Evolving Internet I
Addressing and Intra-domain routing in the Internet

1.0 Naming and addressing

1.1 Introduction

Consider a user, who walks up to a computer connected to the Internet and types *ftp research.att.com,* thus asking for a file transfer session to be initiated between the local computer and a computer with the name *research.att.com.* The local computer must first translate from a human-understandable *name* (research.att.corn) to a numerical identifier called the destination's *address* (here, 135.104.117.5). Understandably, both names and addresses must be globally unique. *Naming is* the process of assigning unique names to endpoints, and *addressing is* the process of assigning unique addresses. We call the process of translating from a name to an address *resolution.*

After figuring out the address, the network must find a *route,* that is, a way to get from the source to the destination. A route may be found before data transfer, as in a connection-oriented network, or may be found on-the-fly, when a packet arrives at a router, as in a datagram network. In either case, given a destination address, switches or routers need a way to find the next hop on which to forward a call-setup packet or a datagram. This is the function of *routing.* In section 1, we study naming and addressing, and in section 2 we study routing.

1.2 Naming and addressing

If names and addresses both serve to identify a destination uniquely, why do we need them both? There are two reasons. First, names are usually human-understandable, therefore variable-length, and potentially rather long. For example, you could name your computer *very-long-name-and-I-dare-you-to-change-it,* and that would be perfectly acceptable. However, every packet in a datagram network must carry an indication of its destination. If we use names in packet headers, the source and destination fields in the packet header must be variable-length, and possibly quite long. Not only does this waste bandwidth, but also it would be more complicated for a router receiving the packet to look up the name in a routing table. Using fixed-length identifiers (that is, addresses) in packet headers is more efficient. (This restriction is less applicable in connection-oriented networks, where the destination identification is carried only once during connection establishment.)

The second reason to separate names and addresses is that this separation provides a level of indirection, giving network administrators some leeway in independently reorganizing names and addresses. For example, if a computer moves from one location to another within the same building or campus, an administrator can change its address without changing its name. If we inform the name-to-address translation mechanism about the change, users of the computer's name are not affected.

1.3 Hierarchical naming

Suppose you were given the job of uniquely naming every computer in the Internet. One way to do this would be to give each machine a name from a dictionary, choosing names in some order, and crossing off names as you assign them. This is easy enough when only a single naming authority (that is, you) is allowed to choose names. However, if many authorities were to choose names in parallel, conflicts are likely. For example, you may choose to name a computer *rosebud* exactly at the same time as another authority. You could come up with a protocol where you circulate a proposed name to every other naming authority before finalizing it. This would ensure global uniqueness, but can be somewhat time-consuming, especially if there are many widely separated naming authorities. A better solution is possible if we invoke a higher authority that assigns each naming authority a unique prefix. For example, you may be allowed to name machines with names starting with *a,* and another authority may be allowed to name machines with names starting with *b.* If both of you obey this rule, you can choose any name you wish, and conflicts will never arise.

Partitioning the set of all possible names (the *name space)* into mutually exclusive portions (or *domains)* based on a unique prefix simplifies distributed naming. The unique prefix introduces a *hierarchy* in the name space, as shown in Figure B3T6. Here, we use the rule that names starting with *a* are written in the form *a.name,* where the "." is a special character showing the domain boundary. We can generalize the name assignment rule to multiple levels of hierarchy. For example, you may choose to give your subordinates portions of the name space prefixed by *a.a, a.b, a.c,* etc. If they start their names with these prefixes, the names they generate are guaranteed to be universally unique.

We call the first level in the hierarchy the *top-level domain.* The rule for naming is, therefore, that a global authority assures uniqueness of names in the top level. Authorities in a given domain can give away part of the name space to lower-level naming authorities, if the prefixes handed to these authorities are unique within that domain. This rule applies recursively at all levels, and therefore we can make the naming hierarchy arbitrarily deep.

A hierarchically organized name space scales without bound, yet allows names to be chosen without consulting every other naming authority on the planet. Because of these wonderful properties, it is used in every large network, including the Internet, the telephone network, and ATM networks.

- Names on the Internet follow the conventions established by the *domain name system,* or DNS. A global authority assigns top-level domains with unique names, such as *edu, com, cz, in,* or *net.* Naming authorities in charge of each domain then parcel out names within that domain to lowerlevel authorities.
- The telephone network identifies endpoints with hierarchical telephone numbers. Since these are not understandable by normal human beings, the telephone network, arguably, has no naming, only addressing.

Example 1.1

The name administrator at UC Berkeley might be given the authority to give names in the domain *berkeley.edu.* (Note that Domain Name System uses a unique suffix rule, instead of a unique prefix rule. This is an equally valid way to create a hierarchy.) The campus naming administrator, might, in turn, give the naming authority in the Mechanical Engineering Department fiefdom over the domain *mech.berkeley.edu.* A lowly graduate student lording it over a roomful of PCs might receive the domain *pclab. mech.berkeley.edu,* and may name the machines, prosaically, *pc1.pclab.mech.berkeley.edu, pc2.pclab.mech.berkeley.edu*, etc. By construction, these names are unique.

1.4 Addressing

Addresses are numerical identifiers that, like names, are globally unique. Therefore, just like names, we usually organize them as a hierarchy. Another important reason for hierarchical addresses is that they allow aggregation in routing tables, as shown in Figure B3T8. We now study the relationship between hierarchical addressing and aggregation.

Example 1.2

In Figure B3T8 left, we see a 10-node network where the addresses have been chosen from a nonhierarchical (or flat) name space. Suppose computer 3 wanted to send a message to computer 5. Since 3 and 5 are not directly connected, 3 must choose to send the message to either 2 or 4, which, in turn will forward it to 5. It turns out that if 3 sends it to 2, the packet will take at least four hops, but if it sends it to 4, it can take as few as two hops. Thus, 3 should send it to 4. Usually, 3 needs to maintain a routing table that shows the next hop for every destination in the network. Here, because 3 has nine possible destinations, the routing table will have nine entries.

Although having one entry per destination is reasonable for networks with hundreds, or even thousands of destinations, it is impractical for networks with tens of millions of destinations, such as the Internet. To work around this problem we must aggregate addresses into clusters by using a hierarchical address space.

Example 1.3

In Figure B3T8 right, we have renamed each node with a two-part address. We now call node 5 node 2.3, which we interpret as computer 3 in subnetwork 2. Certain nodes in each subnetwork are shaded, to mark them as special nodes. We call these *border routers,* and they carry all the traffic into and out of the subnetwork. Now, when node 3 in Figure B3T8 left (now called 1.2) wants to contact 2.3, it simply sends the packet to its border router, 1.1. The routing table in 1.1 shows that the shortest path to subnetwork 2 is via computer 2.1, which, in turn, routes the packet through either 2.4 or 2.2 to 2.3. Router 1.1 only stores a route to *subnetwork* 2, instead of to every router in subnetwork 2. Thus the router *aggregates* routes to subnetwork 2, making its routing table smaller.

Note that the route from 1.2 to 2.3 is four hops long, which is longer than the two-hop shortest path. On the other hand, every nonborder router has only one entry in its routing table, the address of its border router. Each border router has an entry for every computer in its subnetwork, but only one entry for every other subnetwork. Thus, we have traded off some inefficiency in routing for a dramatic reduction in the number of routing entries. This reduction is possible because the addresses are hierarchical, and the network is partitioned along the same lines as the addresses. If computers in subnetwork 3 could have arbitrary addresses, border routers in other subnetworks would need to have one routing entry for each computer in 3. We can reduce the number of entries in a border router only because every node in subnetwork 3 has a prefix of 3. We refer to addresses in subnetwork 3, and the subnetwork itself, as 3*.

1.5 Addressing on the Internet

The Internet addresses host interfaces instead of endpoints. Thus, a computer with multiple interfaces on the Internet has an address for each interface. Internet addresses are used by the Internet Protocol (IP), so they are usually called IP addresses. There are two versions of IP addresses: version 4, also called IPv4 addresses, and version 6, called IPv6. IPv4 addresses are 4 bytes long and are divided in a two-part hierarchy. The first part is called the *network* number, and the second part is a *host* number (although it addresses an interface). A *subnet mask* describes the partition between the two (Figure B3T9). The logical AND of the IP address and the associated subnet mask is the network number, and the remaining portion is the host number. The subnet mask allows arbitrary partitioning of the address into the network and host number. The Internet Numbers Authority guarantees that subnet numbers are globally unique, and within each subnet, individual network administrators guarantee that IP addresses are unique.

Since IP addresses are partitioned, they can be aggregated, and routing tables in the core of the network need only store a specific route to the router responsible for a particular network. For example, routers in the core of the Internet store a route only to the network number 135.104.*. The router in charge of this network routes the packet to the final destination within 135.104, such as 135.104.53.100.

1.5.1 Address classes

The first cut at IP addresses had a fixed network-host partition with only 8 bits of network number. The designers did not envisage more than 256 networks ever joining the Internet "experiment"! They soon generalized this to allow the address to be partitioned in one of three ways:

- Class A addresses have 8 bits of network number and 24 bits of host number
- Class B addresses have 16 bits of network and host number
- Class C addresses have 24 bits of network number and 8 bits of host number

The classes are distinguished by the leading bits of the address. If the address starts with a 0, it is a Class A address. It has 7 bits for a network number, and 24 bits for a host number. Thus, there can be 128 Class A networks, each with 2^{24} hosts (actually, there can be only 126 networks, since network numbers 0 and 127 have special significance). If the address starts with a 10, it is a Class B address; if it starts with a 110, it is a Class C address. Two special classes of addresses are those that start with 1110 (Class D), which are used for multicast, and those that start with 1111 (Class E), which are reserved for future use.

This solution proved adequate until 1984, when the growth of the Internet forced the first of three changes: subnetting, described in Section 1.5.2; CIDR*,* described in Section 1.5.3; and *dynamic host configuration,* described in Section 1.5.4.

1.5.2 Subnetting

Owners of Class B addresses had always wanted to further partition their set of host numbers into many smaller *subnets* Then, routing tables within the network need only store routes to subnets, instead of to individual hosts. The class structure is too coarse to deal with this, because the Class B address administrator cannot use the class structure to further partition its address space. The solution is to introduce the subnet mask that we studied at the beginning of this section. Note that the subnet mask chosen within a network is not visible outside the network. Core routers route packets to the border router responsible for the network, where the incoming packet's IP address is interpreted according to the subnet mask valid in that particular network.

Example 1.4

Suppose you are the administrator for the Class B address 135.104.*. If you do not partition the address, every router in your network has to know the route to every host, because you have no way to describe aggregates of hosts within your network. For instance, hosts 135.104.5.0, 135.105.5.6, and 135.105.5.24 may lie in the same physical LAN segment and may all be reachable from router 135.105.4.1. There is no way to express this using the class notation, because these computers all have the same Class B address. You need some extra information to describe this aggregation, which is the subnet mask. Suppose you decide that no subnet is likely to contain more than 256 hosts. Then, you could partition the 65,536 addresses in your domain into 256 subnets, each with 256 addresses. This is expressed by the subnet mask 255.255.255.0 (1111 1111 1111 1111 1111 1111 0000 0000), as shown in Figure B3T12. Addresses within your local Class B network would then be treated as if the network number were the first 24 bits (instead of the first 16 bits), and the host number were the last 8 bits (instead of the last 16 bits). Because the hosts 135.104.5.0, 135.104.5.1, and 135.104.5.2 all lie in the same subnet, that is, 135.104.5.*, routing tables within your Class B network need only have an entry for 135.104.5.* (expressed as 135.104.5.* plus the subnet mask 255.255.255.0) pointing to 135.105.4.1 (the next hop), instead of entries to each of the individual computers. This saves routing table space and route table computation time.

1.5.3 CIDR

As the Internet grew, most networks were assigned Class B addresses, because their networks were too large for a Class C address, which can hold only 256 hosts, but not large enough for a Class A address, which can hold more than 4 million hosts. In 1991, it became clear that the 16,382 Class B addresses would soon run out, which resulted in a crisis in the Internet community. Some network engineers noted that addresses from the enormous Class C space were rarely allocated, since most networks have more than 256 hosts. Their solution was to allocate new networks contiguous subsets of Class C addresses instead of a single Class B address (the allocated set of Class C addresses usually spans a smaller portion of the address space than a single Class B address). To aggregate routes to sets of Class C addresses, routers in the core of the network must now carry a *prefix indication,* just as routers within a network carry subnet masks. The prefix indication is the number of bits of the network address that should be considered part of the network number. Routing protocols that substitute multiple Class C addresses for a Class B address are said to obey *classless interdomain routing* or *CIDR* (pronounced "cider").

Example 1.5

With CIDR, a network could be allocated eight Class C networks, spanning the 2048 addresses from 201.10.0.0 to 201.10.7.255, instead of a single Class B network, with 65,536 addresses. Since the network administrator is allocated eight Class C networks, which use three bits of the Class C space, the remaining 21 bits must be the network number. The address and prefix describing the network are, therefore, 201.10.0.0 and 21, usually written 201.10.0.0/21 (Figure B3T14).

1.5.4 Dynamic host configuration

A third technique to extend the life of the v4 address space is to dynamically allocate hosts with IP addresses. In many situations, a computer, such as a laptop, may access the Internet only once in a while. These hosts need an IP address only when they are active.

Thus, a clever network operator can share the same IP address among different computers, as long as they are not simultaneously active. The protocol to do so is called the *Dynamic Host Configuration Protocol,* or *DHCP.*

In DHCP, a newly booted computer broadcasts a *DHCP discover* packet on its local LAN. This packet contains host-specific information, such as its hardware address. DHCP servers that receive this packet reply with a *DHCP offer* packet that contains an offered IP address and other configuration parameters. A host that receives one or more replies selects a server and IP address, and confirms its selection by broadcasting a *DHCP request* packet that contains the name of the selected server. The server confirms receipt of this message and confirms its offer with a *DHCP ack.* Other servers, on hearing a DHCP request, automatically withdraw their offers. When a host is done, it sends a *DHCP release* message to its selected server, which releases its IP address for use by other hosts.

DHCP has the notion of a *lease,* which is the time for which a host's IP address is valid. A host can guess the time for which it wants an address and put this request in its request packet, can ask for an infinitely long lease, or can periodically renew its lease. When a lease expires, the DHCP server is free to reassign the IP address to other hosts. A wise server should reuse least-recently-used addresses first, to deal with forgetful hosts that may retain their IP addresses past their lease.

A similar technique is used to dynamically allocate IP addresses to computers that access the Internet on a dial-up line. The widely-used *Point-to-Point Procotol (PPP)* allocates a temporary IP address to a host when it dials in. Since only a fraction of dial-up hosts are simultaneously active, an Internet Service Provider can share a pool of IP addresses among a larger set of dial-up hosts.

1.5.5 IPv6

Although CIDR bought the Internet Numbering Authority some breathing space, the 32-bit IP address space will eventually run out. This problem is being rectified in IP version 6, which uses 128-bit addresses. It has been calculated that, even in the most pessimistic scenario, this will result in more than 1500 IP addresses available for each square meter of surface area on the planet. Like v4, v6 distinguishes between multicast and unicast addresses based on a well-known prefix. Version 6 has address prefixes, as in CIDR, to allow aggregation without reference to classes. Subnetting with subnet masks is also allowed.

IP version 6 distinguishes among three types of addresses: unicast, anycast, and multicast. Unicast addresses are defined in RFC2374. Multiple levels of aggregation are defined (top-level aggregation, next-level aggregation and site-level aggregation).

Anycast addresses correspond to more than one interface. The idea is that a packet sent to an anycast address is sent to *one* of the interfaces sharing the address.

Multicast addresses correspond to multicast groups. They start with the bit sequence FF. A packet sent to a multicast address is routed to every member of the corresponding multicast group. Multicast addresses contain a flag to show whether the group is well known and therefore permanent, or whether it is transient.

To ease the transition from IPv4 to IPv6 addresses, a special form of the v6 address allows v4 addresses to be encapsulated. In these v6 addresses, the first 80 bits are zero and the next 16 bits are all ones. A complicated set of rules allows interoperation between v6 and v4 hosts through intermediate routers that may or may not understand v6 addresses.

1.6 Name resolution

Users typically supply applications with a destination's name. Applications use *name servers* to resolve a name to an address. A name server is like a telephone directory that stores name-to-address translations. An application resolves a name by sending a query to the name server, which responds with the translation.

The simplest design for a name server is to have a single name server for the entire network. This has the advantage that the resolution is always guaranteed to be consistent, because only one copy of the translation exists. On the other hand, the central name server is not only a single point of failure, but also a choke point, since every endpoint directs its name-resolution queries to it. Thus, we usually compose name servers from a set of distributed agents that coordinate their actions to provide the illusion of a single translation table. Perhaps the most successful distributed name service is the Internet's *Domain Name System (DNS),* which we will use as an example of good name server design.

Recall that DNS partitions names, such as *pc1.pclab.mech.berkeley.edu,* into several hierarchically organized domains, such as *mech, berkeley,* and *edu.* The hierarchy can span arbitrarily many levels. The DNS consists of many name servers, each responsible for a subtree of the name space corresponding to a domain boundary (Figure B3T18). Each server may delegate part of the name space to other servers. For example, the name server responsible for the *berkeley.edu* domain gives responsibility for names ending with **.mech.berkeley.edu* to the *mech.berkeley.edu* name server. This delegation of authority allows domains to match administrative boundaries, considerably simplifying namespace management. DNS administrators arrange things so that every DNS name is guaranteed to be correctly translated by at least one *Authoritative Server* for that name.

When an endpoint wants to translate a name, it sends a query to the server serving the root of the name space. The root parses the name right-to-left, determines the server responsible for the name (the name after the last "."), and forwards the query to that server. For example, if the query is for the name *pc1.pclab.mech.berkeley.edu,* the root server forwards the query to the server responsible for the *edu* domain. This server, in turn, hands off the query to the *berkeley* name server, and so on.

Although the scheme described so far allows each authoritative server to independently update its translation table, it still places a heavy load on the root name server. Moreover, if the root server fails, the entire name resolution process will come to a halt. DNS uses two techniques to combat these problems: *replication* and *caching.*

- *Replication:* DNS allows more than one server to handle requests for a domain, since these servers coordinate among themselves to maintain consistency. In particular, the root name server itself is highly replicated, so that a single failure will not cause any problems. A name resolution query can be made to any of the replicated servers, and an end-system typically chooses the nearest or least loaded one.
- *Caching:* When an endpoint (or an agent acting on its behalf) resolves a name, it stores the result in a cache.

Thus, if the query is repeated, it is answered without recourse to the root name server. Endpoints can cache not only the results of a query, but also the addresses of the Authoritative Servers for commonly queried domains. Thus, future queries need not go through the root server. This has been found to reduce name-resolution traffic dramatically. Cached entries are flushed after a time specified in the name-translation table at authorized servers, so that changes in the table are eventually reflected in the entire Internet.

1.7 Datalink-layer addressing

Although the bulk of our discussion of routing deals with network-layer addresses, we will delve briefly into datalink-layer addressing for two common datalink layers-Ethernet and FDDI- because they are commonly used for routing and bridging within a local-area network (LAN).

Both FDDI and Ethernet follow an addressing specification laid down by the 802 committee of the Institute of Electrical and Electronics Engineers (IEEE). This committee is responsible for most LAN standards, and every IEEE 802 LAN obeys the 802 addressing scheme (FDDI is not an IEEE 802 standard, but it uses 802 style addressing anyway).

An 802 address is 6 bytes long (Figure B3T20 top). The IEEE globally assigns the first 3 bytes, and host-adaptor manufacturers uniquely assign the next 3 bytes, imprinting the resulting address on a ROM on the host-adaptor card. The last two bits of the first byte of the address are used for the *global/local* and *group/individual* flags. If the global flag bit is set, then the address is guaranteed to be globally unique; otherwise, it has only local significance. Thus, an experimental prototype card can be assigned a temporary (but still valid) 802 address, if the local bit is set. The *group* bit marks the address as a datalinklayer broadcast or multicast address. Packets with this bit set can be received by more than one destination (but they must be previously configured to accept packets from this address). The group bit allows efficient datalink-level multicast and broadcast, since an 802 host adaptor needs to match the address on an incoming packet with a set of active multicast addresses only when the bit is set.

1.8 Finding datalink-layer addresses

As we saw in Section 1.6, distributed name servers allow us to translate from a name to a network-level address. This network-level address is used to locate a unique destination in the network. However. the destination's host-interface card may only recognize and accept packets labeled with its datalink-layer address (this problem only arises in broadcast LANs; in point-to-point LANs, a destination does not require a datalink-layer address). Thus, the last router along the path to the destination must encapsulate an incoming IP packet in a datalink-layer header that contains a datalink-layer address corresponding to the final destination (Figure B3T20 bottom). Symmetrically, if a source connects to a router within an 802 LAN, it must encapsulate the outgoing packet with a datalink-layer header containing the datalink layer address of the router's host-interface. We usually call the datalink-layer address the *Medium Access Control* or *MAC* address.

If both the source and destination of a packet are on the same broadcast LAN, we can easily translate from an IP address to a MAC. This is because the authoritative translation from a network-layer (IP) address to a MAC address is always available from at least one of the hosts on the LAN. When a source wants a translation, it broadcasts a query on the LAN, and the computer that owns the network-layer address replies with its MAC address. The Internet protocol that carries out this operation is called the *Address Resolution Protocol* or *ARP*.

An ARP packet contains the MAC address of the sender and the IP address that the sender wants to resolve. The packet is addressed with the LAN's broadcast address. Every host on the LAN is required to listen to ARP broadcasts and respond to an ARP query containing its own IP address with its MAC address. The reply is sent to the querier's MAC address, available from the ARP request. The originator of an ARP reply saves the reply in an *ARP cache so* that future translation will not require a broadcast. Cache entries are discarded after a while, so that if the MAC address of a host changes, this will eventually be noticed by every other host on the LAN.

ARP works well only on broadcast LANs. In point-to-point LANs, ARP queries are replied to by an *ARP server,* which is essentially a name server. When a host boots up, it registers its MAC address with the ARP server. When a host wants to find out a destination's address, it asks the ARP server, instead of initiating a broadcast. This technique *was* used when carrying IP over ATM LANs.

1.9 Summary

Naming and addressing are essential to the operation of any network. Although names can be arbitrarily long and are (usually) human understandable, addresses are usually fixed-length and are meant to be easily parsed by routers. Both names and addresses must be unique, yet should not require a naming authority to consult with every other naming authority in the network. For these reasons, both names and addresses are organized hierarchically. Moreover, hierarchical addressing allows us to aggregate sets of computers when describing routes to them in a routing table. Naming and addressing are closely tied to routing, which we study in the next section.

The Internet uses IP addresses, which come in two flavors, version 4 and version 6. Version 4, which is the current standard, divides the address into two parts, called the network number and the host number. A variable number of bits can be assigned to each portion by using a subnet mask. Sets of addresses can also be grouped together to form a larger address space by specifying an address prefix. Moreover, a host can be dynamically assigned an IP address when it joins the Internet. These three innovations allow us to manage the v4 address space more effectively, but it is still running out of addresses. Version 6 addresses, which are four times the size of version 4 addresses, promise to solve the address scarcity problem once and for all.

Name resolution is the process by which names are associated with addresses. A common solution is to distribute this functionality among a set of name servers. Replicating servers and caching replies are two techniques to increase reliability and to avoid excessive name-resolution traffic.

In broadcast LANs, we need to translate a network-layer address to a datalink-layer address. This is done in the Internet with the Address Resolution Protocol. A host broadcasts a resolution request on its LAN and receives a reply from any system that knows the answer. The telephone and ATM networks do not need address resolution because they are carried over point-to-point networks that do not need a datalink-layer address.

2.0 Routing

2.1 Introduction

Routing is the process of finding a path from a source to every destination in the network. It allows users in the remotest part of the world to get to information and services provided by computers anywhere in the world. Routing is what makes networking magical: allowing telephone conversations between Botswana and Buenos Aires, and video clips from the space shuttle to be multicast to hundreds of receivers around the world! How does a network choose a path that spans the world? How does the routing system scale to describe paths to many millions of endpoints? How should the system adapt to a failed link? What if the user wants to choose a path that has the least delay, or the least cost, or the most available capacity? What if the users are themselves mobile, attaching to the network from different wired access points? These are the sorts of questions we will study in this section.

Routing is accomplished by means of *routing protocols* that establish mutually consistent *routing tables* in every router (or switch controller) in the network (Figure B3T24). A routing table contains at least two columns: the first is the address of a destination endpoint or a destination network, and the second is the address of the network element that is the next hop in the "best" path to this destination. When a packet arrives at a router (or when a call-setup packet arrives at a switch controller), the router or switch controller consults the routing table to decide the next hop for the packet.

Example 2.1

An example of a routing table for a toy network is shown in Figure B3T24. We see that the routing table for node 1 has one entry for every other node in the network. This allows it to choose the next hop for every possible destination. For example, packets that arrive at node 1 destined for node 10 are forwarded to node 2.

Notice that node 1 has only two choices: to forward a packet to 2 or to forward it to 3. This is a *local* routing choice. Yet this choice depends on the *global* topology, because the destination address by itself does not contain enough information to make a correct decision. For example, the shortest path from node 1 to node 6 is through 2, and the shortest path to 11 is through 3. Node 1, just by looking at the destination address "6," cannot decide that node 2 should be the next hop to that destination. *We conclude that any routing protocol must communicate global topological information to each routing element to allow it to make local routing decisions.* Yet global information, by its very nature, is hard to collect, subject to frequent change, and voluminous. How can we summarize this information to extract only the portions relevant to each node? This lies at the heart of routing protocols.

A routing protocol asynchronously updates routing tables at every router or switch controller. For ease of exposition, in the remainder of the bloc, we will refer to both routers and switch controllers as "routers." Note that switch controllers are called upon to route packets only at the time of call setup, so that they route connections, instead of packets.

2.2 Routing protocol requirements

A routing protocol must try to satisfy several mutually opposing requirements:

- *Minimizing routing table space:* We would like routing tables to be as small as possible, so that we can build cheaper routers with smaller memories that are more easily looked up. Moreover, routers must periodically exchange routing tables to ensure that they have a consistent view of the network's topology: the larger the routing table, the greater the overhead in exchanging routing tables. We usually require a routing table to grow more slowly than the number of destinations in the network.

- *Minimizing control messages:* Routing protocols require control message exchange. These represent an overhead on system operation and should be minimized.

- *Robustness:* The worst thing that a router can do is to misroute packets, so that they never reach their destination. (They are said to enter a *black hole.)* Routers in error may also cause loops and *oscillations* in the network. Black holes, loops, and oscillations are rare under normal conditions, but can show up if routing tables are corrupted, users specify incorrect information, links break or are restored, or routing control packets are corrupted. A robust routing protocol should protect itself from these types of problems by periodically running consistency tests, and by careful use of checksums and sequence numbers as described in bloc 4.

Example 2.2

If routing tables are inconsistent, loops can easily be formed. For example, router A may think that the shortest path to C is through B, and B may think that the shortest path to C is through A. Then, a packet to C loops back and forth between A and B until some other procedure (such as a "time to live" reaching zero, as described in bloc 4) detects the loop and terminates the packet.

Oscillations can be caused if the routing protocol chooses paths based on the current load. Consider routers A and B connected by paths P1 and P2. Suppose P1 is heavily loaded and P2 is idle. The routing protocol may divert all traffic from P1 to P2, thus loading P2. This makes P1 more desirable, and traffic moves back to P1! If we are not careful, traffic oscillates from P1 to P2, and the network is always congested.

- *Using optimal paths: To* the extent possible, a packet should follow the "best" path from a source to its destination. The "best" path may not necessarily be the shortest path: it may be a path that has the least delay, the most secure links, or the lowest monetary cost, or one that balances the load across the available paths. Routers along the entire path must collaborate to ensure that packets use the best possible route to maximize overall network performance.

As always, these requirements represent trade-offs in routing protocol design. For example, a protocol may trade off robustness for a decrease in the number of control messages and routing-table space. Many common protocols trade off a dramatic reduction in routing-table space for slightly longer paths.

2.3 Choices

Designers of routing protocols have many mechanisms available to them. In this section, we will describe some commonly available choices for routing. These choices also represent a rough taxonomy to categorize routing protocols.

- *Centralized versus distributed routing:* In centralized routing, a central processor collects information about the status of each link (up or down, utilization, and capacity) and processes this information to compute a routing table for every node. It then distributes these tables to all the routers. In distributed routing, routers cooperate using a distributed routing protocol to create mutually consistent routing tables. Centralized routing is reasonable when the network is centrally administered and the network is not too large, as in the core of the telephone network. However, it suffers from the same problems as a centralized name server: creating a single point of failure, and the concentration of routing traffic to a single point.

- *Source-based versus hop-by-hop:* A packet header can carry the entire route (that is, the addresses of every router on the path from the source to the destination), or the packet can carry just the destination address, and each router along the path can choose the next hop. These alternatives represent extremes in the degree to which a source can influence the path of a packet. A source route allows a sender to specify a packet's path precisely, but requires the source to be aware of the entire network topology. If a link or a router along the path goes down, a sourcerouted packet will not reach its destination. Moreover, if the path is long, the packet header can be fairly large. Thus, source routing trades off specificity in routing for packet-header size and extra overhead for control messages. An intermediate solution is to use a *loose source route.* With loose source routes, the sender chooses a subset of routers that the packet should pass through, and the path may include routers not included in the source route. Loose source routes are supported in the IP version 4 and 6 headers.

- *Stochastic versus deterministic:* With a deterministic route, each router forwards packets toward a destination along exactly one path. In stochastic routing, each router maintains more than one next hop for each possible destination. It randomly picks one of these hops when forwarding a packet. The advantage of stochastic routing is that it spreads the load among many paths, so that the load oscillations characteristic of deterministic routing are eliminated. On the other hand, a destination may receive packets along the same connection out of order, and with varying delays. Consequently, modern networks usually use deterministic routing.

- *Single versus multiple path:* In single-path routing, a router maintains only one path to each destination. In multiple-path routing, a router maintains a *primary* path to a destination, along with *alternative* paths. If the primary path is unavailable for some reason, routers may send packets on the alternative path (with stochastic routing, routers may send packets on alternative paths even if the primary path is available). Single-path routing is used on the Internet, because maintaining alternative paths requires more routing table space. Telephone networks usually use multiple-path routing, because this reduces the call blocking probability, which is very important for customer satisfaction.

- *State-dependent versus state-independent:* With state-dependent or *dynamic* routing, the choice of a route depends on the current (measured) network state. For example, if some links are heavily loaded, routers may try to route packets around that link. With state-independent or *static* routing, the route ignores the network state. For example, a shortest-path route (where we measure the path length as the number of hops) is state independent. State-dependent routing usually finds better routes than state-independent routing, but can suffer from problems caused by network dynamics (such as the routing oscillations described earlier). It also requires more overhead for monitoring the network load. The Internet uses both state-dependent and stateindependent routing. Telephone network routing used to be state independent, but statedependent routing with multiple paths is now the norm.

Having broadly considered the choices in routing protocol design, the rest of the bloc deals with specific routing protocols that make a selection from the choices described earlier. The literature on routing (both in the telephone network and in the Internet) is vast. We focus on the study of the routing in the Internet.

2.4 Distance-vector routing

Telephone network routing is specialized to take advantage of the unique features of the telephone network, such as a predictable traffic flow, and a relatively small network core. Large packet networks, such as the Internet, present a very different environment. In the Internet, links and routers are unreliable, alternative paths are scarce, and traffic patterns can change unpredictably within minutes. It is not surprising that routing in the Internet, and in ATM networks, which are likely to have Internet-like characteristics, follows a different path. The two fundamental routing algorithms in packet-switched networks are *distance-vector* and *link-state.*

Both algorithms assume that a router knows (a) the address of each neighbor, and (b) the cost of reaching each neighbor (where the cost measures quantities like the link's capacity, the current queuing delay, or a per-packet charge). Both algorithms allow a router to find *global* routing information, that is, the next hop to reach every destination in the network by the shortest path, by exchanging routing information with only its neighbors. Roughly speaking, in a distance-vector algorithm, a node tells its *neighbors* its distance to *every* other node in the network, and in a link-state algorithm, a node tells *every* other node in the network its distance to its *neighbors.* Thus, both routing protocols are *distributed* and are suitable for large internetworks controlled by multiple administrative entities. In this section, we will focus on distance vector algorithms. We will study link-state algorithms in Section 2.5.

2.4.1 Distance-Vector Algorithm

In distance-vector routing, we assume that each router knows the *identity* of every other router in the network (but not necessarily the shortest path to it). Each router maintains a *distance vector,* that is, a list of <*destination, cost*> tuples, one tuple per destination, where *cost* is the current estimate for the sum of the link costs on the shortest path to that destination. Each router initializes the cost to reach all nonneighbor nodes to a value higher than the expected cost of any route in the network (commonly referred to in the routing literature as *infinity*[1]*). A* router periodically sends a copy of its distance vector to all its neighbors. When a router receives a distance vector from a neighbor, it determines whether its cost to reach any destination would decrease if it routed packets to that destination through that neighbor (Figure B3T31). It can easily do so by comparing its current cost to reach a destination with the sum of the cost to reach its neighbor and its neighbor's cost to reach that destination.

Example 2.3

In Figure B3T31, if router A has an initial distance vector of (<A, 0>, <B, 1>, <C, 4>, <D,-> we see that the arrival of a distance vector from B results in A updating its costs to C and D. If a neighbor's distance vector results in a decrease in a cost to a destination, that neighbor is chosen to be the next hop to get to that destination. For example, in Figure B3T31, the distance vector from B reduced A's cost to D. Therefore, B is the next hop for packets destined for D. A router is expected to advertise its distance vector to all its neighbors every time it changes.

We can show that even if nodes asynchronously update their distance vectors, the routing tables will eventually converge. The intuition behind the proof is that each router knows the true cost to its neighbors. This information is spread one hop with the first exchange of distance vectors, and one hop further on each subsequent exchange. With the continued exchange of distance vectors, the cost of every link is eventually known throughout the network. The distance-vector algorithm is also called the Bellman-Ford algorithm.

2.4.2 Problems and solutions with distance-vector routing

The distance-vector algorithm works well if nodes and links are always up, but it runs into many problems when links go down or come up. The root cause of problems is that when a node updates and distributes a distance vector, it hides the sequence of operations it used to compute the vector. Thus, downstream routers do not have sufficient information to figure out whether their choice of a next hop will cause loops to form. This will become clear when we look at the *count-to-infinity* problem.

[1] Infinity = 16, RIP uses 16 as the infinite metric.

Count-to-infinity

We illustrate this problem with the next example.

Example 2.4

Consider the simple network shown in Figure B3T33. Initially, A routes packets to C via B, and B uses its direct path. Now, suppose the BC link goes down. B updates its cost to infinity, and tells this to A. Suppose, in the meantime, A sends its distance vector to B. B notices that A has a two-hop path to C. Therefore, it updates its routing table to reflect this information, and tells A that it has a three-hop path to C. In the previous exchange, A discovered that B did not have a path to C any more, and had updated its table to reflect that. When B joyfully announces that it does indeed have a path, A updates its routing table to show a four-hop path to C. This process of increasing the hop count to C continues until the hop count reaches infinity, when both nodes realize that no route to C exists after all. Note that during the process of counting-to-infinity, packets from A or B destined to C are likely to loop back and forth between A and B. Thus, if the counting process takes a while, many packets may wander aimlessly in the network, making no progress, and causing congestion for everyone else. It makes sense to try to avoid counting to infinity.

Path vector

The reason for count-to-infinity is that when B updated its path to C to go through A, it did not realize that A's path to C was through B. In other words, the distance vector that A sent B hid the fact that B was on A's route to C. There are several ways to add information to the distance vector to solve this problem. One solution is to annotate each entry in the distance vector with the path used to obtain this cost. For example, in Step 2 of Figure B3T33, A can tell B that its cost to C is 2, and the path to C was C-B. When B sees this, it realizes that no route to C exists, and the count-to-infinity problem goes away. This solution is also called the *path-vector* solution, since routers annotate the distance vector with a path. The path-vector approach is used in the *border gateway protocol (BGP)* in the Internet core. Note that path vectors trade off a larger routing table and extra control overhead for robustness.

Split horizon

The problem with path vectors is that the vectors require large table sizes, which can prove expensive. Several other solutions to the count-to-infinity problem avoid this overhead. In one solution, called *split-horizon routing,* a router never advertises the cost of a destination to its neighbor N, if N is the next hop to that destination. For example, in Figure B3T33, this means that A does not advertise a cost for C to B because it uses B as its next hop to C. This trivially solves the count-to-infinity problem in Figure B3T33. However, split horizon works only when two adjacent routers count to infinity: it is ineffective when three routers mutually do so.

A variant of split-horizon, called *split horizon with poisonous reverse,* is used in the *Routing Information Protocol (RIP)* on the Internet. When A routes to C via B, it tells B that it has an *infinite* cost to reach C (with normal split horizon, A would not tell B of a path to C at all). Though this sometimes accelerates convergence, it does not prevent three-way counting to infinity.

Triggered updates

While the classical distance-vector algorithm prescribes that a router should advertise its distance vector every time it changes, this can lead to a flurry of updates every time a link cost changes. If, for example, the cost measures link delay, a router may update its distance vector quite often. To prevent this, most distance vector algorithms prescribe that distance vectors be advertised only once in about 30 seconds. This adversely affects the time taken to recover from a count-to-infinity situation. Consider the situation in Figure B3T33, where each node must count from 1 to infinity. If we define infinity to be 16, then it will converge only 15 * 30 seconds later = 7.5 minutes. The network will be in an unstable situation during this entire interval. To avoid this, we can trigger distance vector changes immediately after a link is marked down. This rapid propagation removes some race conditions required for count-to-infinity and is adopted in the Internet RIP protocol.

Source tracing

The key idea for source tracing is to augment a distance vector so that it carries not only the cost to a destination, but the router *immediately preceding* the destination. We can show that this information is sufficient for a source to construct the entire path to the destination.

When a router updates its distance vector, if its cost to a destination decreases, it replaces the preceding-router field for that destination in its routing table with the corresponding value in the incoming distance vector. Distance vector with source tracing is guaranteed to be loop free if routers follow the rule that if a router changes its notion of the next hop for a destination D, then it should use the same neighbor for all destinations for which

D lies along the shortest path.

Example 2.5

Consider Figure B3T35 which shows the routing table for router 1. Suppose we want to trace the path to router 6. First, we locate 6 in the routing table and see that the preceding router on the path to 6 is 5. We now look for 5, and find 4 as the preceding router. Continuing in this fashion, it is easy to compute the path as 1-2-4-5-6. This allows us to get the same information as a path vector, but with very little additional table space.

DUAL

The *distributed update algorithm (DUAL) is* a technique to assure loop-free routing tables even in the presence of rapid changes in network topology. With DUAL, a router maintains a pared down version of the network topology by storing the distance reported by each of its neighbors to each destination (this is just the union of their distance vectors). If the cost from a particular router R to a destination D *decreases* because of the receipt of a distance vector from a neighbor N, then it is impossible for a loop to form if R updates its tables based on that distance vector. The reason is that if a loop forms, the reported cost to D from N in the incoming distance vector C2 must *include* the previously reported cost from R to D, Cl. Thus, C2 must be larger than Cl. If R updates its tables only for distance vectors such that C2 < Cl, then loops will not form.

Now, suppose R receives an update such that the distance to a destination increases because of an increase in a link's cost or because of a link failure. Then, R first checks if it can find a shorter path to this destination through another neighbor using its topology table. If not, *it freezes its* routing table and distributes the new distance vector to all its neighbors. The neighbors check whether this increases their cost to D. If so, they freeze their tables in turn and spread the vector to their neighbors. The computation *expands* in this manner until all the routers affected by the change (that is, all routers whose distance to any endpoint increases because of this change) know of it. If all the neighbors of a router already know of the change or are unaffected, they inform the router that they are done. The router unfreezes its state and informs the router that previously informed it of the change, which, in turn, propagates this information. Thus, the computation *contracts* until the router that first detected the change knows that the effect of the change has propagated to every router that ought to know of it. This is called a *diffusion computation.* It can be shown that the DUAL algorithm results in loop-free routing. DUAL is implemented in the Extended Interior Gateway Routing Protocol (EIGRP), a proprietary routing protocol from Cisco Systems.

2.5 Link-state routing

In distance-vector routing, a router knows only the cost to each destination or, sometimes, the path to the destination. This cost or path is partly determined on its behalf by other routers in the network. This hiding of information is the cause of many problems with distance-vector algorithms.

In contrast, the philosophy in link-state routing is to distribute the topology of the network and the cost of each link to all the routers. Each router independently computes optimal paths to every destination. If each router sees the same cost for each link and uses the same algorithm to compute the best path, the routes are guaranteed to be loop free. Thus, the key elements in link-state routing are a way to distribute knowledge of network topology to every router in the network, and a way to compute shortest paths given the topology. We will study these in turn.

2.5.1 Topology dissemination

Each router participating in the link-state algorithm creates a set of *link-state packets (LSPs)* that describe its links. An LSP contains the router's ID, the neighbor's ID, and the cost of the link to the neighbor. The next step is to distribute a copy of every LSP to every router using *controlled flooding.* The idea is that when a router receives a new LSP, it stores a copy of the LSP in an *LSP database,* and forwards the LSP to every interface other than the one on which it arrived. It can be shown that an LSP is never transferred over the same link twice in the same direction. Thus, if a network has *E* edges, flooding requires at most *2E* transfers.

Example 2.6

In Figure B3T39, A creates two LSPs, <A, B, 1> and <A, C, 4>. The other nodes in the network create similar LSPs. Let us trace the path taken by the LSP <A, B, 1> that originates from A. In the first step, the LSP reaches B, which in turn forwards it to C and D, but not to A, because it arrived from A. When C gets the LSP, it forwards it to A and D. A does not forward the LSP further, because its database already contains the LSP. If D got the LSP from B before it got it from C, D detects that the LSP is a duplicate and does nothing. Otherwise, it forwards it to B, who does nothing. Thus, in a few short steps, the LSP reaches every router in the network.

Sequence numbers

Although flooding is easy to understand when links and nodes stay up, as with distance vector algorithms, complexity creeps in when links or routers can go down. For example, in Figure B3T39, suppose link AB goes down. We would like the LSP corresponding to AB to be removed from all the other routers. Router B detects that link AB is down and sends an LSP with an infinite cost for AB to all the other routers. The other routers must somehow determine that this LSP overrides the information already existing in their databases. Therefore, every LSP must have a sequence number, and LSPs with newer sequence numbers override LSPs with older sequence numbers. This allows us to purge the old LSPs from each router's database.

Wrapped sequence numbers

Unfortunately, every sequence number has finite length and therefore is subject to wraparound. We have to ensure that a new LSP that has a numerically lower (but wrapped-around) sequence number still overrides an old LSP that has a numerically higher sequence number. For example, if sequence numbers are three bits long, thus spanning the space from 0 to 7, we would like a newer LSP with sequence number 0 to override an older LSP with sequence number 7. We can solve this problem by using very large sequence numbers. Then it is almost certain that if the difference between the sequence numbers of an existing and an incoming LSP is large, so that the numerically smaller LSP is actually newer. For example, if sequence numbers are 32 bits long, then they span the space from 0 to 4,294,967,295. If an old LSP in the database has a sequence number toward the end of the space, say, 4,294,967,200, and a new LSP has a sequence number toward the beginning of the space, say, 20, then the difference is 4,294,967,180. We therefore declare the LSP with sequence number 20 to be the newer one. More precisely, if the LSP sequence number space spans *N* sequence numbers, sequence number *a* is older than sequence number *b* if:

$$a < b \text{ and } |b-a| < N/2,$$

$$\text{or } a > b \text{ and } |b-a| > N/2.$$

Initial sequence number

When a router starts, it must choose a sequence number such that its LSPs in other routers' databases are overridden. If the router does not know what LSPs it used in the past, it may risk flooding new LSPs that are always ignored. For example, with the 0 to 2^{32}-1 sequence space, the LSPs in the databases may have a sequence number 5. If the router comes back up and chooses to start numbering LSPs with sequence number 0, other routers ignore the new LSPs. There are two ways to solve this problem: *aging* and a *lollipop sequence space*.

Aging

With *aging,* the creator of an LSP sets a field in the LSP header to a maximum age (MAX_AGE). A router receiving this LSP copies the current age to a per-LSP counter in its database and periodically decrements it. If decrementing an LSP counter makes it zero, the router purges the LSP from its database. To preserve a consistent view of the network topology, the router should quickly request the rest of the network to discard this LSP. It does so by initiating flooding with the zero-age LSP. When a router gets an LSP with zero age and the latest sequence number, it purges the LSP and floods the zeroage LSP to its neighbors. This quickly restores consistency. After a purge of an LSP from a particular router, any subsequent LSP from that router will automatically enter the LSP database. Thus, if a newly booted router waits for a while before sending new LSPs, it knows that its old LSPs will be purged, and its new LSPs will override all LSPs from its previous incarnation.

Although this scheme does work, the choice of initial LSP age is problematic. We would like the latest LSP from a router to be flooded throughout the network before its previous one times out. Otherwise, some routers may purge the LSP from their databases before the new LSP reaches them, leading to inconsistent routes. To minimize the overhead of sending LSPs frequently, we should use a fairly large initial LSP age, on the order of an hour or so[2]. However, to allow purging of old LSPs, after rebooting, a router must wait for a time on the order of the initial LSP age before it can start sending new LSPs. So, we cannot simultaneously minimize control overhead and the dead time after a router reboots.

Lollipop sequence space

A better solution is for newly booted routers to use a sequence number that uniquely differentiates it from every other sequence number that it could have used in the past. Although this is impossible with a circular sequence space, we can achieve this using a *lollipop sequence space* (Figure B3T42). Here, we have partitioned the sequence space of size *N* into three parts: a negative space from *-N/2* to 0, the sequence number 0, and a positive space of size *N/2 - 1.* When a router comes up, it uses the sequence number *-N/2* for its LSPs, and subsequent LSPs use *-N/2 + 1, -N/2 + 2,* etc. When the sequence number becomes positive, subsequent sequence numbers

[2] If the initial LSP age is small, then the time interval between the creation of two LSPs must be small; otherwise, a distant router may time out an LSP before the next LSP reaches it. This increases routing overhead.

wrap around in the circular part of the space. An LSP with sequence number *a* is older than an LSP with sequence number *b* if:

- $a < 0$ and $a < b$, or
- $a > 0$, $a < b$, and $b - a < N/4$, or
- $a > 0$, $b > 0$, $a > b$, and $a - b > N/4$

Note that $-N/2$ is therefore the oldest sequence number.

We add the rule that if a router gets an LSP from another router that has an older sequence number than the one in its database, it informs the other router of its sequence number. Because a newly booted router always generates a packet with the oldest sequence number, it is guaranteed to be told by its neighbors of the sequence number it had used before it crashed. It then jumps to a sequence number 1 larger than this, so that subsequent LSPs override its past LSPs. For example, as shown in Figure B3T42, a newly booted router starts with sequence number -4. If existing routers have an LSP from this router with a sequence number *2,* they inform the newly booted router of this. The newly booted router then uses 3 for its newer LSPs and continues to number packets as 3, 0, 1, 2, 3*,* 0, etc., until it boots again. This solution does not require the newly booted router to wait for its LSPs to require. Intuitively, the neighbors of a router act as a distributed memory recording its actions. By sending a unique packet (with sequence number $-N/2$), a newly booted router can access this memory to regain its past state.

Recovering from a partition

LSP databases remain coherent if the network does not partition into two or more fragments. However, when recovering from a partition, databases may become inconsistent. We illustrate this with the next example.

Example 2.7

Consider the network shown in Figure B3T44. Assume that at the start of time, all routers have consistent LSP databases. Now, suppose link 4-5 breaks. This partitions the network into two independent fragments. If links 7-8 and 1-2 break later, the databases in each fragment evolve independently of each other. For example, node *2* is unaware of the break in link 7-8. This does not pose a problem if 4-5 stays down. However, when it comes up, routers on each side of the partition must update their view of the other side; otherwise, routing loops are possible (for example, 2 may route packets to 8 via 7, not knowing that link 7 - 8 is down).

Routers on each side of the newly restored link cooperate to restore LSP databases. Each LSP in the database is associated with a link ID and a version number. Routers increment the version number each time the LSP value changes. A set of *database descriptor records,* also maintained in the database, describe the link IDs and version numbers in the database. Database descriptor records are like link-state packets, except that they store far less information, so that exchanging these records is less expensive than exchanging LSPs. When a link comes up, routers at each end exchange a complete set of database descriptor records. By comparing them, each determines the set of records that are either nonexistent or out-of-date in their database. They request their peer router to send them these LSPs, which they then flood into their respective fragments. This restores a uniform view of the topology to the entire network.

Link or router failure

When a link fails, the routers on either side notice this and can flood the network with this information. Thus, link failures are relatively easy to recover from. However, when a router fails, there is no direct way to detect this. Most link-state protocols require routers to exchange HELLO packets with their neighbors. If a router does not respond to a series of HELLOs, it is likely to be down. The neighbors should immediately flood this information.

It is possible to construct scenarios where, because of a series of failures, the HELLO protocol does not detect a dead router (for example, an undetectable corruption in the source address may make a HELLO packet from a router that is alive look like a HELLO from a dead router). To prevent databases from becoming corrupted without explicit failure detection, LSP records are usually aged (even with lollipop-space sequence numbers). When an LSP times out at some router, the router immediately floods the network with a special packet that informs every other router that the LSP timed out, and that they should delete this (stale) LSP from their LSP database. This allows the network eventually to recover from almost every possible sequence of failures.

Securing LSP databases

Loop-freeness in link-state routing requires that all routers share a consistent view of the network. If a malicious agent injects spurious LSP packets into a router, routing becomes unstable. Thus, routers must actively protect their LSP database not only from corruption, but also from malicious interference. Several techniques for securing LSP databases are well known. First, link-state packets are protected by a checksum, not only when sent over a transmission link, but also when stored in the database. This detects corruption on the link or on a disk. Second, the receiver acknowledges LSP exchanges, so that a sender can recover from link losses using timeouts and retransmissions. Third, LSP exchanges are authenticated by a password known only to routing administrators. This makes it harder for malicious users to inject LSPs into a database. Several other techniques to ensure security in LSP exchange are described in reference.

2.5.2 Computing shortest paths

Thus far, we have seen how every router in the network obtains a consistent copy of the LSP database. We now study how a router can use this database to compute optimal routes in the network. A router typically uses Dijkstra's shortest-path algorithm to do so.

Dijkstra's algorithm

Dijkstra's algorithm computes the shortest path from a *root* node (corresponding to the router where the algorithm is being run) to every other node in the network. The key idea is to maintain a set of nodes, P, for which the shortest path has already been found. Every node outside P must be reached by a path from a node already in P. We find out every way in which an "outside" node *o* can be reached by a one-hop path from a node already in P, and choose the shortest of these as the path to *o*. Node *o* can now be added to P, and we continue in this fashion until we have the shortest path to all the nodes in the network.

More precisely, we define two sets P and T (standing for permanent and temporary). Set P is the set of nodes to which shortest paths have been found, and set T is the set of nodes to which we are considering shortest paths. We start by initializing P to the current node, and T to null. The algorithm repeats the following steps:

1. For the node *p* just added to P, add each of its neighbors *n* to T such that (a) if *n* is not in T, add it, annotating it with the cost to reach it through *p* and *p*'s ID, and (b) if *n* is already in T and the path to *n* through *p* has a lower cost, then remove the earlier instance of *n* and add the new instance annotated with the cost to reach it through *p* and *p*'s ID.

2. Pick the node *n* that has the smallest cost in T and, if it is not already in P, add it to P. Use its annotation to determine the router p to use to reach *n.* If T is empty, we are done.

When the algorithm stops, we have, for each router, the router on the shortest path used to reach it. As we did with source tracing for distance-vector routing, this allows us to compute the next hop on the shortest path for every destination in the network. Figure B3T48 shows an example of Dijkstra's algorithm.

2.5.3 Link state versus distance vector

Given a choice between link-state and distance-vector routing, which style should we prefer? Conventional wisdom is that link state algorithms are more stable because each router knows the entire network topology. On the other hand, transient routing loops can form while the new topology is being flooded. If the network is so dynamic that links are always coming up or going down, then these transients can last for a long time, and the loop-free property is lost. Moreover, as we have seen, simple modifications to the vanilla distance-vector algorithm can prevent routing loops. Thus, one should not prefer link-state protocols for loop-freeness alone.

A second argument in favor of link-state algorithms is that they allow multiple routing metrics. The idea is that each LSP can carry more than one cost. Thus, each router can compute multiple shortest-path trees, one corresponding to each metric. Packets can then be forwarded on one of the shortest-path trees, which they can select with a flag in the header. For example, an LSP may carry a delay cost and a monetary cost. This would allow every router to compute a shortest-delay tree and a lowest-monetary-cost tree. Incoming packets that prefer lower delays (and, perhaps, are willing to pay for it) would be routed according to the shortest-delay path.

Although this sounds attractive at first, it assumes that every router will agree to report the same set of metrics. If some routers do not report some metrics, this is not a disaster if all the other routers assign it a consistent default. However, the benefits from multiple metrics seem more tenuous if a considerable fraction of the routers along the path choose not to report one or more metrics of interest. Moreover, the benefits of multiple metrics can be realized by path-vector-type distance-vector algorithms.

Third, we prefer link-state algorithms because, after a change, they usually converge faster than distance-vector algorithms. It is not clear that this holds if we use a distance-vector algorithm with triggered updates and one of the several algorithms to ensure loop-freeness (and therefore, absence of counting to infinity). Convergence depends strongly on the network topology, the load on the routing protocol, and the exact sequence of link failure and recovery. Thus, it is impossible to argue convincingly for either link state or distance vector.

Distance-vector algorithms do seem to have two advantages over link-state algorithms. First, much of the overhead in link-state routing is in the elaborate precautions necessary to prevent corruption of the LSP database.

We can avoid these in distance-vector algorithms because we do not require that nodes independently compute consistent routes. Second, distance-vector algorithms typically require less memory for routing tables than do link-state protocols. Again, this is because they do not need to maintain an LSP database. On the other hand, this advantage disappears when we use path-vector-type distance-vector algorithms.

Because there is no clear winner, both distance-vector and link-state algorithms are commonly used in packet-switched networks. For example, in the Internet, the two "modern" routing protocols are Open Shortest Path First (OSPF), which is a link-state protocol, and Border Gateway Protocol (BGP), which is a path-vector protocol (more about these in Section 2.9). Examples of both algorithms will probably exist in datagram networks for many years to come.

2.6 Choosing link costs

Thus far, we have assumed that network administrators somehow assign a reasonable cost to each link in the network, which they then distribute to other routers in the network. We have not really considered how the choice of a link cost affects the flow of traffic in the network. As we see next, the cost of a link and the load on it are coupled in a manner reminiscent of the Erlang map, which couples routing strategies and blocking probabilities in the telephone network. The key idea is that the choice of link costs implicitly defines the way in which traffic load is distributed in the network. The lower the cost of a given link, the higher the probability that it is a part of a shortest path to some destination, and the higher the expected load on it. Therefore, if link costs depend on the current load on the link (which is usually a good idea), a high cost lowers the load on the link, which, in turn, lowers its cost. Our goal is to choose an appropriate cost function so that the load and cost converge on a desirable fixed point. A poor cost function leads to routing oscillations, which are highly undesirable.

2.6.1 Static metrics

For the moment, let us ignore the dynamics of routing and focus on the simplest possible way to assign weights, the *hop-count* metric. Here, we give every link a unit weight, so that the shortest-cost path is also the path with the smallest hop count. Allocating all links a unit weight is reasonable when the links are homogeneous. However, it makes little sense if some links run at DS3 speeds (45 Mbps), while others are DS1 (1.5 Mbps). Here, we should probably give links with lower bandwidth a higher cost, so that the load is mostly carried on high-capacity links. This is illustrated in the next example.

Example 2.8

Consider the network in Figure B3T52. Here, links AB, AC, and BD are T3 links, and BC and CD are T1 links. If we assign all links a unit cost, then traffic from B to C will go over the BC link. All other things being equal, it is a better idea to route B-C traffic on the path B-A-C, because it has nearly thirty times the capacity. Therefore, we could assign T1 links a weight of 10, and T3 links a weight of 1. Then, links BC and CD are never used (unless one or more of the T3 links goes down). Unfortunately, even if link AB is highly congested and BC is idle, traffic will still take the B-A-C path instead of the BC path. This points out the inherent problems of statically assigning weights to links. It *may* be a better idea to assign link costs *dynamically*, based on the current load on the link.

2.6.2 Original ARPAnet dynamic metrics

One of the earliest dynamic cost allocation techniques was used in the original ARPAnet. In this scheme, the cost of a link is directly proportional to the length of a router's output queue at the entrance to that link. If a link has a long queue, no matter the link's capacity, it is considered overloaded and given a higher cost. Continuing with Example 2.8 and Figure B3T52, assume that link A-B was heavily loaded in the A-to-B direction. Then, the queue at router A for that link would be long. If A therefore advertises a higher cost for A-B, this would divert the C-to-B traffic to the path C-B from C-A-B, reducing the load on A-B.

Although the idea of a dynamic link cost is a good one, the original ARPAnet implementation is a case study in the unintended consequences of a complex design. In its defense, the scheme did work well when the network was lightly loaded. However, many problems appeared under a heavy load. First, the link cost depended on the queue length averaged over 10 seconds. Since the backbone ran at only 56 Kbps, this represented too small a time granularity at which to measure queue lengths. Thus, tran sient spikes in the queue length could trigger major rerouting in the network. Second, link costs had a wide dynamic range (that is, they could be very low or very high). Consequently, it turned out that the network completely ignored paths with high costs. Although we should avoid high-delay links, they should not be unused! Third, the queue length was assumed to be a predictor for future loads on the link. In other words, if the queue length was long, the link cost was increased in the expectation that the link would continue to be overloaded in the future. In fact, the opposite was true. When a link's cost was large, it was avoided, so that when routes were recomputed, the link's load dramatically decreased. Fourth, there was no restriction on the difference between successively reported costs for a link. This allowed link costs to oscillate rapidly. Finally, all the routers tended to recompute routing tables simultaneously. Thus, links with low costs would be chosen to be on the shortest path simultaneously by many routers, flooding the link.

2.6.3 Modified ARPAnet metrics (or Dynamic metrics II)

The modified version of the ARPAnet link-cost function avoided many errors made in the first version and was much more successful. In this scheme, link costs are a function not only of the measured mean queue length, but also of the link's capacity. When the link's load is low, its cost depends entirely on the link's capacity, and the queue length comes into play only at higher loads. Thus, at low loads, network routing essentially uses static costs, making it stable. Moreover, link costs are hop *normalized,* that is, the weight of a link is measured in "hops". Traversing a link with a weight c is as expensive as traversing c links with unit weight. The higher the advertised hop-normalized cost, the greater the barrier to using the link, but the barrier is not overwhelmingly high.

Two schemes were also added to dampen the oscillation in link costs. First, the dynamic range of link costs was reduced from a range of 127:1 to 3:1 (the worst cost a link can advertise is that it equals 3 hops). Second, a router was allowed to change the link cost by only half a hop in successive advertisements. With these changes, and a few others, routing oscillations were nearly eliminated even under heavy load.

2.6.4 Routing dynamics

We mentioned earlier that the load on a link and the probability of its use in shortest-path routes are tightly coupled. We illustrate this by using two functions called the *metric map* and the *network response map*. The metric map translates the load on a link to its link-cost metric and is the link-cost function we described in the previous paragraph. The network response map translates from a given link metric to the expected load on that link, given the current topology and traffic load. Although the metric map is precisely defined, the network response map is empirically determined by modifying the cost of one link at a time and measuring the additional traffic due to that change, then averaging this over all links. We show the general form of these maps in Figure B3T55. In Figure B3T55 top (a) we see that as the load increases, the link cost first is flat (as explained earlier), and then rises linearly to 3, where it saturates. In Figure B3T55 top (b), we show a family of curves, each corresponding to an overall network load, which plot the load on an "average" link as a function of the cost of that link. We see (in the case of 25% load) that as the link cost increases, the mean load on a link decreases from 0.2, when the cost is close to 0, to nearly 0, when the cost is 5.

We can envision the dynamics of routing in the network by putting these two maps together, as shown in Figure B3T55 bottom. Paths in this map show the evolution of the system. We choose an arbitrary initial load in the system, and compute the corresponding cost metric by drawing a horizontal line and choosing its intercept on the metric curve. The corresponding load in the next time step can now be obtained by drawing a vertical line through that metric and noting the intercept on the load curve. By repeating these steps, we can determine the dynamics of the system starting from an arbitrary initial load.

For example, consider the path marked *a* in the figure. This represents a link at equilibrium, where the link load and its cost metric suffer from a bounded oscillation. Note that the link cost is allowed to change by only half a hop in successive advertisements, which tightly bounds the range of oscillations. This fact is dramatically illustrated by the path marked *b*. Here, we see the effect of introducing a new link into the system. We artificially start the link off with a high cost, reducing the cost by half a hop each time step. Because of the high initial cost metric, the initial load on the link is low. Each subsequent advertisement therefore reduces the metric by half a hop, gradually increasing the load. The network eventually stabilizes with a small, bounded oscillation. If link costs were allowed to change by larger amounts in successive advertisements, link loads and metrics would suffer from large oscillations. System evolution diagrams such as these are very useful in evaluating heuristics for link-cost metrics.

2.7 Hierarchical routing

If a network with N nodes and E edges uses link-state routing, it can be shown that computing shortest paths takes $O(E \log E)$ computation at each router, and the routing table requires $O(N)$ storage. E is at least the same size as N, because even for a tree-shaped graph, which requires the smallest number of edges for a given number of nodes $E = N - 1$. Clearly, the computation and space requirements for a routing protocol become excessive when N is large. Because both the Internet and the telephone network are expected to grow to several billion endpoints, we must use *hierarchical routing* to rein in routing costs.

We alluded to hierarchical routing when we discussed hierarchical addressing and address aggregation in Section 1. The idea is to partition the network into multiple hierarchical levels. A handful of routers in each level are responsible for communication between adjacent levels. Thus, at each level, only a few hundred routers need to maintain shortest-path routes to each other. A router that spans a hierarchy boundary agrees to route packets from the rest of the network to every router in its "area", and from every router in its area to the rest of the network.

2.7.1 Features of hierarchical routing

Figure B3T59 shows a detailed view of how a nation-wide Internet Service Provider might put together a hierarchically routed network. First, note that we have partitioned the network into four routing levels. Each level contains only a few routers, thus making it easy to compute routing tables.

Second, the network is not a strict hierarchy, because more than one router may advertise reachability to the same part of the address space. For example, both routers in Los Angeles, named LA0 and LA1, advertise that they can carry traffic for addresses of the form 6.*. When a packet with an address in 6.* arrives at San Francisco, it should forward the packet to LA0, but the router at Atlanta should forward it to LA1. Making the hierarchy looser lets the network survive faults more easily. For example, if LA0 went down, traffic for destination in 6.* could be routed via LA1.

Third, routers that span levels, such as 6.0.0.0 and 6.4.0.0, participate in routing protocols at both levels. For example, router 6.4.0.0 discovers, using a level-3 routing protocol, that the shortest path to networks advertised by router 6.3.0.0 is its direct link to 6.3.0.0. If the link goes down, level-3 routing informs 6.4.0.0 that it should use router 6.1.0.0 instead. 6.4.0.0 also participates in level-2 routing to find, for instance, that the shortest path to 6.4.2.0 is through 6.4.3.0. The routing protocols at the two levels may be completely different (one may be link-state, and the other distance-vector). Therefore, routers that route between levels must be prepared to speak multiple protocols.

Finally, note that we have a router in level 3 marked 21.1.2.3. Why would a router with address 21.* be placed under a router with address 6.*? This might be because of address-space exhaustion in the 6.* space. Or, a company that had obtained the address space 21.1.2.* for its computers from a service provider with authority over 21.*, then moved to a new location, might want to retain its old addresses (because renumbering computers on the Internet requires manual reconfiguration of every end-system and router). In any case, router 6.2.0.0 must advertise reachability to 21.1.2.3. Moreover, routers 6.0.0.1 and 6.0.0.2 at the network core must also advertise that they can route to 21.1.2.3, and every other router in the core must know that packets for 21.1.2.3 must be forwarded to one of these two routers. The lesson is that if we introduce a router at a lower level whose address cannot be aggregated into an existing address space, then each router in the core of the network must contain a routing table entry for it.

In the current Internet, addresses obey a three-level hierarchy (network number, subnet number, and host number). Because the highest possible degree of aggregation of addresses is at the network level, routers in the network core, which benefit most from aggregation, advertise routes to networks. For example, routers at the core will usually advertise routes to network 135.104.*, instead of to 135.104.53, 135.104.52, etc., which are subnets within 135.104. This approach to aggregation works well when the number of networks is small and routers at a lower level can handle routing within a network. Unfortunately, because of exhaustion of Class B addresses, many networks received multiple Class C network addresses instead. Consequently, routers in the core need to store table entries for routes to thousands of Class C networks. Each core router carried routes to more than 80,000 networks. Thus, even if addresses are hierarchical, they must be carefully managed, or routing tables and route computation can still be expensive. The CIDR scheme for addressing, discussed in Section 1, alleviates some of these problems.

2.7.2 External and summary records

Consider the four level-3 routers in Figure B3T59 with addresses 6.1.0.0, 6.2.0.0, 6.3.0.0, and 6.4.0.0. Suppose they use link-state routing to compute routes. What should be the next hop for a packet arriving at 6.4.0.0 that is destined to an address in 5.*? From the topology of the network, note that if the network uses a least-hop cost metric, then the next hop should be 6.3.0.0. Thus, we want router 6.4.0.0 to discover that there is a 3-hop path through router 6.3.0.0 to 5.0.0.0, whereas the path through 6.2.0.0 is at least 4 hops long. Unfortunately, since router 5.0.0.0 is not part of the level-3 network, 6.4.0.0 would not ordinarily be aware of its existence. We need a mechanism that allows routers that participate in level-4 routing to advertise paths to routers that are external to the level-3 network. This is done by using *external records* in the LSP database.

For example, router 6.0.0.0, which knows that it has a 1-hop path to 5.0.0.0 using level-4 routing, creates an *external LSP* that advertises a link to 5.0.0.0 with a link cost of 1 and floods this within the level-3 network.

Similarly, 6.0.0.1 uses level-4 routing to learn that its least-cost path to 5.0.0.0 is 2 hops and floods this information in the level-3 network. Thus, routers 6.0.0.1 and 6.0.0.2 pretend that 5.0.0.0 is a level-3 router that happens to be connected to them with a 1- and 2-hop path, respectively. When this information is propagated within the level-3 network, 6.4.0.0 automatically discovers that its shortest path to 5.0.0.0 is through 6.3.0.0, as we wanted. External LSPs, therefore, allow optimal routes to be computed despite the information-hiding inherent in hierarchical routing.

An external record allows routers within level 3 to discover shortest paths to *external* networks. The symmetrical problem is for external networks to discover shortest paths to level-3 networks that are not visible at level 4. This is done using summary records. For example, 6.0.0.0 advertises to level-4 routers that it has a path to 6.1.0.0 that is of length 2, to 6.2.0.0 of length 3, to 6.3.0.0 of length 1, and to 6.4.0.0 of length 2. It is as if these are single links with higher costs. Level-4 routers do not need to know the exact topology within the level-3 network, just the costs. Note that cost information in a summary record is functionally equivalent to a distance vector, because it summarizes the distance from a level-4 router to every level-3 router connected to it.

The network uses summary records to compute optimal paths. For example, the Atlanta router knows from 6.0.0.1's summary records that it has a 1-hop path to 6.2.0.0. It also knows (from level-4 routing) that it has a 1-hop path to 6.0.0.1. Therefore, its cost to reach 6.2.0.0 through 6.0.0.1 is 2 hops. In contrast, its path to 6.2.0.0 via 6.0.0.0 is (from these same sources of information) 5 hops. Therefore, it routes packets destined to 6.2.0.0 through 6.0.0.1, as we wanted. In "summary", external records inform "lower level" routers about "higher level" topology and summary records inform "higher level" routers about "lower level" topology. Note also that border router belong to both levels. Continuing with our example, the LSP database at (border) router 6.0.0.0 therefore contains the following:

- LSP records for every link in its level-3 network
- LSP records for every link in its level-4 network
- External records that summarize its cost to reach every router in the level-4 network (that will be flooded to other level 3 routers)
- Summary records for virtual links that connect it to every level-3 router in its area (that will be flooded to other level 4 routers)
- Summary records received from other level-4 routers for virtual links to their level-3 routers (that will be flooded to other level 3 routers).

These records allow a level 3 router to compute optimal paths not only to other level-4 routers, but also to level-3 routers that can be reached via other level-4 routers.

2.7.3 Interior and exterior protocols

In the Internet, we distinguish between three levels of routing (corresponding roughly to the three-level address hierarchy), where we allow each level to use a different routing protocol. The highest level is the Internet backbone, which interconnects multiple autonomous systems (ASs) (Figure B3T63). Routing between autonomous systems uses the exterior gateway protocol. The name reflects the history of the Internet, when a gateway connected university networks to the ARPAnet. The protocol that gateways spoke to each other therefore was the exterior gateway protocol. Symmetrically, the protocol that the gateway spoke to routers within a campus (and now, within an AS) is called the interior gateway protocol. At the lowest level, we have routing within a single broadcast LAN, such as Ethernet or FDDI, In this section, we will discuss the requirements for interior and exterior protocols, and problems with their interconnection.

Exterior protocols

Although all the routers within an AS are mutually cooperative, routers interconnecting two ASs may not necessarily trust each other. Exterior protocols determine routing between entities that can be owned by mutually suspicious domains. An important part of exterior protocols, therefore, is configuring *border gateways* (that is, gateways that mediate between interior and exterior routing) to recognize a set of valid neighbors and, valid paths. This is illustrated in Figure B3T63.

Example 2.9

In Figure B3T63, assume that border routers A and B belong to AT&T, and router D belongs to MCI. Say that the AB link goes down. A can still reach B through D, and a generic link-state routing protocol will easily find this path. However, the thought that internal AT&T packets traverse MCI's router may upset both MCI's and AT&T's managements! Therefore, the exterior protocol must allow A and B to state that if the A-B link goes down, the A-D-B path is unacceptable. Of course, for packets destined to D, the A-D link is perfectly valid, and, similarly, the D-B link may also be independently valid. It is only their combination, A-D-B, that is prohibited.

Accounting for administrative issues such as these complicates the design of exterior routing protocols, and these protocols often require manual configuration and intervention.

A related problem is that of *transit.* Suppose autonomous system A and autonomous system C set up a backdoor link between A.2 and C.1 for their own purposes. Since B knows from its interior routing protocol that C.1 is reachable through the backdoor link, it might advertise this to the rest of the Internet. This might cause A's facility to be used for packets destined for neither A nor C, which might annoy their administrators. Therefore, B should know that some links advertised by an interior protocol are special and should not be advertised (summarized) externally. This is another problem that usually requires manual intervention.

Exterior gateway protocols must be suspicious of routing updates. It should not be possible for malicious users to bring down the Internet by sending spurious routing messages to backbone gateways. Typically, every routing exchange is protected by a link password. Routing updates that fail the password check are rejected.

Interior protocols

Interior protocols are largely free of the administrative problems that exterior protocols face. just as autonomous systems hierarchically partition the Internet at the top level, interior routing protocols typically hierarchically partition each AS into *areas.* However, the same interior protocol routes packets both within and among areas. The issues in generating external and summary records, which we studied in Section 2.7.2, apply to routing among areas, in the same way as they do to routing between autonomous systems.

Issues in interconnecting exterior and interior routing protocols

The key problem in interconnecting exterior and interior protocols is that they may use different routing techniques and different ways to decide link costs. For example, the exterior protocol may advertise a 5-hop count to another AS. However, each of these hops may span a continent and cannot be compared with a 5-hop path in the interior of the AS. How is a router to decide which is the shortest path when routers use link costs that cannot be compared? The solution is to use the least common denominator, usually a hop-count metric, when computing routes outside the AS. This is not necessarily the optimal path, but at least it is a path that works!

A similar problem arises if the interior and exterior routing protocols use different routing schemes. For example, the exterior protocol may use path-vector routing, and the interior may use link-state routing. Thus, the border gateway must convert from an LSP database to a set of distance vectors that summarize paths to its interior. In the other direction, it must convert from distance-vector advertisements to external records for the interior routing protocol. Things are easier if both the interior and exterior routing protocols use the same basic routing scheme.

The bottom line is that interconnecting a given interior and exterior protocol requires a fair amount of manual intervention, and frequent monitoring to ensure that the network stays up. This is a direct consequence of the heterogeneity in the administration of the Internet, and of its decentralized control.

2.8 Common routing protocols

This section presents a highly abbreviated introduction to Internet routing protocols. Details on Internet routing can be found in references famous book wrote by Christian Huitema, Le routage dans l'Internet.

The Internet distinguishes between interior and exterior routing protocols because of the different demands that they pose on the routing system. Two protocols are commonly used as interior protocols. These are the Routing Information Protocol (RIP) and the Open Shortest Path First protocol (OSPF). The protocols commonly used for exterior routing are the Exterior Gateway Protocol (EGP) and the Border Gateway Protocol (BGP).

2.8.1 RIP

RIP, a distance-vector protocol, was the original routing protocol in the ARPAnet. It uses a hop-count metric, where infinity is defined to be 16. Peer routers exchange distance vectors every 30 s, and a router is declared dead if a peer does not hear from it for 180 s. The protocol uses split horizon with poisonous reverse to avoid the count-to-infinity problem. RIP is useful for small subnets where its simplicity of implementation and configuration more than compensates for its inadequacies in dealing, with link failures and providing multiple metrics.

2.8.2 OSPF

OSPF, a link-state protocol, is the preferred interior routing protocol on the Internet. It uses the notion of *areas* to route packets hierarchically within an AS. It also uses all the techniques for achieving LSP database consistency described in Section 2.5. Consequently, it is rather complex to describe and implement.

2.8.3 EGP

The original exterior protocol in the Internet was the distance-vector-based *Exterior Gateway Protocol* or EGP. EGP allows administrators to pick their neighbors in order to enforce inter-AS routing policies. To allow scaling, EGP allows address aggregation in routing tables.

EGP routers propagate distance vectors that reflect a combination of preferences and policies. For example, in the NSFnet backbone, a router advertises the distance to another AS as 128 if the AS is reachable, and 255 otherwise. This reduces EGP to a reachability protocol rather than a shortest-path protocol. Therefore, unless we structure the network backbone as a tree, EGP leads to routing loops! The reason for choosing 128 as the standard inter-AS distance is that it enables *backdoors.* Backdoors between autonomous systems are always given a cost smaller than 128, so that the two ASs sharing the backdoor will use it while keeping the backdoor invisible to outside systems. EGP is no longer widely used because of many deficiencies, particularly its need for loop-free topologies.

2.8.4 BGP

The preferred replacement for EGP is the Border Gateway Protocol, version 4*,* commonly referred to as BGP4. BGP4 is a path-vector protocol, where distance vectors are annotated not only with the entire path used to compute each distance, but also with certain policy attributes. An exterior gateway can usually compute much better paths with BGP than with EGP by examining these attributes. Since BGP uses true costs, unlike EGP, it can be used in non-tree topologies. Its use of a path-vector guarantees loopfreeness, at the expense of much larger routing tables. BGP routers use TCP to communicate with each other, instead of layering the routing message directly over IP, as is done in every other Internet routing protocol. This simplifies the error management in the routing protocol. However, routing updates are subject to TCP flow control, which can lead to fairly complicated and poorly understood network dynamics. For example, routing updates might be delayed waiting for TCP to time out. Thus, the choice of TCP is still controversial.

If an AS has more than one BGP-speaking border gateway, path vectors arriving at a gateway must somehow make their way to all the other gateways in the AS. Thus, BGP requires each gateway in an AS to talk to every other gateway in that AS (also called *internal peering).* BGP4 is hard to maintain because of the need to choose consistent path attributes from all the border routers, and to maintain clique connectivity among internal peers.

2.9 Routing within a broadcast LAN

Thus far we have looked at the routing problem for the network as a whole. In this section, we view the routing problem from the perspective of an endpoint-that is, how should the routing module at an endpoint decide where to forward a packet that it receives from an application?

An endpoint connected to a router by a point-to-point link (as in an ATM network) simply forwards every packet to that router. However, if the endpoint is part of a broadcast LAN, we can exploit the LAN's inherent routing capacity to reduce the load on routers. Specifically, the routing module must make four decisions:

- Is the packet meant for a destination on the same LAN?
- If so, what is the datalink-layer (MAC) address of the destination?
- If not, to which of the several routers on the LAN should the packet be sent?
- What is the router's MAC address?

Example 2.10

Consider host H1 shown in Figure B3T73 If it wants to send a packet to H2, it should determine that H2 is local, then figure out H2's MAC address. If it wants to send a packet to H3, it should find out that the next hop should be R1. If R1 goes down, it should send the packet to R2. Similarly, packets for H4 should go to R2, unless it is down, in which case it should be sent to R1.

These decisions typically require a combination of addressing conventions, explicit information, and exploiting the broadcast nature of the LAN. In the Internet, the first problem is solved by agreeing that all hosts that have the same network number must belong to the same broadcast LAN. (Although a single physical LAN may carry more than one IP subnet, hosts on different subnets on the same LAN communicate only through a router.) Thus, a host can determine whether the destination is local simply by using its subnet mask to extract the network number of the destination and comparing this with its own network number. For example, if host 135.104.53.100, with a subnet mask of 255.255.255.0, wants to send a packet to 135.104.53.12, it uses the subnet mask to determine that the destination's network number is 135.104.53. Since this matches its own network number, the destination must be on the local LAN.

The sending host must next determine the MAC address of the host to which it wants to send. It does so using the Address Resolution Protocol described in Section 1. It installs the MAC address in a local ARP cache and uses it for further transmission.

2.9.1 Router discovery

If a packet's destination address is nonlocal, then the host must send the packet to one of the routers on the LAN. A host can discover all the routers on the local LAN by means of router advertisement packets that each router periodically broadcasts on the LAN. A router advertisement has a preference level and a time-to-live. Hosts first check that the router corresponds to their own subnet by masking the router's IP address with their subnet mask and comparing with their subnet number. They then install a default route to the router with the highest preference. All nonlocal packets are sent to the default router, if necessary, resolving the router's MAC address with an ARP request.

A router advertisement is placed in a cache and flushed when its time-to-live expires. The time-to-live is typically around half an hour, and routers send advertisements about once every 10 minutes. The idea is that if a router dies, the host deletes old state information automatically. If a newly booted host does not want to wait several minutes for a router advertisement, it can force all routers to send an advertisement using a router solicitation packet.

If a default router goes down, then the host must somehow determine this and switch to an alternative router, if one exists. Otherwise, all packets from the host will be lost without trace (the *black hole* problem). The Internet protocol suite does not specify any single algorithm to cover black hole detection, though several heuristics were proposed in the literature. The general idea is that if a host does not hear anything from a router (such as a reply to an ARP request) for some time, it should assume that the router is down, and it can force routers to identify themselves with a router solicitation message. To prevent network load, hosts are required to send no more than three solicitation messages before they give up and assume that no router is available.

2.9.2 Redirection

With a default route, a host sends all nonlocal packets to only one of possibly many routers that share its subnet on the LAN. It may happen that, for a particular destination, it ought to use another router. To solve this problem, if a host's default router is not the right choice for a given destination, the default router sends a control message (using the *Internet Control Message Protocol* or ICMP) back to the host, informing it of a better choice. This *redirect* message is stored in the host's routing table for future use.

Example 2.11

In Figure B3T73, host HI may have selected R1 as its default router. It may then send a packet for H4 to R1. R1 can reach R4 either through the broadcast LAN and R2, or through R3. Assume, for the moment, that R1's next hop to R4 is through R2. When R1 gets a packet for H4, it can detect that R2 is a better routing choice for HI, because R1's next hop for H4 is R2, which has the same network address as R1 and Ell. It therefore sends an ICMP redirect message to H1, asking it to use R2 in the future, and hands the packet to R2 for transmission. In this way, hosts automatically discover the best path to remote destinations.

2.10 Summary

In this section, we studied many aspects of routing in the Internet in detail. The hard problem in routing is summarizing volatile and voluminous global state to something that a router can use in making local decisions. This problem exists both in the Internet and in the telephone and ATM networks. However, in the latter two networks, switch controllers route calls, instead of packets.

We would like a routing protocol to be robust, minimize its use of memory in routers, choose optimal paths, and require the least overhead. We have several choices in designing such a protocol, including centralized or distributed routing, source-based or hop-by-hop routing, single or multiple-path routing, and static or dynamic routing. Different combinations make different trade-offs in their complexity and use of resources.

The two fundamental ways to route packets in the Internet are to use distance-vector and link-state routing. Distance-vector routing is easy to implement, but suffers from problems such as counting to infinity. We can overcome these problems using techniques such as path-vector, source-tracing, and diffusion-update algorithms. Link-state routing allows each router to get its own copy of the global topology. We have to be careful in disseminating topology to avoid corruption of the individual copies of the topology. This is done with error detection techniques, as well as the lollipop sequence space and aging link-state packets to remove stale information. Once we have the topology, we can compute shortest paths using Dijkstra's algorithm.

Both link-state and distance-vector routing have their pros and cons, and neither seems uniformly superior. They are both common in the Internet.

Choosing the cost of a link is a fundamentally hard problem. The cost influences the shortest routes in the network, and these, in turn, affect the load on a link, and hence its cost (this is similar to the Erlang map in telephone networks). The network response map and the link metric map allow us to find cost metrics that guarantee convergence of routes and link costs.

In a large network, a router cannot store information about every other router. Instead, we divide the network into a hierarchy of levels, and each router knows only about other routers in its own level of the hierarchy. This reduces routing table sizes, though at the expense of suboptimal routing. Border routers participate in routing in more than one level, mediating exchange of information across levels to minimize the effects of hierarchical routing.

Many of the techniques used for point-to-point wide-area networks are not directly applicable to broadcast LANs, where broadcast and multicast are cheap. The Internet uses a set of special protocols in the local area to efficiently exploit these properties. These include router discovery and path redirection.

Routing is a rich field for study, and we have only touched on some essentials.

LECTURE 2
Interdomain Internet Routing

The goal of this lecture is to explain how routing between different administrative domains works in the Internet. We discuss how Internet Service Providers (ISPs) exchange routing information (and packets) between each other, and how the way in which they buy service from and sell service to each other and their customers influences the technical research agenda of Internet routing in the real-world. We discuss the salient features of the Border Gateway Protocol, Version 4 (BGP4), the current interdomain routing protocol in the Internet.

■ 2.1 Autonomous Systems

An abstract, highly idealized view of the Internet is shown in Figure 2-1, where end-hosts hook up to routers, which hook up with other routers to form a nice connected graph of essentially "peer" routers that cooperate nicely using routing protocols that exchange "shortest-path" or similar information and provide global connectivity. The same view posits that the graph induced by the routers and their links has a large amount of redundancy and the Internet's routing algorithms are designed to rapidly detect faults and problems in the routing substrate and route around them. In addition to routing around failures, clever routing protocols performing load-sensitive routing dynamically shed load away from congested paths on to less-loaded paths.

Unfortunately, while simple, this abstraction is actually quite misleading as far as the wide-area Internet is concerned. The real story of the Internet routing infrastructure is that the Internet service is provided by a large number of commercial enterprises, generally in competition with each other. Cooperation, required for global connectivity, is generally at odds with the need to be a profitable commercial enterprise, which often occurs at the expense of one's competitors—the same people with whom one needs to cooperate. How this "competitive cooperation" is achieved in practice (although there's lots of room for improvement), and how we might improve things, provides an interesting study of how good technical research can be shaped and challenged by commercial realities.

A second pass at developing a good picture of the Internet routing substrate is shown in Figure 2-2, which depicts a group of Internet Service Providers (ISPs) somehow cooper-

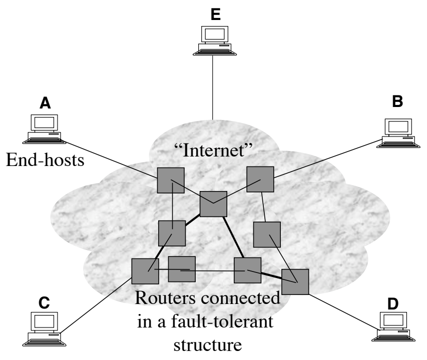


Figure 2-1: This is a rather misleading abstraction of the Internet routing layer.

ating to provide global connectivity to end-customers. This picture is closer to the truth, but the main thing it's missing is that not all ISPs are created equal. Some are bigger and more "connected" than others, and still others have global reachability in their routing tables. There are names given to these "small," "large," and "really huge" ISPs: *Tier-3 ISPs* are ones that have a small number of usually localized (in geography) end-customers; *Tier-2 ISPs* generally have regional scope (e.g., state-wide, region-wide, or non-US country-wide), while *Tier-1 ISPs*, of which there are a handful, have global scope in the sense that their routing tables actually have routes to all currently reachable Internet prefixes (*i.e.,* they have no default routes). Figure 2-3 shows this organization.

The current wide-area routing protocol, which exchanges *reachability information* about routeable IP-address prefixes between routers at the boundary between ISPs, is *BGP* (Border Gateway Protocol, Version 4) [10, 11]. More precisely, the wide-area routing architecture is divided into *autonomous systems* (ASes) that exchange reachability information. An AS is owned and administered by a single commercial entity, and implements some set of policies in deciding how to route its packets to the rest of the Internet, and how to export its routes (its own, those of its customers, and other routes it may have learned from other ASes) to other ASes. Each AS is identified by a unique 16-bit number.

A different routing protocol operates within each AS. These routing protocols are called *Interior Gateway Protocols* (IGPs), and include protocols like Routing Information Protocol (RIP) [7]. Open Shortest Paths First (OSPF) [8], Intermediate System-Intermediate System (IS-IS) [9], and E-IGRP. In contrast, interdomain protocols like BGP are also called EGPs (Exterior Gateway Protocols). Operationally, a key difference between EGPs like BGP and IGPs is that the former is concerned with providing *reachability information* and facilitating *routing policy* implementation in a *scalable* manner, whereas the latter are typically concerned with optimizing a path metric. In general, IGPs don't scale as well as BGP does.

The rest of this lecture is in two parts: first, we will look at inter-AS relationships (transit and peering); then, we will study some salient features of BGP. We don't have time to

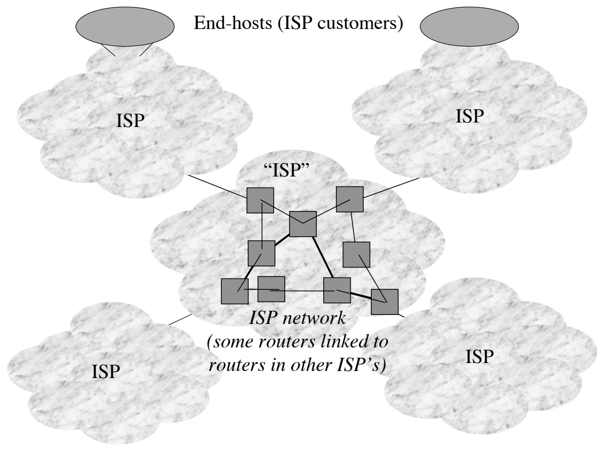


Figure 2-2: The Internet is actually composed of many competing Internet Service Providers (ISPs) that co-operate to provide global connectivity. This picture suggests that all ISPs are "equal," which isn't actually true.

survey IGPs in this lecture, but you should be familiar with the more well-known ones like RIP and OSPF (or at least with distance-vector and link-state protocols). To learn more about IGPs if you're not familiar with them, read a standard networking textbook (e.g., Peterson & Davie or Kurose & Ross).

■ 2.2 Inter-AS Relationships: Transit and Peering

The Internet is composed of many different types of ASes, from universities to corporations to regional Internet Service Providers (ISPs) to nationwide ISPs. Smaller ASes (*e.g.*, universities, corporations, etc.) typically purchase Internet connectivity from ISPs. Smaller regional ISPs, in turn, purchase connectivity from larger ISPs with "backbone" networks.

Consider the picture shown in Figure 2-4. It shows an ISP, *X*, directly connected to a *provider* (from whom it buys Internet service) and a few *customers* (to whom it qsells Internet service). In addition, the figure shows two other ISPs to whom it is directly connected, with whom *X* exchanges routing information via BGP.

The different types of ASes lead to different types of business relationships between them, which in turn translate to different policies for exchanging and selecting routes. There are two prevalent forms of AS-AS interconnection. The first form is *provider-customer transit* (aka "transit"), wherein one ISP (the "provider" *P* in Figure 2-4) provides access to all (or most) destinations in its routing tables. Transit almost always is meaningful in an inter-AS relationship where financial settlement is involved; the provider charges its customers for Internet access, in return for forwarding packets on behalf of customers to destinations (and in the opposite direction in many cases). Another example of a transit relationship in Figure 2-4 is between *X* and its customers (the C_is).

The second prevalent form of inter-AS interconnection is called *peering*. Here, two ASes (typically ISPs) provide mutual access to a subset of each other's routing tables. The subset

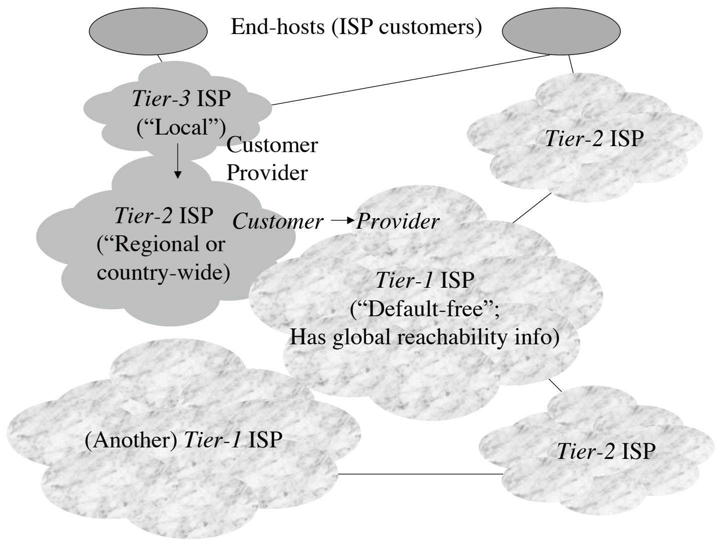

End-hosts (ISP customers)
Tier-3 ISP ("Local")
Customer
Provider
Tier-2 ISP ("Regional or country-wide)
Customer → Provider
Tier-2 ISP
Tier-1 ISP ("Default-free"; Has global reachability info)
(Another) Tier-1 ISP
Tier-2 ISP

Figure 2-3: A more accurate picture of the wide-area Internet routing infrastructure, with various types of ISPs defined by their respective reach. *Tier-1* ISPs have "default-free" routing tables (i.e., they don't have any default routes), and typically have global reachability information. There are a handful of these today (about five or so).

of interest here is their own transit customers (and the ISPs own internal addresses). Like transit, peering is a business deal, but it may not involve financial settlement. While paid peering is common in some parts of the world, in many cases they are reciprocal agreements. As long as the traffic ratio between the concerned ASs is not highly asymmetric (e.g., 4:1 is a commonly believed and quoted ratio), there's usually no financial settlement. Peering deals are almost always under non-disclosure and are confidential.

■ 2.2.1 Peering v. Transit

A key point to note about peering relationships is that they are often between business competitors. The common reason for peering is the observation by each party that a non-trivial fraction of the packets emanating from each one is destined for the other's direct transit customers. Of course, the best thing for each of the ISPs to try to do would be to wean away the other's customers, but that may be hard to do. The next best thing, which would be in their mutual interest, would be to avoid paying transit costs to *their* respective providers, but instead set up a transit-free link between each other to forward packets for their direct customers. In addition, this approach has the advantage that this more direct path would lead to better end-to-end performance (in terms of latency, packet loss rate, and throughput) for their customers. It's also worth noticing that a Tier-1 ISP usually will find it essential to be involved in peering relationships with other ISPs (especially other Tier-1 ISPs) to obtain global routing information in a default-free manner.

Balancing these potential benefits are some forces against peering. Transit relationships generate revenue; peering relationships usually don't. Peering relationships typically need to be renegotiated often, and asymmetric traffic ratios require care to handle in a way that's

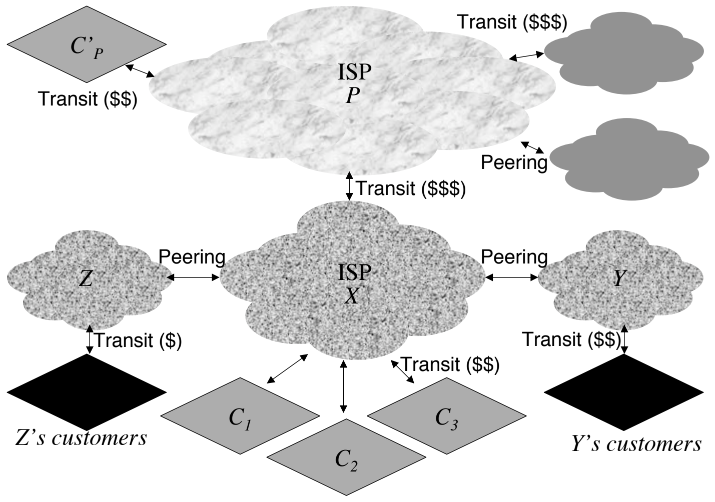


Figure 2-4: Inter-AS relationships; transit and peering.

mutually satisfactory. Above all, these relationships are often between competitors vying for the same customer base.

In the discussion so far, we have implicitly used an important property of current interdomain routing: *A route advertisement from B to A for a destination prefix is an agreement by B that it will forward packets sent via A destined for any destination in the prefix.* This (implicit) agreement implies that one way to think about Internet economics is to view ISPs as charging customers for entries in their routing tables. Of course, the data rate of the interconnection is also crucial, and is the major determinant of an ISP's pricing policy.

■ 2.2.2 Exporting Routes: Route Filtering

Each AS (ISP) needs to make decisions on which routes to export to its neighboring ISPs using BGP. The reason why export policies are important is that no ISP wants to act as transit for packets that it isn't somehow making money on. Because packets flow in the opposite direction to the (best) route advertisement for any destination, an AS should advertise routes to neighbors with care.

Transit customer routes. To an ISP, its customer routes are likely the most important, because the view it provides to its customers is the sense that *all* potential senders in the Internet can reach them. It is in the ISP's best interest to advertise routes to its transit customers to as many other connected ASes as possible. The more traffic that an ISP carries on behalf of a customer, the "fatter" the pipe that the customer would need, implying higher revenue for the ISP. Hence, if a destination were advertised from multiple neighbors, an ISP should prefer the advertisement made from a customer over all other choices (in particular, over peers and transit providers).

Transit provider routes. Does an ISP want to provide *transit* to the routes exported by its provider to it? Most likely not, because the ISP isn't making any money on providing such transit facilities. An example of this situation is shown in Figure 2-4, where C'_P is a customer of P, and P has exported a route to C'_P to X. It isn't in X's interest to advertise this route to everyone, e.g., to other ISPs with whom X has a peering relationship. An important exception to this, of course, is X's transit customers who are paying X for service—the service X provides its customers C_i's is that they can reach any location on the Internet via X, so it makes sense for X to export as many routes to X as possible.

Peer routes. It usually makes sense for an ISP to export only selected routes from its routing tables to other peering ISPs. It obviously makes sense to export routes to all of ones transit customers. It also makes sense to export routes to addresses within an ISP. However, it does not make sense to export an ISP's transit provider routes to other peering ISPs, because that may cause a peering ISP to use the advertising ISP to reach a destination advertised by a transit provider. Doing so would expend ISP resources but not lead to revenue.

The same situation applies to routes learned from other peering relationships. Consider ISP Z in Figure 2-4, with its own transit customers. It doesn't make sense for X to advertise routes to Z's customers to another peering ISP (Y), because X doesn't make any money on Y using X to get packets to Z's customers!

These arguments show that most ISPs end up providing *selective transit*: typically, full transit capabilities for their own transit customers in both directions, some transit (between mutual customers) in a peering relationship, and transit only for one's transit customers (and ISP-internal addresses) to one's providers.

The discussion so far may make it sound like BGP is the only way in which to exchange reachability information between an ISP and its customers or between two ASes. That is not true—a large fraction of end-customers (typically customers who don't provide large amounts of further transit and/or aren't ISPs) do not run BGP sessions with their providers. The reason is that BGP is complicated to configure, administer, and manage, and isn't particularly useful if the set of addresses in the customer is relatively invariant. These customers interact with their providers via *static routes*. These routes are usually manually configured. Of course, information about customer address blocks will in general be exchanged by a provider using BGP to other ASes (ISPs) to achieve global reachability to the customer premises.

■ 2.2.3 Importing Routes

The previous section described the issues considered by an AS (specifically, routers in an AS involved in BGP sessions with routers in other ASes) while deciding which routes to export. In a similar manner, when a router hears many possible routes to a destination network, it needs to decide which route to install in its forwarding tables.

Deciding which routes to import is a fairly involved process in BGP and requires a consideration of several attributes of the advertised routes. At this stage, we consider only one of the many things that a router needs to consider, but it's the most important consideration, viz., who advertised the route? Typically, when a router (e.g., X in Figure 2-4) hears advertisements to its transit customers from other ASes (e.g., because the customer is multi-homed), it needs to ensure that packets to the customer do not traverse additional ASes unnecessarily. This requirement usually means that customer routes are prioritized over routes to the same network advertised by providers or peers. Second, peer routes are likely more preferable to provider routes, because the purpose of peering was to exchange reachability information about mutual transit customers. These two observations imply that typically routes are imported in the following priority order:

$$\textit{customer} > \textit{peer} > \textit{provider}$$

This rule (and many others like it) can be implemented in BGP using a special attribute that's locally maintained by routers in an AS, called the LOCAL PREF attribute. The first rule in route selection with BGP is to pick a route based on this attribute. It is only if this attribute is *not* set for a route, are other attributes of a route even considered. Note, however, that in practice most routes in most ASes are not selected using the LOCAL PREF attribute; other attributes like the length of the AS path tend to be quite common. We discuss these other route attributes and the details of the BGP route selection process, also called the *decision process*, in the next section.

■ 2.3 BGP

We now turn to how reachability information is exchanged using BGP, and how routing policies like the ones explained in the previous section can be expressed and enforced. We start with a discussion of the main design goals in BGP and summarize the protocol. Most of the complexity in wide-area routing is not in the protocol, but in how BGP routers are configured to implement policy, and in how routes learned from other ASes are disseminated within an AS. The rest of the section discusses these issues.

■ 2.3.1 Design Goals

In the old NSFNET, the backbone routers exchanged routing information over a tree topology, using a routing protocol called EGP. (While the modern use of the term EGP is as a family of exterior gateway protocols, its use in the context of NSFNET refers to the specific one used in that network.) Because the backbone routing information was exchanged over a tree, the routing protocol was relatively simple. The evolution of the Internet from a singly administered backbone to its current commercial structure made the NSFNET EGP obsolete and required a more sophisticated protocol.

The design of BGP4 was motivated by three important needs:

1. **Scalability.** The division of the Internet into ASes under independent administration was done while the backbone of the then Internet was under the administration of the NSFNet. An important requirement for BGP4 was to ensure that the Internet routing infrastructure remained scalable as the number of connected networks increased, accommodating complex inter-AS topologies.

2. **Policy.** The ability for each AS to implement and enforce various forms of routing policy was an important design goal. One of the consequences of this was the development of the BGP attribute structure for route announcements, and allowing route filtering.

3. **Cooperation under competitive circumstances.** BGP was designed in large part to handle the transition from the NSFNet to a situation where the "backbone" Inter-

net infrastructure would no longer be run by a single administrative entity. This structure implies that the routing protocol should allow ASes to make purely local decisions on how to route packets, from among any set of choices.

Note that routing security—i.e., ensuring the authenticity and integrity of messages—wasn't a requirement. Efforts to secure BGP, notably S-BGP [?], generally involve external registries and infrastructure to maintain mappings between prefixes and ASes, as well as the public keys for ASes. These approaches have not been deployed in the Internet, despite the occurrence of malicious activity and outage-causing misconfigurations from time to time.

■ 2.3.2 The Protocol

As protocols go, BGP is not an overly complicated protocol (as we'll see later, what makes its operation complicated is the variety and complexity of BGP router configurations, which network operators use to specify how they want routing messages handled). The basic operation of BGP—the protocol state machine, the format of routing messages, and the propagation of routing updates—are all defined in the protocol standard (RFC 1771) [10]. BGP runs over TCP on a well-known port (179). To start participating in a *BGP session* with another router, a router sends an OPEN message after establishing a TCP connection to it on the BGP port. After the OPEN is completed, both routers exchange their tables of all active routes (of course, applying all applicable route filtering rules). This process may take several minutes to complete, especially on sessions that have a large number of active routes.

After this initialization, there are two main types of messages on the BGP session. First, BGP routers send route UPDATE messages sent on the session. These updates only send any routing entries that have changed since the last update (or transmission of all active routes). There are two kinds of updates: *announcements*, which are changes to existing routes or new routes, and *withdrawals*, which are messages that inform the receiver that the named routes no longer exist. A withdrawal usually happens when some previously announced route can no longer be used (e.g., because of a failure or a change in policy). Because BGP uses TCP, which provides reliable and in-order delivery, routes do not need to be periodically announced, unless they change.

But, in the absence of periodic routing updates, how does a router know whether the neighbor at the other end of a session is still functioning properly? One possible solution might be for BGP to run over a transport protocol that implements its own "is the peer alive" message protocol. Such messages are also called "keepalive" messages. TCP, however, does not implement a transport-layer "keepalive" (with good reason), so BGP uses its own. Each BGP session has a configurable keepalive timer, and the router guarantees that it will attempt to send at least one BGP message during that time. If there are no UPDATE messages, then the router sends the second type of message on the session: KEEPALIVE messages. The absence of a certain number BGP KEEPALIVE messages on a session causes the router to terminate that session. The number of missing messages depends on a configurable times called the *hold timer*; the specification recommends that the hold timer be at least as long as the keepalive timer duration negotiated on the session.

More details about the BGP state machine may be found in [2, 10].

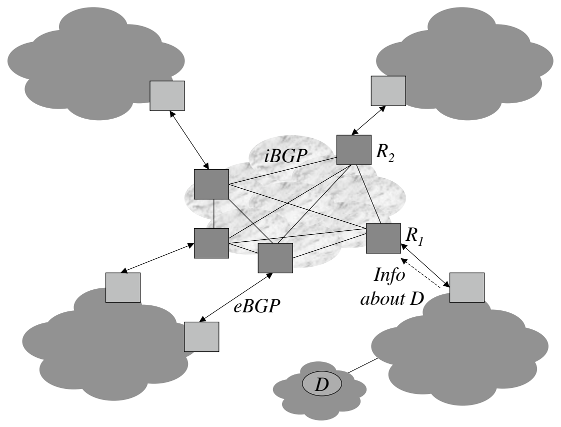


Figure 2-5: eBGP and iBGP.

Unlike many IGP's, BGP does not simply optimize any metrics like shortest-paths or delays. Because its goals are to provide reachability information and enable routing policies, its announcements do not simply announce some metric like hop-count. Rather, they have the following format:

$$IP\ prefix : Attributes$$

where for each announced IP prefix (in the "A/m" format), one or more attributes are also announced. There are a number of standardized attributes in BGP, and we'll look at some of them in more detail here.

Recall that we already talked about one BGP attribute, LOCAL PREF. This attribute isn't disseminated with route announcements, but is an important attribute used locally while selecting a route for a destination. When a route is advertised from a neighboring AS, the receiving BGP router consults its configuration and may set a LOCAL PREF for this route.

■ 2.3.3 Disseminating Routes within an AS: eBGP and iBGP

There are two types of BGP sessions: *eBGP* sessions are between BGP-speaking routers in different ASes, while *iBGP* sessions are between BGP routers in the same AS. They serve different purposes, but use exactly the same protocol.

eBGP is the "standard" mode in which BGP is used; after all BGP was designed to exchange network routing information between different ASes in the Internet. eBGP sessions are shown in Figure 2-5, where the BGP routers implement route filtering rules and exchange a subset of their routes with routers in other ASes.

In general, each AS will have more than one router that participates in eBGP sessions with neighboring ASes. During this process, each router will obtain information about some subset of all the prefixes that the entire AS knows about. Each such eBGP router must disseminate routes to the external prefix to all the other routers in the AS. This dissemination must be done with care to meet two important goals:

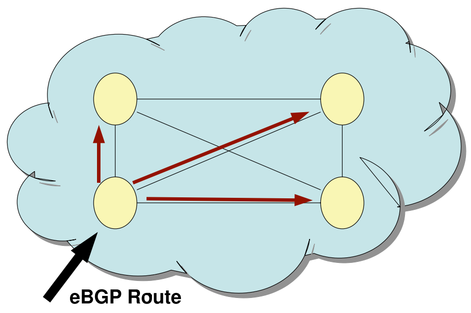


Figure 2-6: Small ASes establish a "full mesh" of iBGP sessions. Each circle represents a router within an AS. Only eBGP-learned routes are re-advertised over iBGP sessions.

1. *Loop-free forwarding.* After the dissemination of eBGP learned routes, the resulting routes (and the subsequent forwarding paths of packets sent along those routes) picked by all routers should be free of deflections and forwarding loops [3, 6].

2. *Complete visibility.* One of the goals of BGP is to allow each AS to be treated as a single monolithic entity. This means that the several eBGP-speaking routes in the AS must exchange external route information so that they have a complete view of all external routes. For instance, consider Figure 2-5, and prefix D. Router R_2 needs to know how to forward packets destined for D, but R_2 hasn't heard a direct announcement on any of its eBGP sessions for D.[1] By "complete visibility", we mean the following: *for every external destination, each router picks the same route that it would have picked had it seen the best routes from each eBGP router in the AS.*

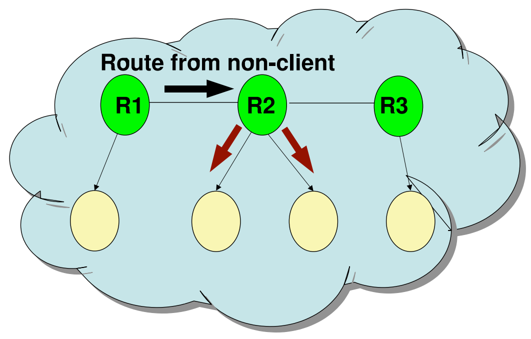
   The dissemination of externally learned routes to routers inside an AS is done over *internal BGP* (iBGP) sessions running in each AS.

An important question concerns the topology over which iBGP sessions should be run. One possibility is to use an arbitrary connected graph and "flood" updates of external routes to all BGP routers in an AS. Of cours, an approach based on flooding would require additional techniques to avoid routing loops. The original BGP specification solved this problem by simply setting up a *full mesh* of iBGP sessions (see Figure 2-6, where every eBGP router maintains an iBGP session with every other BGP router in the AS. Flooding updates is now straightforward; an eBGP router simply sends UPDATE messages to its iBGP neighbors. An iBGP router does not have to send any UPDATE messages because it does not have any eBGP sessions with a router in another AS.

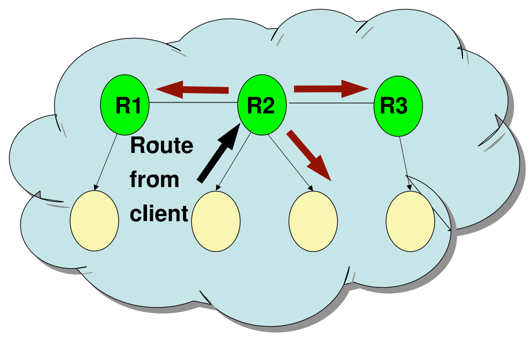
It is important to note that *iBGP is not an IGP* like RIP or OSPF, and it cannot be used to set up routing state that allows packets to be forwarded correctly between internal nodes in an AS. Rather, iBGP sessions, running over TCP, provide a way by which routers inside an AS can use BGP to exchange information about external routes. In fact, iBGP sessions and messages are themselves routed between the BGP routers in the AS via whatever IGP is being used in the AS!

One might wonder why iBGP is needed, and why one can't simply use whatever IGP

[1]It turns out that each router inside doesn't know about all the external routes to a destination. Rather, we will strive for each router being able to discover the best routes of the egress routers in the AS for a destination.



(a) Routes learned from non-clients are re-advertised to clients only.



(b) Routes learned from clients are re-advertised over all iBGP sessions.

Figure 2-7: Larger ASes commonly use route reflectors, which advertise some iBGP-learned routes, as described above. Directed edges between routers represent iBGP sessions from route reflectors to clients (*e.g.*, router $R2$ is a route reflector with two clients). As in Figure 2-6, all routers re-advertise eBGP-learned routes over all iBGP sessions.

is being used in the AS to also send BGP updates. There are several reasons why introducing eBGP routes into an IGP is inconvenient. The first reason is that most IGPs don't scale as well as BGP does, and often rely on periodic routing announcements rather than incremental updates (*i.e.*, their state machines are different). Second, IGPs usually don't implement the rich set of attributes present in BGP. To preserve all the information about routes gleaned from eBGP sessions, it is best to run BGP sessions inside an AS as well.

The requirement that the iBGP routers be connected via a complete mesh limits scalability: a network with e eBGP routers and i other interior routers requires $e(e-1)/2 + ei$ iBGP sessions in a full-mesh configuration. While this quadratic scaling is not a problem for a small AS with onlly a handful of routers, large backbone networks typically have more several hundred routers, requiring tens of thousands of iBGP sessions. This quadratic scaling does not work well in those cases.

As a result, two methods to handle this have arisen, both based on manual configuration into some kind of hierarchy. The first method is to use *route reflectors* [1], while the second sets up *confederations* of BGP routers [12]. We briefly summarize the main ideas in route reflection in this lecture, and refer the interested reader to RFC 3065 [12] for a discussion of BGP confederations.

A route reflector is a BGP router that can be configured to have *client* BGP routers. A route reflector selects a single best route to each destination prefix and announces that route to all of its clients. An AS with a route reflector configuration follows the following rules in its route updates:

1. If a route reflector learns a route via eBGP or via iBGP from one of its clients, the it re-advertises that route over all of its sessions to its clients.

2. If a route reflector learns a route via iBGP from a router that is not one of its clients, then it re-advertises the route to its client routers, *but not over any other iBGP sessions*.

Having only one route reflector in an AS causes a different scaling problem, because it may have to support a large number of client sessions. More importantly, if there are multiple egress links from the AS to a destination prefix, a single route-reflector configuration may not use them all well, because all the clients would inherit the single choice made by the route reflector. To solve this problem, many networks deploy multiple route reflectors, organizing them hierarchically. Figure 2-7 shows an example route reflector hierarchy and how routes propagate from various iBGP sessions.

BGP route updates propagate differently depending on whether the update is propagating over an eBGP session or an iBGP session. An eBGP session is typically a *point-to-point* session: that is, the IP addresses of the routers on either end of the session are directly connected with one another and are typically on the same local area network. There are some exceptions to this practice (*i.e.*, "multi-hop eBGP" [4]), but directly connected eBGP sessions is normal operating procedure. In the case where an eBGP session is point-to-point, the next-hop attribute for the BGP route is guaranteed to be reachable, as is the other end of the point-to-point connection. A router will advertise a route over an eBGP session regardless of whether that route was originally learned via eBGP or iBGP.

On the other hand, an iBGP session may exist between two routers that are *not* directly connected, and it may be the case that the next-hop IP address for a route learned via iBGP is more than one IP-level hop away. In fact, as the next-hop IP address of the route is typically one of the border routers for the AS, this next hop may not even correspond to the router on the other end of the iBGP session, but may be several *iBGP* hops away. In iBGP, the routers thus rely on the AS's internal routing protocol (*i.e.*, its IGP) to both (1) establish connectivity between the two endpoints of the BGP session and (2) establish the route to the next-hop IP address named in the route attribute.

Configuring an iBGP topology to correctly achieve loop-free forwarding and complete visibility is non-trivial. Incorrect iBGP topology configuration can create many types of incorrect behavior, including persistent forwarding loops and oscillations [6]. Route reflection causes problems with correctness because not all route reflector topologies satisfy visibility (see [5] and references therein).

■ 2.3.4 BGP Policy Expression: Filters and Rankings

BGP allows policy expression by allowing network operators to configure routers to *manipulate* route attributes when disseminating routes. Network operators can configure routers to perform the following policy-driven tasks:

1. Control how a router ranks candidate routes and select paths to destinations.

2. Control the "next hop" IP address for the advertised route to balance load.

3. "Tag" a route to control how the ranking and filtering functions on other routers treat the route.

We're now in a position to understand what the anatomy of a BGP route looks like and how route announcements (and withdrawals) allow a router to compute a forwarding table from all the routing information. This forwarding table typically has one chosen path in the form of the egress interface (port) on the router, corresponding to the next neighboring IP address, to send a packet destined for a prefix. Recall that each router implements the longest prefix match on each packet's destination IP address.

Route Attribute	Description
Next Hop	IP Address of the next-hop router along the path to the destination. On eBGP sessions, the next hop is set to the IP address of the border router. On iBGP sessions, the next hop is not modified.
AS path	Sequence of AS identifiers that the route advertisement has traversed.
Local Preference	This attribute is the first criteria used to select routes. It is not attached on routes learned via eBGP sessions, but typically assigned by the import policy of these sessions; preserved on iBGP sessions.
Multiple-Exit Discriminator (MED)	Used for comparing two or more routes from the same neighboring AS. That neighboring AS can set the MED values to indicate which router it prefers to receive traffic for that destination. *By default, not comparable among routes from different ASes.*

Table 2-1: Important BGP route attributes.

■ 2.3.5 Exchanging Reachability: NEXT HOP Attribute

A BGP route announcement has a set of attributes associated with each announced prefix. One of them is the NEXT HOP attribute, which gives the IP address of the router to send the packet to. As the announcement propagates across an AS boundary, the NEXT HOP field is changed; typically, it gets changed to the IP address of the border router of the AS the announcement came from.

The above behavior is for eBGP speakers. For iBGP speakers, the first router that introduces the route into iBGP sets the NEXT HOP attribute to its so-called loopback address (the address that all other routers within the AS can use to reach the first router). All the other iBGP routers within the AS *preserve* this setting, and use the ASes IGP to route any packets destined for the route (in the reverse direction of the announcement) toward the NEXT HOP IP address. In general, packets destined for a prefix flow in the opposite direction to the route announcements for the prefix.

Length of AS Paths: ASPATH Attribute

Another attribute that changes as a route annoucement traverses different ASes is the ASPATH attribute, which is a *vector* that lists all the ASes (in reverse order) that this route announcement has been through. Upon crossing an AS boundary, the first router prepends the unique identifier of its own AS and propagates the announcement on (subject to its route filtering rules). This use of a "path vector"—a list of ASes per route—is the reason BGP is classified as a *path vector protocol*.

A path vector serves two purposes. The first is *loop avoidance*. Upon crossing an AS

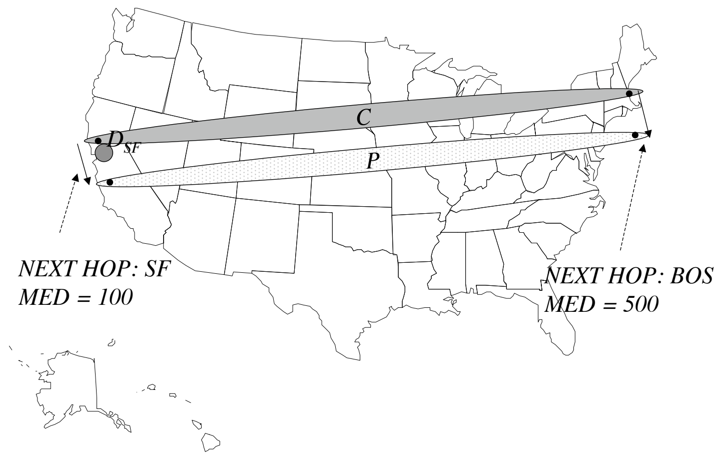


Figure 2-8: MED's **are useful in many situations, e.g., if C is a transit customer of P, to ensure that cross-country packets to C traverse P's (rather than C's wide-area network). However, if C and P are in a peering relationship,** MED **may (and often will) be ignored. In this example, the** MED **for D_SF is set to 100 at the SF exchange point, and 500 in Boston, so P can do the right thing if it wants to.**

boundary, the router checks to see if its own AS identifier is already in the vector. If it is, then it discards the route announcement, since importing this route would simply cause a routing loop when packets are forwarded.

The second purpose of the path vector is to help pick a suitable path from among multiple choices. If no LOCAL PREF is present for a route, then the ASPATH length is used to decide on the route. Shorter ASPATH lengths are preferred to longer ones. However, it is important to remember that BGP isn't a strict shortest-ASPATH protocol (classical path vector protocols would pick shortest vectors), since it pays attention to routing policies. The LOCAL PREF attribute is always given priority over ASPATH. Many routes in practice, though, end up being picked according to shortest-ASPATH.

Choosing Between Multiple Exit Points: MED Attribute

There are many situations when two ASes are linked at multiple locations, and one of them may prefer a particular transit point over another. This situation can't be distinguished using LOCAL PREF (which decides which AS' announcement to import) or shortest ASPATH (since they would be equal). A BGP attribute called MED, for *multi-exit discriminator* is used for this.

It's best to understand MED using an example. Consider Figure 2-8 which shows a provider-customer relationship where both the provider P and customer C have national footprints. Cross-country bandwidth is a much more expensive resource than local bandwidth, and the customer would like the provider to incur the cost of cross-country transit for the customer's packets. Suppose we want to route packets from the east coast (Boston) destined for D_{SF} to traverse P's network and not C's. We want to prevent P from transiting the packet to C in Boston, which would force C to use its own resources and defeat the purpose of having P as its Internet provider.

A MED attribute allows an AS, in this case C, to tell another (P) how to choose between multiple NEXT HOP's for a prefix D_{SF}. Each router will pick the smallest MED from among multiple choices coming from the same neighbor AS. No semantics are associated with how MED values are picked, but they must obviously be picked and announced consistently amongst the eBGP routers in an AS. In our example, a MED of 100 for the SF NEXT HOP for prefix D_{SF} and a MED of 500 for the BOS NEXT HOP for the same prefix accomplishes the desired goal.

An important point to realize about MED's is that they are usually ignored in AS-AS relationships that don't have some form of financial settlement (or explicit arrangement, in the absence of money). In particular, most peering arrangements ignore MED. This leads to a substantial amount of *asymmetric routes* in the wide-area Internet, as we'll see in the next lecture. For instance, if P and C were in a peering relationship in Figure 2-8, cross-country packets going from C to P would traverse P's wide-area network, while cross-country packets from P to C would traverse C's wide-area network. Both P and C would be in a hurry to get rid of the packet from their own network, a form of routing sometimes called *hot-potato routing*. In contrast, a financial arrangement would provide an incentive to honor MED's and allow "cold-potato routing" to be enforced.

The case of large content hosts peering with tier-1 ISPs is an excellent real-world example of cold-potato routing. For instance, an ISP might peer with a content-hosting provider to obtain direct access to the latter's customers (popular content-hosting sites), but does not want the hosting provider to free-load on its backbone. This can be achieved by insisting that its MEDs be honored.[2]

Putting It All Together

So far, we have seen the most important BGP attributes: LOCAL PREF, ASPATH, and MED. We are now in a position to discuss the set of rules that BGP routers in an AS use to select a route from among multiple choices.

These rules are shown in Table 2-2, in priority order. These rules are actually slightly vendor-specific; for instance, the Router ID tie-break is not the default on Cisco routers, which select the "oldest" route in the hope that this route would be the most "stable."

■ 2.4 Failover and Scalability

BGP allows multiple links (and eBGP sessions) between two ASes, and this may used to provide some degree of fault tolerance and load balance. Overall, however, BGP wasn't designed for rapid fault detection and recovery, so these mechanisms are generally not particularly useful over short time scales. Furthermore, upon the detection of a fault, a router sends a withdrawal message to its neighbors. To avoid massive route oscillations, the further propagation of such route announcements is *damped*. Damping causes some delay (configurable using a timer) before problems can be detected and recovery initiated, and is a useful mechanism for scalability.

[2]I thank Nick Feamster for educating me about this operational use of MEDs.

Priority	Rule	Remarks
1	LOCAL PREF	Highest LOCAL PREF (§2.2.3). E.g., Prefer transit customer routes over peer and provider routes.
2	ASPATH	Shortest ASPATH length (§2.3.5) *Not* shortest number of Internet hops or delay.
3	MED	Lowest MED preferred (§2.3.5). May be ignored, esp. if no financial incentive involved.
4	eBGP > iBGP	Did AS learn route via eBGP (preferred) or iBGP?
5	IGP path	Lowest IGP path cost to next hop (egress router). If all else equal so far, pick shortest internal path.
6	Router ID	Smallest router ID (IP address). A random (but unchanging) choice; some implementations use a different tie-break such as the oldest route.

Table 2-2: How a BGP-speaking router selects routes. There used to be another step between steps 2 and 3 in this table, but it's not included in this table because it is now obsolete.

With BGP, faults may take minutes to detect and it may take several minutes for routes to converge to a consistent state afterwards.

■ 2.4.1 Multi-homing: Promise and Problems

Multi-homing typically refers to a technique by which a customer can exchange routes and packets over multiple distinct provider ASes. An example is shown in Figure 2-9, which shows the topology and address blocks of the concerned parties. This example uses *provider-based addressing* for the customer, which allows the routing state in the Internet backbones to scale better because transit providers can aggregate address blocks across several customers into one or a small number of route announcements to their respective providers.

Today, multi-homing doesn't actually work while still preserving the scalability of the routing infrastructure. Figure 2-9 shows why. Here the customer (C) address block 10.0.0.0/16 needs to be advertised not only from provider P_2 to the rest of the Internet, but *also* from provider P_1. If P_1 didn't do so, then longest prefix matching would cause all packets to the customer to arrive via P_2's link, which would defeat the purpose of using P_2 only as a backup path.

Now, given that this route needs to be advertised on both paths, how does C ensure that both paths aren't used? One hack to achieve this is by *padding* the exported ASPATH attribute. On the path through P_1, the normal ASPATH is announced, while on the path through P_2, a longer path is advertised by padding it with C's AS number multiple times.

A good way to do extensive multi-homing without affecting routing scalability is a good open problem. In addition to the fact that customer routes must be advertised along multiple paths, effective multi-homing today is often not possible unless the customer has a large address block. To limit the size of their routing tables, many ISPs will not ac-

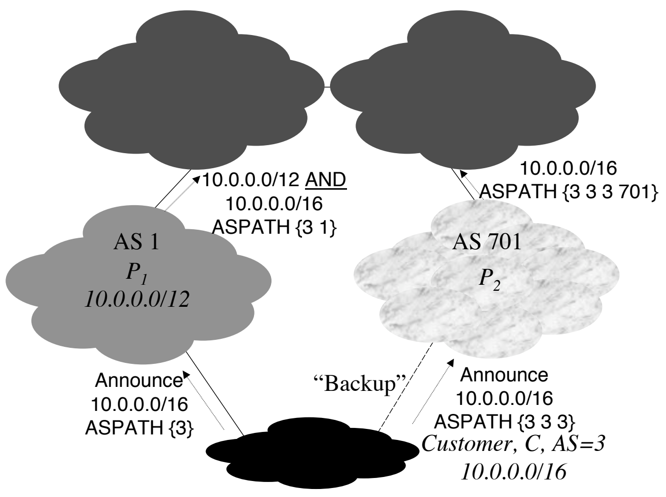


Figure 2-9: Customer C is multi-homed with providers P_1 and P_2 and uses provider-based addressing from P_1. C announces routes to itself on both P_1 and P_2, but to ensure that P_2 is only a backup, it uses a hack that pads the ASPATH attribute. However, notice that P_1 must announce (to its providers and peers) *explicit* routes on both its regular address block *and* on the customer block, for otherwise the path through P_2 would match based on longest prefix in the upstream ASes!

cept routing announcements for fewer than 8192 contiguous addresses (a "/19" netblock).[3] Small companies, regardless of their fault-tolerance needs, do not often require such a large address block, and cannot effectively multi-home. Notice that provider-based addressing doesn't really work, since this requires handling two distinct sets of addresses on its hosts. It is unclear how *on-going* connections (e.g., long-running ssh tunnels, which are becoming increasingly common) on one address set can seamlessly switch on a failure in this model.

■ 2.4.2 Convergence Problems

BGP does not always converge quickly after a fault is detected and routes withdrawn. Depending on the eBGP session topology between ASes, this could involve the investigation of many routes before route convergence occurs. The paper by Labovitz *et al.* from ACM SIGCOMM 2000 explains this in detail, and shows that under some conditions this could take a super-exponential number of steps.

In practice, it's been observed that wide-area routes are often (relative to what's needed for "mission-critical" applications) unavailable. Although extensive data is lacking, the observations summarized in Table 2-3 are worth noting.[4]

[3]This used to be the case, although it seems as if they have relaxed this to /24 in many cases now.

[4]These are, unfortunately, a few years old. I need to update the table to include newer results. Nick Feamster's dissertation has newer failure data.

Researchers	Finding	Time-frame
Paxson	Serious routing pathology rate of 3.3%	1995
Labovitz *et al.*	10% of routes available less than 95% of the time	1997
Labovitz *et al.*	Less than 35% of routes available 99.99% of the time	1997
Labovitz *et al.*	40% of path outages take 30+ minutes to repair	2000
Chandra *et al.*	5% of faults last more than 2 hours, 45 minutes	2001
Andersen *et al.*	Between 0.23% and 7.7% of overlay "path-hours" experienced serious 30-minute problems in 16-node overlay	2001

Table 2-3: Internet path failure observations, as reported by several studies.

■ 2.5 Summary

This lecture looked at issues in wide-area unicast Internet routing, focusing on real-world issues. We first looked at inter-AS relationships and dealt with transit and peering issues. We then discussed many salient features and quirks of BGP, the prevalent wide-area routing protocol today.

BGP is actually a rather simple protocol, but its operation in practice is extremely complex. Its complexity stems from configuration flexibility, which allows for a rich set of attributes to be exchanged in route announcements. There are a number of open and interesting research problems in the area of wide-area routing, relating to policy, failover, scalability, configuration, and correctness. Despite much activity and impressive progress over the past few years, interdomain routing remains hard to understand and model.

■ Acknowledgments

This lecture has evolved over the past few years, thanks in large part to the wonderful collaboration that Nick Feamster and I have had. A few subsections and two figures of these notes are edited from Nick's dissertation. Thanks also to Jennifer Rexford and Mythili Vutukuru for several discussions on interdomain routing.

References

[1] T. Bates, R. Chandra, and E. Chen. *BGP Route Reflection - An Alternative to Full Mesh IBGP*. Internet Engineering Task Force, Apr. 2000. RFC 2796. (Cited on page 35.)

[2] I. V. Beijnum. *BGP*. O'Reilly and Associates, Sept. 2002. (Cited on page 32.)

[3] R. Dube. A Comparison of Scaling Techniques for BGP. *ACM Computer Communications Review*, 29(3):44–46, July 1999. (Cited on page 34.)

[4] Cisco IOS IP Command Reference, ebgp-multihop. `http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_command_reference_chapter09186a00800ca79d.html`, 2005. (Cited on page 36.)

[5] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 43–56, Boston, MA, May 2005. (Cited on page 36.)

[6] T. Griffin and G. Wilfong. On the Correctness of IBGP Configuration. In *Proc. ACM SIGCOMM*, pages 17–29, Pittsburgh, PA, Aug. 2002. (Cited on pages 34 and 36.)

[7] C. Hedrick. *Routing Information Protocol*. Internet Engineering Task Force, June 1988. RFC 1058. (Cited on page 26.)

[8] J. Moy. *OSPF Version 2*, Mar. 1994. RFC 1583. (Cited on page 26.)

[9] D. Oran. *OSI IS-IS intra-domain routing protocol*. Internet Engineering Task Force, Feb. 1990. RFC 1142. (Cited on page 26.)

[10] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Mar. 1995. RFC 1771. (Cited on pages 26 and 32.)

[11] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Oct. 2004. `http://www.ietf.org/internet-drafts/draft-ietf-idr-bgp4-26.txt` Work in progress, expired April 2005. (Cited on page 26.)

[12] P. Traina, D. McPherson, and J. Scudder. *Autonomous System Confederations for BGP*. Internet Engineering Task Force, Feb. 2001. RFC 3065. (Cited on page 35.)

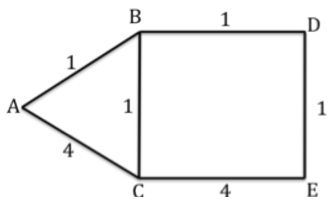
Evolving Internet I

Exercices on Routing

1) Distance Vector Routing

Consider the network in the figure below. It uses the distance vector technique between routers A, B, C, D and E.

1. We consider first that we use the basic technique without any improvement. At startup, nodes send their distance vector (DV) in the following order: at t = 0, for A, at t = 5sec for B, at t = 10sec for C, at t = 15sec for D and at t = 20sec for E. Then the same cycle is repeated every 30 seconds. What are the initial distance vectors (i.e. before starting) sent by all the nodes, and what are their final DVs (i.e. after the convergence).



2. Now consider that the technique of split horizon with poison reverse is used for all the rest of this exercise. Why is this technique used? After convergence of system: what is the DV sent by D to B? And the DV sent by E to C?

3. At time t = 102 sec, the BA link fails (and will not be restored). Describe what happens until convergence. Is there going to be counting up to infinity?

4. At time t = 287 sec, the link AC also falls down. Describe what's happening until convergence. (Nodes keep the same cycle of transmission of DV except that A disappears).

5. We consider that the option "triggered updates" is supported too. What will happen then there after the failure of link AC at t = 287 sec? Explain the origin of the problem. Will the DUAL version solve this problem? And if yes how?

2) BGP Routing

In BGP, border routers exchange route announcements that contain a prefix and an AS path. The semantics of a route announced by a router to its neighbor is then: I agree to relay traffic coming from your to the network identified by the prefix attached and along the path indicated by the indicated AS path. As we saw in the course, there are three steps to handle route advertisements: first, the route import policy which determines the routes to exclude, the decision process which consists in ordering routes and choosing the best route that will be adopted and the route export policy which consists to choose which neighbors to advertise the adopted route to. To achieve a routing policy, BGP routers can control any of these three steps: Change the import policy to exclude routes that they do not want to use, modify some attributes to select certain routes instead of others and decide not to export routes to some neighbors.

1. What does the term provider-customer transit link between two autonomous systems mean? And the term peering link?

2. Explain how the BGP route import and selection rules (Customer > peer > provider) and the BGP route export rules (explained in the course) allow taking into account commercial relations between customers and providers.

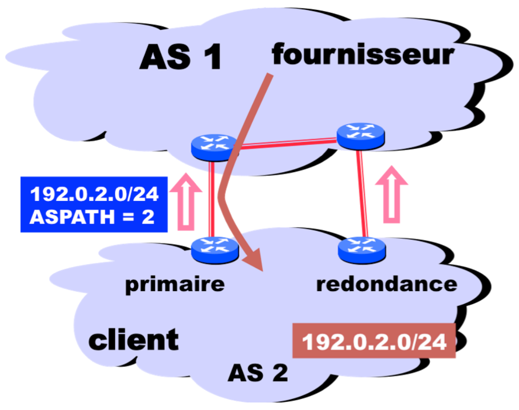


Figure (a)

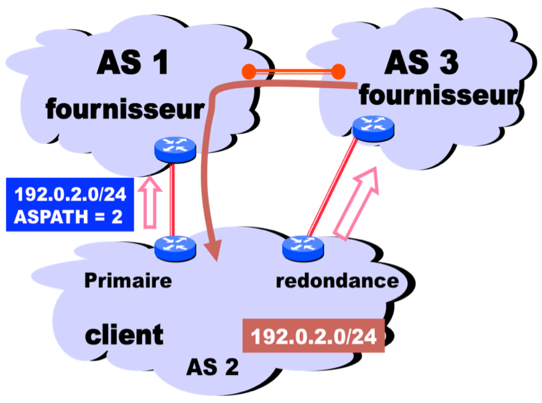


Figure (b)

3. See the network of figure (a) above. The agreement between the customer and the provider states that traffic to the customer's network should pass through the primary link except in the case of failure. In this case, the back-up link can be be used. How to reflect this in the route announcement done by the customer's router connected to the back-up link? (Consider using one of three attributes LOCAL_PREF, AS_PATH or MED).

4. Now consider the network of figure (b) above. Providers AS1 and AS3 have a peering agreement. The client AS2 announces its prefix to both providers. To reflect the fact that the link to AS3 should be used as a back-up, the client chooses to adopt the same solution as in 3 (Figure (a) with a single provider). Does this solution work in this case? (Consider the order of priority for route import and selection rules).

5. What is the attribute MED (Multi Exit Discriminator) used for?

3) Interdomain Routing

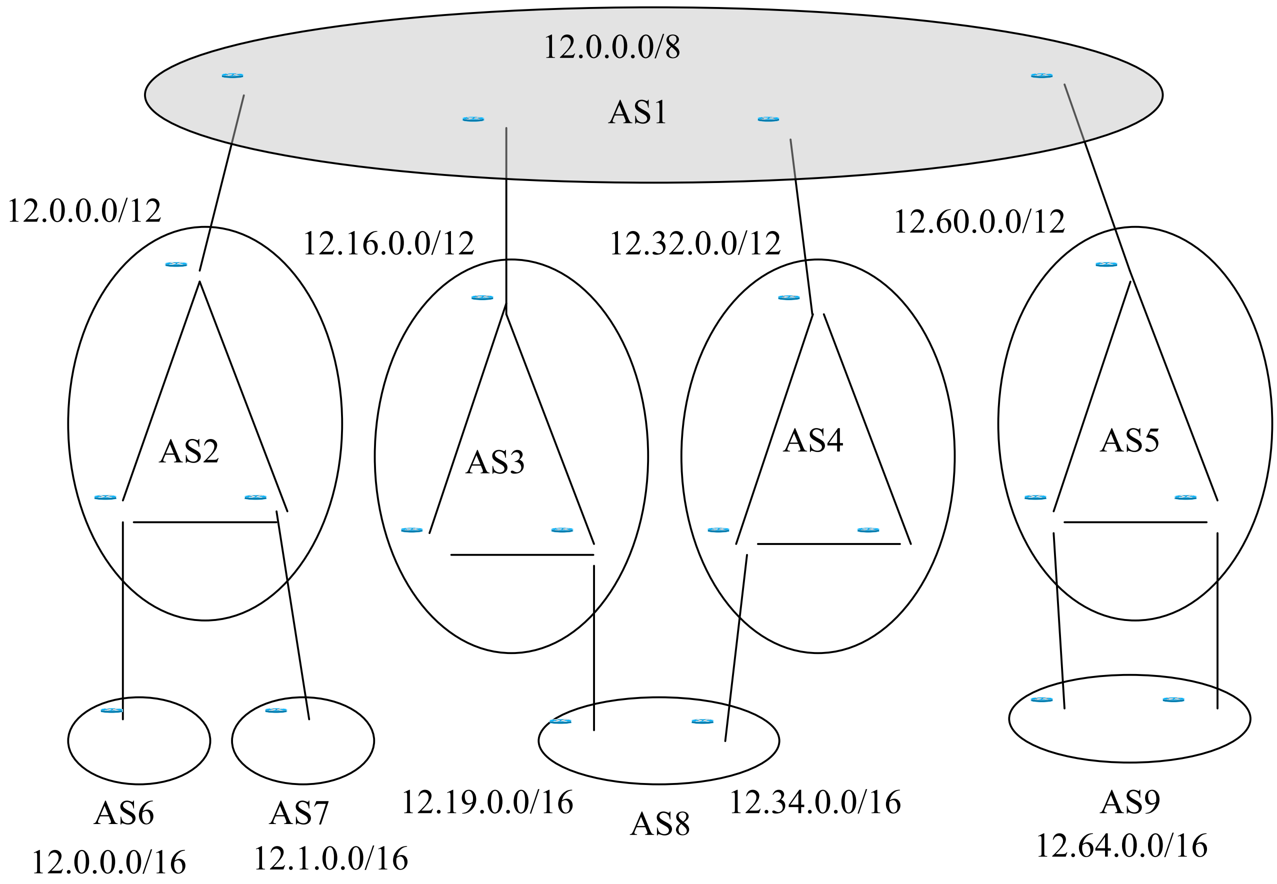
We consider the networks in the figure below. AS1 is a Internet Service Provider with international connectivity (Tier 1). AS2, AS3, AS4 and AS5 are national service Internet Service Providers. AS6, AS7, AS8 and AS9 are regional Internet Service Providers. The prefixes held by different ISPs are shown in the figure. AS1 routers are all connected to each other (these connections are not shown in the figure). We assume that the routing tables of the various routers within an AS are identical. BGP is the protocol used (path vector protocol). CIDR addressing with the longest prefix match for routing. The AS8 is "multihomed": it is connected to two suppliers and has two different prefixes. Routing is not the same for all customers of AS8, it depends on the source address.

a) Valley free routing is applied. Give the routing tables of all AS.

b) AS2 and AS3 establish between themselves a relationship of "peering" in order to avoid to use their common provider networks for their mutual internal clients traffic. Indicate how this change does the routing tables of AS.

c) The AS7 change provider and becomes customer ofAS3. How does this impact the routing tables?

d) Ten thousand users of the BitTorrent protocol are uniformly distributed in the various AS Access (AS6, AS7, AS8 and AS9) regardless of the size of the AS. Knowing that every BitTorrent client open four connections with other clients and customers sharing 10 MB on each connection, calculate the total amount of data exchanged on the links between the service providers AS national and international AS. Hint : pay attention to the routing.



4) CIDR

Consider three Tier 1 providers P, Q and R. P owns the prefix C1.0.0.0/8, Q owns the C2.0.0.0/8 and R owns C3.0.0.0/8. P has two clients Tier 2 clients PA and PB. PA owns the C1.A3.0.0/16 prefix and PB owns the C1.B0.0.0/12 prefix. Q has two Tier 2 clients QA and QB. QA owns the C2.0A.10.0/20 prefix and QB owns the C2.0B.0.0/16 prefix. There are no other providers or clients in the network.

a) All Tier 1 providers are connected in a full mesh. Give the most efficient routing tables.
b) P is connected to Q and Q is connected to R, but P is not directly connected to R. Give P and R routing tables.
c) Consider now that PA becomes multihomed (buying a direct connection through Q) and that QA does the same with P (in addition to the existing links as in b) above). Give P and Q routing tables (ignoring R).