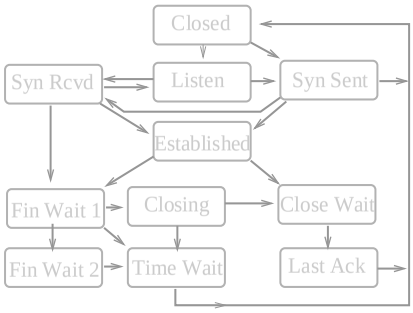
Using FSM to define a protocol



- The TCP connection FSM
  - But how do we create that?

Designing a FSM

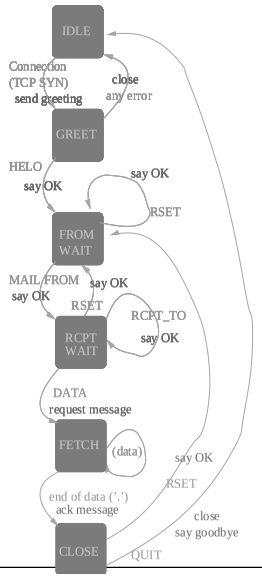
- Know the intended protocol
- List the states
  - for each party to communication
- List the events
  - packets
  - timeouts
  - application requests ...
- Determine which events possible in each state
  - and the effect of each event
- That's it...
  - write it down
  - or draw it.

Example FSM ... SMTP

- Not necessarily very useful, but
  - designing an FSM for SMTP
    - for the receiving MTA (server)
    - The client is too boring...
      - And normally no overlap
- The basic protocol
  - HELO
  - MAIL FROM
  - RCPT TO (repeated)
  - DATA
  - .
  - QUIT

Example FSM ... SMTP (2)

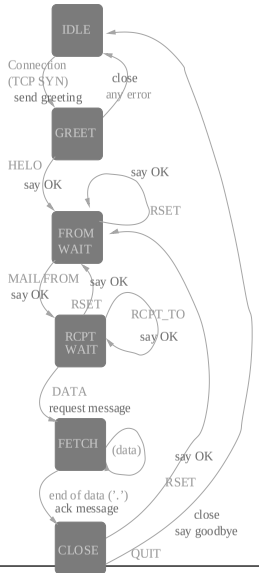
- The states
  - IDLE
  - GREETING
  - FROM_WAIT
  - RCPT_WAIT
  - FETCHING
  - CLOSING
- Input Events
  - Application_requests;
  - TCP_errors;
  - Incoming Protocol messages
    - HELO
    - MAIL_FROM
    - RCPT_TO
    - RSET



Machine Readable Representation

```
Initial_State := IDLE;

States [
    IDLE:
        On SYN_RCVD do
            send_greeting();
            goto GREETING;
        done
    GREETING:
        On HELO do
            say_OK();
            goto FROM_WAIT:
        done
        On * do
            close_Connection();
            goto IDLE;
        done
    FROM_WAIT:
        On MAIL_FROM do
            If (sender_ok())
                say_OK();
                goto RCPT_WAIT;
            else error_reply();
        done
```



Alternate Representation

```
State := IDLE;   /* The Initial State */

ForEver {
    Switch (State) {
        case IDLE:
            switch (next_event()) {
                case SYN_RCVD:
                    transmit_greeting();
                    State := GREETING;
                ;;
                /* ignore anything else */
            }
        ;;
        case GREETING:
            switch (next_event()) {
                case HELO:
                    say_OK("Nice to meet you");
                    State := FROM_WAIT;
                ;;
                default:
                    close_connection();
                    State := IDLE;
                ;;
            }
```

The Internet Protocol

- RFC 791 (1981)

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks.

There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

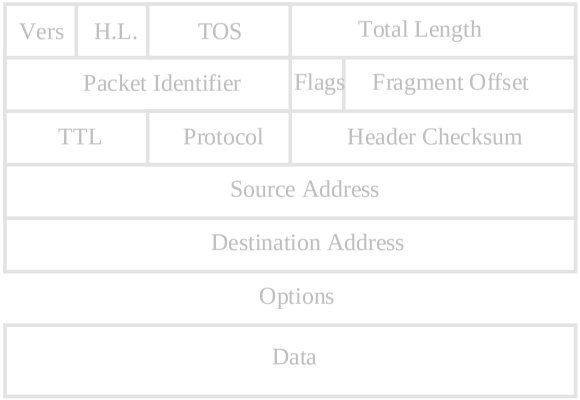
IP Specification

1.4. Operation

The internet protocol implements two basic functions:
addressing and fragmentation.

- Addressing
- Fragmentation
  - Examined soon
- First the rest that makes those useful

IP Header



- Will examine some of these fields
  - Others later

IP Header Length

- Header Length
  - Just like TCP
    - Counts 32 bit words
    - Minimum value 5
    - Maximum value 15

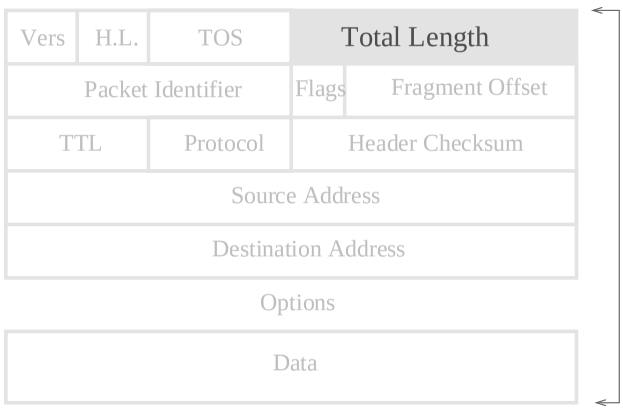
IP Specification

IHL: 4 bits

Internet Header Length is the length of the internet 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.

IP Length



- Length
  - Includes
    - IP header
      - including options
    - Any other headers (other protocols) & their data
  - 16 bits

IP Specification

Total Length: 16 bits

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

This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks.

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 sized data block 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.

IP Specification

Total Length: 16 bits

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

This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks.

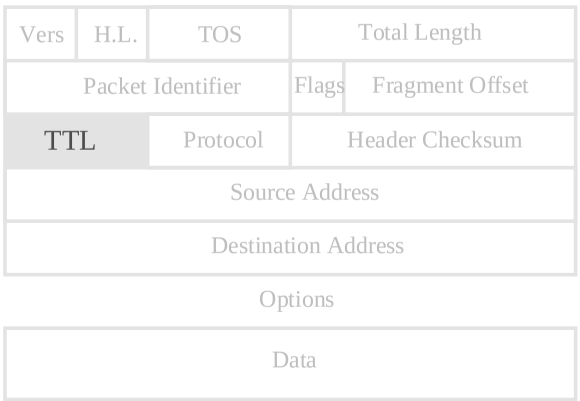
All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments).

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 sized data block 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.

IP Time to Live



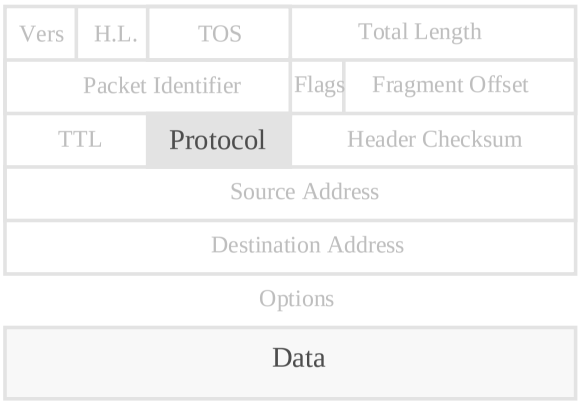
- TTL
  - Counts seconds
    - As each second passes, count decremented
  - Impractical
    - Time not synchronised
    - Nodes do not know when second started

IP Specification

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed
to remain in the internet system.
If this field contains the value zero, then the datagram must be destroyed.
This field is modified in internet header processing.
The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist.
The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

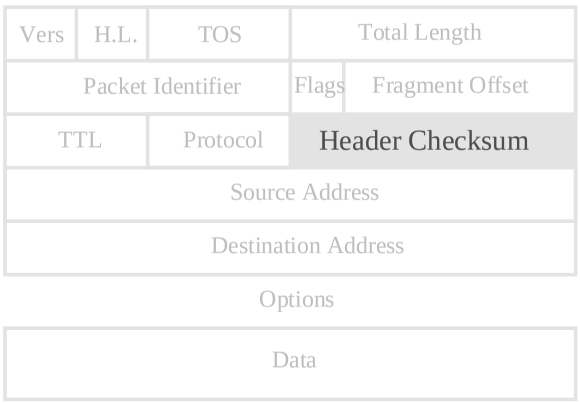
IP Protocol



Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram.
The values for various protocols are specified in
"Assigned Numbers"

IP Checksum



- Checksum
  - Of the IP header only
    - Protocol data not included
  - Standard IP checksum algorithm

IP Specification

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

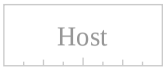
The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header.

For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

IP Addresses

- Originally



  - 8 bit host number
  - Max 256 hosts

- Net too small
  - switch from 8 bit to 32 bit addresses

- Predates current IP protocol
  - IPv4 came from this update (and more)

IP Addressing

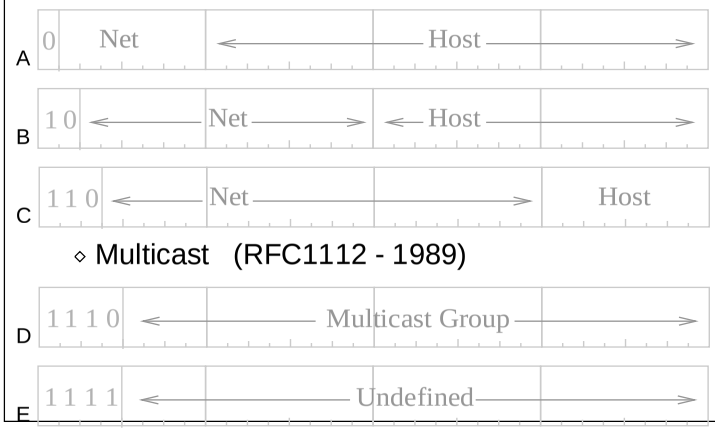
- ARPANET addresses
    - see RFC796 for this & others

  - 32 bit net & host number
    - 8 bit net number
    - 24 bit host number
  - ARPANET net number 10
  - Link level address is host part

- More Generally

IP addressing evolution

- Initially very few nets expected
  - Small number of large nets
- Became clear that more networks would exist



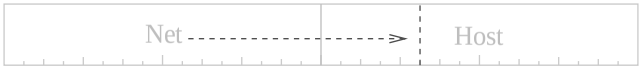
- Multicast (RFC1112 - 1989)

Available Addresses

- A: 127 nets (1..127) 4M hosts/net
- B: 16384 nets, 65536 hosts/net
- C: 2M nets, 256 hosts/net
  - ~ 2M nets total (2113664)
- Smallest allocation, 1 net
  - ~ 2M organisations

Internal Addressing

- Net Number assigned
- Plenty of host numbers
  - Only one net
  - Only one LAN



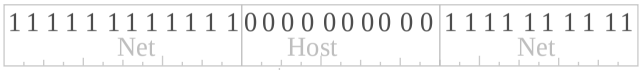
- Now many net numbers
  - Less hosts on each net

Subnet Masks

- RFCs 917 922 925 (1984) 932 936 940 950 (1985)
  - First major change to IP after RFC791

- Subnet Mask

  - Bits set indicate net number
  - Bits clear indicate host
- Non-contiguous masks



  - Was used & Works fine
  - No advantages, Harder to understand
    - Abandoned

Address Boundaries

- To Administrator

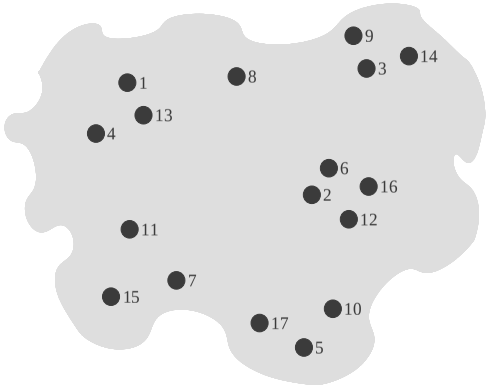
- Outside the network

- Inside the Network

- Implementation sees just one boundary
  - net / host
    - except at border

Address Assignment

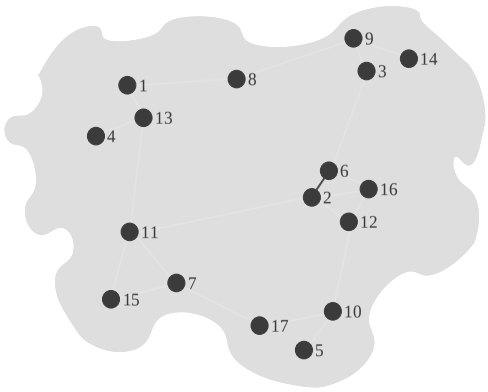
- As network grew...



- Addresses assigned arbitrarily
  - in allocation order
    - now meaningless

Impact of Assigment Method

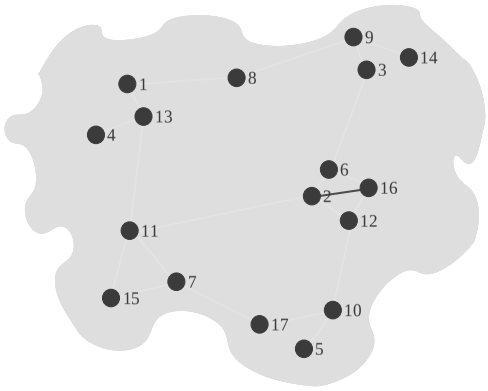
- Addresses used
  - To identify nodes
  - To find path to reach nodes
- Eg: From node 2



  - To reach node 6
    - send directly on link to 6

Impact of Assigment Method

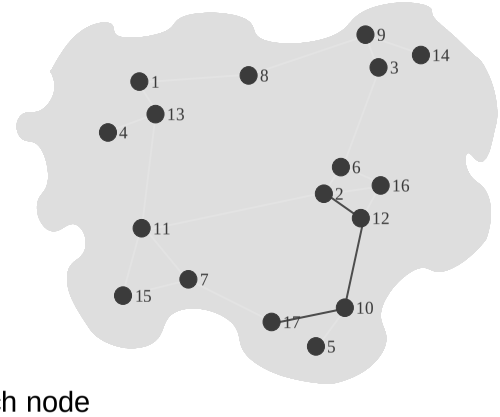
- Similarly (from node 2)



  - To reach node
    - 11: send directly to 11
    - 12: send directly to 12
    - 16: send directly to 16

Impact of Assigment Method

- And for other destinations
  - From node 2



  - To reach node
    - 1: send to node 11 (it will forward)
    - 3: send to node 6
    - 4: send to node 11