Routing Issues

Norman Matloff
Dept. of Computer Science
University of California at Davis

August 8, 2000

1 Hopping Among Various Networks

Recall that the term, *the Internet*, refers to the interconnection of large numbers of **networks**, where one network refers to one Ethernet (or one FDDI or other LAN). Each network will have at least one **router** node, which connects that network to other networks. To get from the source node to the destination, a packet will typically hop from one network to another to another, etc., being passed from one router to another, until it reaches its destination node.

1.1 IP Network Numbers, Subnets Etc.

Recall that an IP address consists of a 32-bit number. This number is further broken down into a network number and a host number within that network. IP allows these into three major categories:

- Class A networks, consisting of a 0 bit, then a 7-bit network number, then a 24-bit host number.
- Class B networks, consisting of a 10 bit field, then a 14-bit network number, then a 16-bit host number.
- Class C networks, consisting of a 110 bit field, then a 21-bit network number, then an 8-bit host number.

A network administrator, say a university campus or a large corporation, will be allocated a block of network numbers, which he/she will then assign to the networks and hosts over which he/she has control. Clearly, Class A networks allow for the largest numbers of hosts, and thus should be given to the largest user entities.

However, even then the above setup would be wasteful. A Class B network, for instance, has number space for as many as 2^{14} or about 16,000 hosts—far more than we could physically place on an Ethernet. Thus there is also something called a **subnet**, which is used to further subdivide what appears to be just one network into many networks, as follows.

The IP address of my machine, heather.cs.ucdavis.edu, is 169.237.6.184, i.e. 0xa9ed06b8, thus a Class B address. Thus the 169.237 is the network number, and in fact this is the network number of all machines at

UCD. But UCD also has a **subnet mask** which applies to those machines, with the value 0xffffff00. This value is known to all routers on campus (though not to routers outside the campus), and is used to divide the huge UCD virtual "network," number 169.237 on the Internet, into the actual physical networks. The actual network number is obtained by taking the bitwise AND of 0xffffff00 and the IP address of the given host. In our case at UCD, that means that the actual network number consists of the first 24 bits of the IP address, rather than the first 16 bits as given in the definition of Class B. The extra 8 bits are called the **subnet ID**.

So, my machine can be described as host 184 on the network 6 at UCD, with the collection of all networks at UCD being (virtual) network 169.237 on the Internet.

1.2 Determining the IP Number of the Destination

(Review from our unit NetIntro.tex.)

One of the things which will have to be done is to get the IP address for our destination. Recall that the function gethostbyname() does this—but how?

On a UNIX system, the function first may check the local file /etc/hosts, where some frequently accessed destination information might be stored. If not, then it probably will use the Domain Name System (DNS). (Some Microsoft Windows networks use Microsoft's own name lookup system, WINS.) Briefly, your OS will have one or more DNS machines which it can query to convert the "English" name to an IP number. (They in turn may have to check other DNS hosts to get the information you want.) The addresses of these machines may be stored in a file, say /etc/resolv.conf, or might be specified in other ways.

1.3 Routers

A router is a machine that, as its name implies, routes packets through the Internet. It may be an ordinary workstation or PC, or it could be a special computer, say produced by Cisco, whose hardware and software are developed specifically for fast routing. Recall from our handout titled "Overview of Computer Networks" that networks are typically interconnected in one of two ways:

- A router could be be attached to two Ethernets, serving as a gateway between them. Here the router could consist of a computer with two Ethernet cards, each attached to some Ethernet. The Network layer running on that machine would read packets coming in on one card and, if routing is needed, write them to the other card.
- A router could be connected to another router via a high-speed phone line. Here a router would read packets from its Ethernet card and send them through the phone line (say using HDLC or PPP), via which they would be received by the other router and relayed on to its Ethernet.

Say machine X, on an Ethernet connected to the Internet, has gotten a message ready to send, say to machine Y. The TCP/IP software there will first put the message in a TCP or UDP packet (typically in response to a call from a socket function), which will in turn encapsulate the packet in an IP wrapping, which will in turn have an Ethernet frame wrapped around it. The question now is, Which destination Ethernet address will it put in that frame?

To determine this, X will first look at the destination IP address. It will consist of a network number and a host number within the network. (Here when I say "network number," I am including any subnetting known to this router.) If the destination Y's network number matches that of X, then the IP layer at X will say, "Hey, we are lucky; the destination machine is on the same Ethernet as us." It will then place the Ethernet ID of the destination machine in the Ethernet frame which it gives to the Link layer at X. That frame will be put on that Ethernet, and will thus be seen by all hosts on that Ethernet, in particular our destination, which will grab the frame and thus receive our message.

But what if the destination is not on the same network? The system on X will maintain a **routing table**, showing the addresses of all routers on this Ethernet. The software will check its routing table, and send the packet to a router on that Ethernet (so the Ethernet address in this packet will be that of the router). The router will check to see whether the destination machine, Y, is on any of the Ethernets to which the router is attached. If so, the router will send the packet directly to Y; otherwise, the router will send the packet to another router on one of the Ethernets to which the router is attached. In the latter case, again the choice of router will be made according to the routing table at that node. This process will be repeated, with the packet hopping from one network to another, until it reaches the destination.

1.4 Finding the Ethernet IDs of Other Machines on This Ethernet

In our story above, if Y is on the same Ethernet as X, X will sent to Y directly, by putting Y's Ethernet ID into the packet. But how does X know Y's Ethernet ID?

This is determined via the Address Resolution Protocol (ARP): Suppose X does not already know Y's Ethernet ID from calling ARP previously, in which case it may be in X's **ARP cache**. Then X will send an ARP frame out onto its Ethernet, asking "Y, please let me know your Ethernet ID." (This frame will be distinguished from an ordinary IP frame, since the latter uses 0x0800 as the protocol ID, while ARP uses 0x0806.) Again, since the Ethernet is a broadcast medium, X will use the Ethernet's broadcast address, so all nodes on the Ethernet will see this frame, including Y, and Y will then place a frame of its own on the Ethernet in response. (Y will not use the broadcast address, as it will know X's Ethernet address from the frame X sent.)

1.5 Software Tools for Troubleshooting Your Machine/Network

UNIX machines have a rich array of programs which you can use for troubleshooting, or for just learning about your system. (The latter is important; it will really make the concepts concrete, so give it a try!)

You can view a UNIX machine's routing table with the command "netstat -r". A **netstat** command is provided by Windows systems too.

You can determine the broadcast address for your Ethernet by using the UNIX **ifconfig** command. (On UNIX machines, the -i option of **netstat** gives similar information, so you may be able to get something like this from the **netstat** command on Windows platforms.) The command will also give you other information, such as the IP address of your machine, the value of subnetting mask and so on.

On some machines, the **netstat** command's -p option will give you the current ARP table.

If you have just subscribed to a new ISP but can't access all (or some) machines on the Internet, try doing so using their numerical IP address. If the latter works, then you likely have some kind of DNS problem. On a Linux system, for example, this may be due to your not having put the DNS addresses in the file /etc/resolv.conf.

If you are having DNS troubles only with a specific remote host you can troubleshoot using the UNIX **nslookup** command. There is a version in Windows NT too. There is no version included with Windows 95/98, but you can get similar programs for these platforms on the Web; for example, do a search for "nslookup" at www.tucows.com.

2 Illustration of Hopping: traceroute

The program **traceroute** illustrates routing through the Internet, and can be used (as a first approximation, at least) to investigate bottlenecks. This program is available from the Internet, say from

```
ftp://ftp.umiacs.umd.edu/pub/SUGstuff
```

Note, though, that most of these tools are runnable only via root privileges (e.g. by setting the set-user-ID bit in the file permissions). They do not (as far as I know) present any security risks, but they use the SOCK_RAW protocol in TCP/IP, which requires root access. If you have a Linux system and a SLIP or PPP account (or if you are a UCD CS student, just log in to one of the CSIF Linux PCs), you may wish to experiment with them. (If you have a Windows system, there is a program named **tracert** which is similar to the UNIX **traceroute**.)

Here is a sample output from the program:

```
heather:NetworkInfo/Tools/TraceRoute% traceroute nbc.com
traceroute to nbc.com (192.35.39.100), 30 hops max, 40 byte packets
 1  ph-254-subnet56.cs.ucdavis.edu (128.120.56.254)  *  0 ms  4 ms
 2  eu2-gw.ucdavis.edu (128.120.66.254)  4 ms  16 ms  8 ms
 3  telcofddi-gw.ucdavis.edu (128.120.128.27)  4 ms  4 ms  4 ms
 4  barrn-gw.ucdavis.edu (128.120.250.1)  4 ms  4 ms  8 ms
 5  paloalto-cr2.bbnplanet.net (131.119.2.37)  55 ms  31 ms  51 ms
 6  paloalto-mci.bbnplanet.net (131.119.0.202)  35 ms  20 ms  16 ms
 7  borderx1-hssi2-0.SanFrancisco.mci.net (204.70.158.101)  12 ms  23 ms  16 ms
 8  core2-fddi-0.SanFrancisco.mci.net (204.70.158.49)  16 ms  16 ms  12 ms
 9  core2.WestOrange.mci.net (204.70.4.185)  78 ms  133 ms  86 ms
10  core1-hssi-4.NewYork.mci.net (204.70.1.97)  82 ms  82 ms  78 ms
11  border2-fddi-0.NewYork.mci.net (204.70.3.18)  86 ms  82 ms  113 ms
12  jvncnet-ges-ds3.NewYork.mci.net (204.70.45.10)  82 ms  109 ms  98 ms
13  130.94.40.252 (130.94.40.252)  86 ms  82 ms  86 ms
14  ge-gateway.jvnc.net (130.94.12.114)  98 ms  98 ms  86 ms
15  peacock.nbc.com (192.35.39.100)  90 ms  90 ms *
```

Here we see the path taken from my UCD machine to NBC's Internet node, comprising 15 hops. The first four hops are to UCD machines, ending up at UCD's gateway to BARRNET (hop 5); the latter was (at the time I obtained this output) a network involving a number of sites in the greater San Francisco Bay region. The **traceroute** program sends three separate messages to each hop, reporting the round-trip transit time for each in milliseconds.

By the way, it is instructive to note how **traceroute** works. In order to prevent having an IP datagram circulate endlessly in some accidental routing loop, it contains a Time to Live field, which states how many more hops through the Internet this datagram is allowed to take. That number is decremented by 1 at each router it passes through, and when it reaches 0, the router at which this occurred sends an Internet Control Message Protocol (ICMP) message back to the original source node. The author of **traceroute** exploited this fact in a clever way, as follows.

Say we wish to know location of the fifth router on our way to nbc.com. Then **traceroute** simply initializes the Time to Live field in a test datagram to 5. The datagram will then expire at the fifth router, paloalto-cr2.bbnplanet.net in our example above, and that router will send an ICMP "error" message back to us; **traceroute** will then say to itself, "Aha! The fifth router is paloalto-cr2.bbnplanet.net."

Supposedly there are some routers which wish to remain "anonymous," and thus will not send back an ICMP message, so as to keep their IP addresses secret. This presumably is due to security concerns on the part of the administrator of such a router.

3 Hierarchical Route Determination

In large networks, it is infeasible to do routing computations for all nodes. Instead, **hierarchical routing** is used. This means that the network is broken up into **domains**, and that all nodes in a domain are treated as identical for routing purposes. One node in the domain acts as a representative for all nodes in the domain, and it will be the only node from the domain to enter the routing computations. Since the overall network will contain far fewer domains than nodes, the routing computation is much easier and quicker.

This setup is natural anyway. Consider a domain which consists of a single Ethernet, and on which the node g is the **gateway** to the outside world. In other words, g is the only node on this Ethernet which is physically connected to at least one other network. Then it is natural to choose g as the "representative" mentioned above, and in fact to set things up so that any packet from other nodes on this Ethernet which will go to the outside world will have its routing determined by g.

An **autonomous system** (AS) is a network or collection of networks under the administration of one central entity, say a university campus. ASs form a natural hierarchy for the hierarchical routing notion: Within an AS, routes may be optimized using some mathematical algorithm, say the Dijkstra algorithm described below. Then routing between ASs will be handled cooperatively using some other protocol, say the Border Gateway Protocol.

4 Least-Cost Routes

We will discuss **least-cost** routing algorithms. The term "cost," though, is an abstraction which could have various definitions. We define a cost according to the goal we have. If for instance we wish to find a path from a given source to a given destination which has the least number of "hops" (i.e. least number of links traversed), then we assign a "cost" of 1 to each link. If we wish to minimize overall queueing delay, we could define the cost of a link to be the current length of the queue for that link. In this case, our routing strategy will be **dynamic**, meaning that we will frequently update the routes as conditions change.

The routing tables throughout the Internet attempt to forward packets only paths which have minimum "cost," where that could be defined in terms of the bits-per-second ratings of various links, the number of hops, the estimated queuing times on the links, and so on.

There are two famous classes of algorithms for finding least-cost paths. In both cases, we discuss how to find the least-cost path from a source s to a destination node n. Let N be the set of all nodes, and let $c(i, j)$ be the "cost" (however defined) associated with the link from node i to node j, if there is a direct link between them. If there is no link between i and j, set the cost to infinity.

Note that both algorithm classes operate on a **distributed** basis. This means that each node computes its own routing tables, as opposed to the **centralized** case, where one node computes the tables for all other nodes. Centralized routing is considered infeasible for large networks.

4.1 Link-State Algorithms

The name here refers to the fact that the calculation relies on knowledge of the "state" (cost) of all links in the entire network. A well-known algorithm of this class is that of Dijkstra, originally used in the ARPANET and TYMNET II networks, and now used widely in a modified form call the Open Shortest Path First (OSPF) algorithm. It works as follows.

Let S be a subset of N; S will initially be equal to $\{s\}$, but we will add one node to S in each iteration. At any given time, for any node n we will have the least-cost path p(s,n) from s to n, among paths which use only nodes within the current S. Denote the cost of that path by m(s,n). The algorithm terminates when S = N, and we have the overall least-cost path.

Here is the corresponding pseudocode:

```
set S to {s}, m(s,n) to infinity for all n not equal to s
do
   find the node w not in S for which m(s,w) is smallest
   add w to S
   for each n not in S
      if m(s,w)+c(w,n) < m(s,n) then
         set m(s,n) to m(s,w)+c(w,n)
         set p(s,n) to p(s,w) concatenated with the link from w to n
until S = N
```

(In the first iteration, the third line of code will simply find the w for which c(s,w) is minimum, since the set S will consist only of s at that time.)

The principal drawback of this class of algorithms is that any given source must know the entire network topology and all link costs. These may change: For example, a link or node may go down, and if cost is defined in terms of something like queue length, the cost will change frequently. Thus update information must be periodically transmitted to the entire network, which itself is a problem due to the extra traffic it generates.

4.2 Distance Vector Algorithms

The word "vector" in this name stems from the fact that instead of directly concerning itself with a full least-cost path to a node n, s will simply determine the first step of such a path, i.e. "which direction to go first." The word "distance" refers to the fact that the total cost of a path to n is often casually called the distance between the two nodes.

The chief advantage of the distance-vector algorithm approach is that s does not have to know all the c(i,j) values, but rather need know only c(s,k), for those nodes k which are immediate neighbors of s. This will have important practical considerations, as we will see later.

The most famous member of this algorithm class is the Bellman-Ford algorithm, which works in the following manner.

The algorithm is again iterative. Any given source s knows only the costs of the links to its immediate neighbors, plus whatever new information the neighbors have provided. To explain the latter, suppose k is a neighbor of s. The node s will ask k, "What is the path to n which is of least cost, **as far as you know**?" For whichever neighbor k reports the least cost (plus the cost of going to k), s will set its own best path to n to be "Go first to k, and then follow directions to n from there." Note that the situation will be the same at k: It will set its best path to be of the form "Go first to j, and then follow directions to n from there."

To state the algorithm more precisely, retain the notation m( ), p( ) etc. introduced earlier, and for any nodes

s and n, let f(s,n) be the "first step" from s to n, as with k above. Here then is the algorithm:

```
for all s
   for all n not equal to s
      if n is an immediate neighbor of s then
         set f(s,n) to n
         set m(s,n) to c(s,n)
      else
         set f(s,n) to "unknown"
         set m(s,n) to infinity
while (true)
   for all s
      for all neighbors k of s
         send f(s,r) and m(s,r) to k for all r
         receive f(k,r) and m(k,r) from k for all r
      for all n
         for all neighbors k of s
            if c(s,k) + m(k,n) < m(s,n) then
               set f(s,n) to k
               set m(s,n) to c(s,k) + m(k,n)
```

Note that node s will only know the values f(s,n) and m(s,n), for varying n. In other words, if we were to represent f and m as tables, node s would know only its own row in each table.

Let us examine what is happening in the **while** loop. First, each pair of neighboring nodes exchanges information about the best paths known to each node so far.[1] Then each node will update its best path to a given node if the neighbor's information indicates a better one.

Assume for the moment that the link costs do not change, and no nodes or links go down. Then after i iterations of this algorithm, each node will know the best path of i hops or less to any other node. Thus, this process will converge after at most D iterations, where D is the **diameter** of the network, i.e. the largest number of hops between any two nodes in the network.

However, we have written the code above so that the iterating in the **while** loop continues forever. It is implemented this way in practice, since the link costs may change, and thus routes will need to be updated. In the Routing Information Protocol commonly used in Internet routing, for instance, there is an information exchange between neighbors once every 30 seconds.[2]

The Bellman-Ford algorithm has its drawbacks too. One problem is that it reacts slowly to changes. Say nodes u and v are i hops apart, and that a link leading out of v changes cost. This news will not reach u until i iterations later.

[1] Though we have shown this exchange as occurring at the beginning of an iteration, in practice the exchange is asynchronous.

[2] Optimal paths are periodically computed for Dijkstra's algorithm too. However, in that case there are no natural recomputation points within the code, whereas in the Bellman-Ford case the natural recomputation point is the exchange of information between neighboring nodes.