UCLA
DEPARTMENT OF COMPUTER SCIENCE

CSC 118

Computer Networking, IP Addressing and Neighbor Routing

In these we quickly give an overview of IP addressing and Neighbor Routing.

Routing consists of:

- IP addressing
- IP Neighbor Routing
- Route Computation
- Forwarding

These notes are on IP Addressing and Neighbor Routing. The next set of notes will deal with route computation.

1 Connectionless Routing

The Internet Protocol (IP) does what we call *connectionless* routing. By contrast, the telephone network (and some data networks, namely virtual circuit networks like ATM) do *connection oriented* routing in that a call or connection is first set up, and then all data is sent along the path set up by the call. By contrast, in connectionless routing, no call is set up: each packet is on its own, and finds a path through the network. In practice, all packets in a given TCP connection almost always follow the same path; however, it is possible after a failure that some packets in the TCP connection be rerouted after the failure.

Before we study how routing is done, we must realize that routing is done to addresses. If you point your browser to `cnn.com`, the Domain name `cnn.com` is first mapped using the Domain Name Service to a 32-bit IP address and all packets are sent to this address. Thus routing in the Internet is always to 32-bit destination addresses. If IP addresses were allocated to computers without any structure, then core routers would have to keep track of billions of IP addresses with constant churn as addresses kept coming up and going down.

Instead, clever use of hierarchy allows core routers to keep track of only hundreds of thousands of so-called *prefixes*. The hierarchy used in IP is not a simple fixed hierarchy as in the telephone network (area codes, country codes) but a more subtle variable length hierarchy of arbitrary length prefixes that allows much more efficient allocation of the address space. While hierarchy is a common idea in computer science, the idea of *flexible* hierarchies is more interesting.

Once we define IP addresses and their structure, we can look at the next problem of computing routes. Even here, we separate out the problem of finding out how to route to one's direct neighbors. While this is trivial when neighbors are connected by a point-to-point link, it is slightly more interesting when the neighbors are connected by an Ethernet. We study this in the second part of these notes.

2 The structure of Addresses

The most important thing about addresses is hierarchy. If addresses are hierarchical, then the routing protocol can be hierarchical. We see this in the Post Office where most San Diego post offices only know how to route mail to San Diego, a few area post offices know how to route mail to a central post office in other cities in the U.S. Hierarchy breaks up a complex problem (routing to all post offices in the world) into bite size pieces.

Am immediate question is how many levels of hierarchy? How many bytes in each level address. Unlike telephones, we want addressing structure that fits a number of different networks and needs. Thus we need *flexible hierarchies.* This is illustrated in Figure 1. IP offers some flexibility. The OSI Routing protocol, which we will not study (but is described in Perlman's textbook), offers even more.

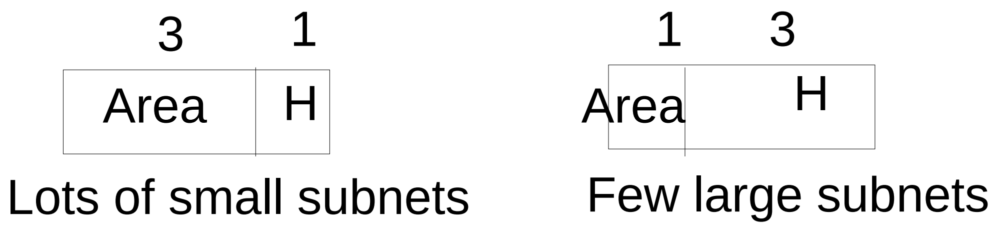

Figure 1: The need for flexible hierarchies: its hard to know whether to divide into lots of small subnets or a few large subnets.

2.1 IP Addresses: from Classful to Classless Addresses

How did the Internet Protocol evolve to having variable length prefixes as a form of flexible hierarchy? The answer comes from the history of Internet addressing. The Internet began with a simple hierarchy in which 32 bit addresses were divided into a network address and a host number; routers only stored entries for networks. Initially, the network number was only 1 byte and the host address was 3 bytes for a total of a 4 byte address (32 bits). The Internet addressing scheme evolved in response to 3 challenges:

- *Challenge 1, Ethernets*: The early addressing model assumed a few large networks, each of which had a lot of hosts. When Ethernets were introduced and each Ethernet was possibly a separate network, this had to be rethought. The Internet designers realized that the few 8 bit network numbers they had assigned started with 0. Thus, they added a hack to allow more flexibility. If the first few bits of an address started with 0, it was supposed to be a Class A address with a network number of 8 bits (this allowed compatibility with existing networks).

  If the first 2 bits started with 10, it was supposed to be a Class B address with a network number of 16 bits; finally if the first 3 bits started with 110, it was supposed to be a Class C address with a network number of 24 bits. Class C addresses in particular were introduced for organizations with small networks; they allowed a large number of such networks with 256 hosts each. Class B allowed a reasonable number of medium size networks. Depending on its needs, an organization could be allocated a Class A, Class B, or Class C with Class A addresses reserved for very large networks. This a limited amount of flexibility was allowed.

Router forwarding tables were still easy to implement. A core router only had to have 3 sets of hash tables, one for Class A, one for Class B, and one for Class C. When a packet arrived, the destination IP address was parsed (based on its first 3 bits) to see if it was Class A, B, or C. Depending on which it was, the network number (first 8, 16, or 24 bits of Destination IP address) was extracted and looked up in the appropriate table to see what the next hop, or next router was. Finally, if it was the final hop, the packet was sent by what IP called network specific forwarding; for example, for many networks like Ethernet, this is done using so-called ARP (described later).

- *Challenge 2, Class B Address Depletion:* Any network with more than 255 addresses needs a class B address. In the Classful Addressing scheme, one address network wastes 254 addresses, 256 address network wastes 64,000 addresses! Thus since most organizations wanted a Class B address to handle growth, the Class B addresses began to run out The naive solution to this was to assigning lots of Class C network numbers to every new organization. For example, UC Merced, as a new UC, may be assigned 32 Class C addresses, if it needs 32 * 256 = 8K IP addresses. Unfortunately, this will require Core Routers to keep track of 32 entries for UC Merced. This will greatly increase core router forwarding table size and router control traffic.

  To cope with exhaustion of Class B addresses, the CIDR scheme (Classless Internet Domain Routing) assigns new organizations multiple *contiguous* Class C addresses that can be aggregated by a common prefix. This reduces core router table size.

  This is shown at the bottom of Figure 2. Thus UC Merced is assigned 32 consecutive Class C addresses (/24's) from $Y00000$ to $Y11111$. By using supernetting or CIDR, UC Merced can be represented by their common 19 bit prefix Y for core routers. The top of Figure 2 shows that supernetting can work in reverse. An organization with a /16 can decide to use *subnetting* to divide the /16 into several /8's (for instance), for example, one for each department such as Computer Science and Arts and Science. Thus the core routers in the organization only need to keep track of the department prefixes instead of all the IP addresses in the organization.[1]

- *Depletion of the Entire Address Space:* Today, the potential depletion of the address space has led to Internet registries being very conservative in the assignment of IP addresses. A small organization may be given only a small portion of a Class C address, perhaps a /30, which only allows 4 IP addresses within the organization. Many organizations are coping with these sparse assignments by sharing a few IP addresses among multiple computers using schemes such as Network Address Translation or NAT.

  NAT works by using TCP port numbers (destination port is 16 bits, source port is 16 bits) in every packet to effectively extend the IP address space from 32 bits to 32 + 32 = 64 bits. For example, consider an organization with 16 clients sitting besides a NAT box. The NAT box is given 1 IP address for the entire organization. The first idea is that the organization uses *private IP addresses* locally. A portion of the IP address is reserved for so-called private addresses. These are addresses that are not globally unique but each organization can reuse

[1]The picture contains an error, the prefix X shown in the picture is not 24 bits. What should it be?

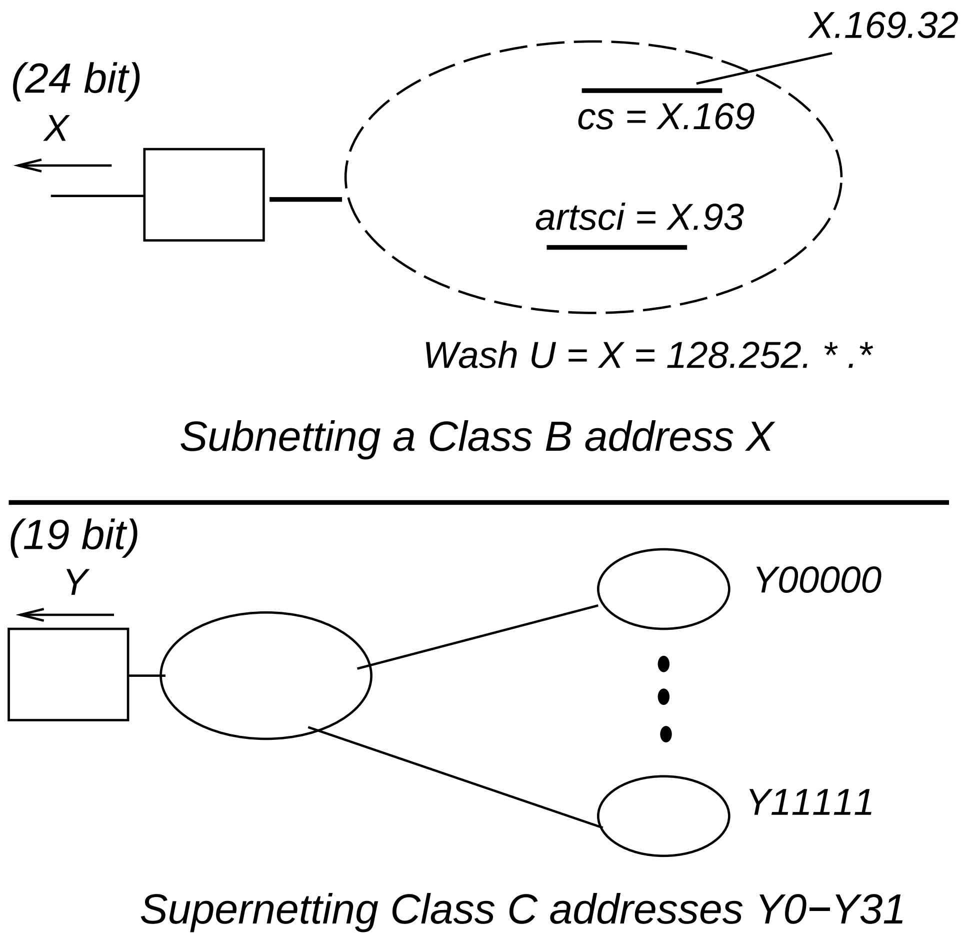
X.169.32
(24 bit)
X
cs = X.169
artsci = X.93
Wash U = X = 128.252. * .*
Subnetting a Class B address X
(19 bit)
Y
Y00000
Y11111
Supernetting Class C addresses Y0−Y31

Figure 2: Subnetting (top) and Supernetting (bottom)

locally. Thus Organization A can have addresses P1 through P8, and Organization B could also use addresses P1 through P8. This works fine as long as Organization A stations communicate locally; but it does not work well if Organization A wants to talk to other parties on the Internet such as a web server. How can the web server tell if a packet with address P1 is coming from Organization A or B? This is where the NAT box comes in.

The NAT box for Organization A is given a single global address G which is globally unique. Thus when a packet from Host 1 in the organization wishes to connect to the outside world, the NAT box sends it on its way but after translating the source IP address from P1 to G. That works fine until the reply comes back. If a packet from Host 2 (with address P2) to the same server has also been sent out, when a reply comes back (it will be to destination G, since the remote server does not know about all this NAT trickery) how can the NAT box tell that its for Host 1 or 2? This is where the port numbers are used.

If the communication of Host 1 and Host 2 used different port numbers in either the destination or source port, the NAT box can remember this when the packet is sent. When the reply comes back, the NAT box can distinguish from the port numbers in the reply whether is for Host 1 or 2. There is a small chance that two hosts will talk to the same server using exactly the same destination ports (say port 80) and the same source port (unlikely). In that case, the NAT server has a final ace up its sleeve. When forwarding the packet on, it not only translates the destination IP address, but it also translates the source port to pick any currently unused port. By doing so, it can ensure that any set of concurrent conversations via the NAT box can be told apart when the replies come back.

In conclusion, CIDR and NAT have helped the Internet handle exponential growth with a finite 32-bit address space. While there are plans for a new IP protocol (IPv6) with a 128 bit address, the effectiveness of NAT in the short run and the complexity of rolling out a new protocol, has made IPv6 deployment currently slow. Despite this, a brave new world containing billions of wireless sensors may lead to an IPv6 resurgence.

The bottom line is that the decision to deploy CIDR helped save the Internet, but has introduced the complexity of longest matching prefix lookup.

2.2 Prefix Notation

We have seen that CIDR (Supernetting) and Subnetting allow more careful allocation of the address space at the cost of the elimination of simple classes (A, B, and C). What is left is a set of prefixes of various lengths mostly from 8 to 32 that are kept by core routers. Internet core routers currently store around 200,000 such prefixes.

Internet prefixes are defined using bits and not alphanumerical characters, of up to 32 bits in length. To confuse matters, however, IP prefixes are often written in dot-decimal notation. Thus, the 16 bit prefix for UCSD at the time of writing is 132.239. Each of the decimal digits between dots represents a byte. Since 132 is 100000100 in binary and 239 is 11101111 in binary, the UCSD prefix in binary can also be written as 1000010011101111*, where the wildcard character * is used to denote that the remaining bits do not matter. All UCSD hosts have 32-bit IP addresses beginning with these 16 bits.

Because prefixes can be variable length, a second common way to denote a prefix is by slash notation of the form A/L. In this case A denotes a 32 bit IP address in dot-decimal notation and L denotes the length of the prefix. Thus the UCSD prefix can also be denoted as 132.239.0.0/16, where the length 16 indicates that only the first 16 bits (i.e., 132.239) are relevant. A third common way to describe prefixes is to use a mask in place of an explicit prefix length. Thus the UCSD prefix can also be described as 128.239.0.0 with a mask of 255.255.0.0. Since 255.255.0.0 has 1's in the first 16 bits, this implicitly indicates a length of 16 bits.[2]

Of these three ways to denote a prefix (binary with a wildcard at the end, slash notation and mask notation), the last two are more compact for writing down large prefixes. However, for pedagogic reasons, it is much easier to use small prefixes as examples and to write them in binary.

[2]The mask notation is actually more general because it allows non-contiguous masks where the 1's are not necessarily consecutive starting from the left. Such definitions of networks actually do exist. However, they are becoming increasingly uncommon, and are non-existent in core router prefix tables. Thus we will ignore this possibility.

3 Neighbor Routing

Before we study neighbor routing, we first make a simple distinction between endnodes and routers.

Endnodes are hosts, user workstations. Large numbers (e.g., 150,000), dedicated to computing. Thus endnode routing protocol should be simple. *Routers* (IMPs, Gateways). Smaller numbers (e.g., 200). Dedicated to communication. Thus it makes sense for them to bear major load of routing calculation. Thus makes sense to have endnode routing, forwarding, and neighbor discovery to be as simple as possible.

Neighbor routing on a point-to-point link is very easy. If an endnode is attached to a router, the endnode simply sends all its data packets to the router. It can be configured (or learn from Hellos) the routers Data Link and IP address. A hello protocol can be used between the two nodes (whether router to router, or router to endnode) on a point-to-point link to see if the link is up and to exchange Data Link and IP addresses.

On the other hand, neighbor routing on a LAN is a slightly more complicated beast. This is because in a LAN there are choices as to who to send a packet to. Also, a simple hello protocol between every endnode (there could be 1000's of endnode) and every router would be unscalable.

3.1 Neighbor Routing on a LAN

There are six problems for neighbor routing on a LAN as shown in Figure 3

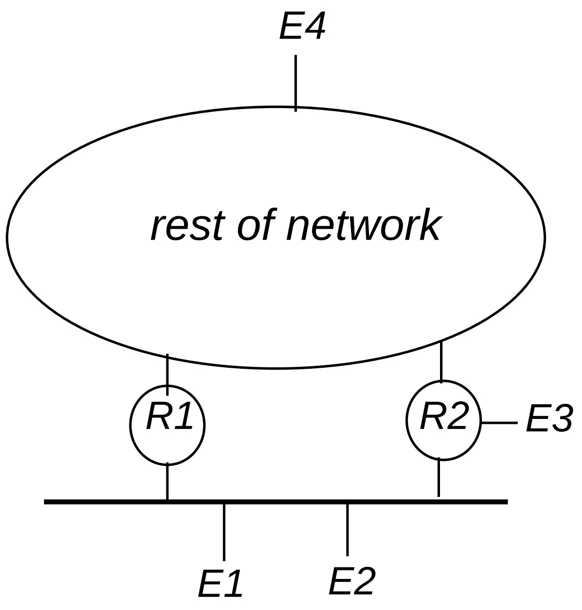

Figure 3: An example topology to demonstrate the six problems that Neighbor Routing must solve

- Routers need network addresses of endnodes. Clearly, when a router gets a packet for an endnode say E1 on its LAN, it needs to know that the endnode is on the LAN so that the routing process can terminate and the final packet be launched on the LAN.

- Routers need Data Link addresses of endnodes. Unfortunately, even when a router gets a packet and it knows that endnode E1 is on its LAN, it needs to know E1's MAC Address. Recall that every node on a LAN has a 48-bit LAN address and this is in no way related to its IP address. One cannot send a packet to a station on a LAN and have that station pick up the packet without sending to the station's MAC address.

- Endnodes need Data Link address of at least one router. Endnodes need to be simple and so it suffices for them to know the IP and Data Link address of at least one router so that they can send all packets to be routed to that router.

- E1 to E2 traffic should not have to go through a router. If E1 sends to E2 via any of the routers, then this is wasteful. Every packet between them is sent twice on the LAN doubling traffic on the LAN. If most traffic is local, this is a poor use of LAN bandwidth. Ideally, this traffic should eventually or always go directly.

- E1 to E3 traffic should go through R2 eventually. In the case of E1 to E3, let us assume that E1 is configured with R1's address. Then E1 will send a packet to E3 first to R1 who then sends the packet back on the LAN to R2. This is again sub-optimal because traffic froM E1 to E3 is being sent twice on the LAN. Ideally, traffic from E1 to E3 should eventually go through E3. Note that the issue is that we don't want a poor endnode like E1 to know the ideal router to use to get to every destination in the Internet. If it did, E1 would have to keep a massive table to get optimal routing. We want a more scalable solution.

- If R1, R2 are down, E1 and E2 should be able to communicate.

The IP solution to these problems are as follows. Perlman's book describes the OSI Routing solution to the same problems.

- Routers need network addresses of endnodes. This is easy in IP because of the convention that all endnodes on the same attached LAN use the same prefix. The router is configured with the prefix of every interface. Thus by checking whether the prefix of an endnode matches the prefix associated with an interface, the router knows that the endnode (if it is up) is on the interface.

- Routers need Data Link addresses of endnodes. Unfortunately, even when a router gets a packet and it knows that endnode E1 is on its LAN, it needs to know E1's MAC Address. IP takes the position that a hello protocol between 1000's of endnodes and several routers is unscalable. Thus IP prefers a more demand-driven strategy. When an IP nodes wants the MAC address of another node (whether router or endnode), it simply broadcasts a so-called ARP request to the entire LAN! An ARP request is a broadcast packet sent to the all 1's MAC address which makes sure that every station on the LAN picks it up. ARP stands for Address Resolution Protocol. Suppose R1 wants E1's MAC address.

Then R1 essentially sends an ARP request with its own MAC address say r as the Data Link header source, and with the all 1's address in the Data Link layer destination address. Inside the packet (at the routing layer) is a field that says this is an ARP packet and the 32-bit IP address of E1. Essentially, an ARP request is a way of asking the entire LAN "If you are E1, please reply with your MAC address". All stations pick up the broadcast on the LAN. All stations besides E1 discard the packet (a bit wasteful, this).

However, E1 replies with a so-called ARP response. E1 puts in the MAC source address sent in the ARP request (R1's MAC address) as the *destination address* and places its own MAC address in the MAC source address. The ARP response contains E1's IP address in the routing header. Thus when R1 gets the ARP it can put two and two together and realize that the MAC address corresponding to the IP Address E1 is in the source MAC address of the ARP response. Since ARP is such a high overhead process, R1 will *cache* (i.e., remember in a table for some specified time period, often a few hours) the mapping between E1's routing and MAC address. All subsequent packets from R1 to E1 will not need the ARP protocol until the ARP cache times out by simply consulting the cache.

- Endnodes need Data Link address of at least one router. IP endnodes are either statically configured (the old way) or are provided with a router address by an auto-configuring protocol called DHCP. DHCP works using so-called DHCP servers on the LAN; DHCP servers can be found using some common multicast addresses. With DHCP, a new endnode can get both its prefix and its router from the DHCP server.

- E1 to E2 traffic should not have to go through a router. In IP, E1 does not send traffic to E2 via any of the routers. Instead, E1 checks whether E2 matches the same prefix which it has been configured with (for example, if it uses a mask it simply uses the mask to AND both addresses E1 and E2, and notices that the masked value is the same) before sending to a router. If they have the same prefix, E1 assumes E2 is on the same LAN and tries to send directly. If it does not have a cached MAC address for E2, E1 does ARP just as the routers do.

- E1 to E3 traffic should go through R2 eventually. Suppose E1 sends a packet to E3 first to R1 who then sends the packet back on the LAN to R2. Whenever a router sends a packet back on the same interface it arrived it on, the router sends a second packet called a Redirect packet. The Redirect packet is sent directly to the source of the packet and essentially says "Please be a good fellow and send directly to R2 in the future, without bothering me unnecessarily". On receiving a redirect, E1 maintains a small cache that maps from E3 to R2. Endnodes always consult this "preferred router cache" before sending a packet to a destination IP address. This may seem like cheating because we said earlier that we did not want E1 to maintain the optimal router to all endnodes, and here we are keeping a cache. The difference is that the cache for an endnode will consist of tens of nodes (for clients) and say 1000's of nodes (for servers) of all the nodes the endnode is talking to concurrently. It does not have to keep track of the millions of possible nodes on the Internet.

- If R1, R2 are down, E1 and E2 should be able to communicate. In IP this happens naturally because of the IP solution to Problem 4. In OSI, where endnodes always go through a router to communicate the solution is trickier and requires some form of broadcasting very similar to ARP.

We have seen so far how addresses are assigned in an IP network and how endnodes and routers talk to neighbors, with the case of LAN neighbors being the only non-trivial issue. The stage is now set for route computation and forwarding.