# Distributed Operating Systems

CHAPTER 16

A *distributed system* is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication networks, such as high-speed buses or telephone lines. In this chapter, we discuss the general structure of distributed systems and the networks that interconnect them. We contrast the main differences in operating-system design between these types of systems and the centralized systems with which we were concerned previously. Detailed discussions are given in Chapters 17 and 18.

**16.1** The advantages are that dedicated hardware devices for routers and gateways are very fast as all their logic is provided in hardware (firmware.) Using a general-purpose computer for a router or gateway means that routing functionality is provided in software—which is not as fast as providing the functionality directly in hardware.

A disadvantage is that routers or gateways as dedicated devices may be more costly than using off-the-shelf components that comprise a modern personal computer.

**16.2** All broadcasts would be propagated to all networks, causing a *lot* of network traffic. If broadcast traffic were limited to important data (and very little of it), then broadcast propagation would save gateways from having to run special software to watch for this data (such as network routing information) and rebroadcast it.

**16.3** Delegating the retransmission decisions to the network layer might be appropriate in many settings. In a congested system, immediate retransmissions might increase the congestion in the system, resulting in more collisions and lower throughput. Instead, the decision of when to retransmit could be left to the upper layers, which could delay the retransmission by a period of time that is proportional to the current congestion in the system. An exponential backoff strategy is the most commonly used strategy to avoid over-congesting a system.

**16.4** A certain network layered protocol may achieve the same functionality of the ISO in fewer layers by using one layer to implement functionality provided in two (or possibly more) layers in the ISO model. Other models may decide there is no need for certain layers in the ISO model. For example, the presentation and session layers are absent in the TCP/IP protocol. Another reason may be that certain layers specified in the ISO model do not apply to a certain implementation. Let's use TCP/IP again as an example where no data link or physical layer is specified by the model. The thinking behind TCP/IP is that the functionality behind the data link and physical layers is not pertinent to TCP/IP—it merely assumes some network connection is provided, whether it be Ethernet, wireless, token ring, etc.

A potential problem with implementing fewer layers is that certain functionality may not be provided by features specified in the omitted layers.

**16.5** Many applications might not require reliable message delivery. For instance, a coded video stream could recover from packet losses by performing interpolations to derive lost data. In fact, in such applications, retransmitted data are of little use since they would arrive much later than the optimal time and not conform to realtime guarantees. For such applications, reliable message delivery at the lowest level is an unnecessary feature and might result in increased message traffic, most of which is useless, thereby resulting in performance degradation. In general, the lowest levels of the networking stack needs to support the minimal amount of functionality required by all applications and leave extra functionality to be implemented at the upper layers.

**16.6** The advantage is that all files are accessed in the same manner. The disadvantage is that the operating system becomes more complex.

**16.7** A token ring is very effective under high sustained load, as no collisions can occur and each slot may be used to carry a message, providing high throughput. A token ring is less effective when the load is light (token processing takes longer than bus access, so any one packet can take longer to reach its destination) or sporadic.

**16.8** For the same operating system, process migration is relatively straightforward, as the state of the process needs to migrate from one processor to another. This involves moving the address space, state of the CPU registers, and open files from the source system to the destination. However, it is important that identical copies of the operating system are running on the different systems to ensure compatibility. If the operating system are the same, but perhaps different versions are running on the separate systems, then migrating processes must be sure to follow programming guidelines that are consistent between the different versions of the operating system.

Java applets provide a nice example of process migration between different operating systems. To hide differences in the underlying system, the migrated process (i.e., a Java applet) runs on a virtual machine rather than a specific operating system. All that is required is for the virtual machine to be running on the system the process migrates to.

**16.9** A fully connected network provides the most reliable topology since if any of the links go down, it is likely there exists another path to route the message. A partially connected network may suffer from the problem that if a specific link goes down, another path to route a message may not exist. Of the partially-connected topologies, various levels of reliability exist. In a tree-structured topology, if any of the links goes down, there is no guarantee that messages may be routed. A ring topology requires two links to fail for this situation to occur. If a link fails in a star network, the node connected to that link becomes disconnected from the remainder of the network. However, if the central node fails, the entire network becomes unusable.

Regarding available bandwidth for concurrent communications, the fully connected network provides the maximum utility followed by partially connected networks. Tree-structured networks, rings, and star networks have a linear number of network links and therefore have limited capability with regard to performing high-bandwidth concurrent communications. Installation costs follow a similar trend, with fully connected networks requiring a huge investment, and trees, rings, and stars requiring the least investment.

Fully connected networks and ring networks enjoy symmetry in the structure and do not suffer from hot spots. Given random communication patterns, the routing responsibilities are balanced across the different nodes. Trees and stars suffer from hotspots: the central node in the star and the nodes in the upper levels of the tree carry much more traffic than the other nodes in the system and therefore suffer from load imbalances in routing responsibilities.

**16.10** Dynamic routing might route different packets through different paths. Consecutive packets might therefore incur different latencies and there could be substantial jitter in the received packets. Also, many protocols, such as TCP, that assume that reordered packets imply dropped packets, would have to be modified to take into account that reordering is a natural phenomenon in the system and does not imply packet losses. Realtime applications such as audio and video transmissions might benefit more from virtual routing since it minimizes jitter and packet reorderings.

**16.11** Despite the connectionless nature of UDP, it is not a serious alternative to TCP for the HTTP. The problem with UDP is that it is unreliable, documents delivered via the web must be delivered reliably. (This is easy to illustrate—a single packet missing from an image downloaded from the web makes the image unreadable.) One possibility is to modify how TCP connections are used. Rather than setting up—and breaking down—a TCP connection for everyweb resource, allow *persistent* connections where a single TCP connection stays open and is used to deliver multiple web resources.

**16.12** Circuit switching guarantees that the network resources required for a transfer are reserved before the transmission takes place. This ensures that packets will not be dropped and their delivery would satisfy quality of service requirements. The disadvantage of circuit switching is that it requires a round-trip message to set-up the reservations and it also might overprovision resources, thereby resulting in suboptimal use of the resources. Circuit switching is a viable strategy for applications that have constant demands regarding network resources and would require the resources for long periods of time, thereby amortizing the initial overheads.

**16.13** Name servers require their own protocol, so they add complication to the system. Also, if a name server is down, host information may become unavailable. Backup name servers are required to avoid this problem. Caches can be used to store frequently requested host information to cut down on network traffic.

**16.14** An ARP translates general-purpose addresses into hardware interface numbers so the interface can know which packets are for it. Software need not get involved. It is more efficient than passing each packet to the higher layers. Yes, for the same reason.

**16.15** Process migration is an extreme form of computation migration. In computation migration, an RPC might be sent to a remote processor in order to execute a computation that could be more efficiently executed on the remote node. In process migration, the entire process is transported to the remote node, where the process continues its execution. Since process migration is an extension of computation migration, more issues need to be considered for implementing process migration. In particular, it is always challenging to migrate all of the necessary state to execute the process, and it is sometimes difficult to transport state regarding open files and open devices. Such a high degree of transparency and completeness is not required for computation migration, where it is clear to the programmer that only a certain section of the code is to be executed remotely. programmer.

**16.16** As of December 2004, the corresponding IP addresses are

a. www.wiley.com—`208.215.179.146`

b. www.cs.yale.edu—`128.36.229.30`

c. www.javasoft.com—`192.18.97.39`

d. www.westminstercollege.edu—`146.86.1.2`

e. www.ietf.org—`132.151.6.21`

**16.17** Three common failures in a distributed system include: (1) network link failure, (2) host failure, (3) storage medium failure. Both (2) and (3) are failures that could also occur in a centralized system, whereas a network link failure can occur only in a networked-distributed system.

**16.18** Faster systems may be able to send more packets in a shorter time. The network would then have more packets traveling on it, resulting in more collisions, and therefore less throughput relative to the number of packets being sent. More networks can be used, with fewer systems per network, to reduce the number of collisions.

**16.19** Hierarchical structures are easier to maintain since any changes in the identity of name servers require an update only at the next-level name server in the hierarchy. Changes are therefore localized. The downside of this approach, however, is that the name servers at the top level of the hierarchy are likely to suffer from high loads. This problem can be alleviated by replicating the services of the top-level name servers.

**16.20** One technique would be for B to periodically send a *I-am-up* message to A indicating it is still alive. If A does not receive an *I-am-up* message, it can assume either B—or the network link—is down. Note that an *I-am-up* message does not allow A to distinguish between each type of failure. One technique that allows A better to determine if the network is down is to send an *Are-you-up* message to B using an alternate route. If it receives a reply, it can determine that indeed the network link is down and that B is up.

If we assume that A knows B is up and is reachable (via the *I-am-up* mechanism) and that A has some value *N* that indicates a normal response time, A could monitor the response time from B and compare values to *N*, allowing A to determine if B is overloaded or not.

The implications of both of these techniques are that A could choose another host—say C—in the system if B is either down, unreachable, or overloaded.