Internet Protocol

The goal of IP is to interconnect networks of diverse technologies and create a single, virtual network to which all hosts connect. Hosts communicate with other hosts by handing datagrams to the IP layer; the sender does not worry about the details of how the networks are actually interconnected. IP provides unreliable, connectionless delivery service. IP defines a universal packet called an *Internet Datagram*. All Internet hosts and gateways process IP datagrams.

Datagrams contain the following fields:

Version number (4-bits): The current protocol version is 4. Including a version number allows a future version of IP be used along side the current version, facilitating migration to new protocols.

Header length (4-bits): Length of the datagram header (excluding data) in 32-bit words.

The minimum length is 20 bytes, but will be longer if options are used. In practice, the length field is used to locate the start of the data portion of the datagram.

Total length (16-bits): Total length of the IP datagram (in bytes), including data and header. The size of the data portion of the datagram is the total length minus the size of the header.

Source address (32-bits): Original sender's address.

Destination address (32-bits): Datagram's ultimate destination. Note: When a gateway forwards a frame to another gateway, it forwards an Ethernet *frame*; the IP datagram contains the source of the original sender (not the forwarding gateway) and the destination address of the ultimate destination.

Time-to-live (8-bits): A hopcount that is decremented by each gateway. Should the hopcount reach 0, discard the datagram.

Originally, the time-to-live field was intended to reflect real time. In practice, it is now a hopcount. The time-to-live field squashes looping packets. It is also needed to guarantee that packets don't stay around in the network for longer than 255 seconds, a property needed by higher layer protocols that reuse sequence numbers.

Protocol (8-bits): What type of data the IP datagram carries (e.g., TCP, UDP, etc.).

Checksum (16-bits): A checksum of the IP header (excluding data). The IP checksum is computed as follows:

1. Treat the data as a stream of 16-bit words (appending a 0 byte if needed).
2. Compute the 1's complement sum of the 16-bit words.
3. Take the 1's complement of the computed sum.

Note: the Internet checksum is much weaker than the CRCs we have studied. However, it has the property that $(A + B) + C = A + (B + C)$. That is, the order in which the 16-bit words are summed is irrelevant.

Benefit? We can place the checksum in a fixed location in the header, set it to zero, compute the checksum, and store its value in the checksum field. On receipt of a datagram, the computed checksum calculated over the received packet should be zero.

Why checksum only header? Finally, checksumming only the header reduces the processing time at each gateway, but forces transport layer protocols to perform error detection (if desired).

Fragment offset (13-bits), Flags (3-bits), Identifier (16-bits): These three fields are used for fragmentation and reassembly. Gateways are free to fragment datagrams as needed, and hosts are required to reassemble fragments before passing complete datagrams to the higher layer protocols.

Each fragment contains a complete copy of the original datagram's header plus some of the data.

How can a receiving host match arriving fragments with the proper original datagram? That is, how can one distinguish those fragments from one datagram from those that belong to another?

1. All fragments of a datagram will have the same source and destination IP address. However, other datagrams between those two machines will share these fields as well, so this is not enough.
2. The *identifier* field uniquely identifies fragments of the same original datagram. Whenever a hosts sends a datagram, it sets the identifier field of the outgoing datagram and increments its local identifier counter. Thus, all fragments of a datagram contain the same source and identifier fields, but fragments from different datagrams won't.

How does one know where a fragment fits within the original datagram?

The *offset* field serves this purpose. When a gateway fragments a datagram, it sets the *offset* field of each fragment to reflect at what offset with respect to the original datagram the current fragment belongs.

Can gateways further fragment fragments? Yes. A 200-byte fragment having an offset of 200 could be split into two 100-byte fragments having offsets of 200 and 300, respectively.

Finally, how do we know when we have received all of the fragments? The *flags* field may contain:

1. A *don't fragment* indication (set by host, honored by gateways). (A 1-bit flag.)
2. When set, the *fragment more* field indicates that another fragment follows this one. That is, this fragment is not the last fragment of the original datagram.

   What fields does an unfragmented datagram have set? An *offset* of 0, and a *fragment more* bit of 0.

   What fields does the last fragment of a fragmented datagram contain? Neither flag would be set, and the offset would be non-zero.

   Note: The total length field of the IP header refers to the current datagram, not the original. Thus, the *fragment more* bit is needed in order for the recipient host to determine when it has all fragments of a datagram.

Type-of-service (8-bits): A hint to the routing algorithms as to what type of service we desire.

Precedence (3-bits): A priority indication, where 0 is the lowest and means normal service, while 7 is highest and is intended for network control messages (e.g., routing, congestion control).

Delay (1-bit): Application requests low delay service (e.g., for interactive applications).

Throughput (1-bit): Application requests high throughput.

Reliability (1-bit): Application requests high reliability.

Note: The three TOS bits will generally be mutually exclusive.

Does setting the low-delay bit guarantee getting such service? No. The *type-of-service* field is meant as a request or hint to the routing algorithms, but does not guarantee that your request can be honored (e.g., there may not be a low-delay path available).

IP Options

IP datagrams allow the inclusion of optional, varying length fields that need not appear in every datagram. The idea is that we may sometimes want to send special information, but since we don't send it very often, we don't want to dedicate a field in the packet header for this purpose.

Options start with a 1-byte *option code,* followed by zero or more bytes of *option data.* The option code byte contains three parts:

copy flag (1 bit): If 1, replicate option in each fragment of a fragmented datagram. That is, this option should appear in every fragment as well. If 0, option need only appear in first fragment.

option class (2 bits): Purpose of option:

Class	Meaning
0	network control
1	reserved for future use
2	debugging and measurement
3	reserved for future use

option number (5 bits): A code indicating the option's type.

The number and format of the rest of the bytes depend on the exact option. Defined options include:

End of option list (1 byte): Last option in the option list.

No option (1 byte): "no op" used for padding. Recall that the size of an IP header must be a multiple of 32-bit words.

Record route (variable length): Record the actual path taken by the datagram.

The option consists of a code of 7, a 1-byte length field, and a 1-byte pointer into a list of addresses that follows the option. The length gives the total length of the option, including the space for holding the list of visited gateways.

The sending host creates the record route option, allocating enough space to hold the IP address of each gateway through which it travels. As a gateway (or host) processes the datagram, it records the address of the interface that the datagram is forwarded out on in the data space of the option and increments the pointer by 4 bytes.

The record route option is useful for debugging, as it allows one to determine the actual path a datagram takes. Note that in the Internet, each gateway makes its own routing decision, making it difficult to determine what route a datagram is actually taking.

Loose source and record route: Provide a source route for the datagram to take. The option is called *loose* because the source route doesn't specify *every* intermediate gateway, just some of them. The variable-length portion of the option contains a list of IP addresses, and the datagram is to be forwarded to the first address in the list, then the second, and so forth until it reaches the last address in the list.

As with record route, gateways (and hosts) also record the actual route taken, overwriting the gateway used as the previous destination. Note that each gateway has (at least) two addresses; the source route will contain the interface the packet arrives on, while the recorded route contains the interface the datagram left on. This allows the recipient to use the recorded route in received datagrams as a source route for the datagrams it returns.

Timestamp: Similar to the record route option, except that each gateway records its current time to the option list. Times are given as seconds past midnight, Universal Time. The timestamp option provides a mechanism for determining how time a datagram spends traversing each hop along a path (assuming that the clocks on all machines are reasonably synchronized).

ICMP

The Internet Control Message Protocol (ICMP) allows gateways and hosts to send network control information to each other.

From a layering point of view, ICMP is a separate protocol that sits above IP and uses IP to transport messages. In practice, ICMP is an integral part of IP and all IP modules must support the ICMP protocol.

ICMP datagrams are encapsulated within IP datagrams and processed by IP in the same way as TCP and UDP datagrams; if special processing is needed, the IP type-of-service (TOS) field could be used.

There are two general types of ICMP messages:

1. Information messages, where a sender sends a query to another machine (either host or gateway) and expects an answer. For example, a host might want to know if a gateway is alive.
2. Error indication messages, where the IP software on a host or gateway has encountered a problem processing an IP datagram. For example, it may be unable to route a datagram to its destination.

ICMP messages begin with an 8-byte header, followed by zero or more bytes of message-specific data:

Type (1 byte): What kind of message (e.g., echo request, etc.).

Code (1 byte): Further information about the message type.

Checksum (2 bytes): Checksum of the ICMP message (excluding the IP header). Calculation is performed using the same algorithm as with the IP header.

Type specific data (4 bytes): Data specific to the message type; not used by all messages.

Echo Requests

The ICMP *echo request* and *echo reply* messages are two of the most useful ICMP messages for network debugging. If machine A sends an echo request message to machine B, machine B is required to respond with an ICMP echo reply.

ICMP echo messages have the following format:

Type (1 byte): 8 for echo request, 0 for echo reply.

Code (1 byte): Always zero.

Checksum (2 bytes): Checksums data only.

Identifier (2 bytes), Sequence number (2 bytes): Supplied by the sender, used to match responses with requests.

Data (arbitrary length): Data that must be returned in the echo reply. Might include information such as a timestamp, to aid in computing round trip times.

Most systems supply an application program that sends and receives ICMP echo messages. In Unix, the program *ping* allows a user to check whether a machine is reachable and functioning.

Because ICMP messages are handled just like other IP datagrams, ICMP echo messages test the *reachability* of any host. Also, because ICMP is an integral part of IP, all hosts *and* gateways must implement ICMP.

Timestamp Messages

ICMP *timestamp messages* are used to estimate the transmission delays between machines and to synchronize clocks:

Type (1 byte): 13 for timestamp request, 14 for reply.

Code, checksum, identifier, and sequence number: Same as in echo request case.

Originate timestamp (32 bits): Time that the timestamp request message is sent. All times are in milliseconds past midnight.

Receive timestamp (32 bits): Time that the timestamp message was received at the destination.

Transmit timestamp (32 bits): Time that the timestamp reply message was sent by the recipient of the timestamp request message.

Including both the receive and transmit timestamp allows the sending host to determine the fraction of time spent transmitting vs. processing the request.

By averaging the measurements of several messages, the sender can estimate the *offset* between its local clock and that on the remote machine. Note: it is quite feasible to synchronize the clocks of all machines on a LAN to within several milliseconds of each other.

Error Processing

When an IP module encounters an error processing a datagram, it sends an ICMP error message back to the original sender of the datagram. ICMP error messages have a common message format, which begins with the standard ICMP header described above. ICMP error messages also include part of the datagram that caused the error message to be generated:

Internet header (20 or more bytes): The *complete* IP header of the datagram causing the error.

Data portion of message (64 bits): The first 64 bits of the data portion of the IP datagram. What is this used for? Presumably, the first 64 bits will contain enough transport header information to allow the host that sent the datagram to match the ICMP error message with the specific transport-level "connection" responsible for sending the message.

When a gateway cannot route a datagram (e.g., it doesn't have an appropriate route in its local table), it discards the message and returns an ICMP *destination unreachable* message to the sending host. The intent of the ICMP destination unreachable messages is to inform a sending host that a particular destination is not currently reachable, and that they should try again (sometime) later. They have the following format:

Type (1 byte): Always 3.

Code (1 byte): 0: Network unreachable.

1: Specific host is unreachable.

2: Protocol is unreachable (e.g., the destination host doesn't support TCP).

3: Port unreachable (protocol exists, but specific service does not).

4: Fragmentation needed, but "don't fragment" bit set in IP header.

5: Source route failed.

Which codes are generated by hosts? By Gateways? Codes 0, 1, 4, and 5 are generated by gateways, while codes 2 and 3 are generated by hosts.

IP header + 64 bits of transport header: Datagram that could not be delivered. Note: Here we see why the transport header is needed; it identifies the sending process. The end user will want to know that the destination is unreachable so that it can take appropriate action.

As a datagram is processed, the gateway decrements the its time-to-live (TTL) field. If the TTL value reaches 0, the gateway discards the datagram and sends a *time exceeded*

message to the sender. Time exceeded messages have a type of 11, and the data portion of the message includes part of the offending datagram's header.

When a host or gateway encounters a problem *parsing* an IP datagram, it returns a *parameter problem* message to the datagram's sender:

Type (1 byte): Always 12.

Pointer (1 byte): Pointer into the original datagram that identifies the byte where the error was encountered.

When a gateway becomes congested and runs out of buffer space, it may discard a datagram and return a *source quench* message. Source quench messages are used to request that the sender reduce the rate at which it is sending datagrams.

Note: Source quench messages are meant to direct a sender to reduce its transmission rate. A gateway may send a source quench message even if it does not drop the datagram, but a host cannot rely on source quench messages to detect undelivered datagrams (they may get lost).

Two other ICMP messages, *redirect* and *mask request* messages, will be discussed later.

Note: Because ICMP messages are carried in IP datagrams, they are not reliable. Thus, a host should use them, but cannot depend on always receiving them.

Finally, to prevent error messages from spawning additional error messages, gateways do not generate ICMP error messages for ICMP error messages.

Host Routing

When a host wants to send a datagram, the routing operation will be in one of two forms:

Direct: The destination host resides on the same network as the sender. What happens? The sender simply maps the Internet address into a physical address (e.g., using ARP), encapsulates the datagram in a physical frame, and sends the datagram directly to the destination.

Indirect: The host must forward the datagram to a gateway.

How does one distinguish between the two cases? The sender extracts the network part of the address from the datagram's destination address and compares it against the list of networks to which it connects. If it finds a match, direct routing is in order. If the destination is remote, the host must forward the datagram to a gateway.

Indirect routing is more complicated than direct routing because the host may have several gateways to choose from, and might choose the wrong one.

How can we know which gateway to use?

Possible solution: Have the host participate in the same routing algorithms used by gateways. Disadvantages:

1. Hosts typically communicate with only a few hosts, most of them on local networks. Having hosts run a full-blown routing protocol is overkill. (The current Internet contains over 10,000 networks).
2. Routing protocols are still evolving. Changing a protocol would force every host to implement the new protocol.
3. No one single protocol is in use: hosts would have to implement many of them.
4. Many hosts (e.g., PCs) don't have the resources to run full blown routing protocols. That is, they don't even support multiple processes!

Better solution: Have hosts maintain "default" pointers to gateways, and have them forward all indirect traffic to a gateway. Let the gateway figure out where the datagram needs to go.

ICMP Redirects

What happens when a host chooses the "wrong" gateway? If a host has several gateways to choose from, picking any one gateway leaves open the possibility that the host will forward a datagram to the wrong gateway. That is, it will forward a datagram to gateway *G1*, and *G1* will forward it to gateway *G2*, which the sending host can reach directly.

The Internet architecture includes an ICMP message that helps. Gateways sends an ICMP *redirect* to inform a host that it should forward datagrams for destination *D* to gateway *G2*.

ICMP redirect message format:

Code (1 byte): always 5.

Code (1 byte): 0: Redirect datagrams for the network.

1: Redirect datagrams for just the host.

2: Redirect datagrams for TOS and network.

3: Redirect datagrams for TOS and host.

Gateway (32 bits): IP address of gateway that should be used.

Data: IP header + 64 bits of datagram that generated the redirect. In particular, the destination address is needed to update the routing table to use the specified gateway.

Each host maintains a cache of routes that have been built up from ICMP redirects. Thus, a host need only maintain a default pointer to a gateway (it doesn't even matter if its not the "best" one!) and a cache of routes for those destinations reachable through some gateway other than the default gateway.

Subnetting

Earlier, mentioned that the Internet consists of over 10,000 networks. In reality, the number is much larger. The figure refers to the number of distinct class A, B, and C networks for which routing table entries exist in the core gateways.

Consider the following:

- A large organization or campus might have 30 or more LANs (one for each department).
- An organization will probably have only a single connection to the rest of the Internet.
- In order for every local host to be able to communicate with other Internet machines, routing entries for each of the 30 networks must exist in the core gateways. In order for other sites to be able to respond to our queries, they must be able to route packets back to us.
- Wouldn't it be nice if we only needed to advertise a single network number for all 30 networks?

Subnet addressing is a technique that allows a set of multiple, interconnected networks to be covered by a single IP network number. IP addresses have a well-defined structure that allows a gateway to extract the network portion of an address by simply looking at its class.

With subnetting, the local part of an IP address is further subdivided into a network and a host part:

- Consider two addresses 128.204.2.29 and 128.204.1.109. Are they on the same network? Yes and no. They refer to hosts on the same IP network (128.204), but they can actually be on different ethernets connected by a bridge.
- We can divide the local part (the two bytes to the right of 128.204) into a 1-byte network part and a 1-byte host part.
- When sending data to 128.204.1.109 local gateways first route datagrams to the (sub)network 128.204.1 rather than (IP network) 128.204.

128.204.2 and 128.204.1 are distinct (sub)networks. To the outside world, there is only a single network 128.204. Each of the individual networks is called a *subnet*.

To implement subnetting, hosts and gateways use a *subnet mask* to extract the network part of an IP address. To distinguish between direct and indirect routing:

1. How does one determine the subnetwork number of a network interface? *Each* network interface has a network mask. The network mask ANDed with the interface address yields the network number of the interface.
2. For each of the machine's interfaces (hosts usually have only one):
   (a) Extract the destination address DEST from the datagram.
   (b) If ((interface_address & interface_mask) == (DEST & interface_mask)), direct routing is needed.

The routing algorithms described earlier remain essentially the same when subnetting is in use. Main difference? Routing algorithms may need to propagate the mask with a network number in routing updates. They need the mask to extract (sub)network numbers.

Finally, subnetting extends the number of levels in the Internet's hierarchical routing scheme. It trades off optimality of routes vs. table space in gateways.

How does a host determine its network mask? RARP? No. RARP doesn't provide the answer.

Hosts send ICMP *address mask requests*; responses contain the mask for the local network.

Gateway Routing

Unlike hosts, gateways cannot use the ICMP redirect mechanism to build routing tables. In particular, in order to send a redirect to machine *A*, gateway *G1* must realize that a datagram was sent by *A* in the first place:

1. If machine *A* is the original sender, the datagram's source address will contain the same network number as that of one of gateway *G1*'s interfaces.
2. If machine *A* were a gateway, *G1* would have to look at the datagram's physical address to determine where the datagram came from. Problems? This: 1) violates layering, and 2) is impossible if the physical network doesn't include the source address in the frame format.
3. While redirects can communicate information to an attached host, they have insufficient power to propagate general topology information.

Thus, gateways must exchange routing information with each other in order to build routing tables.

When routing a datagram, Gateways perform an iterative lookup algorithm to find an appropriate next hop. Each successive iteration searches an internal routing table using a less restrictive matching criteria.

A gateway (typically) uses the following algorithm to select the next hop for datagram *D*:

1. Extract the destination IP address and network number from the datagram.
2. Can we use direct routing? If so, send to the host directly (using ARP if necessary), otherwise go to next step.
3. Search the table for a *host* match. The entire destination (host and network portion) must match. Thus, it is possible to have host-specific routes in addtion to network routes. If no match is found, go on to the next step.
4. Search the table for a *network* match. Only the network portion of the destination address is used in the lookup operation. If no match is found, go on to the next step.
5. Search the table for a *default*, or *wildcard* route. Wildcard routes act as a catch all, and always match any address. Default routes are advertised by gateways that have large numbers of routes in their tables to keep the size of routing updates small.
6. If all of the above fail, return an ICMP destination unreachable error.

Routing Information Protocol (RIP)

An early routing protocol is the *routing information protocol* (RIP). It was developed by Berkeley as part of their BSD distribution, and became a de facto standard. It is also referred to as "ROUTE DEE", after the Unix program *routed*, which implements the protocol.

RIP is essentially the same algorithm as the "old Arpanet" routing algorithm. (Called "distributed routing" in Tanenbaum's book). Recall that:

- Gateways maintain routing tables that map destination addresses to (next hop, metric) pairs.
- Gateways periodically exchange routing tables with their neighbors. When gateway G receives a table from neighbor N, G switches its route for destination D to go through N if the sum of the cost of reaching N plus N's cost of reaching D is lower than the cost associated with the current path G is using.
- Vector-distance algorithms suffer from the problem: "good news travels quickly, bad news travels slowly".

RIP Features

RIP has the following features:

1. It is a vector-distance algorithm: metrics denote hop counts, and an infinity metric of 16 denotes unreachable destinations. Why 16, a small value for infinity? Using a small infinity metric reduces the number of iterations needed to reach infinity, but limits the diameter of the internet to 15 hops. Note that in the current internet, one-way many paths are longer than 20 hops.

2. It sends unsolicited update messages (using broadcast) every 30 seconds.

3. It ages routes; gateways delete from their tables those routes not appearing in any received update over a 180-second time interval. Aging routes has the advantage that updates do not have to be propagated reliably, but requires that periodic updates contain information on all destinations.

4. To propagate changes quickly, RIP gateways generate "triggered" or "flash" updates when they receive an update that changes their tables. This leads to a burst of update message exchanges, but reduces the time needed to count to infinity.

Note: By convention, RIP uses a destination address of all zeros to denote the wildcard address. This allows a gateway that has complete routing information to advertise a default route to the gateways around it. The other gateways use the default route for all their routing, and don't require the large tables containing a route for every network.

Often, an organization will propagate explicit information about all the local networks at its site, but use a wildcard route for routes to the outside world.

RIP is an example of an Interior Routing Protocol (IGP). It is designed to operate within a subset of the Internet, perhaps on a campus or within a corporation.

Autonomous Systems and EGP

As the Internet grew, it became clear that no one single routing algorithm would be used everywhere.

1. Sites must be able to isolate themselves from other sites. They should be able to keep their local internets operating even when other parts of the Internet have failed.
2. Local gateways (probably) don't want to know (in much detail) about topological changes that take place far away.
3. Sites want administrative control over their gateways and networksand may not want to run the same routing protocols as other sites.

As a result, the Internet developed the notion of *autonomous systems*, administrative regions that contain a set of networks and gateways. A site is free to manage routing within its region any way it wishes, and routing information flows among regions only through carefully controlled mechanisms.

The *Exterior Gateway Protocol* (EGP) is the "glue" that ties autonomous systems together. It:

1. Allows a site to advertise to the rest of the world a path to the networks within its autonomous system.
2. Allows sites to learn about networks located in other autonomous regions.

Border Gateway Protocol (BGP)

Current EGP protocol used.

Distance vector protocol, but not only does it maintain distance, but also the specific routes (to take into account administrative policies).

Open Shortest Path First

Current IGP protocol used.

The history of IP routing can only be described as a mess:

1. Despite the known problems with RIP, it has been used where it shouldn't. For example, Nysernet uses it internally as do many network. Indeed, regional networks have had to come up with horrible kludges to make RIP work for networks having diameters greater than 16 hops.

2. The first two years of NSFnet made use of a protocol called *HELLO*. HELLO is the old Arpanet algorithm (again) but uses delay (rather than hop counts) as its metric. The protocol was so unstable that the network was unusable by many sites.

3. EGP's topology restriction is clearly too inflexible.

4. Despite the fact that hosts shouldn't run gateway routing protocols, most do (or they just use static routing).

5. Now that IP has become popular, the definition of the "routing problem" has become more complex.For example, one issue that has come up is called *policy routing*. An organization running a cross-country network may want to be able to specify that it will forward all packets when it is lightly loaded, but always give IBM (or DEC) sponsored traffic higher priority. Another aspect of concern is how to charge users for network services.

We now consider a relative newcomer (1989) to routing: Open Shortest Path First routing (OSPF):

1. Divides an autonomous system into *areas*.
2. Uses a link-state algorithm within an area.
3. Four classes of routers (see Fig. 5-65)
   (a) internal areas wholly within one area.
   (b) area border routers connecting two or more areas
   (c) backbone routers on the backbone area (area 0)
   (d) AS boundary routers which talk to routers in other ASes.
4. It supports type of service routing. That is, it provides for multiple paths, with gateways choosing paths based on the type of service field in IP headers.
5. It supports multipath routing. That is, it distributes traffic over multiple paths to a destination.
6. It includes integrated support for subnetting (RIP does not). How? Specifically, (network number, network mask) pairs are distributed in updates.
7. Updates are authenticated; unauthenticated updates make the network extremely vulnerable to denial of service attacks (e.g., any workstation can send out bogus updates the break routing).

CIDR

Classless InterDomain Routing (CIDR) developed to make "smarter" use of the limited IPv4 set of addresses.

Basics:

- assign Class C addresses to regions of the world: Europe, North America, Asia, ...
- assign multiple Class C addresses to a single site
- avoid filling up routing tables, but maintaining a mask for each site so that a router can quickly determine if a Class C address is assigned to the site.
- routers maintain base address and mask for each site

IPv6

Motivation:

1. We will run out of Class B addresses soon (within years).
2. The entire address space of 32 bits will eventually be exhausted. Although 32 bits is 4 billion nodes, hierarchical routing by definition doesn't distribute addresses evenly.
3. We simply don't know how to scale routing beyond a few tens of thousands of networks. Thus, increasing the size of IP addresses solves problems 1 and 2, but doesn't help with the scaling problem. This is an engineering problem in the sense that distributing routing updates, computing new routing tables, and holding all routes in memory uses processor and memory resources. We can do that for 10,000 networks, maybe even 100,000, but not 1,000,000. Finding the right balance between these costs is difficult.

Need for more addresses provides an opportunity to improve upon other aspects of current IP (IPv4).

Look at header in Fig. 5-68, address space use in Fig. 5-69 and header extensions in Fig. 5-71.

During transition period, IPv4 addresses will be included in IPv6 addresses.