Protocol Design Concepts, IP and Routing

Section 1: Introduction

A central idea in the design of protocols is that of layering; and a guiding principle of Internet protocols is the "end-to-end" principle. In this chapter, we review these ideas and describe the transport and network layers in the Internet stack.

1.1 Protocols and Layering

Protocols are complex, distributed pieces of software. Abstraction and modular design are standard techniques used by software engineers to deal with complexity. By abstraction, we mean that a subset of functions is carefully chosen and setup as a "black-box" or module (see Figure 1). The module has an interface describing its input/output behavior. The interface outlives the implementation the module in the sense that the technology used to implement the interface may change often, but the interface tends to remain constant. Modules may be built and maintained by different entities. The software modules are then used as building blocks in a larger design. Placement of functions to design the right building blocks and interfaces is a core activity in software engineering.

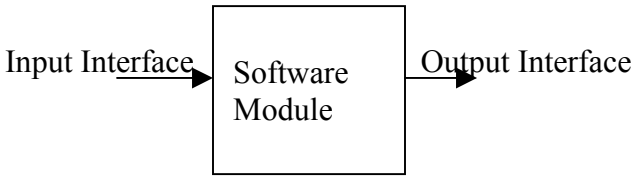

Figure 1: Abstraction of Functionality into Modules

Protocols have an additional constraint of being distributed. Therefore software modules have to communicate with one or more software modules at a distance. Such interfaces across a distance are termed as "***peer-to-peer***" interfaces; and the local interfaces are termed as "***service***" interfaces (Figure 2). Since protocol function naturally tend to be a sequence of functions, the modules on each end are organized as a (vertical) sequence called "***layers"***. The set of modules organized as layers is also commonly called a "***protocol stack***". The concept of layering is illustrated in Figure 3.

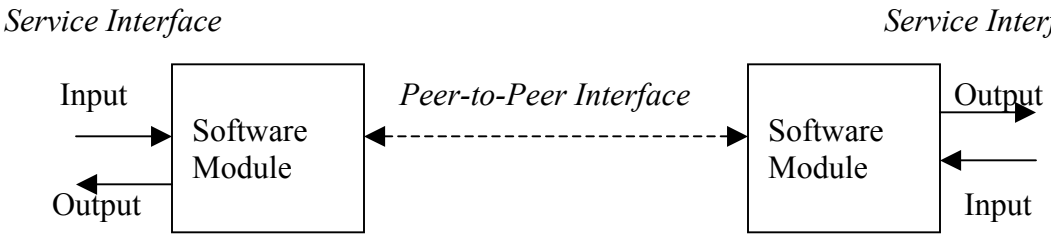

Figure 2: Communicating Software Modules

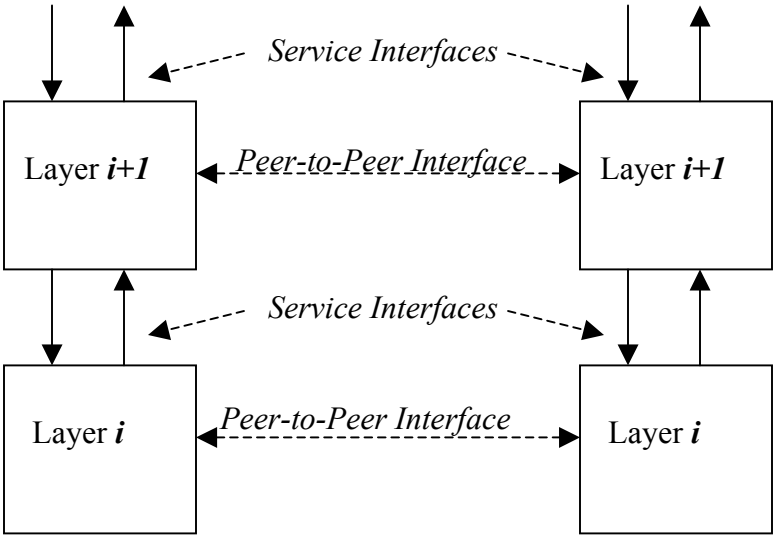

Figure 3: Layered Communicating Software Modules (Protocols)

Over the years, some layered models have been standardized. The ISO Open Systems Interconnection (ISO/OSI) layered model has seven layers and was developed by a set of committees under the auspices of International Standards Organization (ISO). The TCP/IP has a 4-layered stack that has become a *de facto* standard of the Internet. The TCP/IP and ISO/OSI stacks and their rough correspondences are illustrated in Figure 4. The particular protocols of the TCP/IP stack are also shown.

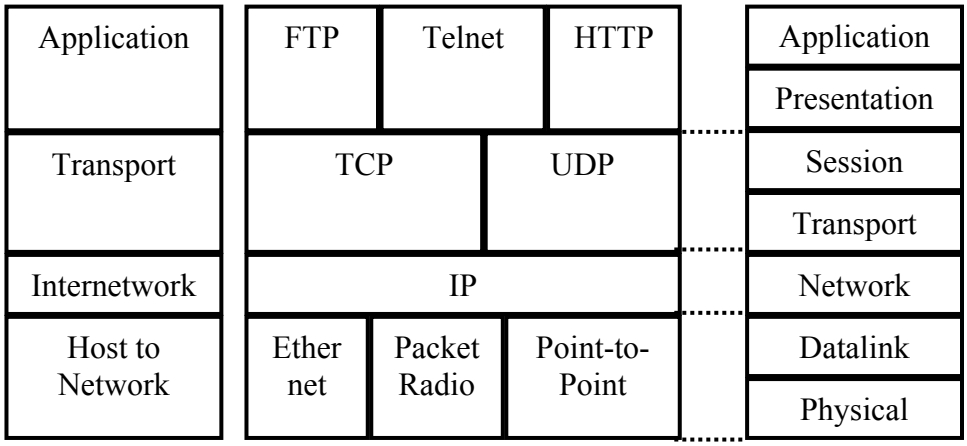

Figure 4: TCP/IP vs ISO/OSI Protocol Stack

The physical layer deals with getting viable bit-transmission out of the underlying medium (fiber, copper, coax cable, air etc). The data-link layer then converts this bit-transmission into a framed-transmission on the link. "Frames" are used to allow multiplexing of several streams; and defines the unit of transmission used in error detection and flow control. In the event that the link is actually shared (i.e. multiple access), the data link layer defines the medium access control (MAC) protocol as well. The network layer deals with packet transmission across multiple links from the source node to the destination node. The functions here include routing, signaling and mechanisms to deal with the heterogeneous link layers at each link.

The transport layer protocol provides "end-to-end" communication services, i.e., it allows applications to multiplex the network service, and may add other capabilities like connection setup, reliability and flow/congestion control. Examples of communication abstractions provided by the transport layer include a reliable byte-stream service (TCP) and an unreliable datagram service (UDP). These abstractions are made available through application-level programming interfaces (APIs) such as the BSD socket interface. The application layers (session, presentation, application) then use the communication abstractions provided by the transport layer to create the basis for interesting applications like email, web, file transfer, multimedia conference, peer-to-peer applications etc. Examples of such protocols include SMTP, HTTP, DNS, H.323 and SIP.

1.2 The End-to-End Principle in Internet Protocol Design

A key principle used in the design of the TCP/IP protocols is the so-called "end-to-end" principle that guides the placement of functionality in a complex distributed system. The principle suggests that "*…functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the lower level…*" In other words, a system (or subsystem level) should consider only functions that can be *completely and correctly* implemented within it. All other functions are best moved to the system level where it can be completely and correctly implemented.

In the context of the Internet, it implies that several functions like reliability, congestion control, session/connection management are best moved to the end-systems (i.e. performed on an "end-to-end" basis), and the network layer focuses on functions which it can fully implement, i.e. routing and datagram delivery. As a result, the end-systems are intelligent and in control of the communication while the forwarding aspects of the network is kept simple. This leads to a philosophy diametrically opposite to the telephone world which sports dumb end-systems (the telephone) and intelligent networks. Indeed the misunderstanding of the end-to-end principle has been a primary cause for friction between the "telephony" and "internet" camps. Arguably the telephone world developed as such due to technological and economic reasons because intelligent and affordable end-systems were not possible until 1970s. Also, as an aside, note that there is a misconception that the end-to-end principle implies a "dumb" network. Routing is a good example of a very complex function that is consistent with the end-to-end principle, but is non-trivial in terms of complexity. Routing is kept at the network level because it can be

completely implemented at that level, and the costs of involving the end-systems in routing are formidable.

The end-to-end principle further argues that even if the network layer did provide connection management and reliability, transport levels would have to add reliability to account for the interaction at the transport-network boundary; or if the transport needs more reliability than what the network provides. Removing these concerns from the lower layer packet-forwarding devices streamlines the forwarding process, contributing to system-wide efficiency and lower costs. In other words, the costs of providing the "incomplete" function at the network layers would arguably outweigh the benefits.

It should be noted that the end-to-end principle emphasizes function placement vis-a-vis correctness, completeness and overall system costs. The argument does say that, *"...sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement..."* In other words, the principle does allow a cost-performance tradeoff, and incorporation of economic concerns. However, it cautions that the choice of such "incomplete versions of functions" to be placed inside the network should be made very prudently. Lets try to understand some implications of this aspect.

One issue regarding the "incomplete network-level function" is the degree of "state" maintained inside the network. Lack of state removes any requirement for the network nodes to notify each other as endpoint connections are formed or dropped. Furthermore, the endpoints are not, and need not be, aware of any network components other than the destination, first hop router(s), and an optional name resolution service. Packet integrity is preserved through the network, and transport checksums and any address-dependent security functions are valid end-to-end. If state is maintained only in the endpoints, in such a way that the state can only be destroyed when the endpoint itself breaks (also termed "*fate-sharing*"), then as networks grow in size, likelihood of component failures affecting a connection becomes increasingly frequent. If failures lead to loss of communication, because key state is lost, then the network becomes increasingly brittle, and its utility degrades. However, if an endpoint itself fails, then there is no hope of subsequent communication anyway. Therefore one quick interpretation of the end-to-end model is that it suggests that only the endpoints should hold critical state. But this is flawed.

Let us consider the economic issues of Internet Service Provider (ISPs) into this mix. ISPs need to go beyond the commoditised mix of access and connectivity services to provide differentiated network services. Providing Quality of Service (QoS) and charging for it implies that some part of the network has to participate in decisions of resource sharing, and billing, which cannot be entrusted to end-systems. A correct application of the end-to-end principle in this scenario is as follows: due to the economic and trust model issues, these functions belong to the network. Applications may be allowed to participate in the decision process, but the control belongs to the network, not the end-system in this matter. The differentiated services architecture discussed later in this chapter has the notion of the "network edge" which is the repository of these functions.

In summary, the end-to-end principle has guided a vast majority of function placement decisions in the Internet and it remains relevant today even as the design decisions are intertwined with complex economic concerns of multiple ISPs and vendors.

Section 3: Network Layer

The network layer in the TCP/IP stack deals with internetworking and routing. The core problems of internetworking are *heterogeneity* and *scale*. Heterogeneity is the problem of dealing with disparate layer 2 networks to create a viable forwarding and addressing paradigm; and the problem of providing meaningful service to a range of disparate applications. Scale is the problem of allowing the Internet to grow without bounds to meet its intended user demands. The Internet design applies the end-to-end principle to deal with these problems.

3.1 Network Service Models

One way of dealing with heterogeneity is to provide translation services between the heterogeneous entities when forwarding across them is desired. Examples of such design include multi-protocol bridges and multi-protocol routers. But this gets too complicated and does not allow scaling because every new entity that wishes to join the Internet will require changes in all existing infrastructure. A more preferable requirement is to be able to "*incrementally upgrade*" the network. The alternative strategy is called an "overlay" model where a new protocol (IP) with its own packet format and address space is developed and the mapping is done between all protocols and this intermediate protocol.

IP has to be simple by necessity so that the mapping between IP and lower layer protocols is simplified. As a result, IP opts for a best-effort, unreliable datagram service model where it forwards datagrams between sources and destinations situated on, and separated by a set of disparate networks. IP expects a minimal link-level frame forwarding service from lower layers. The mapping between IP and lower layers involve address mapping issues (eg: address resolution) and packet format mapping issues (eg: fragmentation/reassembly). Experience has shown that this mapping is straightforward in many subnetworks, especially those that are not too large, and those which support broadcast at the LAN level. The address resolution can be a complex problem on non-broadcast multiple access (NBMA) sub-networks; and the control protocols associated with IP (esp BGP routing) can place other requirements on large sub-networks (eg: ATM networks) which make the mapping problems hard. Hybrid technologies like MPLS are used to address these mapping concerns, and to enable new traffic engineering capabilities in core networks.

For several applications, it turns out that the simple best-effort service provided by IP can be augmented with end-to-end transport protocols like TCP, UDP and RTP to be sufficient. Other applications having stringent performance expectations (eg: telephony) need to either adapt and/or use augmented QoS capabilities from the network. While several mechanisms and protocols for this have been developed in the last decade, a fully

QoS-capable Internet is still a holy grail for the Internet community. The hard problems surround routing, inter-domain/multi-provider issues, and the implications of QoS on a range of functions (routing, forwarding, scheduling, signaling, application adaptation etc).

In summary, the best-effort, overlay model of IP has proved to be enormously successful, it has faced problems in being mapped to large NBMA sub-networks and continues to faces challenges in the inter-domain/multi-provider and QoS areas.

3.2 The Internet Protocol (IP): Forwarding Paradigm

The core service provided by IP is datagram forwarding over disparate networks. This itself is a non-trivial problem. The end-result of this forwarding service is to provide connectivity. The two broad approaches to getting connectivity are: direct connectivity and indirect connectivity. *Direct connectivity* refers to the case where the destination is only a single link away (this includes shared and unshared media). *Indirect connectivity* refers to connectivity achieved by going through intermediate components or intermediate networks. The intermediate components (bridges, switches, routers, NAT boxes etc) are dedicated to functions to deal with the problem of scale and/or heterogeneity. Indeed the function of providing indirect connectivity through intermediate networks can be thought of as a design of a large virtual intermediate component, the Internet. These different forms of connectivity are shown in Figures 5-7.

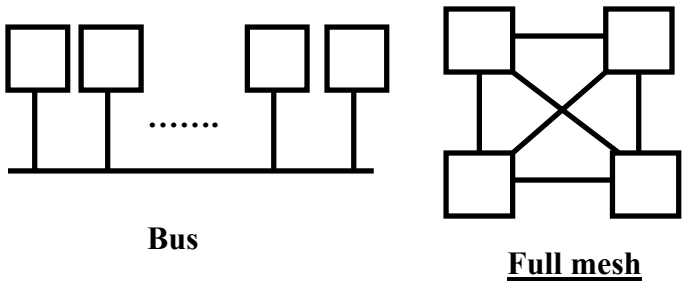

Figure 5: Direct Connectivity Architectures

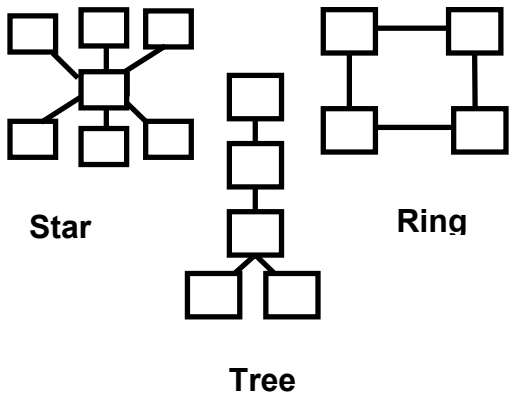

Figure 6: Indirect Connectivity though Intermediate Components

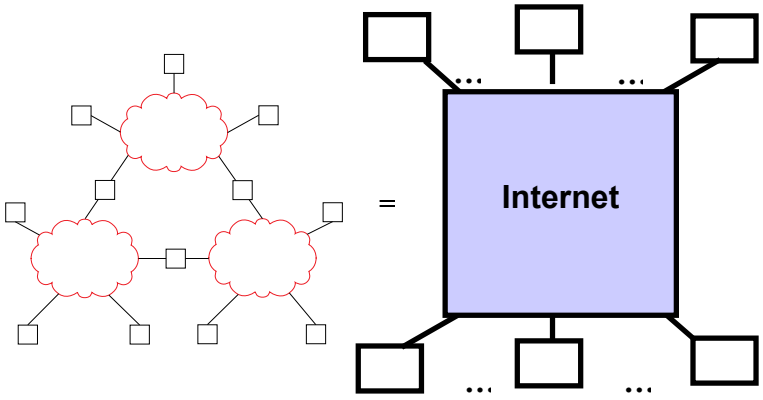

Figure 7: Indirect Connectivity though Intermediate Neworks & Components

The problem of scaling with respect to a parameter (eg: number of nodes) is inversely related to the efficiency characteristics of the architecture with respect to the same parameter. For example, direct connectivity architectures do not scale because of finite capacity of shared medium, or finite interface slots; or high costs of provisioning a full mesh of links. A way to deal with this is to build a switched network, where the intermediate components ("*switches*") provide filtering and forwarding capabilities to isolate multiple networks to keep them within their scaling limits, and yet providing scalable interconnection. In general, the more efficient the filtering and forwarding of these components, the more scalable is the architecture. Layer 1 hubs do pure broadcast, and hence do no filtering, but can forward signals. Layer 2 bridges and switches can filter to an extent using forwarding tables learnt by snooping; but their default to flooding on a spanning tree when the forwarding table does not contain the address of the receiver. This default behavior of flooding or broadcast is inefficient, and hence limits scalability. This behavior if also partially a result of the flat addressing structure used by L2 networks.

In contrast, layer 3 (IP) switches (aka routers) *never* broadcast across sub-networks; and rely on a set of routing protocols and a concatenated set of local forwarding decisions to deliver packets across the Internet. IP addressing is designed hierarchically, and address assignment is coordinated with routing design. This enables intermediate node (or hosts) to do a simple determination: whether the destination is *directly* or *indirectly* connected. In the former case, simple layer 2 forwarding is invoked; and in the latter case, a layer 3 forwarding decision is made to determine the next-hop that is an intermediate node on the same sub-network, and then the layer 2 forwarding is invoked.

Heterogeneity is supported by IP because it invokes only a minimal forwarding service of the underlying L2 protocol. Before invoking this L2 forwarding service, the router has to a) determine the L2 address of the destination (or next-hop) -- an address resolution

problem; and b) map the datagram to the underlying L2 frame format. If the datagram is too large, it has to do something -- fragmentation/reassembly. IP does not expect any other special feature in lower layers and hence can work over a range of L2 protocols.

In summary, the IP forwarding paradigm naturally comes out of the notions of direct and indirect connectivity. The "secret sauce" is in the way addressing is designed to enable the directly/indirectly reachable query; and the scalable design of routing protocols to aid the determination of the appropriate next-hop if the destination is indirectly connected. Heterogeneity leads to mapping issues, which are simplified because of the minimalist expectations of IP from its lower layers (only an forwarding capability expected). All other details of lower layers are abstracted out.

3.3 The Internet Protocol: Packet Format, Addressing, Fragmentation/Reassembly

3.3.1 IP Packet Format

The IP packet format is shown in Figure 8. The biggest fields in the header are the source and destination 32-bit IP *address* fields. The second 32-bit line (*ID, flags, frag offset*) are related to fragmentation/reassembly and will be explained later. The length field indicates the length of the entire datagram, and is required because IP accepts variable length payloads. The *checksum* field covers only the header and not the payload and is used to catch any header errors to avoid mis-routing garbled packets. Error detection in the payload is the responsibility of the transport layer (both UDP and TCP provide error detection). The *protocol* field allows IP to demultiplex the datagram and deliver it to a higher-level protocol. Since it has only 8-bits, IP does not support application multiplexing. Providing port number fields to enable application multiplexing is another required function in transport protocols on IP.

The *time-to-live (TTL)* field is decremented at every hop and the packet is discarded if the field is 0; this prevents packets from looping forever in the Internet. The TTL field is also used a simple way to scope the reach of the packets, and can be used in conjunction with ICMP, multicast etc to support administrative functions. The *type-of-service (TOS)* field was designed to allow optional support for differential forwarding, but has not been extensively used. Recently, the differentiated services (diff-serv) WG in IETF renamed this field to the *DS byte* to be used to support diff-serv. The version field indicates the version of IP and allows extensibility. The current version of IP is version 4. IPv6 is the next generation of IP that may be deployed over the next decade to support a larger 128-bit IP address space. *Header length* is a field used because *options* can be variable length. But options are rarely used in modern IP deployments, so we don't discuss them any further.

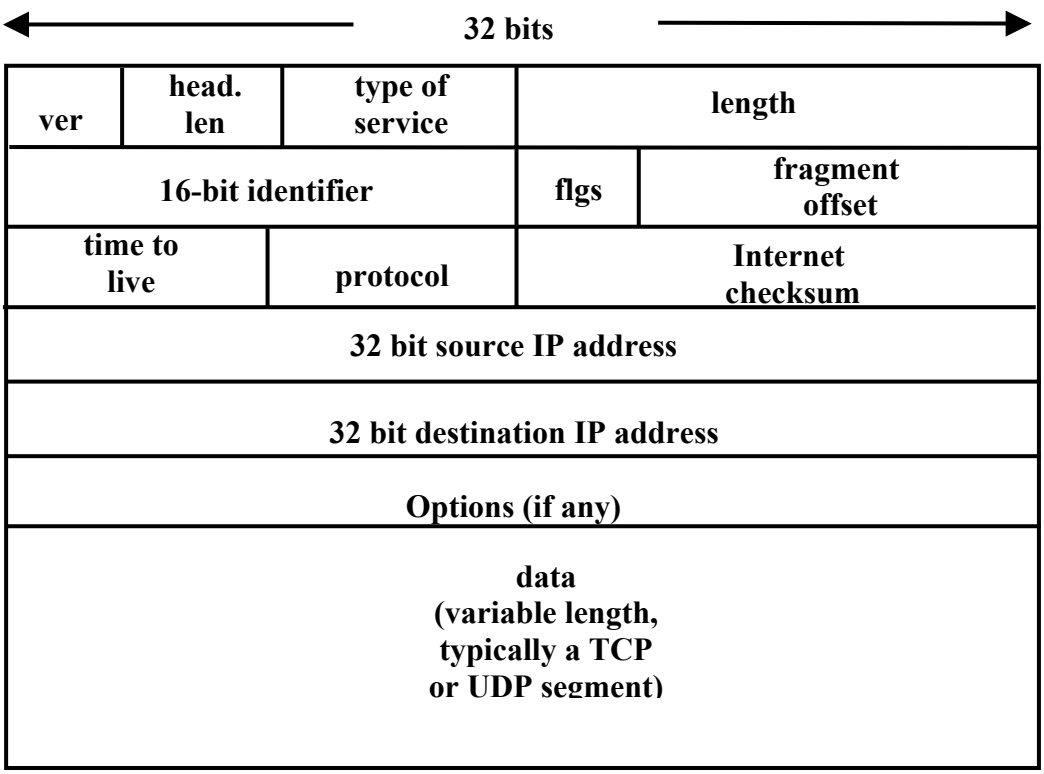

Figure 8: IP Packet Format

3.3.2 IP Addressing and Address Allocation

An address is a *unique "computer-understandable" identifier*. Uniqueness is defined in a domain. Outside that domain one needs to have either a larger address space, or do translation. An address should ideally be valid regardless of the location of the source, but may change if the destination moves. Fixed size addresses can be processed faster. The concept of addresses is fundamental to networking. There is no (non-trivial) network without addresses. Address space size also limits the scalability of networks. A large address space allows a large network, i.e. it is fundamentally required for network scalability. Large address space also makes it easier to assign addresses and minimize configuration. In connectionless networks, the most interesting differences revolve around addresses. After all, a connectionless net basically involves putting an address in a packet and sending it hoping it will get to the destination.

IPv4 uses 32-bit addresses whereas IPv6 uses 128-bit addresses. For convenience of writing, a dotted decimal notation became popular. Each byte is summarized as a base-10 integer, and dots placed between these numbers (eg: 128.113.40.50).

IP addresses have two parts -- a network part (prefix), and a host part (suffix). This is illustrated in Figure 9. Recall that the intermediate nodes (or hosts) have to make a determination whether the destination is directly or indirectly connected. Examining the network part of the IP address allows us to make this determination. If the destination is directly connected, the network part matches the network part of an outgoing interface of the intermediate node. This hierarchical structure of addressing which is fundamental to IP scaling is not seen in layer 2 (IEEE 802) addresses. The structure has implications on address allocation because all interfaces on a single sub-network have to be assigned the same network part of the address (to enable the forwarding test mentioned above).

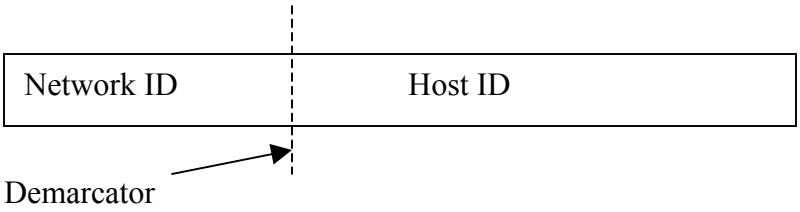

Figure 9: Hierarchical Structure of an IP Address

Unfortunately address allocation was not well thought out during the early days of IP, and hence it has followed a number of steps of evolution. Part of the evolution was forced because of the then unforeseen sustained exponential growth of the Internet. The evolution largely centered around the placement of the conceptual demarcator between the network ID and Host ID as shown in Figure 9.

Initially, the addressing followed a "*classful*" scheme where the address space was divided into a few blocks and static demarcators assigned to each block. Class A has a 8-bit demarcator; Class B has a 16-bit demarcator; Class C has a 24-bit demarcator. Class D was reserved for multicast and Class E for future use. This scheme is shown in Figure 10. This scheme ran into trouble in early 1980s because of two reasons: a) class B's were popular (class Cs largely unallocated) and b) the host space in class As and class Bs were largely unused because no single sub-network (eg: Ethernets) was large enough to utilize the space fully. The solution to these problems is simple -- allow the host space to be further subdivided; and allow demarcators to be placed more flexibly rather than statically.

These realizations led to the development of "subnet" and "supernet" masking respectively. A mask is a 32-bit pattern, the ones of which indicate the bits belonging to the network ID and the zeros indicate the host ID bits. For simplicity, the ones in the masks are contiguous. For example, a subnet mask 255.255.255.0 applied to IP address 128.113.40.50 indicates that the network ID has been extended from 16-bits (since this is a class B address) to 24-bits. Supernet masks are used between autonomous systems to indicate address allocations or to advertise networks for routing. For example the notation 198.28.29.0/18 indicates an 18-bit address space. The supernet mask written as /18 is actually 255.255.192.0. Observe that the 198.28.29.0 belonged to the class C space according to the earlier classful scheme and class C admits only of /24 networks (i.e. with host space of 8 bits).

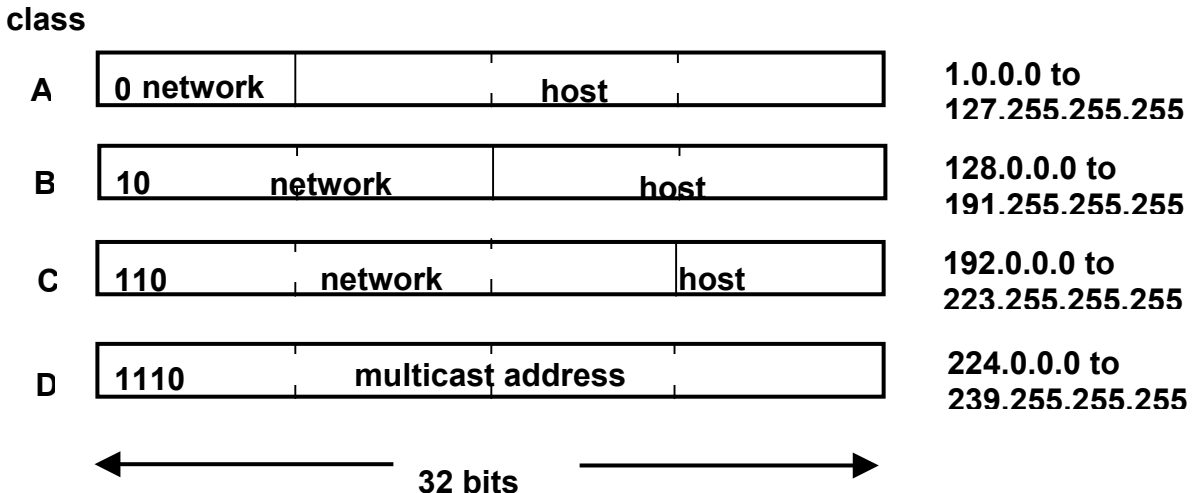

Figure 10: Initial Classful Addressing for IPv4

Since these class boundaries are no longer valid with the supernet masks, this allocation scheme is also called "*classless*" allocation; and the routing scheme which accompanied this development is called *"Classless Inter-Domain Routing" (CIDR)*. One effect of CIDR and supernet masking is that it is possible for a destination address to match multiple prefixes of different lengths. To resolve this, CIDR prescribes that the longest-prefix match be chosen for the L3 forwarding decision. As a result, all routers in the mid 1980s had to replace their forwarding algorithms. Similarly when subnet masking was introduced, hosts and routers had to be configured with subnet masks; and had to apply the mask in the forwarding process to determine the true network ID. Recall that the network ID is used to determine if the destination is directly or indirectly connected. These evolutionary changes are examples of how control-plane changes (CIDR and address allocation) could also affect the data-plane (IP forwarding) operation.

In modern networks, two other schemes are also used to further conserve public address space: DHCP and NAT. The Dynamic Host Configuration Protocol (DHCP) was originally a network "booting" protocol that configured essential parameters to hosts and routers. Now, it is primarily used to *lease* a pool of scarce public addresses among hosts who need it for connecting to the Internet. Observe that the leasing model means that host interfaces no longer "*own*" IP addresses.

The Network Address Translator (NAT) system enables the use of private address spaces within large enterprises. The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets:

10.0.0.0 - 10.255.255.255 (10/8 prefix)
172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

The NAT boxes at the edge of these private networks then translate public addresses to private addresses for all active sessions. Since early applications (eg: FTP) overloaded the semantics of IP addresses and included them in application-level fields, NAT has to transform these addresses as well. NAT breaks certain security protocols, notably IPSEC, which in part tries to ensure integrity of the IP addresses during transmission.

The combination of these techniques has delayed the deployment of IPv6 that proposes a more long-lived solution to address space shortage. IETF and the IPv6 Forum have been planning the deployment of IPv6 for over a decade now, and it remains to be seen what

will be the major catalyst for IPv6 adoption. The potential growth of 3G wireless networks and/or the strain on inter-domain routing due to multi-homing have been recently cited as possible catalysts. ISPs project that the IPv4 address space can be prolonged for another decade with the above techniques.

3.3.3 ARP, Fragmentation and Reassembly

Recall that the overlay model used by IP results in two mapping problems: address mapping and packet format mapping. The address mapping is resolved by a sub-IP protocol called ARP, while the packet mapping is done within IP by the fragmentation/reassembly procedures. The mapping problems in IP are far simpler than other internetworking protocols in the early 80s because IP has minimal expectations from the lower layers.

The address mapping problem occurs once the destination or next-hop is determined at the IP level (i.e. using the L3 forwarding table). The problem is as follows: the node knows the IP address of the next hop (which by definition is directly connected (i.e. accessible through layer 2 forwarding). But now to be able to use L2 forwarding, it needs to find out the next-hop's L2 address. Since the address spaces of L2 and L3 are independently assigned, the mapping is not a simple functional relationship, i.e., it has to be discovered dynamically. The protocol used to discover the L3 address to L2 address mapping is called the Address Resolution Protocol (ARP).

The ARP at the node sends out a link-level broadcast message requesting the mapping. Since the next hop is on the same layer-2 "*wire,*" it will respond with a unicast ARP reply to the node giving its L2 address. Then the node uses this L2 address, and encloses the IP datagram in the L2 frame payload and "*drops*" the frame on the L2 "*wire*." ARP then uses caching (i.e. an ARP mapping table) to avoid the broadcast request-response for future packets. In fact, other nodes on the same L2 wire also snoop and update their ARP tables, thus reducing the need for redundant ARP broadcasts. Since the mapping between L3 and L2 addresses could change (because both L2 and L3 address can be dynamically assigned), the ARP table entries are aged and expunged after a timeout period.

The packet-mapping problem occurs when the IP datagram to be forwarded is larger than the maximum transmission unit (MTU) possible in the link layer. Every link typically has a MTU for reasons such as fairness in multiplexing, error detection efficiency etc. For example, Ethernet has an MTU of 1518 bytes. The solution is for the IP datagram to be fragmented such that each fragment fits the L2 payload. Each fragment now becomes an independent IP datagram; hence the IP header is copied over. However, it also needs to indicate the original datagram, the position (or offset) of the fragment in the original datagram and whether it is the last datagram. These pieces of information are filled into the fragmentation fields in the IP header (ID, flags, frag offset) respectively. The reassembly is then done at the IP layer in the ultimate destination. Fragments may come out-of-order or be delayed. A reassembly table data structure and a time-out per datagram

is maintained at the receiver to implement this function. Reassembly is not attempted at intermediate routers because all fragments may not be routed through the same path.

In general, though fragmentation is a necessary function for correctness, it has severe performance penalties. This is because any one of the fragments lost leads to the entire datagram being discarded at the receiver. Moreover, the remaining fragments that have reached the receiver (and are discarded) have consumed and effectively wasted scare resources at intermediate nodes. Therefore, modern transport protocols try to avoid fragmentation as much as possible by first discovering the minimum MTU of the path. This procedure is also known as "*path-MTU discovery*." Periodically (every 6 seconds or so), an active session will invoke the path-MTU procedure. The procedure starts by sending a maximum sized datagram with the "*do not fragment*" bit set in the flags field. When a router is forced to consider fragmentation due to a smaller MTU than the datagram, it drops the datagram and sends an ICMP message indicating the MTU of the link. The host then retries the procedure with the new MTU. This process is repeated till an appropriately sized packet reaches the receiver, the size of which is used as the maximum datagram size for future transmissions.

In summary, the mapping problems in IP are solved by ARP (a separate protocol) and fragmentation/reassembly procedures. Fragmentation avoidance is a performance imperative and is carried out through path MTU discovery. This completes the discussion of the key data-plane concepts in IP. The missing pieces now are the routing protocols used to populate forwarding tables such that a concatenation of local decisions (forwarding) leads to efficient global connectivity.

3.3 Routing in the Internet

Routing is the magic enabling connectivity. It is the control-plane function, which sets up the local forwarding tables at the intermediate nodes, such that a concatenation of local forwarding decisions leads to global connectivity. The global connectivity is also "efficient" in the sense that loops are avoided in the steady state.

Internet routing is scalable because it is hierarchical. There are two categories of routing in the Internet: inter-domain routing and intra-domain routing. *Inter*-domain routing is performed between autonomous systems (AS's). An autonomous system defines the locus of single administrative control and is internally connected, i.e., employs appropriate routing so that two internal nodes need not use an external route to reach each other. The internal connectivity in an AS is achieved through *intra*-domain routing protocols.

Once the nodes and links of a network are defined and the boundary of the routing architecture is defined, then the routing protocol is responsible for capturing and condensing the appropriate global state into local state (i.e. the forwarding table). Two issues in routing are *completeness* and *consistency*.

In the steady state, the routing information at nodes must be *consistent*, i.e., a series of independent local forwarding decisions must lead to connectivity between any (source, destination) pair in the network. If this condition is not true, then the routing algorithm is said to not have "*converged*" to steady state, i.e., it is in a transient state. In certain routing protocols, convergence may take a long time. In general a part of the routing information may be consistent while the rest may be inconsistent. If packets are forwarded during the period of convergence, they may end up in loops or arbitrarily traverse the network without reaching the destination. This is why the TTL field in the IP header is used. In general, a faster convergence algorithm is preferred, and is considered more stable; but this may come at the expense of complexity. Longer convergence times also limit the scalability of the algorithm, because with more nodes, there are more routes, and each could have convergence issues independently.

Completeness means that every node has sufficient information to be able to compute all paths in the entire network locally. In general, with more complete information, routing algorithms tend to converge faster, because the chances of inconsistency reduce. But this means that more distributed state must be collected at each node and processed. The demand for more completeness also limits the scalability of the algorithm. Since both consistency and completeness pose scalability problems, large networks have to be structured hierarchically (eg: as areas in OSPF) where each area operates independently and views the other areas as a single border node.

3.3.1 Distance Vector and Link-State Algorithms and Protocols

In packet switched networks, the two main types of routing are link-state and distance vector. *Distance vector* protocols maintain information on a *per-node* basis (i.e. a vector of elements), where each element of the vector represents a distance or a path to that node. *Link state* protocols maintain information on a *per-link* basis where each element represents a weight or a set of attributes of a link. If a graph is considered as a set of nodes and links, it is easy to see that the link-state approach has complete information (information about links also implicitly indicates the nodes which are the end-points of the links) whereas the distance vector approach has incomplete information.

The basic algorithms of the distance vector (Bellman-Ford) and the link-state (Dijkstra) attempt to find the shortest paths in a graph, in a fully distributed manner, assuming that distance vector or link-state information can only be exchanged between immediate neighbors. Both algorithms rely on a simple recursive equation. Assume that the shortest distance path from node i to node j has distance D(i,j), and it passes through neighbor k to which the cost from i is c(i,k), then we have the equation:

$$D(i, j) = c(i,k) + D(k,j) \qquad (1)$$

In other words, the subset of a shortest path is also the shortest path between the two intermediate nodes.

The *distance vector (Bellman-Ford) algorithm* evaluates this recursion iteratively by starting with initial distance values:

D(i,i) = 0 ;
D(i,k) = c(i,k) if k is a neighbor (i.e. k is one-hop away); and
D(i,k) = INFINITY for all other non-neighbors k.

Observe that the set of values D(i,*) is a *distance vector at node i*. The algorithm also maintains a nexthop value for every destination j, initialized as:

next-hop(i) = i;
next-hop(k) = k if k is a neighbor, and
next-hop(k) = UNKNOWN if k is a non-neighbor.

Note that the next-hop values at the end of every iteration go into the forwarding table used at node i.

In every iteration each node i exchanges its distance vectors D(i,*) with its immediate neighbors. Now each node i has the values used in equation (1), i.e. D(i,j) for any destination and D(k,j) and c(i,k) for each of its neighbors k. Now if c(i,k) + D(k,j) is smaller than the current value of D(i,j), then D(i,j) is replaced with c(i,k) + D(k,j), as per equation (1). The next-hop value for destination j is set now to k. Thus after *m* iterations, each node knows the shortest path possible to any other node which takes *m* hops or less. Therefore the algorithm converges in O(d) iterations where d is the maximum diameter of the network. Observe that each iteration requires information exchange between neighbors. At the end of each iteration, the next-hop values for every destination j are output into the forwarding table used by IP.

The *link state (Dijkstra) algorithm* pivots around the link cost c(i,k) and the destinations j, rather than the distance D(i,j) and the source i in the distance-vector approach. It follows a greedy iterative approach to evaluating (1), but it collects all the link states in the graph *before* running the Dijkstra algorithm *locally*. The Dijkstra algorithm at node i maintains two sets: set N that contains nodes to which the shortest paths have been found so far, and set M that contains all other nodes. Initially, the set N contains node i only, and the next hop (i) = i. For all other nodes k a value D(i,k) is maintained which indicates the current value of the path cost (distance) from i to k. Also a value p(k) indicates what is the predecessor node to k on the shortest known path from i (i.e. p(k) is a neighbor of k). Initially,

D(i,i) = 0 and p(i) = i;
D(i,k) = c(i,k) and p(k) = i if k is a neighbor of i
D(i,k) = INFINITY and p(k) = UNKNOWN if k is *not* a neighbor of i
Set N contains node i only, and the next hop (i) = i.
Set M contains all other nodes j.

In each iteration, a new node j is moved from set M into the set N. Such a node j has the minimum distance among all current nodes in M, i.e. $D(i,j) = \min_{\{l \in M\}} D(i,l)$. If multiple nodes have the same minimum distance, any one of them is chosen as j. Node j is moved from set M to set N, and the next-hop(j) is set to the neighbor of i on the shortest path to j. Now, in addition, the distance values of any neighbor k of j in set M is reset as:

If D(i,k) < c(j,k) + D(i,j), then D(i,k) = c(j,k) + D(i,j), and p(k) = j.

This operation called "*relaxing*" the edges of j is essentially the application of equation (1). This defines the end of the iteration. Observe that at the end of iteration *p* the algorithm has effectively explored paths, which are *p* hops or smaller from node i. At the end of the algorithm, the set N contains all the nodes, and knows all the next-hop(j) values which are entered into the IP forwarding table. The set M is empty upon termination. The algorithm requires *n* iterations where *n* is the number of nodes in the graph. But since the Dijkstra algorithm is a *local* computation, they are performed much quicker than in the distance vector approach. The complexity in the link-state approach is largely due to the need to wait to get all the link states c(j,k) from the entire network.

The protocols corresponding to the distance-vector and link-state approaches for *intra*-domain routing are RIP and OSPF respectively. In both these algorithms if a link or node goes down, the link costs or distance values have to be updated. Hence information needs to be distributed and the algorithms need to be rerun. RIP is used for fairly small networks mainly due to a convergence problem called "*count-to-infinity*." The advantage of RIP is simplicity (25 lines of code!). OSPF is a more complex standard that allows hierarchy and is more stable than RIP. Therefore it is used in larger networks (esp enterprise and ISP internal networks). Another popular link-state protocol commonly used in ISP networks is IS-IS, which came from the ISO/OSI world, but was adapted to IP networks.