Chapter 8

Navigating the Net

The Internet or any other network that depends on switches or routers to forward a packet through a series of links from its source to its destination must solve a significant navigational problem. There must be a way to find a good path each packet can follow to reach its destination through a potentially large and complex series of network links. In computer networks, this task is called *routing.*

The routing problem has a lot in common with a task we are all familiar with: finding a path to follow when driving to a location we have not previously visited. Thinking about how we formulate driving directions can give useful insights into how the computers in a network can solve the routing problem, particularly if we rule out options like using Mapquest or installing a GPS system in our car.

The first step in deciding how to drive to a new destination is to find a map showing the roads in the area through which we will travel. Then, using this map we can identify possible routes from our starting point to our destination and compare the length and convenience of each possibility to decide which router to follow. Similarly, it is obvious that the computers in a network must have some sort of a map describing the network and some procedure for identifying and comparing possible paths using the information in this map.

In this chapter we will focus on three aspects of how computer networks perform routing. First, we will spend some time thinking about what a map of the network should look like. Maps for humans come in several specialized varieties. While contour lines showing changes in elevation are very important in a map meant for planning a hike, they are not so important in a road map. Not surprisingly, the maps used by computers to find routes will differ from the forms of maps with which we are familiar in several ways.

Second, once we have an understanding of the information that will be available to computers planning routes for packets, we will examine an important algorithm that can be used to find good paths. This algorithm is known as Dijkstra's algorithm or the Shortest Path First algorithm. It is the basis for an important Internet routing protocol known as OSPF (Open Shortest Path First).

Finally, we will consider an issue that most of us don't worry about when planning driving directions. We will discuss how to make maps. Links in a computer network are added and/or fail much more rapidly than changes occur in most road systems. As a result, the maps used for network routing must be updated frequently. In fact, on some networks the information used to make routing decisions is updated every few minutes. This requires an automated process for distributing the routing information that makes up the "map" throughout the network.

8.1 Encoding Routing Information

Although the information computers use to identify good routes in a network is different from the information provided by a typical road map, thinking about road maps is a good way to begin to understand what information is important for network routing. With this in mind, a map of the interstate highways connecting cities in Texas and its northern neighbors is shown in Figure 8.1.

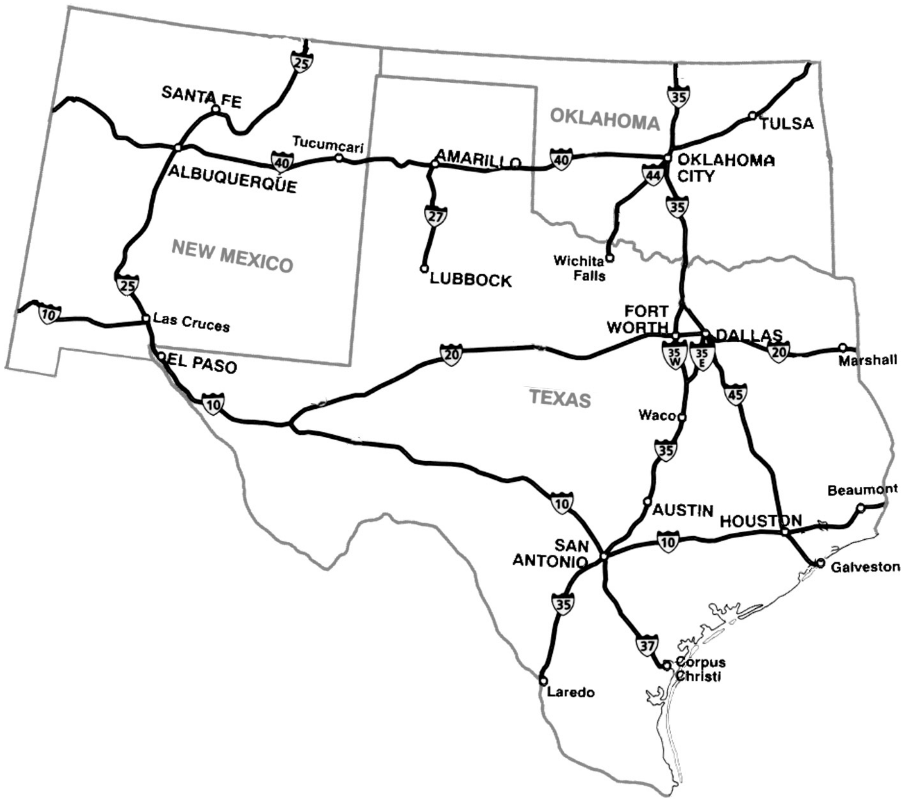

Figure 8.1: Road Map of Texas, New Mexico, and Oklahoma

When using a map like Figure 8.1, we extract two critical types of information. We use a map to determine points between which roads exist and to determine the approximate lengths of those roads. We depend heavily on the physical layout of roads and towns to enable us to estimate the relative distances involved. On many maps, one can find the actual length of various roads, but most drivers don't pay too much attention to these lengths. Instead, knowing that the shortest distance between two points is a straight line, we visualize a straight line between our starting and ending points and then we look for roads that stay reasonably close to this imaginary line. A computer, however, has no eyes. It cannot "look" at an image that encodes information about pathways in a network or roads between cities and visually identify appropriate routes.

In addition, the technique of approximating a straight line is not guaranteed to find the shortest route. To be sure to find the shortest route we would have to determine and add up all the mileage figures for all the roads we might travel. If we took this approach, it would be less important that the map we used provided an accurate description of the physical layout of the roads and more

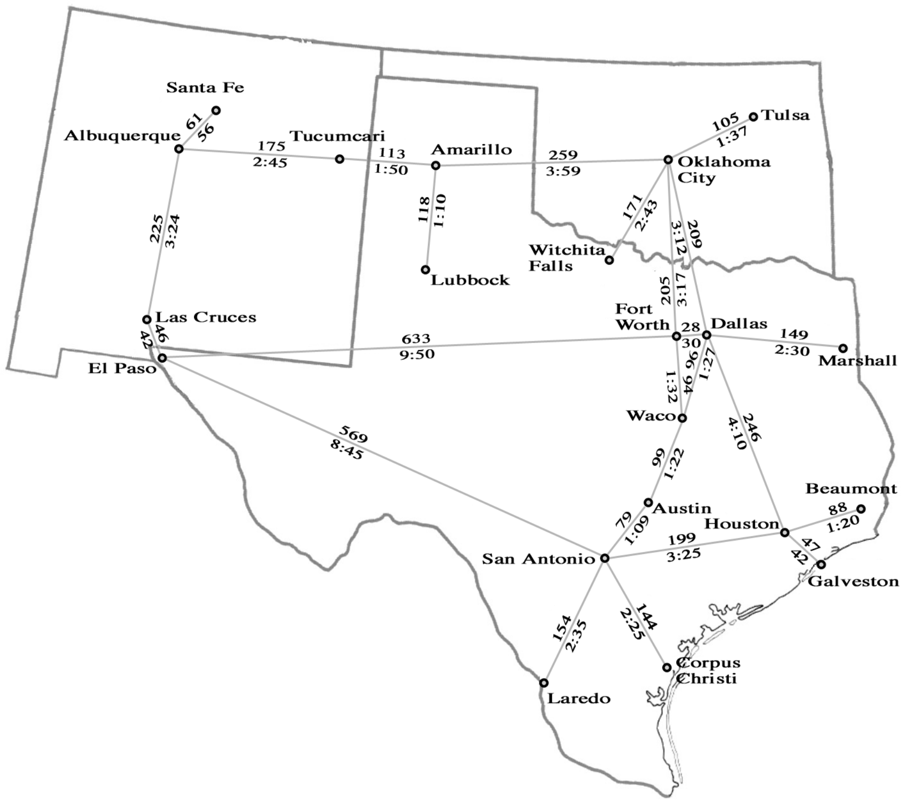

Figure 8.2: Travel times and distances between Southwestern cities

important that it make it easy to extract accurate mileage figures. The map shown in Figure 8.2 is designed with this goal in mind. Rather than showing the twists and turns of the actual roads, it simply uses straight lines to indicate which cities are connected by interstate highways. Each such line is labeled by both the distance between the cities it connects (above the line) and an estimate of the time required to drive between the cities (below the line).

The values associated with each connecting line in Figure 8.2 provide information that could be used to to plan routes with one of two different strategies. If you wanted to save gas or minimize wear on your car by driving as few miles as possible, you would use the number above each line to identify a path that contained as few miles as possible. The times required to drive two different roads, however, may not be directly proportional to their distances. The speed limit on one road may be higher than another or one may be more congested than the other. So, if time rather than mileage was your actual concern, you would ignore the mileage numbers and concentrate on the time estimates while planning your trip.

When we consider exactly how to plan routes through a computer network, we will see that time estimates are much more useful than physical lengths. Messages sent through links in a computer network travel a bit faster than 65 mph. They travel at the speed of light, which is roughly 700 million mph. As a result, the time a given bit actually spends traveling from source to destination is often negligible. What really matters is how long it takes to get the bits onto the communications

	Albuquerque	Las Cruces	Santa Fe	Tucumcari
Albuquerque		3:24	56	2:45
Las Cruces	3:24			
Santa Fe	56			
Tucumcari	2:45			

Figure 8.3: Adjacency table for New Mexico

	Albuquerque	Las Cruces	Santa Fe	Tucumcari
Albuquerque	0	3:24	56	2:45
Las Cruces	3:24	0	4:20	6:09
Santa Fe	56	4:20	0	3:41
Tucumcari	2:45	6:09	3:41	0

Figure 8.4: Travel time table for New Mexican cites

link. For example, if one tries to send a 1,000,000 bit message through a DSL connection running at 128,000 bits per second (a typical speed for upload bandwidth on "economy" DSL service), it will take about 8 seconds to send the message even though each bit will take just a fraction of a second to traverse the link. Accordingly, we will want to find the route that takes the shortest time.

While the map in Figure 8.2 does not show the exact paths followed by the highways it describes, it still provides information through its physical layout. In particular, the positions of the cities shown are still based on their actual physical positions. In a computer representation of the information in such a map, layout cannot be important because, as mentioned above, a computer can't see. To understand what a computer has to work with when trying to find paths through a highway network or a data network, we need to look at non visual ways of representing "maps".

A simple, textual approach to providing the driving time information found in the map from Figure 8.2 is shown on the next page in Figure 8.5. It is a table with one row and one column for each city shown on our original map. Given a pair of cities, the table entry found in the row associated with the first city and the column associated with the second city provides information about interstate highways between the two cities. If there is nothing in the selected table entry, then there is no direct highway connection between the two cities. If there is a number in the table entry, then there is a direct highway connection between the cities and the number in the table entry is the estimated time required to drive from the first city to the second.

It is important to note some differences between the information presented in this table and the information found in relatively similar tables commonly included in automobile road maps. The tables found in road maps typically provide total distances (or travel times) between each pair of cities. To clarify the difference between such tables and the table in Figure 8.5, we will restrict our attention for a moment to a smaller set of cities.

A table using the same format as Figure 8.5 but limited to just the cities from New Mexico is shown in Figure 8.3. A table for the same set of cities providing total travel times between the cities similar to what is typically found in a road map is included in Figure 8.4.

The most striking difference between the two tables is that the table in Figure 8.3 has many

	Albuquerque	Amarillo	Austin	Beaumont	Corpus Christi	Dallas	El Paso	Fort Worth	Galveston	Houston	Laredo	Las Cruses	Lubbock	Marshall	Oklahoma City	San Antonio	Sante Fe	Tucumcari	Tulsa	Wichita Falls	Waco
Albuquerque												3:24					56	2:45			
Amarillo													1:10		3:59			1:50			
Austin																1:09				1:22	
Beaumont										1:20											
Corpus Christi																2:25					
Dallas								30		4:10				2:30	3:12					1:27	
El Paso								9:50				42				8:45					
Fort Worth						30	9:50								3:17					1:32	
Galveston										42											
Houston				1:20		4:10			42							3:25					
Laredo																2:35					
Las Cruces	3:24						42														
Lubbock		1:10																			
Marshall						2:30															
Oklahoma City		3:59				3:12		3:17											1:37		2:43
San Antonio			1:09		2:25		8:45			3:25	2:35										
Santa Fe	56																				
Tucumcari	2:45	1:50																			
Tulsa															1:37						
Waco			1:22			1:27		1:32													
Wichita Falls															2:43						

Figure 8.5: Travel times between major cities in Texas, New Mexico, and Oklahoma

blank entries for which the corresponding entries in Figure 8.4 contain useful values. For example, the entry for the total distance from Santa Fe to Las Cruces in Figure 8.4 is 4:20. The corresponding table entry in Figure 8.3, on the other hand, is empty since there is no direct link between Santa Fe and Las Cruces.

At first, it might appear that a table with many empty cells must provide less information than a table in which all the cells are filled. In fact, in this case the opposite is true. The only form of question we can answer using a table like Figure 8.4 is "How long will it take to get from A to B?" We cannot determine what other cities we might have to pass through to get from A to B or even whether there is a road that connects A and B directly. The information in Figure 8.5 (and Figure 8.3), on the other hand, tells us exactly which cities are directly connected by roads and how long it will take to drive between each connected pair of cities. Using this information, we can still answer the question "How long will it take to get from A to B?" It will, of course, require more effort to answer such a question using Figure 8.3 than it would using Figure 8.4. In the process, however, the information in Figure 8.3 will enable us to determine what other cities we will pass through on the way from A to B.

It may appear that we could save space in Figures 8.4 through 8.5 by including only one entry for each pair of cities. All of these tables include one entry for traveling from A to B and another for traveling from B to A. Typically, the distance between city A and city B is the same as the distance between city B and city A. It might seem reasonable to eliminate half the entries in either of our tables by eliminating this duplication.

In reality, the entry for travel from city A to city B may not be identical to that for travel from city B to city A. In a table that records travel times rather than distances there are many situations in which the direction of travel may be significant. There may be road construction on the northbound lanes of the highway from A to B, making travel in one direction slower than travel in the other. If the road leads to a major city, then travel into the city is likely to take longer than leaving the city during morning rush hour. Such effects may exist only for limited time periods, but they will influence the travel time required and therefore the best route.

Examples of links in which travel time depends on direction also abound in networks. Network links leading to a major web site like yahoo.com are likely to carry less traffic than the same links carry in the opposite direction because the amount of data Yahoo.com sends in response to each user request tends to be larger than the incoming request. As a result, it is likely to take a single packet less time to travel from a user's machine to yahoo.com than it takes for a similar packet to travel in the other direction. Therefore, in representing maps of network connections, we would like to maintain the flexibility to record separate travel time estimates for travel in each direction on a given link.

We can, however, reduce the wasted space in our table travel times by modifying its form. We can just list information about all the pairs of destinations that are directly connected. Any pair that is not included in the list would correspond to an empty cell in the tabular format. We could do this using a list like:

Albuquerque → Las Cruces = 3:24
Albuquerque → Santa Fe = 56
Albuquerque → Tucumcari = 2:45
Amarillo → Lubbock = 1:10
Amarillo → Oklahoma City = 3:59
Amarillo → Tucumcari = 1:50

Cities	Neighbors				
Albuquerque	Las Cruces, 3:24	Santa Fe, 56	Tucumcari, 2:45		
Amarillo	Oklahoma City, 3:59	Tucumcari, 1:50	Lubbock, 1:10		
Austin	San Antonio, 1:09	Waco, 1:22			
Beaumont	Houston, 1:20				
Corpus Christi	San Antonio, 2:25				
Dallas	Oklahoma City, 3:12	Fort Worth, 30	Houston, 4:10	Marshall, 2:30	Waco, 1:27
El Paso	Fort Worth, 9:50	San Antonio, 8:45	Las Cruces, 42		
Fort Worth	Waco, 1:32	Oklahoma City, 3:17	El Paso, 9:50	Dallas, 30	
Galveston	Houston, 42				
Houston	Beaumont, 1:20	San Antonio, 3:25	Galveston, 42	Dallas, 4:10	
Laredo	San Antonio, 2:35				
Las Cruces	Albuquerque, 3:24	El Paso, 42			
Lubbock	Amarillo, 1:10				
Marshall	Dallas, 2:30				
Oklahoma City	Wichita Falls, 2:43	Fort Worth, 3:17	Amarillo, 3:59	Dallas, 3:12	Tulsa, 1:37
San Antonio	Corpus Christi, 2:25	Austin, 1:09	El Paso, 8:45	Houston, 3:25	Laredo, 2:35
Santa Fe	Albuquerque, 56				
Tucumcari	Albuquerque, 2:45	Amarillo, 1:50			
Tulsa	Oklahoma City, 1:37				
Wichita Falls	Oklahoma City, 2:43				
Waco	Austin, 1:22	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.6: Immediate neighbors for southwestern cities

Austin → San Antonio = 1:09
Austin → Waco = 1:22
...

or we could arrange lists of each city's immediate neighbors in a table as shown in Figure 8.6. A list like this provides exactly the same information as the big table.

A very similar table of information can be used as the basis for networking routing. The city names in Figure 8.6 would be replaced by the addresses of network switches and routers. The entries would still include travel times, but the units would probably be quite different. Hours and minutes would likely become milliseconds. A table of such routing information that describes a small, imaginary network (whose topology is remarkably similar to that of highways in the southwest) using IP addresses to identify routers is shown in Figure 8.7. Each row of this table provides information about the router whose address appears in the first column. The remaining columns in a row describe routers that are directly connected to the machine identified in the first column. Each reachable router's address is followed by an estimate of the time required to send a message directly to that router.

8.1.1 One Step at a Time

We have repeatedly suggested that human vision plays an important role in the way we use maps to find routes. If you still do not believe this, compare how you would chose the best route from

Router	Adjacent Routers				
210.44.23.2	145.32.84.1, 193	172.59.12.4, 56	210.3.67.18, 98		
197.210.8.1	137.165.8.2, 157	210.3.67.18, 110	182.27.90.14, 147		
150.100.8.2	133.29.127.4, 154	165.211.34.101, 172			
201.34.77.1	189.32.148.200, 96				
189.32.148.201	133.29.127.4, 207				
197.210.8.2	165.211.34.101, 122	189.32.148.200, 211	137.165.8.2, 312	221.18.94.10, 87	159.250.7.3, 77
162.47.200.3	159.250.7.3, 45	133.29.127.4, 115	145.32.84.1, 142		
159.250.7.3	165.211.34.101, 125	137.165.8.2, 117	162.47.200.3, 122	197.210.8.2, 80	
192.168.4.1	189.32.148.200, 92				
189.32.148.200	201.34.77.1, 170	133.29.127.4, 125	192.168.4.1, 52	197.210.8.2, 110	
205.4.107.6	133.29.127.4, 135				
145.32.84.1	210.44.23.2, 124	162.47.200.3, 72			
182.27.90.14	197.210.8.1, 205				
221.18.94.10	197.210.8.2, 230				
137.165.8.2	153.203.25.13, 137	159.250.7.3, 147	197.210.8.1, 139	197.210.8.2, 92	50.10.4.1, 113
133.29.127.4	189.32.148.201, 125	189.32.148.200, 135	162.47.200.3, 145	205.4.107.6, 167	150.100.8.2, 84
172.59.12.4	210.44.23.2, 156				
210.3.67.18	210.44.23.2, 45	197.210.8.1, 50			
153.203.25.13	137.165.8.2, 107				
50.10.4.1	137.165.8.2, 133				
165.211.34.101	150.100.8.2, 132	159.250.7.3, 175	197.210.8.2, 139		

Figure 8.7: Immediate neighbors for an imaginary network

Amarillo to Corpus Christi using Figure 8.2 to how you would find the shortest path from the router with address 197.210.8.1 to 189.32.148.201 using the information found in Figure 8.7.

As we hinted above, the entries in Figure 8.7 are patterned after those in Figure 8.6. Each router address corresponds to a city shown in Figure 8.2 and the interconnections between routers correspond to the interconnections between cities. The timing estimates included in Figure 8.7, on the other hand, are not related to those in Figure 8.6. As a result, looking at the map in Figure 8.1 does not really help you to find good paths in the network described by Figure 8.7. The only information you can really use to find the best path from 197.210.8.1 to 189.32.148.201 are the numbers in the table.

If you actually try this, you will almost immediately find yourself overwhelmed with possibilities to consider. From 197.210.8.1, you can either go to 137.165.8.2, 210.3.67.18, or 182.27.90.14. From 137.165.8.2, you could go to 153.203.25.13, 159.250.7.3, 197.210.8.2, or 50.10.4.1. Just keeping track of all these possibilities would be complicated enough. In addition, however, you would want to know how long it would take to reach each of these possible routers and which if any of them were getting you close to your ultimate destination, 189.32.148.201.

The good news is that we do not expect people to solve problems like this. We want computers to do it. Unfortunately, to program a computer to solve such problems, we need to devise an algorithm that will guide the computer through a sequence of steps that use the data available in a table like Figure 8.7 or 8.6 to find the best path.

In our introduction to this topic, we used cities and road maps to motivate the routing problem even though our real interest is with routers and network links. In the last few paragraphs, we have focused more on the network version of the problem to reinforce the notion that we need an algorithm that can find paths without the help of the visual cues humans can identify when using a map. As we describe how to solve this problem algorithmically, however, we will return to our examples based on cities and roads. It will be easier for you to read a description of a process involving planning a trip of several hours from El Paso to Amarillo than it would be if the description were filled with network addresses and times in milliseconds. There is, however, one important difference between the networking version of the problem and the road trip version that is important to understand before we proceed.

The fundamental difference is that cars have drivers and packets do not. That is, if a car is driving along the best route from El Paso to Amarillo, it is safe to assume that there is someone at the wheel or sitting in the passenger's seat who is doing the navigating. In fact, chances are that whoever is navigating actually planned the route. As a packet travels along the best route from 197.210.8.1 to 159.250.7.3, there is no internal agent guiding the packet. The packet contains some data and also the address of its destination, but it does not contain any active entity that guides its path. The guidance/intelligence required to get a packet to its destination is provided by the routers in the network, not by the packet itself. In addition, no single router provides the complete path. Each router simply decides which link a packet should be sent through next. Only when all of the decisions made by these independent routers get combined do we see the complete path.

If you imagine driving from Amarillo to El Paso the way a packet travels through the network, the difference will become clear. When you started your journey, you would not plan a route using a map. Instead, you would jump in the car and drive to find a nearby police officer or gas station attendant who was willing to act as a "router" for you. You would ask the police officer or gas station attendant for directions to El Paso. Strangely, the person you asked would not give you complete directions to your destination. Instead, they would act as if you could not remember any more than a single step. Therefore, they would just tell you the next city to go to and the highway to take to get there. That is, they might tell you to "Take interstate 40 until you get to Tucumcari" and nothing else.

Once you reached Tucumcari, you would again look for a friendly stranger and again ask for directions to El Paso. The person you asked would still only tell you the next big step you should take, "Continue on 40 all the way to Albuquerque." In Albuquerque, you would again ask for directions to El Paso and be told to "Follow I-25 South to Las Cruces." Finally, when you asked for help in Las Cruces you would be told to "Take route 10 into El Paso."

This would be a strange approach to take if you were traveling by car, but it is an appropriate way to handle packet traffic. Obviously, packets cannot guide themselves through the network. A packet is just a series of bits. We need a machine that can process these bits to figure out the path they should take through the network. Therefore, the routing must either be done by the machine that originally sends the packet or by the intermediate routers along the path.

If the route was planned by the original sender, it would have to be encoded in binary and included with the bits encoding the data the packet was intended to transport. In fact, this would also be necessary if any of the routers along the way try to figure out the rest of the steps to the destination instead of just worrying about the next step. The encoding of such paths could account for a significant fraction of the total size of many packets. Simply encoding the packet's destination address will certainly take fewer bits.

8.1.2 Routes and Forwarding Tables

If a router does the work required to figure out the best first step to some destination "B", it should save the answer so that it can use it later to handle other packets traveling to B without repeating the work of comparing routes. If it does this, a router will eventually be equipped with a table giving the best next steps for many possible destinations. Such a table is called a *forwarding table.*

We want packets delivered as quickly as possible. If a router has to execute a complex routing algorithm to determine the best path after a packet arrives, that packet will be delayed. A forwarding table makes it possible to determine how to handle packets very quickly. The router just looks up the packet's destination in the table and sends the packet to the indicated first step if a match is found. It would therefore be best if every router had a forwarding table with entries for all possible destinations.

At first, building a complete forwarding table sounds like much more work than finding routes for individual packets as they arrive. In fact, if you think about the process, you will realize that a router can build a complete forwarding table without individually computing the answer to every question of the form "How do you get from here to B?" As a result, building a complete table can be the efficient approach.

Suppose you just started working in a gas station in Amarillo and someone pulled in and asked for directions to El Paso. If you act like a router, the correct answer would be "Take interstate 40 until you get to Tucumcari." To realize this is the correct answer, you would look at the map in Figure 8.1 and notice that the best way to get to El Paso appeared to be to drive through Tucumcari, Albuquerque, and Las Cruces. This means that your answer, "Take interstate 40 until you get to Tucumcari," is not just the right answer to the question "How can I get from here to El Paso?" It is also the answer to the questions "How can I get from here to Las Cruces?" and "How can I get from here to Albuquerque?" Basically, in the process of finding the best route to a specified destination, you inevitably find the best routes to all of the steps along the way. As a result, the effort required to build a complete forwarding table for N destinations will be considerably smaller than the effort required to determine routes to those N destinations independently.

With this in mind, as we approach the problem of finding an algorithm for routing, our goal will not be to answer questions of the form "How do you get from here to B?" We will be looking for an algorithm to answer the question "How do you get from here to everywhere else?" This shift is important. Although finding routes to everywhere may at first sound harder than finding a route to a single destination, it will actually make it easier to find an algorithm to solve the problem.

8.2 The Shortest Path First Algorithm

How can trying to find paths to everywhere be easier than trying to find a path to just one destination? Simple! It gives us flexibility. If we have a specific destination, we have to think about how to get there. If we have to eventually figure out how to get everywhere, we have the freedom to choose which destinations to work on first. If we are smart (or just lazy), we will work on the easy destinations first. This is the basis for an algorithm known as the Shortest Path First algorithm. The algorithm's name is based on the idea that the easiest destinations to reach will be the ones that can be reached through the shortest paths. The algorithm works out these paths first, and only worries about longer paths later. This algorithm was first described in 1959 by Edsger Dijkstra. Accordingly, it is also frequently called Dijkstra's Algorithm.

To be concrete, let us see how this strategy could be applied to finding routes from El Paso, Texas to all of the other cities in the map in Figure 8.8. As we do this, we will try to simulate the process that would be used in a computer by ignoring the physical positions of the cities in the map and only using the data for travel times shown below the lines connecting cities and summarized in Figure 8.9. These two figures describe a subset of the cities and roads shown in Figure 8.2.

Our first goal will be to find the best route to the city that is the shortest distance from El Paso. This has to be one of the cities that is directly connected to El Paso by road. That is, the path we are looking for has to have only one step in it. Given this, we can look at the row for El Paso in Figure 8.9 to determine its immediate neighbors. There are two cities that are just one step away from El Paso. They are Fort Worth and San Antonio. The closest is San Antonio, which is 8 hours and 45 minutes away. Therefore, we know that San Antonio is the city that is closest to El Paso and that the entry in our forwarding table for San Antonio should simply say to drive directly to San Antonio.

Our next goal is to find the second closest city. For the closest city, we knew we only needed to consider cities that were directly connected to El Paso. This is no longer the case for the second closest city. In general, it may be possible to travel from the starting point through the closest city to a city that is not directly connected to the starting point in less time than we can reach any of the other cities that are directly connected to the starting point.

Luckily, the only city that could be an intermediate step on the shortest path to the second closest city is the closest city. This is because any city on the shortest path to a city must be closer to the starting point than the city at the end of the path. Accordingly, the only cities that can be on a shortest path to the second closest city are the starting point and the closest city. Given this fact, the only inter-city connections we need to consider are the direct connections from El Paso and San Antonio to other cities. These connections are all shown in Figure 8.10.

We can see that the total time required to reach Austin by driving from El Paso through San Antonio is just the sum of the time required to reach San Antonio from El Paso and the time to

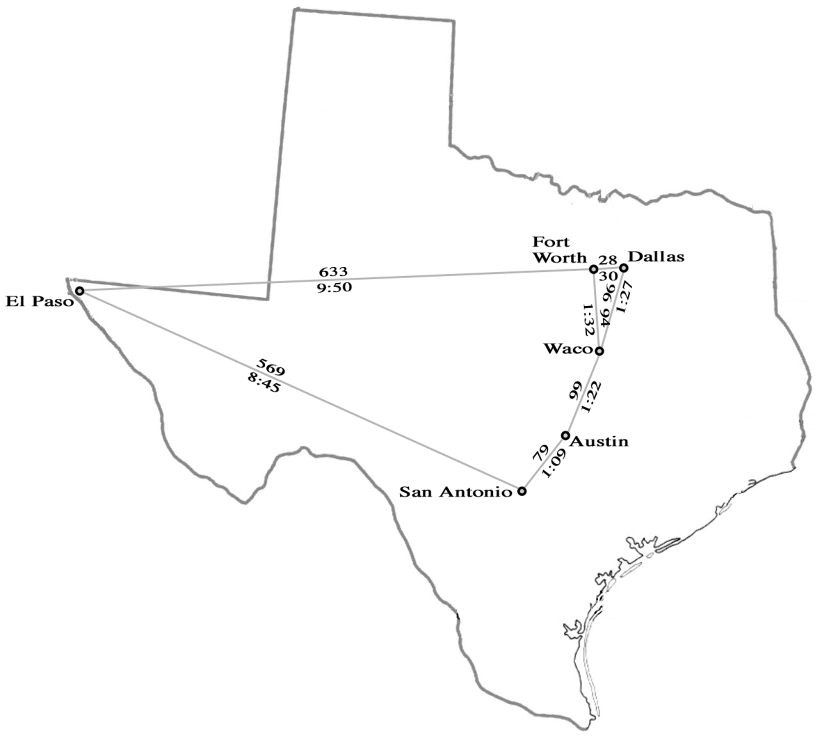

Figure 8.8: Travel times and distances between several Southwestern cities

Cities	Neighbors				
Austin	San Antonio, 1:09	Waco, 1:22			
Dallas	Fort Worth, 30	Waco, 1:27			
El Paso	Fort Worth, 9:50	San Antonio, 8:45			
Fort Worth	Waco, 1:32	El Paso, 9:50	Dallas, 30		
San Antonio	Austin, 1:09	El Paso, 8:45			
Waco	Austin, 1:22	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.9: Immediate neighbors for cities in Figure 8.8

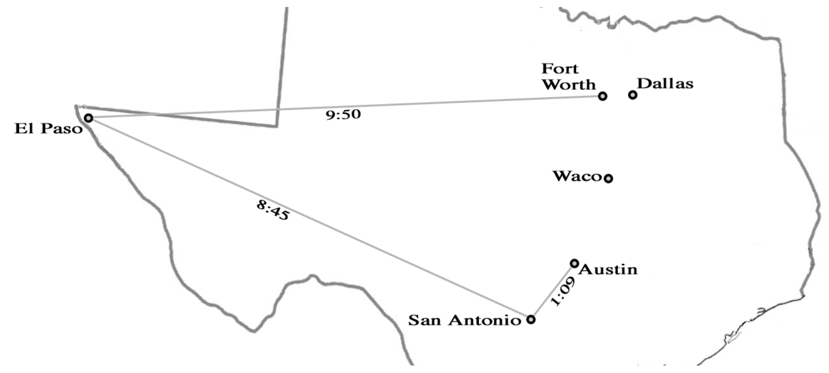

Figure 8.10: Direct connections from El Paso and San Antonio

reach Austin from San Antonio. That is, the total time will be 8 hours and 45 minutes plus 1 hours and 9 minutes or 9 hours and 54 minutes. This total is just a bit longer than the time required to reach Fort Worth directly from El Paso. We can therefore safely conclude that Fort Worth is the second closest city to El Paso.

Not only do we know that Fort Worth is the second closest city to El Paso, we also know what to put in the forwarding table entry for Fort Worth. The first step to get to Fort Worth from El Paso is simply to go to Fort Worth.

At this point, we can rank two cities in terms of their closeness to El Paso and know the first step on the way to each of these cities.

1. San Antonio is closest to El Paso. It can be reached in 8 hours and 45 minutes. The first step is to drive to San Antonio.

2. Fort Worth is the second closest city to El Paso. It can be reached in 9 hours and 50 minutes. The first step is to drive to Fort Worth.

Given the work we did to decide whether Fort Worth or Austin deserved second place, we also know a third fact:

3. There is a way to reach Austin in 9 hours and 54 minutes. The first step on such a path is San Antonio.

This fact is different from the first two. We don't yet know that 9 hours and 54 minutes is the best way to reach Austin or that Austin is the third closest city. We have, however, found at least one way to get to Austin and we should keep track of this fact.

As we proceed to identify the third closest city, the fourth closest, and so on, we will undoubtedly collect additional "facts" about best paths and path lengths. Before we do, we would like to suggest a notation that can be used to describe the information we collect more concisely.

To appreciate how a special notation might help us describe Dijkstra's algorithm, consider a very familiar algorithm, the process you were taught in elementary school for performing division. An example of the application of this algorithm is shown below.

$$\begin{array}{r}
407567 \\
32\,\overline{)13042144} \\
\underline{128} \\
242 \\
\underline{224} \\
181 \\
\underline{160} \\
214 \\
\underline{192} \\
224 \\
\underline{224} \\
0
\end{array}$$

Imagine how you would describe this process using prose as we have been describing the process of building a forwarding table:

> First, we take the divisor, 32, and compare it to the first two digits of the dividend, 13. Since 13 is less than 32, we realize we must examine one additional digit of the dividend, giving 130. The number 130 is larger than 32, so now we must determine the largest value such that 32 times that value is less than 130....

The description would clearly go on for pages. The special notation we use for describing the steps performed as part of the division algorithm makes it easier to complete the process and also leads to a very concise description of the steps that were performed. A standard, tabular approach to organizing the data we collect as we build a forwarding table can provide similar improvements.

The key to the system we will use is the observation that the collection of "facts" we must keep track of during the process has several nice properties. First, we are only interested in keeping track of one fact per city. Our goal is to find the shortest path to each city. Therefore, if we find two paths to a given city, we only need to remember the information describing the shorter path. We can safely discard information about longer paths. Second, we really only need two pieces of information about each path found: its length and its first step.

Given these observations, we can keep track of the information that matters by augmenting the tabular format we used to describe direct connections between cities in Figure 8.2 and 8.9 with two extra columns as shown in Figure 8.11. The two new columns are labeled "First step" and "Route Length". The "Route length" column in a given row will be used to record the length of the shortest path we have found to the city in that row's "Destination" column. The "First step" column will record the first step of this shortest path. As a result, when we are all done, the first two columns of the table will contain all of the information needed to form our forwarding table.

We will record our steps in this table as follows. Once we can identify the Nth closest city to our starting point, we will cross out the "Route length" information for this city. As a result, we will always be able to identify the cities for which we have already identified the best routes. They will be the ones in which the route length information has been crossed out.

Next, we will explore paths that can be formed by taking a single step from this Nth city by considering each of the neighbors listed in the Nth city's row. As we do this, we will update the neighbor's "First step" and "Route length" columns when we find new short routes and, to keep track of the work we have completed, we will cross out the "Neighbors" entries used. For example,

Destination	First step	Route length	Neighbors				
Austin			San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	0:00	Fort Worth, 9:50	San Antonio, 8:45			
Fort Worth			Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio			Austin, 1:09	El Paso, 8:45			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.11: Initial contents of table used to illustrate Dijkstra's algorithm

Destination	First step	Route length	Neighbors				
Austin			San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	9:50	Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio	San Antonio	8:45	Austin, 1:09	El Paso, 8:45			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.12: Table updated to reflect direct connections from El Paso

the first step described above was to determine which cities could be reached from El Paso in a single step. After doing this work, our table would be updated as shown in Figure 8.12.

After each such update, we will then scan the "Route Length" column to find the shortest path length. The "Destination" city in the row which currently has the shortest route length can be identified as the N+1th closest city. For example, based on the information in Figure 8.12 we can see that San Antonio is the closest city to El Paso (as we already knew from our earlier discussion). We will record this by crossing out the "Route length" for San Antonio so that its route length will not be identified as the smallest in future steps. The resulting state of the table is shown in Figure 8.13.

Destination	First step	Route length	Neighbors				
Austin			San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	9:50	Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio	San Antonio	~~8:45~~	Austin, 1:09	El Paso, 8:45			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.13: Table updated to reflect identification of San Antonio as closest to El Paso

Next, having identified San Antonio as the closest city, we have to consider all routes that included San Antonio as an intermediate step. In the tabular form, this corresponds to adding the shortest route length for San Antonio to the lengths of the connections to each of its "Neighbors". As we do this, we will cross out the neighbor information used and update the "First step" and "Route length" column entries for any neighbors for which this total represents the shortest route found.

The first neighbor listed for San Antonio is Austin. At this point, we know of no other path to Austin, so this must be the shortest path we have found to Austin. Accordingly, we will enter the sum of the time to reach San Antonio and the time to travel along the link between San Antonio and Austin, 9 hours and 54 minutes, in Austin's "Route Length" cell. We will also record that San Antonio is the first step on this route. Then, we will cross out the neighbor information for Austin. As a result, the updated table will look like Figure 8.14.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	9:54	San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	9:50	Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	El Paso, 8:45			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.14: Table updated to reflect path from San Antonio to Austin

Next, we will examine the entry for San Antonio's other neighbor, El Paso. Obviously, we can ignore this entry since El Paso is our starting point. We do not, however, have to treat this entry as a special case. Instead, we can go ahead and compute the time that would be required to get from El Paso all the way back to El Paso by going through San Antonio. That is, we can add 8 hours and 45 minutes to itself. The result will be greater than the route length already recorded for El Paso, 0:00. Accordingly, we will not update any entries in El Paso's row. Instead, we will simply cross out the neighbor information for El Paso in San Antonio's row to indicate that this information has been considered by the algorithm. The state of the table after this is done is shown in Figure 8.15. This table accounts for all of the direct links included in Figure 8.10.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	9:54	San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	9:50	Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.15: Table updated to reflect paths to both of San Antonio's neighbors

We now again scan the "Route length" column for the shortest path length. Fort Worth, with a path length of 9:50 is identified as the next closest city. We indicate that we now know the best route to Fort Worth by crossing out its route length as shown in Figure 8.16.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	9:54	San Antonio, 1:09	Waco, 1:22			
Dallas			Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	~~9:50~~	Waco, 1:32	Dallas, 30	El Paso, 9:50		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~			
Waco			Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.16: Table updated to reflect identification of Fort Worth as 2nd closest to El Paso

Next, we look at the list of neighbors that appear in Fort Worth's row. We add 9:50 to each of these entries to determine the total time required to reach each of Fort Worth's neighbors by first traveling to Fort Worth. We discover paths to Waco and Dallas for the first time, so we update the rows for Waco and Dallas as shown in Figure 8.17. Intuitively, the data collected in this table reflects all of the roads shown in the map in Figure 8.18.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	9:54	San Antonio, 1:09	Waco, 1:22			
Dallas	Fort Worth	10:20	Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	~~9:50~~	~~Waco, 1:32~~	~~Dallas, 30~~	~~El Paso, 9:50~~		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~			
Waco	Fort Worth	11:22	Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.17: Table updated to reflect paths to Fort Worth's neighbors

Looking at the route length entries in Figure 8.17, it is now clear that Austin is the 3rd closest to El Paso. We will therefore cross out its route length to mark this fact as shown in Figure 8.19.

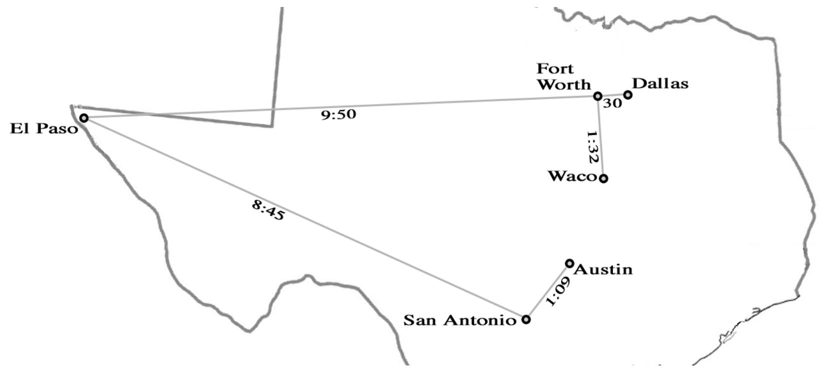

Figure 8.18: Direct connections from El Paso, San Antonio, and Fort Worth

Destination	First step	Route length	Neighbors				
Austin	San Antonio	~~9:54~~	San Antonio, 1:09	Waco, 1:22			
Dallas	Fort Worth	10:20	Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	~~9:50~~	~~Waco, 1:32~~	~~Dallas, 30~~	~~El Paso, 9:50~~		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~			
Waco	Fort Worth	11:22	Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.19: Table updated to reflect identification of Austin as 3rd closest to El Paso

Next, we will add 9:54, the length of the shortest path to Austin to 1:09 and 1:22, the lengths of the neighbors listed in its row to see if there are any routes using Austin as their last step that provide faster ways to reach any cities. The path to Waco through Austin takes 11 hours and 16 minutes. We have already found a path to Waco, but this path through Austin is slightly faster than the 11 hours and 22 minutes required by the earlier route. Therefore, we will replace the information about the shortest path to Waco as shown in Figure 8.20.

Note that in addition to replacing the route length 11:22 with 11:16, we also replace the "First step" with the first step to Austin, San Antonio. If the best path to Waco turns out to be through Austin, then it must start with the same first step as the first step to Austin. Figure 8.21 shows the collection of roads that correspond to the information in our table at this point.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	~~9:54~~	~~San Antonio, 1:09~~	~~Waco, 1:22~~			
Dallas	Fort Worth	10:20	Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~			
Fort Worth	Fort Worth	~~9:50~~	~~Waco, 1:32~~	~~Dallas, 30~~	~~El Paso, 9:50~~		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~			
Waco	~~Fort Worth~~ San Antonio	~~11:22~~ 11:16	Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.20: Table updated to reflect paths to Austin's neighbors

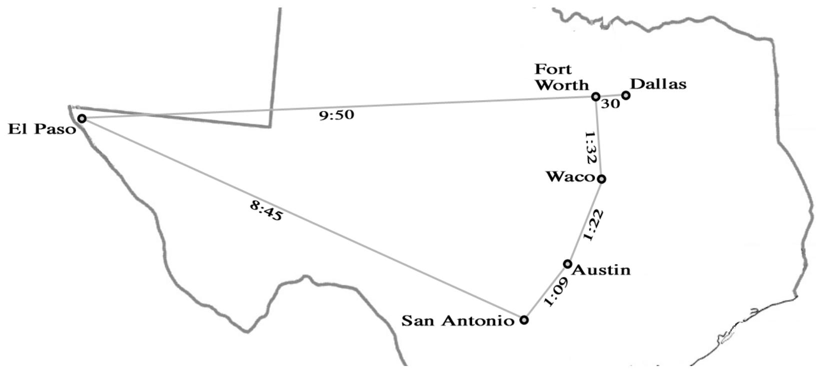

Figure 8.21: Direct connections from El Paso, San Antonio, Fort Worth, and Austin

Even with its updated route length, Waco still takes longer to reach than Dallas according to the numbers remaining in our "Route length" column. Accordingly, our next step is to cross out the route length for Dallas as shown in Figure 8.22.

We then have to add the time required to reach Dallas, 10 hours and 20 minutes to the travel time to its neighbors. The only neighbor of Dallas for which we do not already know the shortest route is Waco, so this is the only chance we have of finding an interesting route in this step. The total time to reach Waco through Dallas is 11 hours and 47 minutes. This is greater than the 11 hours and 16 minute path found in the preceding step. Therefore, we will not update Waco's route information in this step. Instead, all that happens is we get to cross out all of the information for Dallas' neighbors as shown in Figure 8.23.

This table now reflects all of the roads in Figure 8.8. We can identify Waco as the city that is farthest from El Paso and the algorithm is complete. More importantly, if we now discard all but the first two columns of the table we have used (and remove crossed out information), we are left with a complete forwarding table for El Paso, as shown in Figure 8.24.

Destination	First step	Route length	Neighbors				
Austin	San Antonio	~~9:54~~	~~San Antonio, 1:09~~	~~Waco, 1:22~~	·		
Dallas	Fort Worth	~~10:20~~	Waco, 1:27	Fort Worth, 30			
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~	·		
Fort Worth	Fort Worth	~~9:50~~	~~Waco, 1:32~~	~~Dallas, 30~~	~~El Paso, 9:50~~		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~	·		
Waco	~~Fort Worth~~ San Antonio	~~11:22~~ 11:16	Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.22: Table updated to reflect identification of Dallas as 4th closest to El Paso

Destination	First step	Route length	Neighbors				
Austin	San Antonio	~~9:54~~	~~San Antonio, 1:09~~	~~Waco, 1:22~~	·		
Dallas	Fort Worth	~~10:20~~	~~Waco, 1:27~~	~~Fort Worth, 30~~	·		
El Paso	—	~~0:00~~	~~Fort Worth, 9:50~~	~~San Antonio, 8:45~~	·		
Fort Worth	Fort Worth	~~9:50~~	~~Waco, 1:32~~	~~Dallas, 30~~	~~El Paso, 9:50~~		
San Antonio	San Antonio	~~8:45~~	~~Austin, 1:09~~	~~El Paso, 8:45~~	·		
Waco	~~Fort Worth~~ San Antonio	~~11:22~~ 11:16	Austin, 1:32	Fort Worth, 1:32	Dallas, 1:27		

Figure 8.23: Table updated to reflect paths to Dallas' neighbors

The tabular form used in Figure 8.23 and the preceding steps makes it easy to describe the process of applying this algorithm. We start by filling in a table with all of the city names and routing information while leaving the "Route length" and "First step" columns blank. The first step column for our starting point should be left empty and its route length entry should be filled with a crossed-out 0.

The cities directly connected to our starting point must be handled specially. For each such city, we write its own name in its first step column and the length of the direct route from our starting point to the neighbor in its route length entry. We are then ready to begin repeating the following two steps until the table is complete.

Destination	First step
Austin	San Antonio
Dallas	Fort Worth
El Paso	—
Fort Worth	Fort Worth
San Antonio	San Antonio
Waco	San Antonio

Figure 8.24: Forwarding table for traffic passing through El Paso

1. Find the next closest city by looking for the smallest value in the "Route length" column that has not already been crossed out. Call the city listed as the "Destination" in this row the "current destination". Cross out the "Route length" for the current destination to indicate that it has been processed.

2. Process each of the neighbors of the "current destination" identified in step 1 as follows:

   (a) add the route length for the current destination to the time required to travel to the neighbor,

   (b) compare the total to the value recorded as the route length for the neighbor, and

   (c) if the neighbor's current route length is greater than the total computed in step 2(a) or the neighbor's route length entry is still blank, then record the total in the neighbor's route length column and copy the city name in the current destination's first step column to the neighbor's first step column.

While the algorithm is known as the Shortest Path First algorithm, it might be better called the Shortest Path Next algorithm. The key to its operation is not just that it finds the shortest path. Instead, with each iteration of "Step 1" it finds the destination reachable with a path that is the next shortest compared to the previous iteration.

8.3 Routing Information Changes

Dijkstra's algorithm depends upon the availability of a simple but precise description of the network involved. The only information that must be included in the description is an estimate of the travel time between every pair of routers or switches directly connected by the network.

The task of estimating travel time in a data network is significantly different from estimating highway travel times. As we mentioned earlier, in most cases the dominant factors determining highway travel times are the distance to be traveled and the speed limit. Other factors certainly matter. For a more accurate estimate, congestion and road work should be taken into account. In most cases, however, these are secondary.

Because electronic and optical signals travel at close to the speed of light, the component of a network message's travel time that can be computed by dividing the distance to be traveled by the speed at which the signal travels is typically quite small. Other factors, particularly the network equivalent of congestion, are much more significant.

The nature of congestion on a switched computer network is somewhat different from that of highway congestion. Cars on a highway gradually slow down as the traffic gets heavier and heavier. A packet cannot slow down as it is traveling through a cable connecting two computers. The signals involved always move at the speed of light once they have been transmitted. A packet may, however, be delayed because a switch is unable to begin its transmission until other packets that arrived earlier have been sent. In effect, all congestion in a data network occurs on the "entrance ramps" rather than the highway.

Consider the simple network shown in Figure 8.25. Obviously, the best (and only) route from A or B to D goes through the computer C. C must act as a switch for A and D by receiving and then retransmitting any messages they attempt to send to D.

If all the interconnections in this network transmit data at the same rate, a backlog can easily develop. Suppose that both A and B start sending messages to D at full speed. C will forward

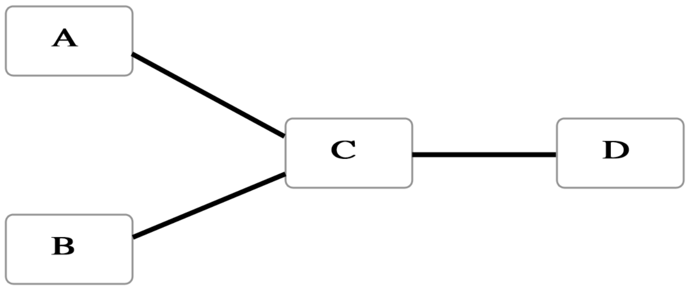

Figure 8.25: A simple four computer network

these messages to C as fast as it can, but in the time it takes C to send one message, two new messages requiring retransmission will arrive. A backlog of messages awaiting transmission to D will quickly build up at C. Each message that arrives will have to wait longer than the preceding message before it gets sent to D.

If this goes on for a long time, C will get hopelessly behind and run out of room for storing the messages awaiting retransmission. This is obviously undesirable. As a result, computer network protocols include mechanisms to ensure that such overloads are short-lived. On the other hand, during normal operation a switch like C will frequently deal with temporary overloads by making newly arrive messages wait in its memory until it can catch up on its backlog. The impact of such congestion on the time required for a packet to travel from one switch to another can be very significant.

If no packets are pending when a packet for D arrives at C, then the time required to deliver the new packet to D will involve two factors. First it will take a certain amount of time to transmit the bits on the outgoing cable. For example, if the packet contains 1000 bits and the data rate is 1,000,000 bits per second, it will take 1 millisecond to transmit the packet. Second, the bits will actually have to travel through the cable. Signals in network cables travel at roughly 200,000,000 meters per second. Therefore, if the cable from C to D is 100 kilometers or 100,000 meters long, it will take the signal about .5 milliseconds to arrive. The total delivery time will therefore be 1.5 milliseconds.

If the same packet for D arrived at C when 100 packets of the same size were backlogged, then the time required for the new packet to get from C to D would be nearly 100 times greater than if it had arrived when there were no pending packets. It would first have to wait for all of the other packets to be transmitted. This would take 100 milliseconds. Then it would take 1.5 milliseconds for the new packet to be transmitted and to travel through the cable. The total time to travel from C to D would be 101.5 milliseconds.

As a result, if we ignore delays caused by congestion while computing routes for a data network, the paths packets follow may be very inefficient. To obtain efficient routes when using Dijkstra's algorithm, we must use a description of the network that includes congestion in its estimates of the time required to travel between switches. Worse yet, estimates of travel time that accurately account for congestion at one point in time may become very inaccurate in the future when the areas of congestion within the network have changed. This implies that if our switches are to choose good routes, they must be regularly provided with new, timely estimates of each link's travel time based on recent information about congestion.

8.4 Link State Updates

We can use the network itself to distribute timing estimate for network links to all of the switches in the network.

It is not hard to imagine how this data could be formatted for transmission in network packets. While discussing the information needed to compute routes, we suggested one simple textual representation that includes a line describing each link. In this format, the first few lines of our map of Texas highways would look like:

Albuquerque → Las Cruces = 3:24
Albuquerque → Santa Fe = 56
Albuquerque → Tucumcari = 2:45
Amarillo → Lubbock = 1:10
Amarillo → Oklahoma City = 3:59
Amarillo → Tucumcari = 1:50
Austin → San Antonio = 1:09
Austin → Waco = 1:22
...

Similar data describing network switch interconnections could easily be encoded in binary for transmission.

In practice, it is best to use separate messages for information describing links that originate at distinct switches. That is, if the cities in our map really did correspond to network switches, we would use one message of the form

Albuquerque → Las Cruces = 3:24
Albuquerque → Santa Fe = 56
Albuquerque → Tucumcari = 2:45

to distribute information about the links from Albuquerque, a separate message of the form

Amarillo → Lubbock = 1:10
Amarillo → Oklahoma City = 3:59
Amarillo → Tucumcari = 1:50

for links from Amarillo, and so on.

To illustrate how such messages are created and distributed, we will use the hypothetical computer network is shown in Figure 8.26. It should look familiar. This computer network happens to have the same topology as the highway system example we have been considering. To make it appear more like a computer network than a road system, we have named the computers using single letters from the alphabet.

Consider the "view" from computer "C" in this network. Computer C will regularly forward message using its links to computers "A", "H", and "I". As it does this, computer C can keep statistics on the average delay caused by the backlog awaiting transmission on each of these links. This will give it enough information to periodically compute accurate travel time estimates for each link. Using this information, it might compose a message of the form:

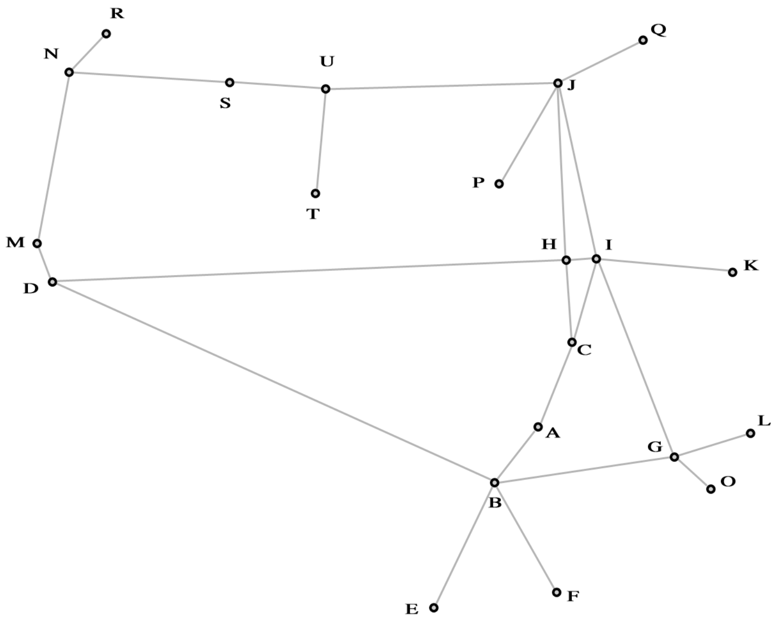

Figure 8.26: A sample network

C → H = 23
C → I = 45
C → A = 35

describing the current travel time estimate for each of its links. Then, it could send copies of this message to every other machine in the network. Such a message is called a *link state update.* If every machine in the network behaved in the same way, then each machine would periodically receive a message from every other machine describing the links connected to the sender. Together, a set of these messages provides the necessary complete description of the network.

8.5 Flooding

The process of delivering link state update methods is not based on the forwarding tables they are used to construct. If it did, we would have a bit of a “Which came first, the chicken or the egg?” problem. We would need to deliver the link state updates to build the forwarding tables, but we would need the forwarding tables to deliver the link state updates.

Even if we could use the forwarding tables to deliver link state updates, it would not be an efficient approach. A forwarding table enables a switch to determine the link along which a packet should be forwarded to reach its destination. This works fine as long as a packet has a single destination, but each link state update has to be delivered to all switches. These messages must in some sense be broadcast throughout the network. Unlike Ethernets, however, routing in a switched network based on forwarding tables does not really support broadcasting.

There is a technique for efficiently broadcasting a message in a switched network called *flooding.*

As the name suggests, the basic idea is to cover the entire network with many copies of a single message rather than sending separate update messages to each destination.

When a packet that is being broadcast using flooding arrives at a switch, the switch sends a copy of the packet out on every link to which it is connected except for the link from which the packet was received. For this to work, switches need to be able to distinguish new packets from old packets or a true flood would quickly result.

Recall that in our example network, routers C, H and I form a triangle. If C receives a routing update from the other machine to which it is connected (A), it would flood the packet by sending copies to H and I. H and I would then flood the packet by sending copies out on all their links except their links to C. Accordingly, H would send a copy to I and I would send a copy to H. These copies would cause H and I to send redundant copies of the update message to C. C would therefore receive two more copies and forward one back to H and the other to I, restarting the whole process. This would go on indefinitely. In addition at each iteration, copies would be sent to the other neighbors of C, H, and I, filling the whole network with unnecessary messages.

Such unnecessary messages can be avoided by having the computer that creates a new routing update message include the time at which the message was created within the message itself. Each switch in the network will keep track of the time contained within the last message received from every other switch. When a routing update arrives, each machine will compare the time contained in the message to its own record of the time found in the last message received from the source of the update. If the time found within the message is earlier or the same as the record of the time in the last message from the same switch, then the update message will be ignored. Otherwise the information in the update message will be recorded by the receiving switch and then the update will be flooded as described above.

Figures 8.27 through 8.31 show how this process might be used to distribute a routing update from A to all the switches in our example network. To begin, A would send copies of its latest estimates of the travel time on its links to its neighbors B and C. The arrows leaving A in Figure 8.27 represent these messages.

Both A and B would use the information included in this message to update their routing information. They would also forward copies of the message to their neighbors, D, E, F, G, H and I. Figure 8.28 show the messages sent by B and C to their neighbors. These message are represented by the black arrows in the diagram. Gray arrows are included to show the original messages from A that were received by B and C.

Next, the recipients of the messages from B and C would forward copies to their neighbors. The messages that would be sent at this stage are represented by the black arrows in Figure 8.29.

At this point, interesting things begin to occur. Switches H and I received copies of A's routing update message during the preceding stage. They now send copies to all their neighbors, including one another. When they each receive these redundant copies, they will each examine the time A stored in the message and compare it to the last copy of a message from A which they received. The times will be the same, so H and I will ignore the duplicates.

It is worth observing that under certain conditions, the set of messages sent might be different. For example, C might flood the message from A to H quickly, but then delay sending a copy to I because it became busy with some other task. If this delay is long, H might actually forward a copy to I before C sent its copy to I. In this case, both I and C would end up sending copies to one another.

To avoid accounting for all such possibilities, we have assumed that whenever a switch floods

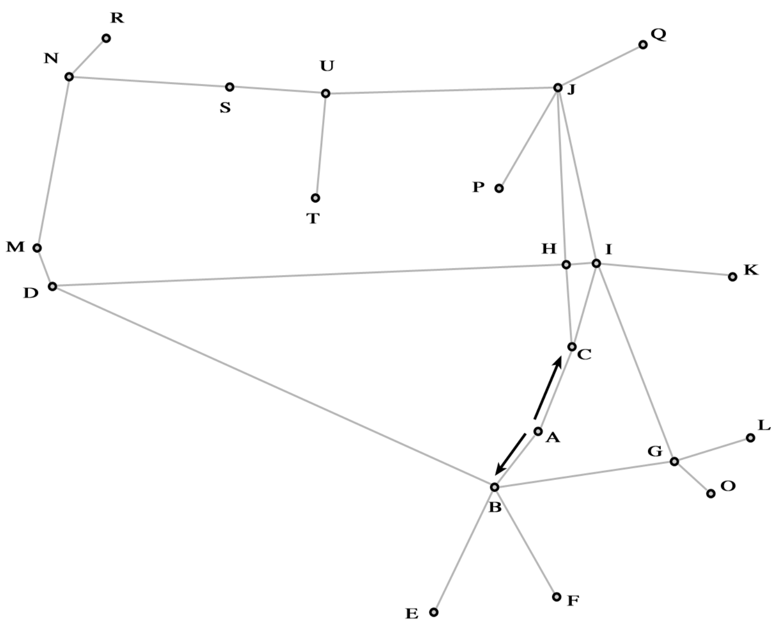

Figure 8.27: Switch A sends a link state update to its immediate neighbors

a messages, all the copies it sends get delivered to all its neighbors at the same time. Under this assumption, H and I will receive copies from C simultaneously and then send copies to one another as described above.

Similar steps will occur between D and H and I and G. Another interesting situation will occur at J. Both H and I will send copies of the message from A to J at this point. Whichever copy arrives first will "win" in this situation. J will record the information found in the first copy and forward the first copy to its neighbors. After checking the time recorded for the last update received from A, it will decide to ignore the second copy it receives.

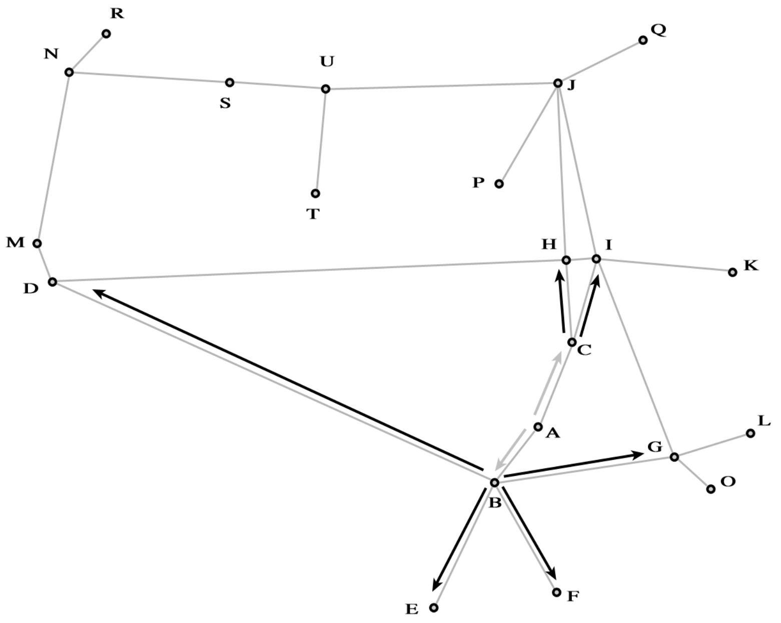

Figure 8.28: Switches B and C flooding a link state update from A

Since D, H, I and G will have ignored duplicate copies of A's message, they will not send copies to any of their neighbors in the next round. M and J, however, will forward copies to N, P, U and Q. These messages are represented by the black arrows in Figure 8.30.

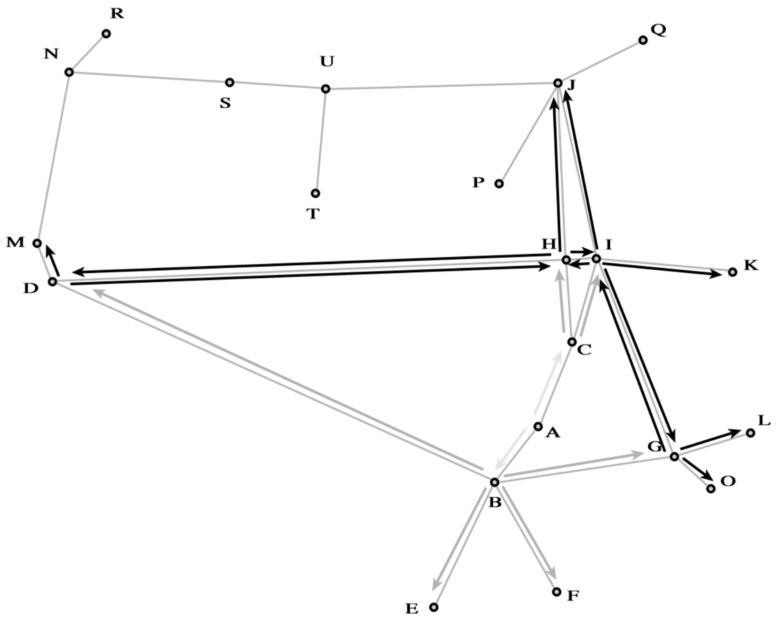

Figure 8.29: Switches D, G, H, and I flooding a link state update from A

U and N then send messages to R, S and T as shown in Figure 8.31.

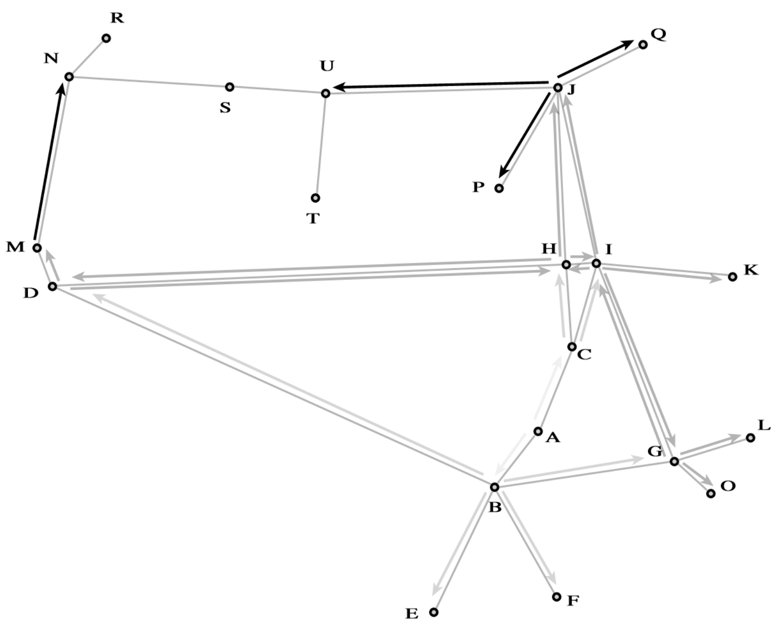

Figure 8.30: Switches M and J flooding a link state update from A

Finally, S sends a message to N which will ignore the message since it has already received a message with the same creation time from A (the copy sent to it by M).

This process is certainly effective. The message from A is delivered to every machine in the network without assuming that the other machines already have accurate routing table and without requiring that A even know how many machines are in the network. All that is assumed is that each switch knows how many links connect it to other machines.

Flooding, together with the algorithm for finding shortest routes provide a way to implement automatic routing in a circuit switched network. As long as any machine is connected, it should continuously collect statistics on the delays experienced by the packets it transmits and periodically send an updated description of the travel time estimates on its links to all other switches using flooding. This process obviously contributes to the amount of data sent through the network and may increase congestion. Flooding routing updates frequently will ensure that other packets are sent on the best routes, but may make all routes more congested. Flooding routing updates infrequently will reduce the accuracy of the routes used for other packets while reducing the congestion caused by routing updates. A balance must be struck.

When a switch receives a routing update, it can use Dijkstra's algorithm to construct a new set of best routes based on the revised estimates of link behavior it has received. It will then use the routes it computes to handle the forwarding of all normal packets until more routing updates arrive.

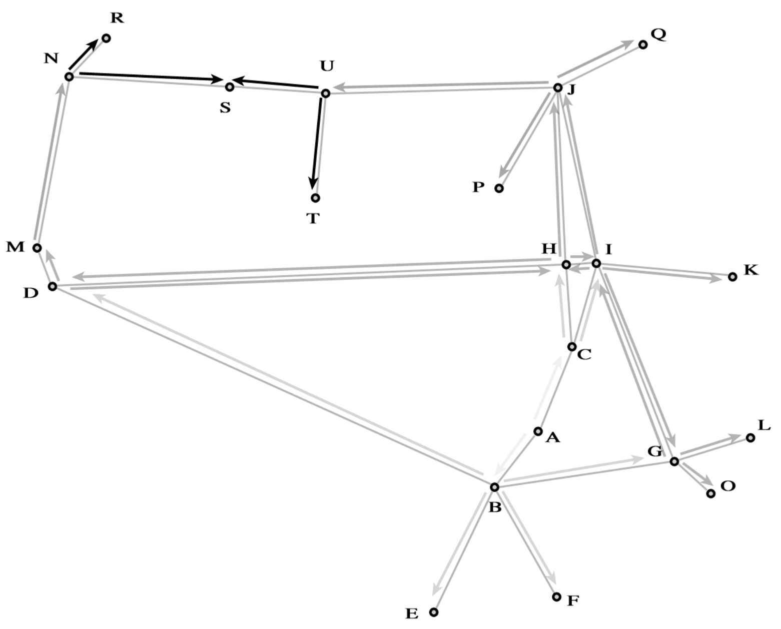

Figure 8.31: Switches U and N flooding a link state update from A

8.6 Summary

In this chapter, we have described the components of a complete system for automatic routing of packets in a switched network. We showed how we could encode a description of the information necessary to find good routes in a form that could be encoded in binary, transmitted through the network and stored on individual switches. We showed how this information could be processed using Dijkstra's algorithm to create forwarding tables that would enable switches to quickly dispatch arriving packets along good paths to their destinations.

All of the mechanisms we described are implemented as part of the routing scheme actually used in the Internet. Internet routing, however, is more complicated than the scheme presented here. In addition to ensuring efficient delivery, routing in the Internet must respect many other issues including non-technical issues dictated by legal contracts between service providers, users, and other service providers.

In practice, the techniques we have discussed are often used within components of the Internet controlled by a single service provider. Such components of the network are called *autonomous systems*. As mentioned earlier, Dijkstra's algorithm is a component of an Internet protocol name OSPF (for Open Shortest Path First), which is used within autonomous systems. For routing between autonomous systems, other protocols and techniques that are beyond the scope of this chapter are employed.