram's transmission path is subject to changes in internet topology and traffic conditions. Inadvertently, a datagram might be routed on a circuitous path to arrive at its destination after a considerable delay. Or, a datagram could loop through the same IP modules without making real progress towards its destination. Such "old datagrams" reduce internet bandwidth and waste processing time.

To prevent these problems, IP provides a mechanism to limit the lifetime of a datagram, called time-to-live. Along with the other sending parameters, a ULP specifies a maximum datagram lifetime in second units. Each IP module on the gateway route decreases the time-to-live value carried in the IP header. If an IP module receives an expired datagram, it discards the datagram. The lifetime limit is in effect until the datagram's data is delivered to the destination ULP. That is, if a datagram is fragmented during transmission, it can still expire during the reassembly process. Section 6.3.4.3 defines the reassembly algorithm use of the time-to-live data.

6.1.5 Type of Service

In support of the virtual network service, the type of service mechanism allows upper layer protocols to select the transmission quality. IP passes the type of service (TOS) command set for service quality to the SNP where it is mapped into subnetwork-specific transmission parameters. Not every subnetwork supports all transmission services, but each SNP on the delivery path should make a best effort to match the available subnet services to the desired service quality.

The TOS command set includes precedence level, reliability level, speed level, resource trade-off, and transmission mode. Several subnetworks offer a precedence service where treating high precedence traffic is processed before other traffic. A few networks offer a stream service, whereby one can achieve low delay and constant datagram interarrival time by reserving network resources. Another choice involves a low-delay vs. high reliability trade-off. Usually subnetworks invoke more complex and delay producing mechanisms as the need for reliability increases.

6.1.6 Data Options

Motivated by the virtual network service, IP provides a mechanism, called options, to carry certain identification and timing data in a standard manner through the internet. The use of this mechanism by the ULPs is optional, as the name implies, but all options must be supported by each IP implementation. No performance penalty is exacted from other services because the option

data requires no additional processing by IP; it is simply passed on to the receiving ULP.

The data options carry three kinds of information: security, stream identification, and timing. The security data is used by DOD hosts needing to transmit security and TCC (closed user groups) parameters through the internet in a standard manner. The stream identification option provides a way for a SATNET stream identifier to be carried both through stream-oriented subnetworks and subnets not supporting the stream concept. The timing data, useful for testing and diagnostics, includes internet timestamps and satellite timestamps.

6.1.7 Error Report Datagrams

The error reporting service motivates a mechanism to generate and process error information. The error mechanism uses the datagram delivery service to transfer the errors between IP modules.

An IP module encounters errors from three sources: ULPs, SNPs, and other IP modules. Errors from the first two appear across IP's interfaces and are handled on the same medium. But errors from IP modules are found in or caused by datagrams and are reported to the source IP module in an "error report datagram."

An error report datagram is composed of a minimal IP header and a data portion to carry error information. An error report datagram is distinguished from normal datagrams by the protocol field being equal to the Gateway-Gateway Protocol's identifier[13]. The error information includes a general error type, an error code, related error information, and portions of the discarded or erroneous datagram causing the report. The errors reported include:

a. Destination Unreachable - A gateway or host IP cannot deliver a datagram.

   - subnet unreachable - A gateway IP cannot determine a route to the destination subnetwork.

   - host unreachable - A gateway IP cannot determine a route to the destination host.

   - protocol unreachable - The IP module at the destination address cannot deliver to the unknown or inactive protocol.

   - port unreachable - The IP module at the destination address cannot deliver to the unknown or inactive port.

   - fragmentation needed but DF set - A host or gateway cannot deliver a datagram because fragmentation is required but is prohibited by the don't fragment flag.

b. Time Exceeded - A datagram has exceeded its allowed lifetime.

   - in transit - A gateway or host finds the time to live field in the datagram header is zero. The datagram is discarded.

   - during reassembly - The destination host cannot complete reassembly within the time-to-live time limit due to missing fragments.

c. Parameter Problem - A gateway or host cannot deliver the datagram because of some problem with the header parameters.

   - Problem with an option - An option is either not supported or incorrect. The third octet of the data portion contains the option type of the problem option.

d. Source Quench - A gateway has discarded a datagram due to congestion. This report request the source host to cut back the rate at which it is sending traffic to that destination.

e. Redirect - A gateway has received a datagram from a host on an attached subnet, but must forward it to another gateway on the same subnet. This message requests the source IP to send subsequent datagrams with the same destination to the other gateway. The address of the other gateway appears in octets 4-7.

f. Echo - A host or gateway may use this report to establish connectivity.

g. Type 0 : Echo Reply - A host or gateway responds to a previous Echo.

6.2 Internet Protocol Header Format

A summary of the contents of the IP header follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IP Header Format

Note that each tick mark represents one bit position. Each field description below includes its name, an abbreviation, and the field size. Where applicable, the units, the legal range of values, and a default value appears.

6.2.1 Version

```
  abbrev: VER       field size: 4 bits
```

The Version field indicates the format of the IP header. This document describes version 4.

6.2.2 Internet Header Length

```
  abbrev: IHL       field size: 4 bits
  units : 4-octets  range : 5 - 15      default : 5
```

Internet Header Length is the length of the IP header in 32-bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

6.2.3 Type of Service

```
  abbrev: TOS       field size: 8 bits
```

The Type of Service field contains the IP parameters describing the quality of service desired for this datagram.

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |     |           |     |     |
|   PRECEDENCE    | STRM|RELIABILITY| S/R |SPEED|
|                 |     |           |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+

Bits 0-2:  Precedence.
Bit    3:  Stream or Datagram.
Bits 4-5:  Reliability.
Bit    6:  Speed over Reliability.
Bits   7:  Speed.
```

PRECEDENCE	STRM	RELIABILITY	S/R	SPEED
111-Flash Override	1-STREAM	11-highest	1-speed	1-high
110-Flash	0-DTGRM	10-higher	0-rlblt	0-low
10X-Immediate		01-lower		
01X-Priority		00-lowest		
00X-Routine				

6.2.4 Total Length

```
    abbrev: TL           field size: 16 bits
    units: octets        range : 20 - 65,535    default: 20
```

Total Length is the length of the datagram, measured in octets, including header portion and the data portion of the datagram.

6.2.5 Identification

```
     abbrev: ID     field size : 16 bits
```

A identifying value used to associate fragments of a datagram. This value is usually supplied by the sending ULP as an interface parameter. If not, IP generates datagram identifications which are unique for each sending ULP.

6.2.6 Flags

```
     abbrev:  -     field size :  3 bits
```

This field contains the control flags "don't fragment" which prohibits IP fragmentation and "more fragments" which helps to identify fragments.

```
  0   1   2
+---+---+---+
|   | D | M |
| 0 | F | F |
        +---+---+---+
```

Bit 0: reserved, must be zero
Bit 1: Don't Fragment This Datagram (DF).
Bit 2: More Fragments Flag (MF).

6.2.7 Fragment Offset

```
     abbrev: FO                  field size : 13 bits
     units : 8-octet groups      range : 0 - 8191    default : 0
```

This field indicates the positions of this fragment's data relative to the beginning of the data carried in the original datagram. Both a complete datagram and a first fragment has this field set to zero. Section 6.1.2 describes the fragmentation mechanism.

6.2.8 Time to Live

```
     abbrev : TTL        field size : 8 bits
     units : seconds     range : 0 - 255(=4.25 mins) default : 15
```

This field indicates the maximum time the datagram is allowed to remain in the internet. If the value of this field drops to zero, the datagram should be destroyed. Section 6.1.4 describes the time-to-live mechanism.

6.2.9 Protocol

```
     abbrev : PROT       field size : 8 bits
```

This field indicates which ULP is to receive the data portion of the datagram. The numbers assigned to common ULPs are specified in [13].

6.2.10 Header Checksum

```
     abbrev :  -         field size : 16 bits
```

This field contains the checksum covering the IP header. The checksum mechanism is described in section 6.1.3.

6.2.11 Source Address

```
     abbrev : source     field size : 32 bits
```

This field contains the internet address of the datagram's source host. The first octet is the Source Network, and the following three octets are the Source Subnetwork Address. Internet addressing is discussed in section 6.1.1.

6.2.12 Destination Address

```
     abbrev : dest       field size : 32 bits
```

This field contains the internet address of the datagram's destination host. The first octet is the Destination Network, and the following three octets are the Destination Subnetwork Address.

Internet addressing is discussed in section 6.1.1.

6.2.13 Padding

abbrev : -   field size : variable (8 to 24 bits)

The IP header padding is used to ensure that the IP header ends on a 32-bit boundary. The padding field is set to zero.

6.2.14 Options

abbrev : -   field size : variable

The option field is variable in length depending on the number and type of options associated with the datagram. The options mechanisms are discussed in sections 6.1.1 and 6.1.6.

Options may have two possible formats:

a. a single octet of option-type, or

b. a variable length string containing:

   1. an option-type octet,

   2. an option-length octet - counting the option-type octet and option-length octet as well as the option-data octets, and

   3. the actual option-data octets.

The option-type octet is viewed as having 3 fields:

```
  0  1  2  3  4  5  6  7
+--+--+--+--+--+--+--+--+
|CF|CLASS|   NUMBER     |
+--+--+--+--+--+--+--+--+
```

```
bit  0   - copy flag, if set the option is copied into
             every fragment if fragmentation occurs.
bits 1-2 - option class
bits 3-7 - option number
```

The option classes are:

```
0 = control
1 = internet error
2 = debugging and measurement
3 = reserved for future use
```

The following internet options are defined:

COPY	CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	0	-	End of Option list: This option occupies only 1 octet; it has no length octet.
0	0	1	-	No Operation: This option occupies only 1 octet; it has no length octet.

1	0	2	4	Security: Used to carry Security, and user group (TCC) information compatible with DOD requirements.
1	0	3	var.	Source Routing: Used to route the datagram based on information supplied by the source.
0	0	7	var.	Return Route: Used to record the route a datagram takes.
1	0	8	4	Stream ID: Used to carry the stream identifier.
0	2	4	6	Internet Timestamp.
0	2	5	6	Satellite Timestamp.

6.2.15 Specific Option Definitions

Each option format is defined below. "Option type" indicates the value of the option-type octet, and "length" indicates the value of the length-octet if appropriate.

6.2.15.1 End of Option List

option type : 0   option length : N/A

This one octet option marks the end of the option list when it does not coincide with the four-octet boundary indicated by the IP header length. This field is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the IP header. This option may be introduced or deleted upon fragmentation as needed.

6.2.15.2 No Operation

option type : 1   option length : N/A

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary. This option may be introduced or deleted upon fragmentation as needed.

6.2.15.3 Security

option type : 130   option length : 4

This option provides a way for hosts to send security and TCC (closed user groups) parameters through subnetworks in a standard manner. The format for this option is as follows:

```
 0        1        2        3
 01234567 89012345 67890123 45678901
+--------+--------+--------+--------+
|Opt.Type| Length |xxxxxxSS|  TCC   |
+--------+--------+--------+--------+

    bits 0-7   : Option Type - described above
    bits 8-15  : Length - described above
    bits 16-21 : Not Used - must be zero
    bits 22-23 : SS - security, specifies security level:
                         11-top secret
                         10-secret
                         01-confidential
                         00-unclassified

    bits 24-31 : TCC - transmission control code, provides
                       a means to compartmentalize traffic
                       and define controlled communities
                       of interest among subscribers.
```

(This format is subject to change according to DOD requirements.)

6.2.15.4 Source Route

option type : 131   option length : variable

The source route option provides a means for the source ULP of a datagram to supply routing information to be used by IP modules along the gateway route.

The option begins with the option type code. The second octet is the option length which includes the option type octet, the length octet, and the source route list. A source route list is composed of one or more internet addresses. Each internet address is 4 octets long. When the source route list is empty, the option is removed from the datagram and the remaining routing is based on the destination address field. The source route option is described in section 6.1.1.

6.2.15.5 Return Route

option type : 7   option length : variable

The return route option provides a way to record a datagram's route.

The option begins with the option type code. The second octet is the option length which includes the option type code, the length octet, and the return route list. A return route list is composed of a series of internet addresses. The return route option is described in section 6.1.1.

6.2.15.6 Stream Identifier

option type : 136   option length : 4

This option provides a way for 16-bit stream identifier to be carried through the internet for use by subnetworks supporting the stream concept such as the SATNET.

6.2.15.7 Internet Timestamp

option type : 68   option length : 10

The internet address of the "stamper" is followed by a 32-bit time measured in milliseconds. Time zero is defined as January 1, 1980, GMT modulo 2^32 (=49.71 days).

6.2.15.8 Satellite Timestamp

option type : 69   option length : 10

The internet address of the "stamper" is followed by a 32-bit time measured in milliseconds. Time zero is defined as January 1, 1980, GMT modulo 2^32 (=49.71 days).

6.2.16 Error Report Datagrams

An error report datagram informs source IPs of erroneous or discarded datagrams. Such a datagram is identified by its protocol field being equal to the GGP identifier[13]. An error report datagram is made up of a minimal IP header and a data portion carrying error information. The first octet of the data portion contains the error type octet. The next two octets hold additional error information which depends on the error type. The fourth octet is not used. Beginning with the fifth octet, the erroneous datagram's header and first 64 data octets may appear.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type        |      Code         |               |  Not Used     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
\ Header + 64 data octets from erroneous or discarded datagram  \
\                                                               \
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Error Datagram Data Portion

The error types and related error information are defined below:

Type	Code	Description
3	0	Destination Unreachable due to unreachable subnetwork.
3	1	Dest. Unreachable due to unreachable host.
3	2	Dest. Unreachable due to unreachable or unknown protocol.
3	3	Dest. Unreachable due to unreachable or inactive port.
3	5	Dest. Unreachable due to fragmentation needed but prohibited by don't fragment flag.
11	0	Time Exceeded in transit.
11	1	Time Exceeded during reassembly.
12	0	Parameter Problem due to incorrect or unsupported option.
4	-	Source Quench due to congestion and discarded datagram in a gateway.
5	-	Redirect due to more direct path via alternate gateway. Octets 4-7 carry the address of the alternate gateway. The redirected datagram follows at octet 8.
8	-	Echo used to establish internet topology. No erroneous datagram carried in data portion.
0	-	Echo Reply responds to previous echo. No erroneous

datagram carried in data portion.

6.3 Extended State Machine Representation

The IP protocol mechanism is defined by a state machine representation made up of a set of states, a set of transitions between states, and a set of input events causing the state transitions. In addition, a state machine has an initial state whose values are assumed at state machine instantiation.

6.3.1 State Machine Identification

Each datagram is an independent unit. Therefore, one state machine instance exists for each datagram. Each datagram is uniquely named by the four values:

- o+ source address
- o+ destination address
- o+ protocol
- o+ identification

6.3.2 State Machine Diagram

The following diagram depicts a simplified IP state machine.

```
                 send or receive a complete datagram
                                __
                               |  |
                               |  v
                 *  *  *  *  *  *
               *                  *
              *      INACTIVE      *
               *                  *
                 *  *  *  *  *  *
                  |          ^
         receive  |          |receive complete datagram,
          fragment|          | or final fragment,
                  |          |  or reassembly timeout
                  v          |
                 *  *  *  *  *  *
               *                  *
              *    REASSEMBLING    *
               *                  *
                 *  *  *  *  *  *
                |  ^
                |__|
          receive
          fragments
```

6.3.3 State Vector

A state vector consists of the following elements :

o + STATE NAME = (inactive, reassembling)

o + REASSEMBLY RESOURCES = control information and storage needed to reassemble fragments into the original datagram, including:

- reassembly map : a representation of each 8-octet unit of data and its relative location within the original datagram.

- timer : value of the reassembly timer in unit seconds ranging from 0 to 255.

- total data length : size of the data carried in datagram being reassembled.

- header : storage area for the header portion of the datagram being reassembled.

- data : storage area for the data portion of the datagram being reassembled.

A state machine's initial state is inactive with unused reassem-bly resources.

6.3.4 State Machine Data Structures

The IP state machine references certain data areas corresponding to the state vector, and each interaction primitive : SEND, DELIVER, SNP_SEND, and SNP_DELIVER. For clarity in the events and actions section, data structures are declared in Ada for these data areas. However, a data structure may be partially typed or completely untyped where specific formats or data types are implementation dependent.

6.3.4.1 state_vector

The definition of an IP state vector appears section 6.3.1 above. A state_vector structure is declared as:

```
type state_vector_type is
  record
    state_name : (INACTIVE, REASSEMBLING);
    reassembly_map
    timer
    total_data_length
    header
    data
  end record;
```

6.3.4.2 from_ULP

The from_ULP structure holds the interface parameters and data associated with the SEND primitive, as specified in section 3.1.1. The from_ULP structure is declared as:

```
     type from_ULP_type is
       record
         source_addr
         destination_addr
         protocol
         type_of_service
         identifier
         time_to_live
         dont_fragment
         length
         data
         options
       end record;
```

6.3.4.3 to_ULP

The to_ULP structure holds interface parameters and data associated with the DELIVER primitive, as specified in section 3.1.2. The to_ULP structure is declared as:

```
     type to_ULP_type is
       record
         source_addr
         destination_addr
         protocol
         type_of_service
         length
         data
         options
         error
       end record;
```

6.3.4.4 from_SNP

The from_SNP structure holds the interface parameters and datagram associated with the SNP_DELIVER primitive, as specified in section 5.2.2. The from_SNP structure is declared as:

```
     type from_SNP_type is
       record
          local_destination_addr
          dtgm: datagram_type;
          error
       end record;
```

The dtgm element is itself a structure as specified below.

6.3.4.5 to_SNP

The to_SNP structure holds the data and parameters associated with the SNP_SEND primitive specified in section 5.2.1. The to_SNP structure is declared as:

```
     type to_SNP_type is
       record
         local_destination_addr
         type_of_service_indicators
         length
         dtgm: datagram_type;
       end record;
```

The dtgm element is itself a structure as specified below. specified below.

6.3.4.6 dtgm

A dtgm structure holds a datagram made up of a header portion and a data portion as specified in section 6.2. A dtgm structure is declared as:

```
type datagram_type is
   record
     version : HALF_OCTET;
     header_length : HALF_OCTET;
     type_of_service : OCTET;
     total_length : TWO_OCTETS;
     identification : TWO_OCTETS;
     dont_frag_flag : BOOLEAN;
     more_frag_flag : BOOLEAN;
     fragment_offset : ONE_N_FIVE_EIGHTHS_OCTETS;
     time_to_live : OCTET;
     protocol : OCTET;
     header_checksum : TWO_OCTETS;
     source_addr : FOUR_OCTETS;
     destination_addr : FOUR_OCTETS;
     options : option_type;
     data : array(1..DATA_LENGTH) of INTEGER;
  end record;

subtype HALF_OCTET is INTEGER range 0..15;
subtype OCTET is INTEGER range 0..255;
subtype ONE_N_FIVE_EIGHTHS_OCTETS is INTEGER range 0..8191;
subtype TWO_OCTETS is INTEGER range 0..65535;
subtype FOUR_OCTETS is INTEGER range 0..4294967296;
```

6.3.5 Event List

The following list defines the set of possible events in an IP state machine:

a. SEND from ULP - A ULP passes interface parameters and data to IP for delivery across the internet.

b. SNP_DELIVER from SNP - SNP passes to IP a datagram received from subnetwork protocol.

c. TIMEOUT - The timing mechanism provided by the execution environment indicates a previously specified time interval has elapsed.

6.3.6 State Machine Events and Actions

This section is organized in three parts. The first part contains a decision table representation of state machine events and actions. The decision tables are organized by state; each table corresponds to one event.

The second part specifies the decision functions appearing at the top of each column of a decision table. These functions examine attributes of the event and the state vector to return a set of decision results. The results become the elements of each column. A "--" element represents a "don't care" condition where a decision result does not determine the action procedure chosen.

The third section specifies action procedures appearing at the right of every row. Each row of the decision table combines the decision results to determine appropriate event processing. These procedures specify event processing algorithms in detail.

6.3.6.1 Events and Actions Decision Tables

STATE = INACTIVE

EVENT = SEND from ULP

ULP params valid	where to	need to frag	can frag	
NO	--	--	--	error to ULP(PARAM_PROBLEM)
YES	ULP	--	--	local delivery
YES	IP	--	--	**illegal
YES	REMOTE	NO	--	build&send
YES	REMOTE	YES	NO	error to ULP(CAN'T FRAGMENT)
YES	REMOTE	YES	YES	fragment&send

Comments:
A ULP passes data to IP for internet delivery. IP validates the interface parameters, determines the destination, and dispatches the ULP data to its destination.

STATE = INACTIVE

EVENT = DELIVER from SNP

check-sum valid	SNP params valid	TTL valid	where to	a frag	
NO	--	--	--	--	discard
YES	NO	--	--	--	error to source(PARAM_PROBLEM)
YES	YES	NO	--	--	error to source(EXPIRED_TTL)

YES	YES	YES	ULP	NO	remote delivery
YES	YES	YES	ULP	YES	reassemble;STATE:=REASSEMBLING
YES	YES	YES	IP	NO	analyze
YES	YES	YES	IP	YES	reassemble;STATE:=REASSEMBLING
YES	YES	YES	REMOTE	--	error to source(HOST_UNREACH)

Comments:
  The SNP has delivered datagram from another IP. IP validates the datagram header, and either delivers the data from a complete datagram to its destination within the host or begins reassembly for a datagram fragment.

STATE = REASSEMBLING
====================

EVENT = DELIVER from SNP

check-sum valid	SNP params valid	TTL valid	where to	a frag	reass. done	
NO	--	--	--	--	--	discard
YES	NO	--	--	--	--	error to source(PARAM_PROBLEM)
YES	YES	NO	--	--	--	error to source(EXPIRED_TTL)
YES	YES	YES	ULP	NO	--	remote delivery;state:=INACTIVE
YES	YES	YES	ULP	YES	NO	reassemble
YES	YES	YES	ULP	YES	YES	reass delivery;state:=INACTIVE
YES	YES	YES	IP	NO	--	analyze;state:=INACTIVE
YES	YES	YES	IP	YES	NO	reassemble
YES	YES	YES	IP	YES	YES	analyze;state:=INACTIVE
YES	YES	YES	REMOTE	--	--	error to source(HOST_UNREACH)

Comment:
  The SNP has delivered a datagram associated to an earlier received datagram fragment. IP validates the header and either continues the reassembly process with the datagram fragment or delivers the data from the completed datagram to its destination within the host.

EVENT = TIMEOUT

  reassembly timeout; state:=INACTIVE

Comment:
  The time-to-live period of the datagram being reassembled has elapsed.
  The incomplete datagram is discarded; the source IP is informed.

6.3.6.2 Decision Functions

The following functions examine information contained in interface parameters, interface data, and the state vector to make decisions. These decisions can be thought of as further refinements of the event and/or state. The functions return values represent the possible decisions.

6.3.6.2.1 checksum valid

The checksum_valid function examines an incoming datagram's header to determine whether it is free from transmission errors.

The data effects of this function are:

- Data examined only:

```
     from_SNP.dtgm.version                from_SNP.dtgm.fragment_offset
     from_SNP.dtgm.header_length          from_SNP.dtgm.time_to_live
     from_SNP.dtgm.type_of_service        from_SNP.dtgm.protocol
     from_SNP.dtgm.total_length           from_SNP.dtgm.source_addr
     from_SNP.dtgm.identification         from_SNP.dtgm.destination_addr
     from_SNP.dtgm.dont_frag_flag         from_SNP.dtgm.options
     from_SNP.dtgm.more_frag_flag         from_SNP.dtgm.padding
```

- Return values:

```
     NO   -- checksum did not check indicating header fields
               contain errors
     YES  -- checksum was consistent
```

The algorithm:

```
   --The checksum algorithm is the 16-bit one's complement
   --of the one's complement sum of all 16-bit words in
   --the IP header.  For purposes of computing the checksum,
   --the checksum field is set to zero.

       --implementation dependent action
```

6.3.6.2.2 ULP_params_valid

The ULP_params_valid function examines the interface parameters received from a ULP to see if all values are within legal ranges and desired options are supported.

The data effects of this function are:

- Data examined only:

```
          from_ULP.time_to_live
          from_ULP.options
```

- Return values:

  NO - some value is illegal or a desired option is not supported.

  YES - examine values are within legal ranges and desired
        options can be supported.

The algorithm:

```
     if (
          --The time-to-live value must be greater than zero to
          --allow IP to transmit it at least once.
          (from_ULP.time_to_live < 0 )

       or --The options requested should be checked for consistency.
          --implementation dependent action

       or --Check other implementation dependent values.
         )

     then return NO

     else return YES;

     end if;
```

6.3.6.2.3 SNP_params_valid

The SNP_params_valid function examines the interface parameters and the datagram received from the local subnetwork protocol to see if all values are within legal ranges and no errors have occured.

The data effects of this function are:

- Data examined only:

```
          from_SNP.dtgm.version          from_SNP.dtgm.source_addr
          from_SNP.dtgm.header_length    from_SNP.dtgm.destination_addr
          from_SNP.dtgm.total_length     from_SNP.dtgm.options
          from_SNP.dtgm.protocol         other information/errors from SNP
```

- Return values:

  NO - some value or values are illegal or an error has occured

  YES - examined values are within legal ranges and no
        errors have occured

The algorithm:

```
 if (  --The current IP header version number is 4.
     (from_SNP.dtgm.version /= 4)

       --The minimal IP header is 5 32-bit units in length.
   or (from_SNP.dtgm.header_length < 5)

       --The smallest legal datagram contains only a header and is
```

```
        --20 octets in length.
    or (from_SNP.dtgm.total_length < 20)

        --The legal protocol identifiers are provided in [13].
    or (from_SNP.dtgm.protocol is not one of the acceptable identifiers)

        --The legal network address mappings are provided in [12].
    or (from_SNP.dtgm.source_addr is not an acceptable address)

        --The legal network address mappings are provided in [12].
    or (from_SNP.dtgm.destination_addr is not an acceptable address)
       )

  then return NO

  elseif (any implementation dependent values received from the
           SNP are illegal or indicate error conditions)
      then return NO
      else return YES;   --Otherwise, all values look good.
      endif;
```

```
  endif;
```

6.3.6.2.4 where to

The where_to function determines the destination of the incoming datagram by examining the address fields and options fields of the datagram header.

The data effects of this function are:

- Data examined only:

```
        from_SNP.dtgm.destination_addr
        from_SNP.dtgm.protocol
        from_SNP.dtgm.options
```

- Return values:

```
        ULP - destination is an upper layer protocol at this location
        IP  - destination is this IP module
        REMOTE - destination is some remote location
```

The algorithm:

```
   --The source route influences the datagram's gateway route.

     if ((from_SNP.dtgm.options contains the source routing option)
     then return REMOTE;
     endif;

   --Examine the destination address field of the datagram header.

     if ((from_SNP.dtgm.destination_addr /= this site's address)
     then
         --It's destined for another site.
```

```
          return REMOTE
      else
          --It's destined for this site.
          if (from_SNP.dtgm.protocol = the IP identifier)
          then return IP
          else
              --Verify existence of addressed ULP at this location.
               if (from_SNP.dtgm.protocol exists here)
               then return ULP
               end if;
          end if;
      end if;
```

6.3.6.2.5 TTL valid

The TTL_valid function examines the IP header time-to-live field of an incoming datagram to determine whether the datagram has exceeded its allowed lifetime.

The data effects of this function are:

- Data examined only:

  from_SNP.dtgm.time_to_live

- Return values:

  NO - the datagram has expired
  YES - the datagram has some life left in it

The algorithm:

```
    --Decrement from_SNP.dtgm.time_to_live field by the maximum
    --of either the amount of time elapsed since the last IP module
    --handled this datagram (if known) or one second.

     if (( from_SNP.dtgm.time_to_live
           - maximum(number of seconds elapsed since last IP, 1))
          <= 0 )
     then return NO
     else return YES;
```

6.3.6.2.6 a frag

The a_frag function examines certain fields in an incoming datagram's header to determine whether the datagram is a fragment of a larger datagram.

The data effects of this algorithm are:

- Data examined only:

  from_SNP.dtgm.fragment_offset
  from_SNP.dtgm.more_frag_flag

- Return values:

```
          NO - the datagram has not been fragmented
          YES - the datagram is a part of a larger datagram
```

The algorithm:

```
   if ((from_SNP.dtgm.fragment_offset = 0)     --contains the beginning
       and (from_SNP.dtgm.more_frag_flag = 0)) --and the end of the data

  then return NO   --therefore it is an unfragmented datagram

  else return YES; --otherwise it contains only a portion of the data
                   --and is a fragment.
```

6.3.6.2.7 reassembly done

The reassembly_done function examines the incoming datagram and the reassembly resources to determine whether the final fragment has arrived to complete the datagram being reassembled.

The data effects of this function are:
Data examined only:

```
          state_vector.reassembly_map        from_SNP.dtgm.more_frag_flag
          state_vector.total_length          from_SNP.dtgm.header_length
          from_SNP.dtgm.fragment_offset      from_SNP.dtgm.total_length
```

- Return values:

```
          NO - more fragments are needed to complete reassembly
          YES - this is the only fragment needed to complete reassembly
```

The algorithm:

```
  --The total data length of the original datagram, as computed from
  --"tail" fragment, must be known before completion is possible.

    if (state_vector.total_data_length = 0)
    then
      --Check incoming datagram for "tail."

        if (from_SNP.dtgm.more_frag_flag = FALSE)
        then
          --Compute total data length and see if data in
          --this fragment fills out reassembly map.

            if (reassembly map from 0 to
                 (((from_SNP.dtgm.total_length -          --total data
                    (from_SNP.dtgm.header_length*4)+7)/8) --  length
                 +7)/8  is set )
            then return YES;
            end if;
        else
          --Reassembly cannot be complete if total data length unknown.
             return NO;
        end if;

     else --Total data length is already known.  See if data
          --in this fragment fills out reassembly map.

          if ( all reassembly map from 0 to
                  (state_vector.total_data_length+7)/8 is set)
          then return YES;  --final fragment
          else return NO;   --more to come
          end if;
     end if;
```

6.3.6.2.8 need to frag

The need_to_frag function examines the interface parameters and data from a ULP to determine whether the data can be transmitted as a single datagram or must be transmitted as two or more datagram fragments.

The data effects of this function are:

- Data examined only:

  from_ULP.length
  from_ULP.options

- Return values:

  NO - one datagram is small enough for the subnetwork
  YES - datagram fragments are needed to carry the data

The algorithm:

```
    --Compute the datagram's length based on the length of data,
    --the length of options, and the standard datagram header size.

     if (( from_ULP.length + (number of bytes of option data) + 20 )
             > maximum transmission unit of the local subnetwork )
     then return YES
     else return NO;
     end if;
```

6.3.6.2.9 can frag

The can_frag function examines the don't fragment flag of the interface parameters allows fragmentation.

The data effects of this function are:

- Data examined only:

  from_ULP.dont_fragment

- Return values:

  NO - don't fragment flag is set preventing fragmentation
  YES -don't fragment flag is NOT set to allow fragmentation

The algorithm:

```
     if (from_ULP.dont_fragment = TRUE)
     then return NO
     else return YES
     end if;
```

6.3.6.3 Decision Table Actions

6.3.6.3.1 compute_checksum

The compute_checksum procedure calculates a checksum value for a datagram header so that transmission errors can be detected by a destination IP.

The data effects of this procedure are:

- Data examined:

```
        to_SNP.dtgm.version                to_SNP.dtgm.fragment_offset
        to_SNP.dtgm.header_length          to_SNP.dtgm.time_to_live
        to_SNP.dtgm.type_of_service        to_SNP.dtgm.protocol
        to_SNP.dtgm.total_length           to_SNP.dtgm.source_addr
        to_SNP.dtgm.identification         to_SNP.dtgm.destination_addr
        to_SNP.dtgm.dont_frag_flag         to_SNP.dtgm.options
        to_SNP.dtgm.more_frag_flag         to_SNP.dtgm.padding
```

- Data modified:

```
        to_SNP.dtgm.checksum
```

The procedure:

```
    --The checksum algorithm is the 16-bit one's complement of
    --the one's complement sum of all 16-bit words
    --in the IP header.  For purposes of computing the checksum,
    --the checksum field is set to zero.

          --implementation dependent action
```

6.3.6.3.2 reassemble

The reassemble procedure reconstructs an original datagram from datagram fragments. The data effects of this procedure are:

- Data examined:

```
        from_SNP.dtgm
```

- Data modified:

```
        state_vector.reassembly_map             state_vector.header
        state_vector.timer                      state_vector.data
        state_vector.total_data_length
```

The procedure:

```
  --A local variable is introduced to make the computations more clear.
  --data_in_frag equals the number of octets of data in received fragment.

   data_in_frag := (from_SNP.dtgm.total_length
                               -from_SNP.dtgm.header_length*4);
  --Put data in its relative position in the data area of the state vector.

    state_vector.data[from_SNP.dtgm.fragment_offset*8..
                  from_SNP.dtgm.fragment_offset*8+data_in_frag] :=
                               from_SNP.dtgm.data[0..data_in_frag-1];
  --Fill in the corresponding entries of the reassembly map representing
  --each 8-octet unit of received data.

    for j in (from_SNP.dtgm.fragment_offset)..
             (from_SNP.dtgm.fragment_offset + data_in_frag + 7)/8) loop

      state_vector.reassembly_map[j] := 1;
    end loop;

  --Compute the total datagram length from the "tail-end" fragment.

    if (from_SNP.dtgm.more_frag_flag = FALSE)
    then state_vector.header.total_data_length :=
                       from_SNP.dtgm.fragment_offset*8 + data_in_frag;
    end if;

  --Record the header of the "head-end" fragment.

    if (from_SNP.dtgm.fragment_offset = 0)
    then state_vector.header := from_SNP.dtgm;
    end if;

  --Reset the reassembly timer if its current value is less than the
  --time-to-live field of the received datagram.
```

```
    state_vector.timer := maximum(
                    from_SNP.dtgm.time_to_live, state_vector.timer);
```

A reassembly algorithm may vary according to implementation concerns, but each one must meet these requirements:

1. Every destination IP module must have the capacity to receive a datagram 576 octets in length, either in one piece or in fragments to be reassembled.

2. The header of the fragment with from_SNP.dtgm.fragment_offset equal to zero (i.e. the "head-end" fragment) becomes the header of the reassembling datagram.

3. The total length of the reassembling datagram is calculated from the fragment with from_SNP.dtgm.more_frag_flag equal to zero (i.e. the "tail-end" fragment).

4. A reassembly timer is associated with each datagram being reassembled. The current recommendation for the initial timer setting is 15 seconds. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the subnetwork. That is, data rate multiplied by timer value equals reassembly capacity (e.g.10Kb/s X 15secs = 150Kb).

5. As each fragment arrives, the reassembly timer is reset to the maximum of state_vector.reassembly_resources.timer and from_SNP.dtgm.time_to_live in the incoming fragment.

6. If the reassembly timer expires, the datagram being reassembled is discarded. Also, an error datagram is returned to the source IP to report the "time exceeded during reassembly" error.

6.3.6.3.3 build&send

The build&send procedure builds an outbound datagram in the to_SNP structure from the interface parameters and data in from_ULP and passes it to the SNP for transmission across the subnet.

The data effects of this procedure are:

- Data examined:

```
from_ULP.source_addr                 from_ULP.time_to_live
from_ULP.destination_addr            from_ULP.dont_fragment
from_ULP.protocol                    from_ULP.options
from_ULP.type_of_service             from_ULP.length
from_ULP.identifier                  from_ULP.data
```

- Data modified:

```
to_SNP.dtgm          to_SNP.type_of_service_indicator
to_SNP.length        to_SNP.local_destination_addr
```

The algorithm:

```
 --Fill in each IP header field with information from from_ULP or
 --standard values.

     to_SNP.dtgm.version := 4;     --Current IP version is 4.
     to_SNP.dtgm.type_of_service := from_ULP.type_of_service;
     --If ID is not given by ULP, the IP must supply its own.
     to_SNP.dtgm.identification := from_ULP.identifier;

     to_SNP.dtgm.dont_frag_flag := from_ULP.dont_fragment;
     to_SNP.dtgm.more_frags_flag := 0;
     to_SNP.dtgm.fragment_offset := 0;
     to_SNP.dtgm.time_to_live := from_ULP.time_to_live;
     to_SNP.dtgm.protocol := from_ULP.protocol;
     to_SNP.dtgm.source_addr := from_ULP.source_address;
     to_SNP.dtgm.destination_addr := from_ULP.destination_address;
     to_SNP.dtgm.options := from_ULP.options;
     to_SNP.dtgm.padding := (as needed to end the IP header
                             four octet boundary);
     to_SNP.dtgm.header_length := 5 + (number of bytes of option data)/4;
     to_SNP.dtgm.total_length := (to_SNP.dtgm.header_length)*4
                                                  + (from_ULP.length);

 --Call compute_checksum to to compute and set the checksum.

     compute_checksum;

 --And, fill in the data portion of the datagram.
```

```
     to_SNP.dtgm.data[0..from_ULP.length -1]  := from_ULP.data[0..
                                                  from_ULP.length-1];
 --Set the type of service and length fields for the SNP.
```

```
     to_SNP.type_of_service_indicator := to_SNP.dtgm.type_of_service;
     to_SNP.length := to_SNP.dtgm.total_length;

 --Call the route procedure to determine a local destination
 --from the internet destination address supplied by the ULP.

     route;

 --Request the execution environment to pass the contents of to_SNP
 --to the local subnetwork protocol for transmission.

     TRANSFER to_SNP to the SNP.
```

NOTE: The format of the from_ULP elements is unspecified allowing an implementor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

6.3.6.3.4 route

The route procedure examines the destination address and options fields of an outbound datagram in to_SNP to determine a local destination address.

The data effects of this procedure are:

- Data examined:

  to_SNP.dtgm.destination_addr
  to_SNP.dtgm.options

- Data modified:

  to_SNP.local_destination_addr

The procedure:

```
  --The source routing option influences the path of the datagram.
  --If that option is present, use the top of the source route list
  --as the destination; otherwise use the header's destination
  --address field.

    if (source routing present in options)
    then destination := (first address on source route list)
    else destination := to_SNP.dtgm.destination_addr;
    endif;

    if (the network id field of destination matches the network id
           of the local subnet protocol )
    then
         --Translate the REST field of destination into the subnetwork
         --address of the destination on this subnet.
            --implementation dependent action
    else
         --Find the appropriate gateway and its subnetwork address
         --based on the network id field of the destination.
            --implementation dependent action
    end if;

  --Set the local destination interface parameter.

     to_SNP.local_destination_addr := (subnetwork address found above);
```

6.3.6.3.5 local delivery

The local_delivery procedure moves the interface parameters and data in the from_ULP structure to the to_ULP structure and delivers it to an in-host ULP.

The data effects of this procedure are:

- Data examined:

```
        from_ULP.destination_addr          from_ULP.length
        from_ULP.source_addr               from_ULP.data
        from_ULP.protocol                  from_ULP.options
        from_ULP.type_of_service
```

- Data modified:

```
        to_ULP.source_addr                 to_ULP.length
        to_ULP.destination_addr            to_ULP.data
        to_ULP.protocol                    to_ULP.options
        to_ULP.type_of_service
```

The procedure:

```
  --Move the interface parameters and data from the input
  --structure, from_ULP, directly to the output structure, to_ULP,
  --for delivery to a local ULP.

     from_ULP.destination_addr := to_ULP.destination_addr;
     from_ULP.source_addr      := to_ULP.source_addr;
     from_ULP.protocol         := to_ULP.protocol;
     from_ULP.type_of_service  := to_ULP.type_of_service;
     from_ULP.length           := to_ULP.length;
     from_ULP.data             := to_ULP.data;
     from_ULP.options          := to_ULP.options;

 --Request the execution environment to pass the contents of to_SNP
 --to the local subnet protocol for transmission.

     TRANSFER to_ULP to to_ULP.protocol.
```

6.3.6.3.6 remote delivery

The remote_delivery procedure decomposes a datagram arriving from a remote IP into interface parameters and data and delivers them to the destination ULP.

The data effects of this procedure are:

- Data examined:

```
        from_SNP.dtgm.source_addr          from_SNP.dtgm.total_length
        from_SNP.dtgm.destination_addr     from_SNP.dtgm.header_length
        from_SNP.dtgm.protocol             from_SNP.dtgm.data
        from_SNP.dtgm.type_of_service      from_SNP.dtgm.options
```

- Data modified:

```
     to_ULP.destination_addr       to_ULP.length
     to_ULP.source_addr            to_ULP.data
     to_ULP.protocol               to_ULP.options
     to_ULP.type_of_service
```

The algorithm:

```
to_ULP.destination_addr  :=  from_SNP.dtgm.destination_addr;
to_ULP.source_addr       :=  from_SNP.dtgm.source_addr;
to_ULP.protocol          :=  from_SNP.dtgm.protocol;
to_ULP.type_of_service   :=  from_SNP.dtgm.type_of_service;
to_ULP.length            :=  from_SNP.dtgm.total_length -
                                from_SNP.dtgm.header_length*4;
to_ULP.data              :=  from_SNP.dtgm.data;
to_ULP.options           :=  from_SNP.dtgm.options;
```

NOTE: The format of the to_ULP elements is unspecified allowing an implementor to assign data types for the interface parameters. If those data types differ from the IP header types, the assignment statements above become type conversions.

6.3.6.3.7 reassembled delivery

The reassembled_delivery procedure decomposes the datagram that has been reassembled in the state vector into interface parameters and data, then delivers them to a ULP.

The data effects of this procedure are:

- Data examined:

```
     state_vector.header.destination_addr
     state_vector.header.source_addr
     state_vector.header.protocol
     state_vector.header.type_of_service
     state_vector.header.header_length
     state_vector.header.total_length
     state_vector.header.options
     state_vector.data
```

- Data modified:

```
     to_ULP.destination_addr       to_ULP.length
     to_ULP.source_addr            to_ULP.data
     to_ULP.protocol               to_ULP.options
     to_ULP.type_of_service
```

The procedure:

```
to_ULP.destination_addr  := state_vector.header.destination_addr;
to_ULP.source_addr       := state_vector.header.source_addr;
to_ULP.protocol          := state_vector.header.protocol;
to_ULP.type_of_service   := state_vector.header.type_of_service;
to_ULP.length            := state_vector.header.total_length;
                              - state_vector.header.header_length*4;
to_ULP.options           := state_vector.header.options;
to_ULP.data              := state_vector.data;
```

6.3.6.3.8 fragment&send

The fragment&send procedure breaks data that is too big to be transmitted through the subnetwork as a single datagram into smaller pieces for transmission in several datagrams.

The data effects of the procedure are:

- Data examined only:

```
     from_ULP.source_addr                from_ULP.length
     from_ULP.destination_addr           from_ULP.data
     from_ULP.protocol                   from_ULP.options
     from_ULP.type_of_service
```

- Data modified:

```
     to_SNP.dtgm          to_SNP.type_of_service_indicators
     to_SNP.length        to_SNP.local_destination_address
```

Some "local variables" are used to make the procedure clearer:

```
number_of_fragments -- number of small datagrams created from
                       user data

data_per_fragment -- the number of octets in each small datagram

number_frag_blocks -- the number of 8-octet blocks in each small
                      datagram
data_in_last_frag -- the number of octets in the last datagram
```

The procedure:

```
  --Compute the fragmentation variables.

  --The amount of data per fragment equals the max datagram size less
  --the length of the datagram header.
     data_per_fragment   := maximum subnet transmission unit
                             - (20 + number of bytes of option data);

     number_frag_blocks  := data_per_fragment/8;

     number_of_fragments := (from_ULP.length + (data_per_fragment-1))
                             / data_per_fragment;

     data_in_last_frag := from_ULP.length modulo data_per_fragment;

  --Create the first fragment and transmit it to the SNP.

     to_SNP.dtgm.version := 4;
     to_SNP.dtgm.header_length := 5 + (number bytes of option data/4);
     to_SNP.dtgm.total_length := to_SNP.dtgm.header_length
```

```
                                             + data_per_fragment;
     to_SNP.dtgm.identifier := from_ULP.identification;
     to_SNP.dtgm.dont_frag_flag := from_ULP.dont_fragment;
     to_SNP.dtgm.more_frag_flag := TRUE;
```

```
      to_SNP.dtgm.fragment_offset := 0;
      to_SNP.dtgm.time_to_live := from_ULP.time_to_live;
      to_SNP.dtgm.protocol := from_ULP.protocol;
      to_SNP.dtgm.source_addr := from_ULP.source_addr;
      to_SNP.dtgm.destination_addr := from_ULP.destination_addr;
      to_SNP.dtgm.options := from_ULP.options;
      to_SNP.dtgm.padding := (as needed to end header on 4-octet boundary);

      to_SNP.dtgm.data[0..data_per_fragment-1] :=
                            from_ULP.data[0..data_per_fragment-1];

   --Set the datagram's header checksum field.
      compute_checksum;

   --Call route to determine the subnetwork address of the destination.
      route;

   --Also set the length and type of service indicators.
      to_SNP.length := to_SNP.dtgm.total_length;
      to_SNP.type_of_service_indicators := to_SNP.dtgm.type_of_service;

   --Request the execution environment to pass the first fragment
   --to the SNP.
      TRANSFER to_SNP to the local subnetwork protocol.


   --Format and transmit successive fragments.

     for j in 1..number_of_fragments-1 loop

        --The header fields remain the same as in the first fragment,
        --EXCEPT for:

          if ("copy" flag present in any options)

          then --put ONLY "copy" options into options fields and
               --adjust length fields accordingly.

               to_SNP.dtgm.options := (options with "copy" flag);
               to_SNP.dtgm.header_length := 5 +
                                   (number of copy options octets/4);

          else --only standard datagram header present

               to_SNP.dtgm.header_length := 5;

          endif;
```

```
        --Append data and set fragmentation fields.

         if (j /= number_of_fragments-1)

         then --middle fragment(s)

           to_SNP.dtgm.more_frag_flag := TRUE;
           to_SNP.dtgm.fragment_offset := j*number_frag_blocks;
           to_SNP.dtgm.total_length := to_SNP.dtgm.header_length
                                      + data_per_fragment;
           to_SNP.dtgm.data[0..data_per_fragment-1] :=
                              from_ULP.data[j*data_per_fragment..
                        (j*data_per_fragment + data_per_fragment-1)];
         else --last fragment

           to_SNP.dtgm.more_frag_flag := FALSE;
           to_SNP.dtgm.fragment_offset := j*number_frag_blocks;
           to_SNP.dtgm.total_length := to_SNP.dtgm.header_length*4
                                                + data_in_last_frag;
           to_SNP.dtgm.data[0..data_in_last_frag-1] :=
                            from_ULP[j*data_per_fragment..
                        (j*data_per_fragment+ data_in_last_frag-1)];
         end if;

      --Call checksum to set the datagram's header checksum field.
         checksum;

      --Call route to determine the subnetwork address of the destination.
         route;

      --Also set the length and type of service indicators.
         to_SNP.length := to_SNP.dtgm.total_length;
         to_SNP.type_of_service_indicators := to_SNP.dtgm.type_of_service;

      --Request the execution environment to pass this fragment
      --to the SNP.
         TRANSFER to_SNP to the local subnetwork protocol.

   end loop;
```

A fragmentation algorithm may vary according to implementation concerns but every algorithm must meet the following requirements:

1. A datagram must not be fragmented if dtgm.dont_frag_flag is true.

2. Data must be broken on 8-octet boundaries.

3. The minimum fragment size is 68 octets.

4. The first fragment must contain all options carried by the original datagram, except padding and no-op octets.

5. The security, source routing, and stream identification options (i.e. marked with "copy" flag) must be carried by all fragments, if present in the original datagram.

6. The first fragment must have to_SNP.dtgm.fragment_offset set to zero.

7. All fragments, except the last, must have to_SNP.dtgm.more_frag_flag set true.

8. The last fragment must have the to_SNP.dtgm.more_frag_flag set false.

6.3.6.3.9 analyze

The analyze procedure examines datagrams addressed to IP containing error reports from other IP modules. In general, error handling is implementation dependent. However, guidelines are provided to identify classes of errors and suggest appropriate actions. The errors and error formats are defined in section 6.2.15.

The data effects of this procedure are:

- Data examined:

  from_SNP.dtgm.protocol
  from_SNP.data

- Data modified:

  implementation dependent

For simplicity, it is assumed that the data area can be accessed as a byte array.

The algorithm:

```
  --Examine the first and second octets in the data portion
  --of the error datagram to identify the error reported.

     case from_SNP.dtgm[1] of

          when 3 =>  --Destination Unreachable Message

            --The errors in the "unreachable" class should
            --should be passed to the ULP indicating data delivery
            --to the destination is unlikely if not impossible.

             case from_SNP.dtgm[2] of

               when 0 =>  --net unreachable

               when 1 =>  --host unreachable

               when 2 =>  --protocol unreachable

               when 3 =>  --port unreachable

               when 5 =>  --fragmentation needed and don't fragment
                          --flag set
             end case;


          when 11 =>  --Time Exceeded Message
```

```
           --The "time-out" class of errors are usually not passed to the
           --ULP but should be recorded for network monitoring uses.

             case from_SNP.dtgm[2] of

               when 0 =>  --Time to live exceeded in transit

               when 1 =>  --Fragment reassembly time exceeded

             end case;
```

```
    when 12 =>  --Parameter Problem Message
       --This error is generated by a gateway IP to indicate
       --a problem in the options field of a datagram header.

    when 4  =>  --Source Quench Message
       --This message indicates that a datagram has been
       --discarded for congestion control.  The ULP should
       --be informed so that traffic can be reduced.

    when 5  =>  --Redirect Message
       --This message should result in a routing table update
       --by the IP module.  It is not passed to the ULP.

    when 8  =>  --Echo Datagram
       --Use of this message is implementation dependent.

    when 0  =>  --Echo Reply Datagram
       --Use of this message is implementation dependent.

end case;
```

6.3.6.3.10 error to ULP

The error_to_ULP procedure returns an error report to a ULP which has passed invalid parameters or has requested a service that cannot be provided.

The data effects of this procedure are:

- Parameters:

```
error_param : (PARAM_PROBLEM, CAN'T_FRAGMENT,
                NET_UNREACH, HOST_UNREACH,
                PROTOCOL_UNREACH, PORT_UNREACH);
```

- Data examined:

  implementation dependent

- Data modified:

  to_ULP.error
  implementation dependent parameters

The algorithm:

```
--The format of error reports to a ULP is implementation
--dependent.  However, included in the report should be
--a value indicating the type of error, and some information
--to identify the associated data or datagram.

  to_ULP.error := error_param;
  --implementation dependent action
```

6.3.6.3.11 error to source

The error_to_source procedure formats and returns an error report to the source of an erroneous or expired datagram.

The data effects of this procedure are:

- Parameters:

```
     error_param : (PARAM_PROBLEM, EXPIRED_TTL,
                    HOST_UNREACH, PROTOCOL_UNREACH);
```

- Data examined:

```
     from_SNP.dtgm
```

- Data modified:

```
     to_SNP.dtgm         to_SNP.local_destination_addrs
     to_SNP.length       to_SNP.type_of_service_indicators
```

The algorithm:

```
     --Format and transmit an error datagram to the source IP.

     to_SNP.dtgm.version         :=  4;      --standard IP version
     to_SNP.dtgm.header_length   :=  5;      --standard header size
     to_SNP.dtgm.type_of_service :=  0;      --least quality of service
     to_SNP.dtgm.identification  :=  select new value;
     to_SNP.dtgm.more_frag_flag  :=  FALSE;
     to_SNP.dtgm.dont_frag_flag  :=  FALSE;
     to_SNP.dtgm.fragment_offset :=  0;
     to_SNP.dtgm.time_to_live    :=  60;     --or value large enough to
                                             --allow delivery
     to_SNP.dtgm.protocol        :=  3;      --Gateway-Gateway Protocol ID
     to_SNP.dtgm.source_addr     :=  from_SNP.dtgm.destination_addr;
     to_SNP.dtgm.destination_addr := from_SNP.dtgm.source_addr;

   --The data section carries the error message in the first four
   --bytes, and the header and first 64 bytes of data of the
   --bad datagram.

     case error_param of

        where PARAM_PROBLEM =>
          to_SNP.dtgm.data[0] := 12;    --Gateway type = Parameter Problem
          to_SNP.dtgm.data[1] := 0;     --Code = problem with option


        where EXPIRED_TTL =>
          to_SNP.dtgm.data[0] := 11;    --Gateway type = Time Exceeded
          to_SNP.dtgm.data[1] := 0;     --Code = TTL exceed in transit
```

```
        where HOST_UNREACH =>
          to_SNP.dtgm.data[0] := 3;    --Gateway type = Dest. Unreachable
          to_SNP.dtgm.data[1] := 1;    --Code = host unreachable

        where PROTOCOL_UNREACH =>
          to_SNP.dtgm.data[0] := 3;    --Gateway type = Dest. Unreachable
          to_SNP.dtgm.data[1] := 2;    --Code = protocol unreachable
```

```
      end case;

    --Below, N is assumed to be the length of the bad datagram's header
    --plus the first 64 bytes of its data section ( 84 <= N <= 124);

      to_SNP.dtgm.data[4..N+3] := from_SNP.dtgm[0..N-1];
      to_SNP.dtgm.total_length := to_SNP.header_length*4 + N;

    --Compute checksum, determine the route for the error datagram,
    --the type of service indicators, and the datagram size for the SNP.

      compute_checksum;
      to_SNP.type_of_service_indicators := 0;
      to_SNP.length := to_SNP.dtgm.total_length;
      route;

    --Request the execution environment to pass the contents of to_SNP
    --to the local subnet protocol for transmission.

      TRANSFER to_SNP to the SNP.
```

6.3.6.3.12 reassembly timeout

The reassembly_timeout procedure generates an error datagram to the source IP informing it of the datagram's expiration during reassembly.

The data effects of the procedure are:

- Data examined:

  state_vector.header
  state_vector.data

- Data modified:

  to_SNP.dtgm
  to_SNP.length
  to_SNP.type_of_service_indicators
  to_SNP.local_destination_addrs

The algorithm:

```
      --Format and transmit an error datagram to the source IP.

      to_SNP.dtgm.version         :=  4;       --standard IP version
      to_SNP.dtgm.header_length   :=  5;       --standard header size
      to_SNP.dtgm.type_of_service :=  0;       --least quality of service
      to_SNP.dtgm.identification  :=  new value selected;
      to_SNP.dtgm.more_frag_flag  :=  FALSE;
      to_SNP.dtgm.dont_frag_flag  :=  FALSE;
      to_SNP.dtgm.fragment_offset :=  0;
      to_SNP.dtgm.time_to_live    :=  60;
      to_SNP.dtgm.protocol        :=  3;      --Gateway-Gateway Protocol ID
      to_SNP.dtgm.source_addr     :=  state_vector.header.destination_addr;
      to_SNP.dtgm.destination_addr := state_vector.header.source_addr;

    --The data section carries the error message in the first four
    --bytes, and the header and first 64 bytes of data of the
    --timed-out datagram.
```

```
     to_SNP.dtgm.data[0] := 12;    --Gateway type = Time Exceeded
     to_SNP.dtgm.data[1] := 1;     --Code = fragment reassembly timeout

   --Below, N is assumed to be the length of the expired datagram's header
   --plus the first 64 bytes of its data section ( 84 <= N <= 124 ).

     to_SNP.dtgm.data[4..N+3] := state_vector.data[0..N-1];
     to_SNP.dtgm.total_length := to_SNP.header_length*4 + N;

   --Compute checksum, determine the route for the error datagram,
   --the type of service indicators, and the datagram size for the SNP.
```

```
     compute_checksum;
     to_SNP.type_of_service_indicators := 0;
     to_SNP.length := to_SNP.dtgm.total_length;
     route;

   --Request the execution environment to pass the contents of to_SNP
   --to the local subnet protocol for transmission.

     TRANSFER to_SNP to the SNP.
```

7. EXECUTION ENVIRONMENT REQUIREMENTS

This section describes the facilities required of an execution environment for proper implementation and operation of the Internet Protocol. Throughout this document, the environmental model portrays each protocol as an independent process. Within this model, the execution environment must provide two facilities: inter-process communication and timing.

7.1 Inter-process communication

The execution environment must provide an inter-process communication facility to enable independent processes to exchange variable-length units of information, called messages. For IP's purposes, the IPC facility is not required to preserve the order of messages.

IP uses the IPC facility to exchange interface parameters and data with upper layer protocols across its upper interface and the subnetwork protocol across the lower interface. Sections 3 and 5 specify these interfaces.

7.2 Timing

The execution environment must provide a timing facility that maintains a real-time clock with units no coarser than 1 millisecond. A process must be able to set a timer for a specific time period and be informed by the execution environment when the time period has elapsed. A process must also be able to cancel a previously set timer.

Two IP mechanisms use the timing facility. The internet timestamp carries timing data in millisecond units. The reassembly mechanism uses timers to limit the lifetime of a datagram being reassembled. In the mechanism specification this facility is called TIMEOUT.

8. BIBLIOGRAPHY

1. "DCEC Protocols Standardization Program Preliminary Architecture Report" TM-7038/200/00, Contract No. DCA100-80-C-0044, February 1981.

2. "Protocol Specification Guidelines", TM-7038/204/00, Contract No. DCA100-80-C-0044, June 1981.

3. V. Cerf and R. Kahn, "A Protocol for Packet Network Interconnection", IEEE Transactions on Communications, May 1974.

4. J. Postel (ed.), "DoD Standard Internet Protocol", Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC760, IEN128, January 1980.

5. J. Postel (ed.), "DoD Standard Transmission Control Protocol", Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC761, IEN129, January 1980.

6. V. Cerf and P. Kirstein, "Issues in Packet-Network Interconnection", Proceedings of the IEEE, November 1978, pp.1386-1408.

7. P.T. Kelly, "Public Packet Switched Data Networks, International Plans and Standards", Proceedings of the IEEE, November 1978, pp.1539-1549.

8. D. Boggs, J. Shoch, E. Taft, and R. Metcalfe, "Pup: An Internetwork Architecture", IEEE Transactions on Communications, April 1980, pp.612-624.

9. J. Postel, "Internetwork Protocol Approaches", IEEE Transactions on Communications, April 1980, pp.604-611.

10. J. Shoch, "Inter-Network Naming, Addressing, and Routing", COMPCON, IEEE Computer Society, Fall 1978.

11. J. Shoch, "Packet Fragmentation in Inter-Network Protocols", Computer Networks, vol.3, no.1, February 1979.

12. J. Postel, "Address Mappings", IEN 115, USC/Information Sciences Institute, August 1979.

13. J. Postel, "Assigned Numbers", RFC 762, IEN 127, USC/Information Sciences Institute, January 1980.

9. GLOSSARY

Destination
An IP header field containing an internet address indicating where a datagram is to be sent.

datagram
A self-contained package of data carrying enough information to be routed from source to destination without reliance on earlier exchanges between source or destination and the transporting subnetwork.

datagram fragment
The result of fragmenting a datagram, also simply referred to as a fragment. A datagram fragment carries a portion of data from the larger original, and a copy of the original datagram header. The header fragmentation fields are adjusted to indicate the fragment's relative position within the original datagram.

datagram service
A datagram, defined above, delivered in such a way that the receiver can determine the boundaries of the datagram as it was entered by the source. A datagram is delivered with non-zero probability to the desired destination. The sequence in which datagrams are entered into the subnetwork by a source is not necessarily preserved upon delivery at the destination.

DF
Don't Fragment flag: An IP header field that when set true prohibits an IP module from fragmenting a datagram to accomplish delivery.

fragmentation
The process of breaking the data within a datagram into smaller pieces and attaching new internet headers to form smaller datagrams.

Fragment Offset
A field in the IP header marking the relative position of a datagram fragment within the larger original datagram.

gateway
A device, or pair of devices, which interconnect two or more subnetworks enabling the passage of data from one subnetwork to another. In this architecture, a gateway usually contains an IP module, a Gateway-to-Gateway Protocol (GGP) module, and a subnetwork protocol module (SNP) for each connected subnetwork.

header
Collection of control information transmitted with data between peer entities.

host
A computer which is a source or destination of messages from the point of view of the communication subnetwork.

Identification
An IP header field used in reassembling fragments of a datagram.

IHL
Internet Header Length: an IP header field indicating the number of 32-bit words making up the internet header.

Internet address
A four octet (32 bit) source or destination address composed of a Network field and a REST field. The latter usually contains a local subnetwork address.

internet datagram
The package exchanged between a pair of IP modules. It is made up of an IP header and a data portion.

local address
The address of a host within a subnetwork. The actual mapping of an internet address onto local subnetwork addresses is quite general, allowing for many to one mappings.

local subnetwork
The subnetwork directly attached to host or gateway.

MF
More Fragments flag: an IP header field indicating whether a datagram fragment contains the end of a datagram.

module
An implementation, usually in software, of a protocol or other procedure.

MTU
Maximum Transmission Unit: a subnetwork dependent value which indicates the largest datagram that a subnetwork can handle.

octet
An eight bit byte.

Options
The optional set of fields at the end of the IP header used to carry control or routing data. An Options field may contain none, one, or several options, and each option may be one to several octets in length. The options allow ULPs to customize IP's services. The options are also useful in testing situations to carry diagnostic data such as timestamps.

packet
The unit of data transmitted by a packet-switched network. A packet usually contains nested control information and data from the upper layer protocols using the subnetwork.

packet network
A network based on packet-switching technology. Messages are split into small units (packets) to be routed independently on a store and forward basis. This packetizing pipelines packet transmission to effectively use circuit bandwidth.

Padding
An IP header field, an octet in length, inserted after the last option field to ensure that the data portion of a datagram begins on a 32-bit word boundary. The Padding field value is zero.

Protocol
An internet header field used to identify the upper layer protocol that is the source and destination of the data within an IP datagram.

Precedence
One of the service quality parameters provided by the type of service mechanism. Precedence is a relative measure of datagram importance. This parameter can be set to one of five levels: routine, priority, immediate, flash, or flash override. It

appears as a three bit field within the Type of Service field in the IP header.

reassembly
The process of piecing together datagram fragments to reproduce the original large datagram. Reassembly is based on fragmentation data carried in their IP headers.

Reliability
One of the service quality parameters provided by the type of service mechanism. The reliability parameter can be set to one of four levels: lowest, lower, higher, or highest. It appears as a two bit field within the Type of Service field in the IP header.

reliability
A quality of data transmission defined as guaranteed, ordered delivery.

Rest
The three octet field of the internet address usually containing a local address.

segment
The unit of data exchanged by TCP modules. This term may also be used to describe the unit of exchange between any transport

protocol modules.

Source
An IP header field containing the internet address of the datagram's point of origin.

stream delivery service
The special handling required for a class of volatile periodic traffic typified by voice. The class requires the maximum acceptable delay to be only slightly larger than the minimum propagation time, or requires the allowable variance in packet interarrival time to be small.

SNP
SubNetwork Protocol: the protocol residing in the subnetwork layer below IP which provides data transfer through the local subnet. In some systems, an adaptor module must be inserted between IP and the subnetwork protocol to reconcile their dissimilar interfaces.

TCP
Transmission Control Protocol: a transport protocol providing connection-oriented, end-to-end reliable data transmission in packet-switched computer subnetworks and internetworks.

TCP segment
The unit of data exchanged between TCP modules (including the TCP header).

Total Length
An IP header field containing the number of octets in an internet datagram, including both the IP header and the data portion.

Type of Service
An IP header field containing the transmission quality parame-

ters: precedence level, reliability level, speed level, resource trade-off (precedence vs. reliability), and transmission mode (datagram vs. stream). This field is used by the type of service mechanism which allows ULPs to select the quality of transmission for a datagram through the internet.

ULP
Upper Layer Protocol: any protocol above IP in the layered protocol hierarchy that uses IP. This term includes transport layer protocols, presentation layer protocols, session layer protocols, and application programs.

user
A generic term identifying a process or person employing a protocol.   In IP's case, this term may describe a transport protocol, a presentation layer protocol, a session layer protocol, or an application program.

Version
An IP header field indicating the format of the IP header.