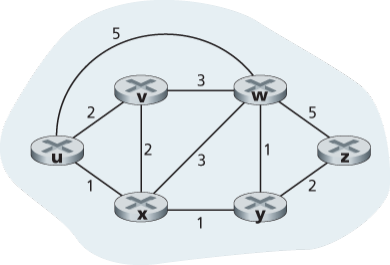
5.2 Routing Algorithms

In this section we’ll study **routing** **algorithms**, whose goal is to determine good paths (equivalently,

routes), from senders to receivers, through the network of routers. Typically, a “good” path is one that has the least cost. We’ll see that in practice, however, real-world concerns such as policy issues (for example, a rule such as “router *x*, belonging to organization *Y*, should not forward any packets originating from the network owned by organization *Z*”) also come into play. We note that whether the network control plane adopts a per-router control approach or a logically centralized approach, there must always be a well-

defined sequence of routers that a packet will cross in traveling from sending to receiving host. Thus, the routing algorithms that compute these paths are of fundamental importance, and another candidate for our top-10 list of fundamentally important networking concepts.

A graph is used to formulate routing problems. Recall that a **graph** G=(N, E) is a set *N* of nodes and a collection *E* of edges, where each edge is a pair of nodes from *N*. In the context of network-layer routing, the nodes in the graph represent



**Figure** **5.3** **Abstract** **graph** **model** **of** **a** **computer** **network**

routers—the points at which packet-forwarding decisions are made—and the edges connecting these

nodes represent the physical links between these routers. Such a graph abstraction of a computer network is shown in [**Figure5.3**](#bookmark1). To view some graphs representing real network maps, see **[Dodge** **2016**,

**Cheswick** **2000]**; for a discussion of how well different graph-based models model the Internet, see **[Zegura** **1997**, **Faloutsos** **1999**, **Li** **2004]**.

As shown in [**Figure** **5.3**](#bookmark1), an edge also has a value representing its cost. Typically, an edge’s cost may

reflect the physical length of the corresponding link (for example, a transoceanic link might have a higher

cost than a short-haul terrestrial link), the link speed, or the monetary cost associated with a link. For our purposes, we’ll simply take the edge costs as a given and won’t worry about how they are determined. For any edge (*x*, *y*) in *E*, we denote *c*(*x*, *y*) as the cost of the edge between nodes *x* and *y.* If the pair (*x*, *y*)

does not belong to *E*, we setc(x, y)= ∞ . Also, we’ll only consider undirected graphs (i.e., graphs whose

edges do not have a direction) in our discussion here, so that edge (*x*, *y*) is the same as edge (*y*, *x*) and

that c(x, y)=c(y, x); however, the algorithms we’ll study can be easily extended to the case of directed links with a different cost in each direction. Also, a node *y* is said to be a **neighbor** of node *x* if (*x*, *y*) belongs to *E*.

Given that costs are assigned to the various edges in the graph abstraction, a natural goal of a routing

algorithm is to identify the least costly paths between sources and destinations. To make this problem

more precise, recall that a **path** in a graph G=(N, E) is a sequence of nodes (x1,x2, … ,xp) such that each of the pairs (x1,x2),(x2,x3), … ,(xp-1,xp) are edges in *E*. The cost of a path (x1,x2, … , xp) is simply the sum of all the edge costs along the path, that is, c(x1,x2)+c(x2,x3)+…+c(xp-1,xp). Given any two nodes *x* and *y*, there are typically many paths between the two nodes, with each path having a cost. One or more of

these paths is a **least-cost** **path**. The least-cost problem is therefore clear: Find a path between the

source and destination that has least cost. In [**Figure** **5.3**](#bookmark1), for example, the least-cost path between source node *u* and destination node *w* is (*u*, *x*, *y*, *w*) with a path cost of 3. Note that if all edges in the graph have the same cost, the least-cost path is also the **shortest** **path** (that is, the path with the smallest number of links between the source and the destination).

As a simple exercise, try finding the least-cost path from node *u* to *z* in [**Figure5.3**](#bookmark1) and reflect for a

moment on how you calculated that path. If you are like most people, you found the path from *u* to *z* by

examining [**Figure** **5.3**](#bookmark1), tracing a few routes from *u* to *z*, and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 17 possible paths

between *u* and *z*? Probably not!) Such a calculation is an example of a centralized routing algorithm—the routing algorithm was run in one location, your brain, with complete information about the network.

Broadly, one way in which we can classify routing algorithms is according to whether they are centralized or decentralized.

 A **centralized** **routing** **algorithm** computes the least-cost path between a source and destination

using complete, global knowledge about the network. That is, the algorithm takes the connectivity

between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain

this information before actually performing the calculation. The calculation itself can be run at one site (e.g., a logically centralized controller as in **Figure** **5.2**) or could be replicated in the routing component of each and every router (e.g., as in **Figure** **5.1**). The key distinguishing feature here, however, is that the algorithm has complete information about connectivity and link costs. Algorithms with global state information are often referred to as **link-state** **(LS)** **algorithms**, since the algorithm must be aware of the cost of each link in the network. We’ll study LS algorithms in **Section** **5.2.1**.

. In a **decentralized** **routing** **algorithm**, the calculation of the least-cost path is carried out in an