17 Quality of Service Routing

The more precisely the position is determined, the less precisely the momentum is known in this instant, and vice versa.

Werner Heisenberg

Reading Guideline

Quality of Service routing includes aspects from both the circuit-switched world and the packet-switched world. Thus, some knowledge of circuit switching is helpful. In addition, understanding the material on dynamic call routing in the telephone network (Chapter 10) and its traffic engineering (Chapter 11), along with the link state routing protocol (Chapter 3) and the shortest and widest path routing algorithms (Chapter 2), is helpful in getting the most out of this chapter.

Quality of Service (QoS) is an important issue in any communication network; typically, this can be viewed from the perception of service quality. Eventually any service perception needs to be mapped to network routing, especially since QoS guarantee is required for a particular service class.

In this chapter, we discuss what QoS routing means and how different routing algorithms covered in this book may be extended to fit the QoS routing framework. Finally, we present a representative set of numerical studies with which we can understand the implications of different routing schemes and roles played by different network controls.

17.1 Background

We start with a brief background on QoS and QoS routing.

Quality of Service

To discuss *Quality of Service routing*, we first need to understand what *Quality of Service* means. Consider a generic request arrival to a network; if this request has certain resource requirements that it explicitly announces to the network at the time of arrival, then QoS refers to the network's ability to meet the resource guarantees for this request.

To understand QoS, we will first consider a network link; no routing is considered at this point. Assume that a request arrives at this network link for a 1-Mbps constant data rate. If the network link had bandwidth available that is more than 1 Mbps, then it can certainly accommodate this request. Thus, the arriving request received the specified QoS. Implicit in this is that the QoS will be continually met as long as this request is active; in other words, for the *duration* of the request, the QoS is met.

Suppose that the network link at the instant of the request arrival has less available bandwidth than the requested bandwidth. In this case, the request cannot be served. When there are many arriving requests requiring resource guarantees and the network link cannot accommodate them, another aspect related to QoS emerges. This aspect of QoS considers that arriving requests usually receive the service guarantee requested with an acceptable probability of not being turned away; in other words, blocking should not be high. That is, the blocking probability of arriving requests is another important consideration in regard to QoS. When we consider from this viewpoint, it is easy to see that traffic engineering and capacity expansion also play crucial parts in regard to QoS since if the network is not engineered with a reasonable capacity level, the likelihood of a request facing blocking would be high. Thus, blocking probability is an important factor in the perception of QoS and is traditionally known as *grade of service* (GoS).

In general, the term QoS is used much more broadly than its use in this chapter in the context of QoS routing. For example, "a network meets QoS" can be interpreted as meeting delay requirements through a network, not necessarily for a specific request.

QoS Routing

Consider now a network instead of just a link. Then for an arriving request that requires guaranteed resources, the network would need to decide what resources it has in its *different* links and paths so that the request can be accommodated. Thus, *QoS routing* refers to a network's ability to accommodate a QoS request by determining a *path* through the network that meets the QoS guarantee. Furthermore, an implicit understanding is that the network's performance is also optimized. In this sense, QoS routing cannot be completely decoupled from traffic engineering.

QoS Routing Classification

What are the types of resource guarantees an arriving request might be interested in? Typically, they are bandwidth guarantee, delay bound, delay jitter bound, and acceptable packet loss. We have already described a bandwidth guarantee. *Delay bound* refers to end-to-end delay being bounded. Jitter requires a bit of explanation. In a packet-based network, packets that are generated at equal spacing from one end may not arrive at the destination with the same spacing; this is because of factors such as delay due to scheduling and packet processing at intermediate routers, interaction of many flows, and so on. In real-time interactive applications such as voice or video, the interpacket arrival times for a call are equally spaced when generated, but may arrive at the destination at uneven time spacing; thus, interpacket delay is known as *jitter*. *Packet loss* refers to the probability of a packet being lost along the path from origin to destination.

Consideration of these four factors would, however, depend on whether the network is a circuit-based network or a packet-based network. To discuss this aspect and the critical elements related to QoS routing, we also need to consider time granularity in regard to an arriving request. By considering three time-related factors, arrival frequency, lead time for set up, and the duration of a session/connection, we broadly classify requests into three types as listed in Table 17.1. There are very specific outcomes of these classifications. In Type A, the network technology is either packet-switched or circuit-switched where circuit-switched networks require bandwidth guarantee while packet-switched networks may have one or all of the requirements: bandwidth guarantee, delay bound, jitter bound, and acceptable packet loss. However, Type B is generally circuit-oriented where a permanent or semi-permanent bandwidth guarantee is the primary requirement; there is very little about on-demand switching. Routing for the Type B classification is traditionally referred to as *circuit routing* (for example, see [596, p. 136]); in recent literature, circuit routing is commonly known as *transport network routing* (covered in Chapter 24). Between Type A and Type B, there is another form of routing where some overlap of time granularity is possible. We classify this type that has overlapping regions as Type C; for example, routing for this type of service can be accomplished in MPLS networks and will be discussed later in Chapter 19.

Of these classifications, QoS routing arises for a Type A classification. It is thus helpful to consider a taxonomy for QoS routing to understand the relationship between networking paradigms and QoS factors (see Figure 17.1). In figure, we have included an identifier

TABLE 17.1 Service request type classification.

Type	Average arrival frequency	Lead time for setup	Duration of session
Type A	Subsecond/seconds time frame	A few seconds	Minutes
Type B	Day/week time frame	Weeks	Months to years
Type C	Multiple times a day	Minutes	Minutes to hours

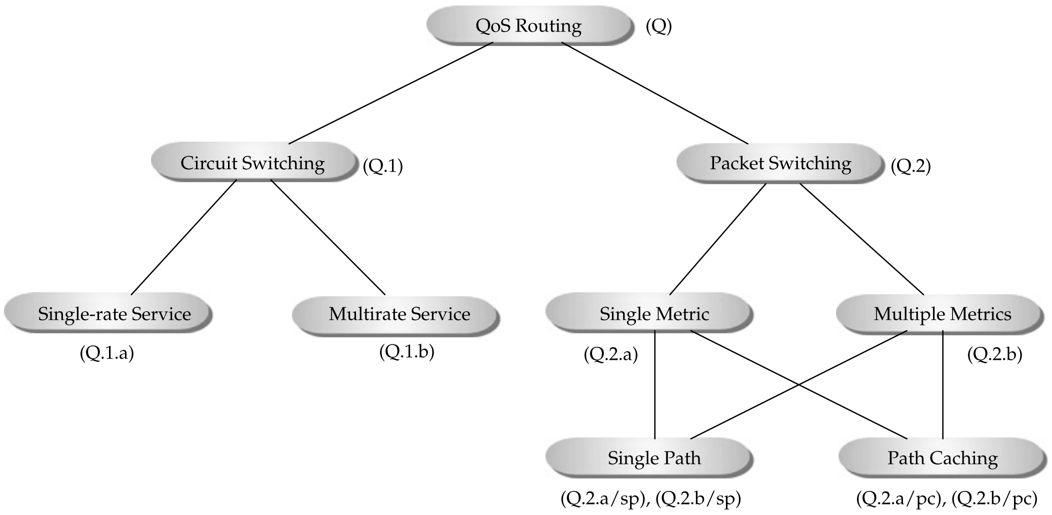

FIGURE 17.1 QoS routing taxonomy.

in parentheses for ease of illustration. First note that classification Q.1.a refers to routing in the current circuit-switched telephone network. You may note that we have already covered hierarchical and dynamic call routing in a telephone network in Chapter 10. An important point to note is that *both* hierarchical and all variations of dynamic call routing fall under Q.1.a in terms of meeting QoS. It may be noted that old hierarchical call routing meets the bandwidth guarantee of a new request if admitted; however, hierarchical call routing is not as flexible as dynamic call routing schemes and requires more bandwidth to provide the same level of service. This then helps in seeing that traffic engineering efficiency is an implicit requirement of QoS routing, a primary reason why dynamic call routing was pursued in the telephone network. A broader point is that QoS routing can be accomplished by different routing *schemes*—the drivers for QoS routing are developing routing schemes that address issues such as performance benefit, cost, routing stability, management, and so on. Thus, in general, dynamic or adaptive routing is preferred over fixed routing. Classification Q.1.a is a very important area in network routing. Besides circuit-switched voice, many problems in optical routing also fall under classification Q.1.a; this will be discussed later in Chapter 25.

Classification Q.1.b is an extension of Q.1.a. Multirate, multiservice circuit-switched QoS routing refers to the case in which there are more than one service classes and an arriving request for each service class has a different bandwidth requirement as opposed to Q.1.a, where all arriving requests have the same bandwidth requirement, for example, the current wired telephone network where per-request bandwidth is 64 Kbps. In case of Q.1.b, service classes are rigid and the bandwidth requirement of a request in each class is an integral multiple of the *base bandwidth rate*. If the base bandwidth rate in the circuit-switched voice network is 64 Kbps, then a switched video service can be defined as, say, 384 Kbps, which is then six times the base bandwidth rate. It may be noted that among the dynamic call routing schemes discussed earlier in Chapter 10, real-time network routing (RTNR), discussed in Section 10.6, has been deployed to handle multiple classes of services.

For the packet-switched branch of QoS routing, there are two aspects to consider: single attribute or multiple attributes. By single attribute, we mean only a single criterion, such as the bandwidth requirement, is used as a metric for a request that is considered for QoS routing. By multiple attributes, we mean that more than one factor, such as bandwidth and delay, is being considered for QoS routing. Note that we do not distinguish here by rates as we have done with Q.1.a and Q.1.b, although theoretically it is possible to discuss single rate and multiple rate. The reason this is grouped together is that packet-switched networks are usually not designed with a single bandwidth rate in mind—any arbitrary bandwidth rate is generally usable due to the packet switching nature. It is, however, indeed possible to deploy a private packet network, for example, a private voice over IP (VoIP) packet network, where all voice calls have the same data rate. Thus, Q.2.a has some aspects of Q.1.b, with the additional flexibility of arbitrary data rate of a request. For classification Q.2.b, multiple criteria are required to handle the decision-making process for an arriving request. This will be discussed in detail later in this chapter.

For both Q.2.a and Q.2.b, there are two possibilities in terms of path consideration: either a single path is considered, or paths are cached for alternate paths consideration. Note that for Q.1.a and Q.1.b, it has been common to consider path caching; in fact, a single path is rarely considered and is not shown in this classification.

Defining a Request and Its Requirement

You may note that so far we have not defined a *request*. Typically, in a circuit-switching context, a request is labeled as a *call*; in a packet-switching context, especially in IP networking, a QoS request is labeled as a *flow*, while terms such as *SIP call* or *VoIP call* are also often used.[1] For a discussion on SIP, refer to Chapter 20. Note that the usage of the term *flow* here is not to be confused with *network flow* or *link flow* described earlier in Chapter 4.

When a call request arrives, there is a call setup phase that can typically perform functions such as route determination, signaling along the path to the destination, and QoS checking before the call is setup; in essence, a call request must always face a call setup time delay before it can be connected—this is also known as postdial delay; certainly, this should be minimized. For the services that require a QoS guarantee, the call setup phase needs to ensure that the QoS guarantee can be provided for the entire duration of the call; otherwise, the call request is denied by the network.

General Observations

It is quite possible that a network may not have the *functionality* to guarantee that it can meet QoS for an arriving request, but yet has the resources to meet the request. An IP network without integrated services functionality falls into this category. For example, a VoIP call can receive QoS in an IP network without the network explicitly having the ability to provide a guarantee at the time of request arrival. This can be possible, for example, if the network is engineered properly, or overprovisioned. In general, overprovisioning is not desirable since, after all, a network does cost real money in terms of capacity cost, switching cost, and so on.

[1]There is yet another terminology in the networking literature: *call flows*. This term refers to the flow or sequence diagram of messages in regard to establishing or tearing down a call over an SS7 network or in a SIP environment or when translation is required at a gateway going from SS7 to SIP, or vice versa.

Finally, much like best-effort traffic services, QoS routing can also have two components: intradomain and interdomain. Most of this chapter is focused on *intradomain* QoS routing. We will briefly discuss interdomain QoS routing at the end.

17.2 QoS Attributes

In the previous section, we mentioned the following factors in terms of attributes: *residual* bandwidth, delay, jitter, and packet loss. Note that any of these attributes are applicable under classification Q.2, while bandwidth is the only one applicable for classification Q.1. We will now discuss how to classify these attributes in terms of metrics.

Suppose that an arriving request has requirements for bandwidth, delay, jitter, and packet loss identified by $\overline{b}$, $\overline{\tau}$, $\overline{\zeta}$, and $\overline{L}$, respectively. The important question is how are measures for these factors accumulated along a path in terms of satisfying the guaranteed requirement of an arriving call? To understand this, we will consider a path that is made up of three links numbered 1, 2, and 3, and current residual bandwidth, delay, jitter, and packet loss measures for link i as b_i, τ_i, ζ_i, and L_i $(i = 1, 2, 3)$, respectively. We can then list the path measures as follows:

Type	Path measure	Requirement
Bandwidth	$\min\{b_1, b_2, b_3\}$	$\geq \overline{b}$
Delay	$\tau_1 + \tau_2 + \tau_3$	$\leq \overline{\tau}$
Jitter	$\zeta_1 + \zeta_2 + \zeta_3$	$\leq \overline{\zeta}$
Packet loss	$1 - (1 - L_1)(1 - L_2)(1 - L_3)$	$\leq \overline{L}$

You can see that the packet loss measure is a nonadditive multiplicative measure; however, it can be looked at from another angle. If L_i $(i = 1, 2, 3)$ is very close to zero, which is typically the case for packet loss, again due to traffic engineering requirements, the expression for path measure can be approximated as follows (see Appendix B.8):

$$1 - (1 - L_1)(1 - L_2)(1 - L_3) \approx L_1 + L_2 + L_3. \tag{17.2.1}$$

Thus, the packet loss measure becomes an additive measure. We can then classify the different attributes into two groups in terms of metric properties:

Additive:	Delay, jitter, packet loss
Nonadditive (concave):	Bandwidth

Broadly, this means that from a routing computation point of view, delay, jitter, and packet loss metrics can be classified under shortest path routing while the bandwidth requirement metric falls under widest path routing. It may be noted that a buffer requirement at routers along a path for an arriving request requiring a QoS guarantee is another possible metric that falls under the nonadditive concave property; however, unlike the rest of the metrics discussed so far, a buffer requirement is checked as the call setup signaling message is propagated along the path chosen, rather than being communicated through a link state advertisement.

To summarize, for classification Q.2, both additive and nonadditive concave metrics are possible, while for classification Q.1 only nonadditive concave is appropriate. In the next section, we will discuss adaptations of shortest path routing and widest path routing for a request requiring a QoS guarantee.

17.3 Adapting Shortest Path and Widest Path Routing: A Basic Framework

Out of different attributes classified into two categories, we will use one metric each from additive and nonadditive (concave) metric properties for our discussion here. Specifically, we will use delay for the additive property and bandwidth requirement for the nonadditive property. We assume here the reader is familiar with shortest path routing and widest path routing described earlier in Chapter 2. You may note that the discussion in this section is applicable only to classification Q.2.

From our discussion in Part I of this book, we know that applicability of a particular routing algorithm for a packet-switched network depends on whether the network is running a distance vector protocol or a link state protocol. While the basic idea of shortest path or widest path routing would work under both these protocol concepts, we will assume here that a link state protocol framework is used since most well-known intradomain routing protocol frameworks are link state based.

17.3.1 Single Attribute

We first consider that requests have a single additive metric requirement in terms of delay attribute. A simple way to adapt the shortest path routing algorithm paradigm here is by using delay as the link cost metric. Suppose a request arrives with the delay requirement no greater than $\overline{\tau}$.

> For an arriving request requiring a guaranteed delay requirement of $\overline{\tau}$, do the following:
> Compute the shortest delay using the shortest path first algorithm (Algorithm 2.4); if the result is less than $\overline{\tau}$, then admit the request; otherwise, deny the request.

Note that the request arrives for a particular destination. Thus, unlike the standard shortest path first (SPF) algorithm, here the shortest path computation must be computed only for the specific destination of a request. Consider again Algorithm 2.4 in Chapter 2; in step 3, once a new node k is identified with the minimum cost path, it can be checked whether this k is the destination of the request; if so, the algorithm can stop. At this point, this delay cost is then compared against the arriving request's delay requirement; if met, the request is accepted, otherwise it is denied.

What if the single metric is in terms of the bandwidth requirement of a request? This scenario is similar to the delay-based scenario. Suppose that an arriving request has a bandwidth requirement of $\overline{b}$. Then, we can use the following rule:

> For an arriving request with a guaranteed bandwidth requirement of $\overline{b}$, do the following:
> Compute the widest path using Algorithm 2.8 for the specific destination; if this value is higher than $\overline{b}$, then admit the request; otherwise, deny the request.

In many instances, it is desirable to obtain the widest path with the least number of hops for the path. Although this is sometimes referred to as the *shortest-widest* path, it is not a good name since shortest does not indicate the context in which this is meant. Thus, we will refer to it as the *least-hop-widest* path. How do we find the widest path with the least number of hops? Consider again Algorithm 2.8; In step 3 of this algorithm, k in $\mathcal{S}'$ with the maximum residual bandwidth is determined. Instead of storing just one k, the list of nodes where the maximum residual bandwidth is attained is determined. If this list happens to have more than one element, then k is chosen so that it is the least number of hops from source node i. In essence, this means that if there are multiple paths with maximum residual bandwidth, choose the one with the least number of hops; if there are still such multiple paths, one is randomly selected. In the same manner, a *least-hop-minimum delay* path can be determined when a delay metric is used.

17.3.2 Multiple Attributes

In this case, consider an arriving request specifying that both the delay as well as the bandwidth requirement must be satisfied. This can be addressed from the point of view of which factor is to be considered the dominant factor: delay or bandwidth; this, however, depends on which is found: a bandwidth feasible path while the delay is minimized, or a delay feasible path while maximizing available bandwidth.

Again, we can adapt the widest path and shortest path routing framework. To determine the minimum delay path that satisfies the bandwidth requirement of a request, we can initialize any link that does not meet the bandwidth requirement temporarily as a link with infinite delay; this method of considering a nonadditive metric requirement with an additive shortest path computation is generally known as *constrained shortest path routing*. Instead, if we were to determine a maximum residual bandwidth, i.e., the widest path while meeting the delay requirement, we can initialize any link that does not meet the delay requirement by temporarily setting the residual link bandwidth to zero; this form can be classified as *constrained widest path routing*. Note that for a constrained shortest path, the constraint is on bandwidth, while for a constrained widest path, the constraint is on delay. For source node i and destination node v, we present both routing algorithms in Algorithm 17.1 and Algorithm 17.2 for completeness. The notations are summarized in Table 17.2. Chapter 2 may also be consulted for comparison.

TABLE 17.2 Notation for QoS routing.

Notation	Remark
i	Source node
v	Destination node
$\mathcal{N}$	List of all nodes
$\mathcal{N}_k$	List of neighboring nodes of k
$\mathcal{S}$	List of nodes considered so far
$\mathcal{S}'$	List of nodes yet to be considered
τ_{ij}	Link delay on link i-j (set to ∞ if the link does exist, or not to be considered)
T_{ij}	Delay from node i to node j
b_{ij}	Residual bandwidth on link i-j (set to 0 if the link does exist, or not to be considered)
B_{ij}	Bandwidth available from node i to node j

ALGORITHM 17.1 **QoS minimum delay path with bandwidth feasibility**

```
$\mathcal{S} = \{i\}$   // permanent list; start with source node $i$
$\mathcal{S}' = \mathcal{N} \setminus \{i\}$   // tentative list (of the rest of the nodes)
for ( $j$ in $\mathcal{S}'$ ) do
  // check if $i$-$j$ directly connected and link has required bandwidth $\overline{b}$
  if ( $\tau_{ij} < \infty$ and $b_{ij} \geq \overline{b}$ ) then
    $T_{ij} = \tau_{ij}$   // note the delay cost
  else
    $\tau_{ij} = \infty; T_{ij} = \infty$   // mark temporarily as unavailable
  endif
endfor
while ($\mathcal{S}'$ is not empty) do   // while tentative list is not empty
  $Ttemp = \infty$   // find minimum-delay neighbor $k$
  for ( $m$ in $\mathcal{S}'$ ) do
    if ( $T_{im} < Ttemp$ ) then
      $Ttemp = T_{im}; k = m$
    endif
  endfor
  if ( $T_{ik} > \overline{\tau}$ ) then   // if minimum delay is higher than delay tolerance
    'No feasible path exists; request denied'
    exit
  endif
  if ( $k == v$ ) then exit   // destination $v$ found, done
  $\mathcal{S} = \mathcal{S} \cup \{k\}$   // add to permanent list
  $\mathcal{S}' = \mathcal{S}' \setminus \{k\}$   // delete from tentative list
  for ( $j$ in $\mathcal{N}_k \cap \mathcal{S}'$ ) do
    if ( $T_{ij} > T_{ik} + \tau_{kj}$ and $b_{kj} > \overline{b}$ ) then   // if delay is less via $k$
      $T_{ij} = T_{ik} + \tau_{kj}$
    endif
  endfor
endwhile
if ( $T_{iv} \leq \overline{\tau}$ ) then   // final check, if the path meets delay requirement
  'Request accepted'
else
  'No feasible path exists; Request denied'
endif
```

17.3.3 Additional Consideration

We next consider a general question: can we provide QoS routing in a packet environment where buffer guarantee at routers is also required? For this, assume that the packet network is an integrated services environment. For a request requiring bandwidth guarantee on demand, we need to consider also whether the router's scheduling algorithm can guarantee requests in terms of buffering, in addition to bandwidth guarantee on links. This brings in the issue of scheduling with routing. It has been shown that this combined problem can be addressed with a polynomial time algorithm that factors in capacity and constrained shortest path [766].

ALGORITHM 17.2 QoS widest path with delay feasibility

```
$\mathcal{S} = \{i\}$    // permanent list; start with source node $i$
$\mathcal{S}' = \mathcal{N} \setminus \{i\}$    // tentative list (of the rest of the nodes)
for ( $j$ in $\mathcal{S}'$ ) do
   // if $i$-$j$ directly connected and link has required bandwidth $\overline{b}$
   if ( $b_{ij} > \overline{b}$ and $\tau_{ij} < \infty$ ) then
      $B_{ij} = b_{ij}$; $T_{ij} = \tau_{ij}$
   else
      $b_{ij} = 0$; $B_{ij} = 0$; $\tau_{ij} = \infty$; $T_{ij} = \infty$    // mark temporarily as unavailable
   endif
endfor
while ($\mathcal{S}'$ is not empty) do    // while tentative list is not empty
   $Btemp = 0$    // find neighbor $k$ with maximum bandwidth
   for ( $m$ in $\mathcal{S}'$ ) do
      if ( $B_{im} > Btemp$ ) then
         $Btemp = B_{im}$; $k = m$
      endif
   endfor
   if ( $B_{ik} < \overline{b}$ ) then    // bandwidth is higher than the request tolerance
      No feasible bandwidth path exists; request denied
      exit
   endif
   if ( $k == v$ ) then exit    // destination $v$ is found; done
   $\mathcal{S} = \mathcal{S} \cup \{k\}$    // add to permanent list
   $\mathcal{S}' = \mathcal{S}' \setminus \{k\}$    // drop from tentative list
   for ( $j$ in $\mathcal{N}_k \cap \mathcal{S}'$ ) do    // path has higher bandwidth
      if ( $B_{ij} < \min\{B_{ik}, b_{kj}\}$ ) then
         $B_{ij} = \min\{B_{ik}, b_{kj}\}$
         $T_{ij} = T_{ik} + \tau_{kj}$
      endif
   endfor
endwhile
if ( $B_{iv} \ge \overline{b}$ ) then    // final check; if path meets bandwidth requirement
   'Request accepted'
else
   'No feasible path exists; Request denied'
endif
end procedure
```

17.4 Update Frequency, Information Inaccuracy, and Impact on Routing

In the previous section, we have provided the computational framework for QoS routing for classification Q.2 by considering single or multiple attributes. What is missing is how often attribute information is obtained and/or when the computation is performed. To discuss these important aspects, we will again assume that a link state framework is used.

Ideally, it appears that if a node knows the state of each link in terms of the applicable attributes (either single or multiple) *instantaneously*, it can then invoke routing computation. There are, however, practical limitations on this utopian view:

- First, an instantaneous update is almost impossible in a real network; very frequent updates can lead to excessive information exchange, which can overload a network. In fact, it has now become a common practice in many routing protocols to include a hold-down time to assert that no updating of information is allowed that is more frequent than the hold-down time. Also note that if a particular link state is advertised too frequently due to a legitimate change in the link state status, some form of dampening is still applied by a receiving node to avoid having an undesirable performance consequence, and before flooding to its neighboring node; as an example of a similar situation, see Section 8.9 addressed for the border gateway protocol (BGP).

- Second, there are two possibilities in regard to routing computation: (1) perform the computation periodically, or (2) perform it on demand for every arriving request. The second option is usually avoided since an important requirement in QoS routing services is that the call setup time, also known as postdial delay, for an arriving request is as small as possible. There is an important lesson to be learned here from RTNR (refer to Section 10.6). In an almost fully mesh network environment with a separate signaling (SS7) network for link state message exchanges, RTNR was initially intended to be deployed with per call computation in mind; in actuality, the computation is based on the information queried for the *previous* call in order to avoid increasing postdial delay to an undesirable level. For a general packet network, performing routing computation on demand for each arriving request can be taxing on the CPU load of the node—thus, this is also not desirable.
- Finally, it is not hard to realize that if the link state information obtained at a node is delayed due to periodic/asynchronous update or dampening, i.e., the link state information is somewhat stale or inaccurate. Due to such inaccurate information, it is questionable if it is worth doing a per-call routing computation.

To summarize, for QoS routing, it is more appropriate to perform a routing computation periodically than on a per-call basis and build a routing table. Taking this entire scenario into account, the arrival of link state information and the timing of the routing computation are depicted in Figure 17.2. It may be noted that due to the periodic computation framework, instead of executing a constrained shortest path or constrained widest path on a per-pair basis, it can be performed on a source to all destination basis, albeit with the option that for a specific pair the computation can be triggered if needed. In any case, it is important to note that if there is a network link failure, usually link state flooding and routing computation are triggered immediately so that changes can be accommodated by each node.

There is, however, an important consequence of periodic/update and periodic routing table computation. Suppose that the routing is hop-by-hop and each node has only one entry for each destination identified by the next hop. When an actual request arrives, there may not be enough resources along the path (dictated by the routing table) to establish the call. Thus, this request is denied entry, which then affects the overall call-blocking probability. Note that just being locked into one path during two consecutive routing computations does not necessarily mean that all arrivals will be blocked during this window; it is important to note that during this time window, some exiting calls might be overreleasing resources that can be used by newly arrived calls (for example, refer to Figure 11.1). In any case, to maintain the GoS aspect of QoS, there are two possibilities: (1) updates must be done frequently enough so that the newly obtained path does not block too many calls, or (2) the network is engineered with enough capacity/resources so that the overall blocking effect is maintained at an accept-

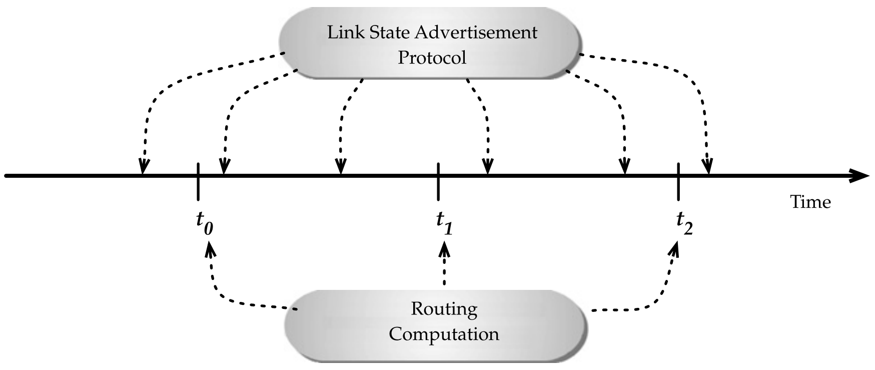

FIGURE 17.2 Time of link state advertisement and routing computing for QoS routing.

able level. The first option belongs to traffic engineering while the second option belongs to capacity expansion. Note that it is also important to maintain the GoS aspect of QoS to avoid excessive user-level retry in case calls are blocked.

Since an important goal of QoS routing is to provide good traffic engineering, we may ask the following question: can we consider more than one path from a source to a destination? This will partly depend on whether the network is capable of providing hop-by-hop routing and/or source routing. In the case of hop-by-hop routing, the only option for multiple paths is if there are two paths of equal cost, i.e., equal-cost multipath (ECMP). It is difficult to find multiple equal-cost paths in a constrained-based routing environment. In a source routing environment, multiple paths can be cached ahead of time, which then leads to the possibility of *alternate routing* options.

17.5 Lessons from Dynamic Call Routing in the Telephone Network

There has been extensive experience with alternate routing for dynamic call routing for the telephone network; see Chapter 10 for details and also Section 11.6. First, we summarize the typical network environment for dynamic call routing in telephone networks:

- The network is fully mesh, or nearly fully mesh.
- Calls attempt a direct link path first (if available); then an alternate path is attempted; alternate paths are made of at most two links.
- The path cost is nonadditive, concave.
- The alternate paths to be considered and the number of such paths to be cached depend on specific routing schemes.
- Call setup can be based on progress call control or originating call control.
- In the presence of originating call control, a call crankback can be performed to try another path, if such a path is listed in the routing table.
- Routing schemes use a link state framework.

- Link state update, setup messages, and crankback messages are carried through out-of-band signaling, for example, using an SS7 network.
- The main performance measure is minimization of call-blocking probability, which can dictate the choice of a routing scheme. However, factors such as messages generated due to call setup, crankback, and link state updates can also be deciding factors.

There are key lessons learned from such a dynamic call routing environment:

- In general, a dynamic call routing scheme increases throughput, but has a metastability problem beyond a certain load to capacity ratio in the network.
- A trunk reservation feature is used to protect a direct link from being excessively used by alternate routed calls to avoid metastable behavior. In essence, trunk reservation works as a link-level admission control. An important consequence that sounds counterintuitive is that a network may not accept a call even if it has capacity under certain conditions.
- For effective traffic engineering, especially under overload conditions, several control measures such as dynamic overload control, call gapping, and hard to reach may need to be invoked.

To reiterate, dynamic call routing in a telephone network is operating for *single-rate* homogeneous service—all are voice calls requiring the same amount of bandwidth for the duration of the call. The first question then is what changes in regard to a heterogeneous service environment where arriving calls require *differing* bandwidth. This is discussed in the next section.

17.6 Heterogeneous Service, Single-Link Case

To understand an important difference going from a single-rate service case to a multiple-rate service case, we illustrate a performance scenario that has important implications for QoS routing. Note that this analysis requires some understanding of offered load in Erlangs (Erls) and call blocking, discussed earlier in Section 11.2 using just a single link without the presence of routing. The results discussed below are summarized in Table 17.3.

Consider a service that requires 1 Mbps of bandwidth during the duration of the call. Assume that the link capacity is 50 Mbps; thus, this link can accommodate at most 50 such calls simultaneously, that is, the effective capacity of the link is 50. Assume that the call arrival pattern is Poisson with an average call arrival rate at 0.38 per second, and that the average duration of a call is 100 seconds. Using Eq. (11.2.2), we can determine that the offered load is $0.38 \times 100 = 38$ *Erls*. Furthermore, using the Erlang-B loss formula Eq. (11.2.3), we can find that 38 *Erls* offered to a link with 50 units of capacity results in a call-blocking probability of 1%. Since most networking environments would like to maintain a QoS performance requirement for call blocking below 1% probability, we can see that users will receive acceptable QoS in this case. Note that to meet QoS, there are two issues that need to be addressed: (1) each call must receive a bandwidth guarantee of 1 Mbps, if admitted, and (2) the call acceptance probability is below 1% so that users perceive that they are almost always going to get a connection whenever they try.

TABLE 17.3 Call blocking for different services under various scenarios.

Link capacity (Mbps)	a_{low} (*Erls*)	a_{high} (*Erls*)	m_{low} (Mbps)	m_{high} (Mbps)	Reservation (Yes/No)	$\mathcal{B}_{\text{low}}$	$\mathcal{B}_{\text{high}}$	$\mathcal{W}_{\text{composite}}$
50	38.0	—	1	—		1.03%	—	1.03%
50	19.0	1.9	1	10	No	0.21%	25.11%	12.66%
85	19.0	1.9	1	10	No	0.05%	0.98%	0.52%
85	22.8	1.9	1	10	No	0.08%	1.56%	0.75%
85	22.8	1.9	1	10	Yes	1.41%	0.94%	1.20%
85	22.8	1.9	1	10	Yes, $Prob = 0.27$	1.11%	1.10%	1.11%

Next, consider the situation where we allow a new 10-Mbps traffic stream on the *same* 50-Mbps link to be shared with the basic 1-Mbps traffic stream. We start by splitting the 38 *Erls* of offered load equally, i.e., 19 *Erls* to the 1-Mbps traffic class and 19 *Erls* to the 10-Mbps traffic class. However, note that each 10-Mbps call requires 10 times the bandwidth of a 1-Mbps call. Thus, a more appropriate equitable load for a 10-Mbps traffic stream would be 1.9 *Erls* (= 19/10) when we consider traffic load level by accounting for per-call bandwidth impact. The calculation of blocking with different traffic streams and different bandwidth requirements is much more complicated than the Erlang-B loss formula; this is because the Erlang-B formula is for traffic streams where all requests have the *same* bandwidth requirement. The method to calculate blocking in the presence of two streams with differing bandwidth is known as the Kaufman–Roberts formula [356], [597]. Using this formula, we can find that the blocking probability for a 1-Mbps traffic class will be 0.21%, while for a 10-Mbps traffic class it is 25.11%.

We can see that for the same amount of load exerted, the higher-bandwidth traffic class suffers much higher call blocking than the lower-bandwidth service in a *shared* environment; not only that, the lower-bandwidth service in fact has much lower blocking than the acceptable 1% blocking. If we still want to keep the blocking below 1%, then there is no other option than to increase the capacity of the link to a higher-capacity link (unless the network is completely partitioned for each different service). After some testing with different numbers, we find that if the link capacity is 85 Mbps, then with 19 *Erls* load of 1-Mbps traffic class and 1.9 *Erls* load of 10-Mbps traffic class, the call blocking would be 0.05% and 0.98%, respectively. The important point to note here is that with the introduction of the higher-bandwidth traffic class, to maintain a 1% call-blocking probability for each class, the link capacity is required to be 70% (= $(85 - 50)/50$) more than the base capacity.

Now, consider a sudden overload scenario for the 1-Mbps traffic class in the shared environment while keeping the overall capacity at the new value: 85 Mbps. Increasing the 1-Mbps traffic class by a 20% load while keeping the higher bandwidth (10 Mbps) traffic class at the same offered load of 1.9 *Erls*, we find that the blocking changes to 0.08% and 1.56%, respectively. What is interesting to note is that although the traffic for the lower-bandwidth call has increased, its overall blocking is still below 1%, while that of the higher-bandwidth call has increased beyond the acceptable threshold level; yet there has been *no* increase in traffic load for this class. These are sometimes known as *mice and elephants* phenomena. Here mice are the lower-bandwidth service calls, while elephants are the higher-bandwidth service calls. However, unlike IP-based TCP flows (see [272]), the situation is quite different in a QoS-based environment—it is the mice that get through while elephants get unfair treatment.

This suggests that some form of admission control is needed so that higher-bandwidth services are not treated unfairly. One possibility is to extend the idea of trunk reservation to *service class* reservation so that some amount of the link bandwidth is logically reserved for the higher-bandwidth service class. Taking this into account, assume that out of 85 Mbps of capacity, 10 Mbps of capacity is reserved for the elephant (10-Mbps) service class; this means that any time the available bandwidth drops below 10 Mbps, no mice (1 Mbps) traffic calls are allowed to enter. With this change in policy, with 20% overload for mice traffic from 19 *Erls*, while elephant traffic class remains at 1.9 *Erls*, we find that the call blocking for mice traffic would be 1.41% and 0.94%, respectively—that is, the elephant traffic class is not affected much; this is then good news since through such a service class–based reservation concept, certain traffic classes may be protected from not getting their share of the resources. Now, if an equitable blocking is still desirable for both service classes, even though only the low-bandwidth stream is overloaded, then some mechanisms are needed to increase the blocking for the elephant service class. A way to accomplish this is to consider a probabilistic admission control; this rule can be expressed as follows:

- *An amount of bandwidth threshold may be reserved for higher-bandwidth calls, which is activated when the available bandwidth of the link falls below this threshold. As a broad mechanism, even when this threshold is invoked, lower-bandwidth calls may be admitted based on meeting the acceptable probabilistic admission value.*

To compute blocking for each traffic class with differing bandwidth *and* a probabilistic admission control and reservation, an approach presented in [480] is used. In Table 17.3 we list the probabilistic admission control case along with reservation and no reservation for the higher-bandwidth traffic class; you can see that equity in call blocking can be achieved when, with reservation, 27% of the time low-bandwidth calls are still permitted to be admitted.

We now consider the other extreme when only high-bandwidth 10-Mbps calls, still with 38 *Erls* of traffic, are offered. To keep call-blocking probability at 1%, with 38 *Erls* of offered load, a link would still need 50 units of *high-bandwidth* call-carrying capacity; this then translates to a raw bandwidth of 50×10 Mbps $= 500$ Mbps. Thus, we can see that depending on whether a network link faces low-bandwidth calls, or a mixture of low- and high-bandwidth calls, or just (or mostly) high-bandwidth calls, for the same offered load exerted, the link requires vastly different raw link bandwidth to maintain a QoS performance guarantee.

Finally, while we discuss call blocking for each individual traffic class, it is also good to have a network-wide performance objective in terms of bandwidth measure. Suppose that a_{low} is the offered load for the low-bandwidth traffic class that requires m_{low} bandwidth per call; similarly, a_{high} is the offered load for high-bandwidth traffic, and m_{high} is the bandwidth requirement per call of high-bandwidth calls, then a bandwidth blocking measure is given by

$$\mathcal{W}_{\text{composite}} = \frac{m_{\text{low}} a_{\text{low}} \mathcal{B}_{\text{low}} + m_{\text{high}} a_{\text{high}} \mathcal{B}_{\text{high}}}{m_{\text{low}} a_{\text{low}} + m_{\text{high}} a_{\text{high}}}. \tag{17.6.1}$$

These composite performance measure values for the cases considered above are also listed in Table 17.3. We can see that while this composite measure is a good overall indicator, it can miss unfair treatment to high-bandwidth calls.

Generalizing from two service classes to the environment where *each* arriving call i has an arbitrary bandwidth requirement m_i, the composite bandwidth blocking measure, known as Bandwidth Denial Ratio (BDR), is given by

$$\mathcal{W}_{\text{composite}} = \frac{\sum_{i \in \text{Blocked Calls}} m_i}{\sum_{i \in \text{Attempted Calls}} m_i}. \tag{17.6.2}$$

However, we have learned an important point from our illustration of low- and high-bandwidth traffic classes that higher-bandwidth classes may suffer higher blocking. We can still consider a simple generalization determine if a similar occurrence is noticed when *each* call has a differing bandwidth. Based on profiles of calls received, they may be classified into two or more groups/buckets in terms of their per-call bandwidth requirements, and then apply the above measure to each such group. For example, suppose that a network receives calls varying from a 64-Kbps requirement to a 10-Mbps requirement; calls may be put into, say, three buckets: 0 to 3 Mbps, 3 Mbps to 7 Mbps, and higher than 7 Mbps. If higher-bandwidth groups have a significantly higher-bandwidth blocking rate than the average bandwidth blocking rate for all calls, then this is an indicator that some form of admission control policy is needed so that the higher-bandwidth call groups do not necessarily have a significantly higher-bandwidth blocking rate.

17.7 A General Framework for Source-Based QoS Routing with Path Caching

We now consider a general alternate call-routing framework where calls are heterogeneous. To consider a general framework, we first summarize several goals of QoS routing:

- Reduce the impact on the call setup time by keeping it as low as possible.
- Minimize user-level retry attempts, i.e., it is preferable to do retry *internally* to the network as long as the call setup time is not drastically affected. It is important to note that user-level retry attempts cannot be completely avoided, at least in a heavily loaded network, i.e., a network where the ratio of traffic to network bandwidth is at a level beyond the normally acceptable tolerance for service guarantee.
- Allow the capability for the source node to select a path from a number of possible routes very quickly for each arriving request. Also, allow *crankback* capability as an optional feature.
- Allow a call admission control feature that can be invoked.

To keep call setup time minimal and the need to minimize user-level retry along with the recognition that on-demand route determination can be taxing suggests that having multiple path choices can be beneficial in a QoS routing environment; this is often referred to as *alternate path routing*. Since path caching is necessary to be able to do alternate path routing, we refer to it as the *path caching option*. With multiple path choices, knowing that due to inaccurate/stale information blocking on a path selected cannot be completely ruled out, crankback is a nice optional feature to try another path quickly, thus avoiding user-level retry.

Finally, a framework should allow the ability to incorporate a number of routing schemes so that network providers can choose the appropriate one depending on their performance and systems configuration goal.

17.7.1 Routing Computation Framework

The basic idea behind this framework addresses the following: how is the selection of paths done, when are they selected, and how are they used by newly arrived requests? For calls requiring bandwidth guarantees, another important component that can complicate the matter is the definition of the cost of a path based on possibly both additive and nonadditive properties. Later, we will consider our framework using an extended link state protocol concept. Before we discuss this aspect, we describe a three-phase framework [469]: (1) Preliminary Path Caching (PPC) phase, (2) Updated Path Ordering (UPO) phase, and (3) Actual Route Attempt (ARA). Each of these phases operates at different time scales.

The first phase, PPC, does a preliminary determination of a set of possible paths from a source to destination node, and their storage (caching). A simple case for this phase is to determine this set at the time of major topological changes. PPC, in the simplest form, can be thought of as topology dependent, i.e., if there is a change in the major topological connectivity, then the PPC phase may be invoked. This can be accomplished by a topology update message sent across the network in a periodic manner. This process can be somewhat intelligent, i.e., if a link availability is expected to be less than a certain threshold for a prolonged duration or if the link is scheduled for some maintenance work, then PPC can also be used for pruning the link and a new topology update, thus letting nodes determine a new set of cached paths. Essentially, PPC uses a coarse-grain view of the network and determines a set of candidate paths to be cached. A simple mechanism to determine the set of paths for each source node to each destination node may be based on hop count or some administrative weight as the cost metric using the k-shortest paths algorithm (refer to Section 2.8). Thus, for this phase, we assume the link cost metric for determining a set of candidate paths to be additive.

The second phase, UPO, narrows the number of QoS acceptable paths; this module uses the most recent status of all links as available to each source node. Since the PPC phase has already cached a set of possible paths, this operation is more of a compare or filter to provide a set of QoS acceptable paths. Furthermore, for a specific service type or class, this phase may also *order* the routes from most acceptable to least acceptable (e.g., based on path residual bandwidth), and will, in general, have a subset of the routes "active" from the list obtained from the PPC phase. In this phase, the cost metric can be either additive, e.g., delay requirement, or nonadditive, i.e., bandwidth requirement, or a combination, where one is more dominant than the other. Another important factor to note about the UPO phase is that the value of the link state update interval may vary, with each node being able to select the interval value; for simplicity, we will refer to this as the routing link state update interval (RUI). This phase should be more traffic dependent (rather than on-demand per call) with a minimum and maximum time window on the frequency of invocation.

The third phase is ARA. From the UPO phase, we already have a reasonably good set of paths. The ARS phase selects a specific route on which to attempt a newly arrived flow. The exact rule for selecting the route is dependent on a specific route selection procedure. The main goal in this phase is to select the actual route as quickly as possible based on the pruned available paths from the UPO phase.

There are several advantages of the three-phase framework:

- Different routing schemes can be cast in this framework.
- It avoids on-demand routing computation; this reduces the impact on the call setup time significantly since paths are readily available; i.e., there is no "cost" incurred from needing to compute routes from scratch *after* a new flow arrives.
- The framework can be implemented using a link state routing protocol with some extension. For the PPC phase, some topology information, for example, needs to be exchanged at coarse-grain time windows. During the UPO phase, periodic update on the status of link usage is needed at a finer grain time window. Since different information about links is needed at different time granularity for use by the PPC and the UPO phase, we refer to this as the *extended* link state protocol concept.
- Each of the three phases can operate independently without affecting the other ones. For example, in the PPC phase, the k-shortest paths can be computed either based on pure hop count or other costs such as link speed–based interface cost. In some schemes the UPO phase may not be necessary.

A possible drawback of the framework is that path caching will typically require more memory at the routers to store multiple paths; this will certainly also depend on how many paths are stored. However, with the drop in memory price, a path caching concept is more viable than ever before. Additionally, there is some computational overhead due to k-shortest path computation on a coarse-scale time window. Our experience has been that k-shortest path computation takes only a few seconds to generate 5 to 10 paths in a 50-node network on an off-the-shelf computer. Thus, this overhead is not remarkable since it is done in the PPC phase. If needed, a router architecture can be designed to include a separate processor to do this type of computational work periodically.

17.7.2 Routing Computation

Consider the source destination node pair $[i,j]$. The set of cached paths for this pair determined at time t (the PPC phase time window) is denoted by $\mathcal{P}_{[i,j]}(t)$ and the total number of paths given by $\#(\mathcal{P}_{[i,j]}(t))$. For path $p \in \mathcal{P}_{[i,j]}(t)$, let $\mathcal{L}^p_{[i,j]}(t)$ denote the set of links used by this path.

Let $b_\ell(t)$ be the available capacity of link ℓ at time t (obtained using the link state protocol for the UPO phase). Then, from a bandwidth availability perspective, the cost of path p for $[i,j]$ is determined by the nonadditive concave property of the available capacity on the bottleneck link along the path:

$$z^p_{[i,j]}(t) = \min_{\ell \in \mathcal{L}^p_{[i,j]}(t)} \{b_\ell(t)\}. \qquad (17.7.1)$$

Since the path is known from the PPC phase, this filter operation is quite simple. If the index p is now renumbered in order of the most available bandwidth to the least available bandwidth at time t, that is, from the widest path, the next widest path, and so on, then we have

$$z^1_{[i,j]}(t) \geq z^2_{[i,j]}(t) \geq \cdots \geq z^{\#(\mathcal{P}_{[i,j]}(t))}_{[i,j]}(t). \tag{17.7.2}$$

Similar to node i, all other source nodes can use the same principle to determine their own ordered path sets.

How is the available capacity of various links known to nodes i? This can be determined by receiving used capacity of various links through a link state protocol, either in a periodic or an asynchronous manner. Note the availability of the bandwidth on a link is dependent on whether trunk reservation is activated. Suppose the capacity of link ℓ is C_ℓ, and the currently occupied bandwidth as known at time t (based on link state update) is $u_\ell(t)$. In the absence of trunk reservation, the available bandwidth on link ℓ is given by

$$b_\ell(t) = C_\ell - u_\ell(t). \tag{17.7.3}$$

If, however, a part of the link bandwidth $r_\ell(t)$ for link ℓ is kept for trunk reservation at time t, then

$$a_\ell(t) = C_\ell - u_\ell(t) - r_\ell(t). \tag{17.7.4}$$

The former is sometimes also referred to as the residual bandwidth and the second as the available or allowable bandwidth.

There are two important observations to note. First, if the last update value of $u_\ell(t)$ changes dramatically, it can affect the decision process. Thus, in practice, an exponential weighted moving average value $\overline{u}_\ell(t)$ is more appropriate to use than the exact value from the most recently obtained measurement; a discussion on how the exponential weight moving average can be computed is given in Appendix B.6. Second, the reservation allocation for different service classes may be different; thus, it may be beneficial to keep different sets of alternate paths to consider for different service classes. This means that each service class is essentially sliced into a virtual topology.

17.7.3 Routing Schemes

The computation described above can be used in a number of ways. An obvious one is the maximum available capacity-based scheme (widest path); furthermore, the availability can be proportioned to different paths to select a weighted path, similar to dynamically controlled routing (DCR) (see Section 10.4).

The decision on computation routes may depend on whether the information is periodically updated. Finally, the crankback feature availability is a factor to consider; here we will assume that the crankback is activated only at the source node. This means that during the call setup phase, an intermediate node does not try to seek an alternate path; instead, it returns the call control to the originating node when the call does not find enough resources on the outgoing link for its destination.

Recall that a fundamental component of the QoS routing framework used here is path caching. With this, in the PPC phase, a k-shortest paths algorithm (refer to Section 2.8) is used to generate a set of paths, which is cached. At this phase, the cost metric used is additive. For the routing schemes, an extended link state protocol is used to disseminate the status of the link (different information) at the PPC phase and the UPO phase. Since paths are already cached, the UPO phase can use a simple filtering mechanism to order paths based on available bandwidth (for services that require bandwidth guarantee for QoS). If there are services that have other QoS requirements such as path delay, these requirements can be easily incorporated in the UPO phase as additional filters.

Recall that an important goal of reducing the impact on flow setup time is addressed by the framework through the notion of path caching. Due to the three-phase framework, the newly arrived flow attempts one of the paths already pruned by the UPO phase—so there is no on-demand route computation delay in this phase. Depending on the periodicity of the UPO phase and the arrival of the link state advertisement, the pruned path set can have outdated information. Thus, some newly arrived flows can be assigned to a path that may not have any available bandwidth at this instant. This cannot be completely avoided unless the frequency of the update interval is reduced; if this is done, then more frequent link state advertisement would be necessary, which leads to an increase in network traffic.

17.7.4 Results

For performance studies, we consider maximum available capacity routing with periodic update and crankback (MACRPC), as well as for no crankback (MACRPNC). Note that MACRPC uses the shortest widest path on residual bandwidth, but with trunk reservation turned on, and the computation is periodic. For comparison, the utopian scheme, maximum available capacity routing with instantaneous computation (MACRIC), is also considered. This is possible since we have used a simulation environment where the instantaneous feature can be invoked. Also, we consider a sticky random routing scheme that extends the dynamic alternate routing scheme (see Section 10.5) to the multiservice case, which is labeled as cached sticky random adaptive routing (CaSRAR). Results presented here are based on call-by-call routing simulation for randomly arriving calls that follow the Poisson process.

Revisit Homogeneous Traffic Case

We first start with results on call blocking for the homogeneous service case as the number of cached paths K changes from 2 to 15 (for a 10-node fully connected network); this is reported in Figure 17.3 for both the case of no reservation and with trunk reservation set at 40%; while a very high trunk reservation value such as 40% is rarely used in an operational network, the intent here is to show how results are influenced, with and without trunk reservation. It is interesting to note that for the no reservation case, the increase of cached paths does not necessarily result in improvement in performance for *all* routing schemes. We see improvement only for MACRPNC. However, with trunk reservation activated, performance can improve with the increase in K for *all* routing schemes. This substantiates the claim on performance degradation in the absence of trunk reservation as reported in [385]. Furthermore, our result shows that this behavior is not necessarily consistent for *all* routing schemes. For the utopian scheme, MACRIC, the performance degrades drastically as K increases when

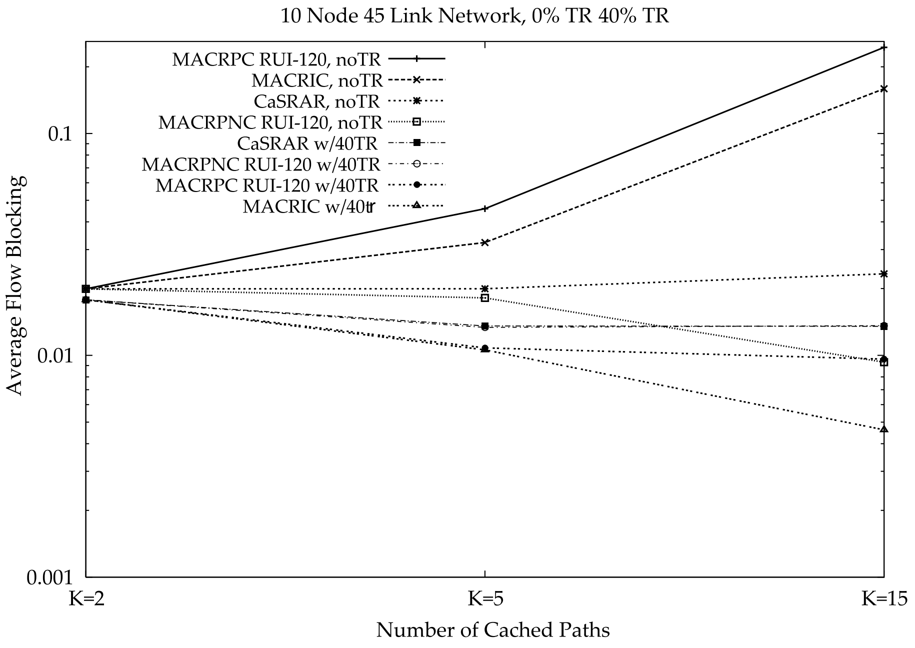

FIGURE 17.3 Homogeneous service fully-connected network (with and without trunk reservation).

there is no trunk reservation. Although this may sound surprising, this is possibly caused by overuse of multiple-link paths through instantaneous checking, which leads to local optimization and bistability. We observe the same problem with MACRPC when there is no trunk reservation. Overall, CaSRAR and MACRPNC are more robust in the absence of trunk reservation. However, in the presence of high trunk reservation, as K increases we found that MACRIC and MACRPC had better performances than CaSRAR and MACRPNC. Overall, these results show that path caching is indeed helpful; however, the actual routing schemes and factors such as trunk reservation do matter.

Service Class Performance

Next we discuss the case of heterogeneous services where three different service classes with differing bandwidth requirements for each service class are offered. We consider two cases: the network capacity in the first one is dimensioned[2] for low BDR (less than 1%) while the second one is dimensioned for moderately high BDR (over 5%). From the scenario where the network is dimensioned for low BDR (Figure 17.4(a)), we found that in the presence of trunk reservation, as K increases the BDR decreases for all schemes (similar to the homogeneous case). However, this is not true when the network is dimensioned for moderate BDR (Figure 17.4(b)), even in the presence of moderate trunk reservation. The pattern is somewhat closer to the homogeneous case with no trunk reservation. What we can infer is that even in

[2]Dimensioning or sizing refers to determining the capacity needed in a network to carry a given traffic offered load at a prespecified level of performance guarantee.

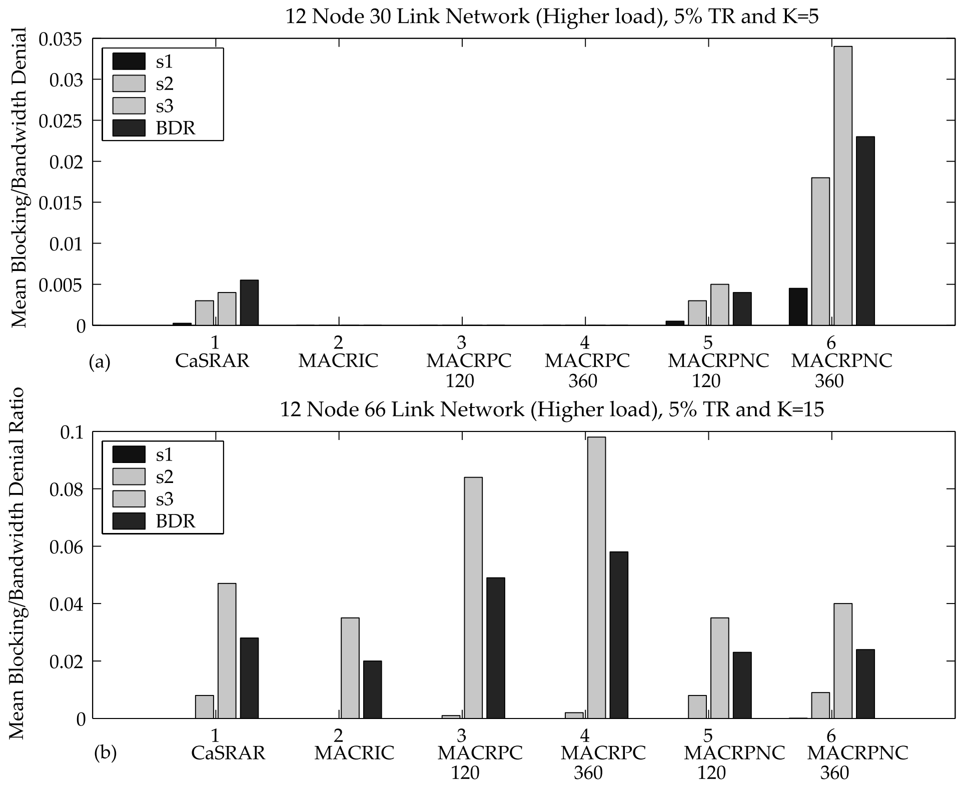

FIGURE 17.4 Performance of different routing schemes (and update periods), (a) low-load, sparsely connected case, (b) higher-load case.

the presence of trunk reservation, the ability to hunt over multiple paths through crankback is beneficial in a network designed for low BDR, but crankback can be detrimental when the network is designed for moderately high BDR as it impacts network performance (and also can lead to higher flow setup time due to frequent path hunting). See [695] for more details.

Now we discuss briefly the role of the UPO phase. Recall that different routing update interval (RUI) parameter values can be used for the UPO phase. As one would guess, with more frequent updates (i.e., for a smaller value of RUI), the inaccuracy in link state information decreases. It is observed that both schemes MACRPC and MACRPNC give better performance with more frequent updates as would be intuitively guessed. However, it appears that inaccuracy in link state information can be well compensated by the availability of crankback in a network designed for low BDR. Specifically, we note that MACRPC with an RUI of 360 seconds has much lower BDR than MACRPNC with an RUI of 120 seconds (Figure 17.4(a)). However, the reverse relation holds when the load is moderately high (Figure 17.4(b)). We also saw in an earlier example (through MACRIC) that instantaneous information update is not always beneficial in terms of network performance (as well as negatively affecting flow setup time considerably). Overall, we can infer that inaccuracy in link state information is not necessarily bad, and in fact, can be well compensated through path caching; in any case, the specifics of the routing scheme do play a role here.

So far we have discussed performance using the network-wide indicator bandwidth blocking rate. We are next interested in understanding the effect on each service class. For this, we have considered three service classes in increasing order of bandwidth requirement, i.e., the first service (s1) class has the lowest bandwidth requirement per flow, while the third service class (s3) has the highest bandwidth requirement per flow. For a network dimensioned for low BDR, we found that with a moderate to large number of path caching, CaSRAR and MACRPNC tend to give poorer performances to the higher bandwidth service class (s3), whether the network is fully or sparsely connected (Figure 17.4(a) is shown here for the sparsely connected case). Furthermore, the inaccuracy of routing information due to the update interval of the UPO phase does not seem to affect MACRPC for different service classes but can noticeably affect MACRPNC (Figure 17.4(a)). To check whether the same behavior holds, we increased the load uniformly for all service classes. We made some interesting observations (Figure 17.4(b)): the lowest bandwidth service (s1) has uniformly low flow blocking for *all* routing schemes; however, the highest bandwidth service class (s3) is affected worst under MACRPC at the expense of the lower bandwidth classes; i.e., MACRPC is more unfair to higher-bandwidth services as the network load uniformly increases. In general, we found that CaSRAR works better than the other schemes in providing smaller variation in performance differences seen by different service classes.

Call Admission Control

While it is known that higher-bandwidth, reservation-based services experience worse performance than lower-bandwidth, reservation-based services in a single-link system [380], these results indicate that this behavior holds as well in a network *with* dynamic routing and trunk reservation. In other words, routing and trunk reservation can*not* completely eliminate this unfairness. Thus, in a network, if fairness in terms of GoS to different service classes is desirable, then additional mechanisms are needed. In this context, a concept called *service reservation* beyond traditional trunk reservation has been proposed [471]. This concept can be manifested, for example, through source-based admission control at the time of flow arrival. While a good source-based admission control scheme for a general topology network in the *presence* of QoS routing operating in a link state protocol environment and trunk reservation remains a research problem, a probabilistic source-based admission control scheme for fully connected networks in the presence of routing and for two services case has been presented [471]. The ability to provide service fairness in terms of fair GoS using this source-based admission control scheme in the presence of routing and trunk reservation is shown in Figure 17.5. This is shown for three different values of network load with two service class scenarios (shown for normal load “lf-1.0,” 5% s2 overload “lf-s2,” and 5% network-wide overload “lf-1.05,” all for MACRPC). The right-most entries (corresponding to $p = 1$) denote the *no* source-based admission control case. As we can see, with the increase in load, the higher-bandwidth service suffers the most in the absence of source-based admission control. As the admission control parameter is tuned (by changing p toward 0.8) to invoke different levels of source-based admission control, it can be seen that service-level fairness in terms of GoS can be achieved.

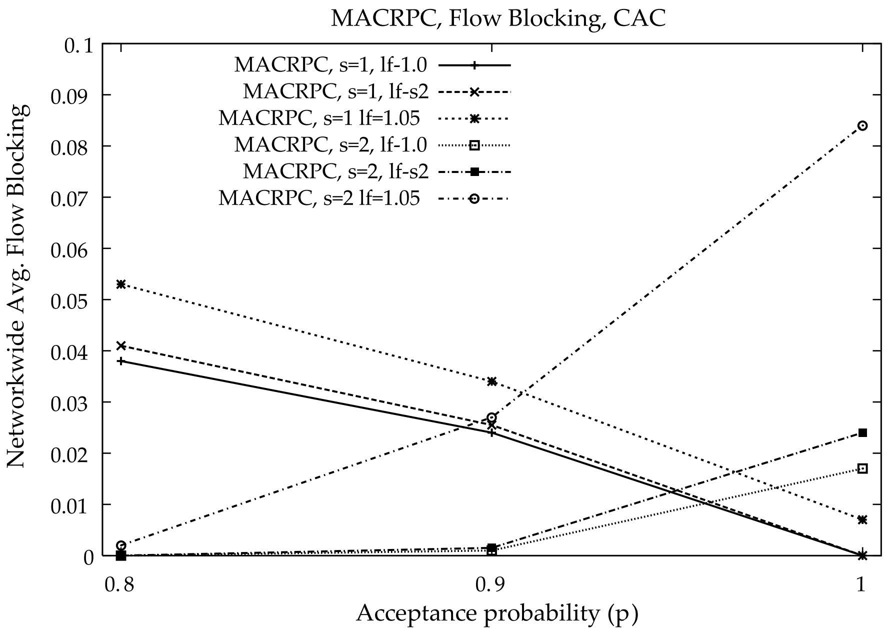

FIGURE 17.5 Performance impact in the presence of source-based admission control.

Dynamic Traffic

Finally, we discuss network performance impact due to network traffic dynamics. To show this we consider a homogeneous service, fully connected network where one source-destination node pair has dynamic traffic while the rest of the traffic pairs have stationary traffic (no source-based admission control is included here). For our study, the dynamic traffic has been represented through a time-dependent, stationary process that follows a sinusoidal traffic pattern. For the case with no trunk reservation, we have found that MACRPC has much worse performance than both CaSRAR and MACRIC as traffic changes for the dynamic traffic class (pair); CaSRAR adapts very well with traffic changes, although it has no UPO phase. It is interesting to note that just the presence of dynamic traffic between a source-destination node pair can cause the rest of the (stationary) traffic to show dynamic performance behavior (Figure 17.6).

We also considered the case in which trunk reservation is imposed; purposefully, we set the reservation at an unusually high value of 40% to understand the performance implication—the result is shown in Figure 17.7; from this figure, we note two phenomena: (1) MACRPC performs better than CaSRAR for dynamic traffic, and (2) the imposition of dynamic performance on the stationary traffic (from the dynamic traffic class) is no longer there. Also, we found that the overall performance improves in the presence of trunk reservation in a dynamic traffic scenario (similar to the stationary traffic case). From these results an important question, although not directly within the purview of routing, arises: should a network allow a dynamic traffic stream/class to impose its behavior on a stationary traffic stream? In other words, should a stationary traffic stream suffer higher flow blocking just because the load for the dynamic traffic stream is increasing? This cannot be addressed alone

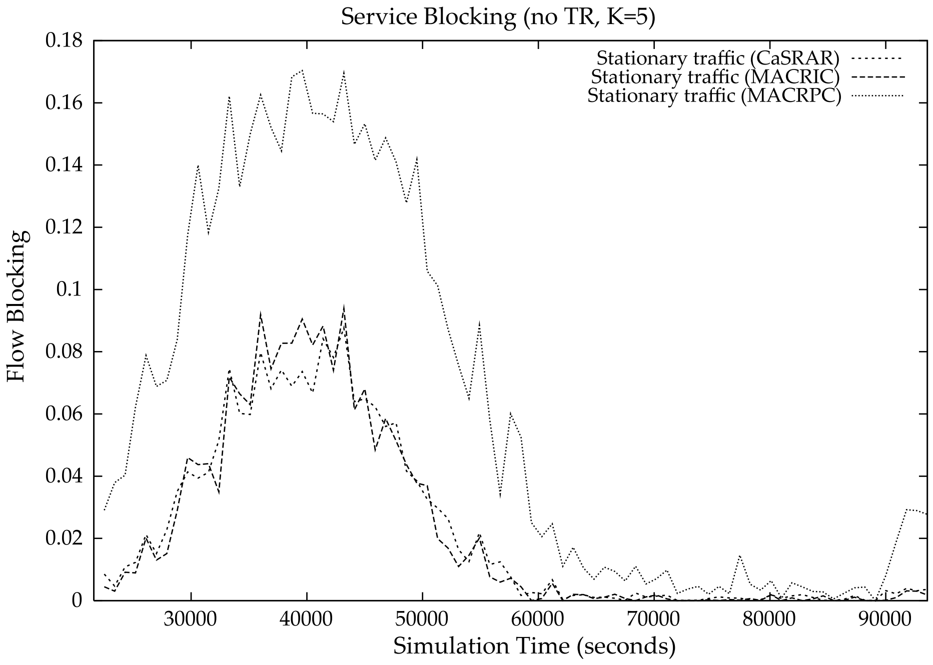

FIGURE 17.6 Dynamic performance behavior of *stationary* traffic due to the influence of dynamic traffic (no trunk reservation).

through the three-phase QoS routing framework or any other QoS routing framework. However, the impact can be controlled through the use of trunk reservation and under controls; this is where lessons on controls discussed earlier in Section 11.6 in Chapter 11 may be taken into consideration.

17.8 Routing Protocols for QoS Routing

17.8.1 QOSPF: Extension to OSPF for QoS Routing

The OSPF extension for QoS routing mechanisms, described in RFC 2676 [23], is commonly referred to as QOSPF. Earlier in Section 6.3, we discussed OSPF packet formats. You may note that every hello, database description, and LSA contains an options field that is 1 byte long. One of the bits in the options field, originally known as the T-bit to indicate if a originating router is capable of supporting Type of Service (TOS), was later removed [505]. Instead, the QOSPF specification proposed to reclaim this bit and renamed it as the Q-bit to indicate that the originating router is QoS routing capable. When this bit is set, two attributes are announced with a link state: bandwidth and delay.

An important aspect about the QOSPF protocol is that it specifies the path computation mechanism, which is divided into the pre-computed option and the on-demand option. For the pre-computed path option, a widest path version of the Bellman–Ford approach based on bandwidth was proposed (refer to Section 2.7.2 for a similar discussion). For the on-demand computation, a widest shortest path version of Dijkstra's algorithm that considered band-