On Network Resource Management for End-to-End QoS*

Ibrahim Matta

Computer Science Department
Boston University
Boston, MA 02215, USA
matta@cs.bu.edu

Abstract. This article examines issues and challenges in building distributed Quality-of-Service (QoS) architectures. We consider architectures that facilitate cooperation between the applications and the network so as to achieve the following tasks. Applications are allowed to express in their own language their QoS (performance and cost) requirements. These application-specific requirements are communicated to the network in a language the network understands. Resources are appropriately allocated within the network so as to satisfy these requirements in an efficient, scalable, and reliable manner. Furthermore, the applications and the network have to cooperate "actively" (or continually) to be able to achieve these tasks under time-varying conditions.

1 Introduction

Advanced network-based applications have received a lot of attention recently. An example of such applications is depicted in Figure 1. Here, a number of scientists are collaborating in a videoconference to control a distributed simulation. This application requires distributing simulator output to the scientists, the exchange of audio and video signals among them, etc. Such application demands some high-quality services from the network. For example, the simulator data should be distributed to the scientists quickly and with no loss. The audio and video signals should be transmitted to other participants in a timely and regular fashion to ensure interactivity, etc. The network in turn should provide these services while utilizing the network resources efficiently (to maximize revenue). Also, the network should deliver these services in a scalable and reliable way.

In this article, we examine architectures designed to provide such advanced applications with varying QoS. We start by discussing in Section 2 traditional network services that do not provide QoS support as well as traditional applications that do not express their QoS or are not aware of the QoS that the network is providing them with. We will argue why we need

* This work was done while the author was with the College of Computer Science of Northeastern University. The work was supported in part by NSF grants CAREER ANIR-9701988 and MRI EIA-9871022.

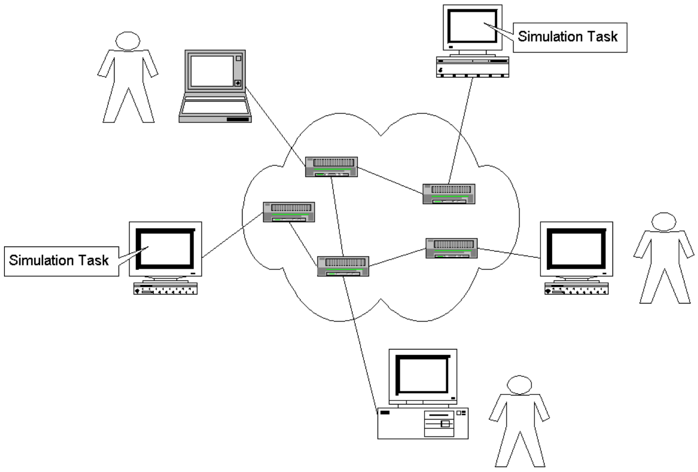

Fig. 1. Example of an advanced network-based application.

the network and applications to be aware of (and sensitive to) QoS, and we discuss in Section 3 how QoS support is achieved. Then, for these QoS-aware applications and network to cooperate, we present in Section 4 a generic integrated architecture and describe its components and discuss its main features. These features include the scalability of the architecture to large networks and large number of applications or users, and also its robustness to various dynamics. We conclude in Section 5.

2 QoS-oblivious Architectures

In this section, we discuss traditional network architectures and their shortcomings due to their lack of QoS support.

2.1 QoS-oblivious Network Services

Traditionally, a network, such as the current Internet, provides only a best-effort service, which means that the data which applications send can experience arbitrary amounts of loss or delays. On the positive side, the network could only implement simple mechanisms for traffic control. For example, any application can access (or send data over) the network at any time. Network devices (like switches or routers) can serve packets carrying application's data in a simple first-come-first-serve fashion. Packets can be routed to their destination over paths that are optimal with respect to a single metric, for example paths that have the minimum number of links (or that traverse the least number of routers). With these mechanisms, however, the application's QoS may not be met. For example, shortest-distance paths may be loaded and not provide the least delay and so may not satisfy the requirements of delay-critical applications. This kind of best-effort network has served quite well traditional data applications (e.g. Telnet, FTP, E-mail) that have relatively modest QoS requirements.

More advanced applications need additional support, for example, for multicasting a data packet to many destinations. The network should then be able to establish a delivery tree rooted at the source and whose branches lead to the various destinations. In the case of many senders, the network can build one tree for each sender. This is what current multicast protocols like DVMRP (Distance Vector Multicast Routing Protocol) [27] and MOSPF (Multicast Open Shortest Path First) [23] do. The tree usually consists of paths with minimum delay from the source to each destination. Figure 2 shows an example three-node network with two senders $S1$ and $S2$ and with a receiver at each node. A packet from $S1$ is replicated on both links to reach the two other nodes/receivers. Similarly, for $S2$. So this multicast uses 4 links or the total cost (in number of links) is 4. This cost may reflect the overhead of replication or the amount of bandwidth consumed by this communication group. Also, each packet experiences a maximum delay of going over 1 link.

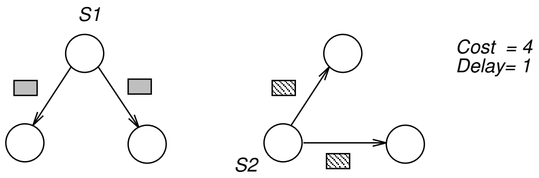

Fig. 2. Sender-based multicast trees.

Other multicast routing protocols would build a single shared tree that spans all members of the group. A major objective of such protocol is to minimize the sum of the link costs, or build a so-called “Steiner tree.” Since it is well known that it is computationally expensive (or NP-complete) to find such minimum-cost tree [11], protocols usually implement heuristics that are less expensive and give close to optimal trees. With such trees, the goal is to minimize the cost of replication and bandwidth, possibly at the expense of higher delays from a source to a destination. Figure 3 shows a shared tree. Here, the data packet from $S2$ needs to be generated only once, as opposed to being replicated in Figure 2 when source-rooted trees are used. This shared tree uses 3 links only, as opposed to 4 links with source-routed trees. However, the packet from $S2$ experiences a maximum delay of traversing 2 links, as opposed to 1 link with source-rooted trees. Some protocols like PIM-sparse [8] and CBT (Core Based Tree) [3] try to achieve a balance between cost and delay by having a single shared tree with the root at a center node and minimum-delay paths built from the center to members of the multicast group.

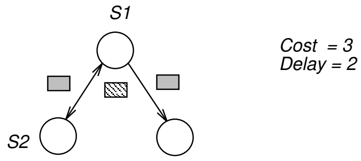

Fig. 3. A shared multicast tree.

One can easily see that regardless of what type of tree a *best-effort* multicast routing protocol builds, this tree may not be appropriate. This depends on several factors such as the location of the group members, the layout (topology) of the network, the requested QoS, etc. Figure 4(a) shows a case where a source-based tree should be built rather than a shared tree (with boldfaced links) as it costs less (using 2 links as opposed to 3). Figure 4(b) shows a case where a shared tree achieves lower cost (using 2 links as opposed to 3). This illustrates the need for adaptive network services that establish appropriate structures so as to efficiently utilize the network resources as well as satisfy the QoS requirements of applications. We later discuss such QoS-sensitive multicast routing protocols.

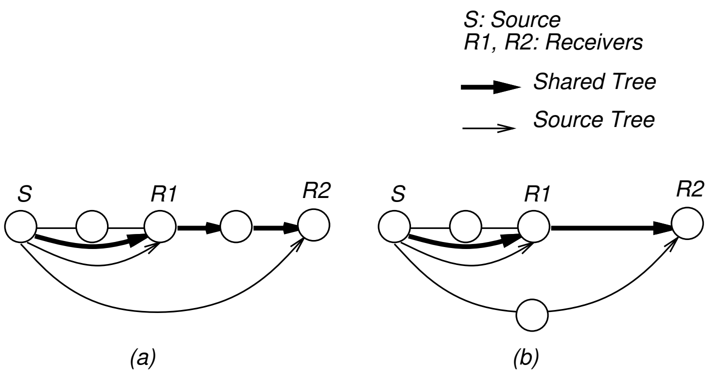

Fig. 4. (a) A source-based tree costs less, (b) a shared tree costs less.

2.2 QoS-oblivious Applications

As we traditionally had networks that are insensitive to various parameters for the state of the network and applications, we traditionally had applications that are insensitive to the state of the network and what kind of QoS they are getting from the network. A major consequence of this is that the application can experience “arbitrary” QoS. For example, under loaded network conditions, a video application can start losing its frames and suffering arbitrary degradation in quality. If the application had adapted its coding strategy to further compress its data and thus sent fewer frames, this may have increased the likelihood that the transmitted frames make it through the network. As a result, the application gets a consistent service (although of lesser quality due to compression). Another example of QoS-oblivious applications is one that "arbitrarily" assigns a distributed computation over the network.

Figure 5 shows a video example where the decoder (receiver) could decode either MPEG- or JPEG-coded streams. MPEG is more expensive to decode than JPEG. Thus, if there is enough computation cycles, and communication is expensive, we could just send MPEG-coded video and have a matching MPEG decoder, where video traffic is routed over the shortest (one-link) path. On the other hand, if there are not enough cycles and communication is cheap, we could use the decoder in JPEG mode to reduce computation cost and transparently insert an intermediate hardware-based (computationally inexpensive) transcoder that translates from MPEG format to JPEG format. Here, video traffic is routed through the transcoder over a longer (two-link) path, which is acceptable as communication is assumed to be cheap. This example illustrates the advantages of having applications that could adapt their operation mode based on the state of the system.

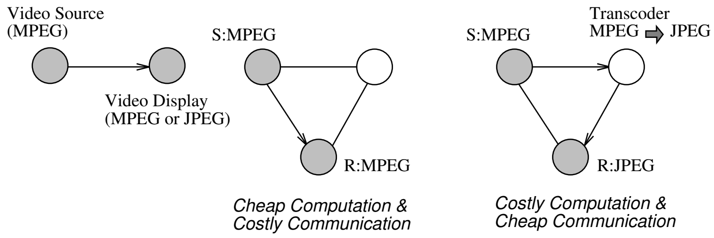

Fig. 5. Video transport example.

3 QoS-sensitive Architectures

Now that we have argued for the flexibility and potential benefits of QoS-sensitive network and applications over traditional ones, we next discuss various issues and challenges that must be addressed to implement them.

The network has to distinguish among traffic streams (or flows) requiring different QoS. A flow generally defines a stream of data packets that belong to the same application or a pre-defined aggregated traffic class. The works of several standardization groups, such as the IETF integrated-services [6] and differentiated-services [4] working groups and the ATM Forum Traffic Management working group [14], are based on allocating each traffic flow a given (absolute or relative) share of resources (bandwidth, buffers, etc.). This provides different flows with different services; better service (but typically more expensive) to higher priority flows at the expense of worse service (and usually cheaper) to lower priority flows.

Figure 6 shows a general architecture of a QoS network. A QoS manager is responsible for receiving requests for some QoS through some signaling protocol, such as the Internet RSVP protocol [7] or the ATM signaling protocol. The manager communicates with a routing component to find the outgoing link(s) or path that can likely satisfy the request and over which the flow would be routed. This path selection is typically based on an outdated view of the state of the network. The manager then communicates with an admission control component which decides whether indeed there are enough resources on the selected link(s) to satisfy the given QoS without violating the QoS already promised to existing flows. If the flow request is accepted (admitted), the QoS manager installs appropriate state in other components: A classifier that recognizes packets belonging to the flow. A route lookup component that forwards the flow's packets over the selected path. A shaper (or dropper) that shapes the flow (or drops excess traffic) according to the initial traffic specifications declared by the request and based on which resource allocations have been made. Finally, the manager has to set the parameters of the scheduler to allocate to the flow the necessary resources that are needed to satisfy the requested QoS.

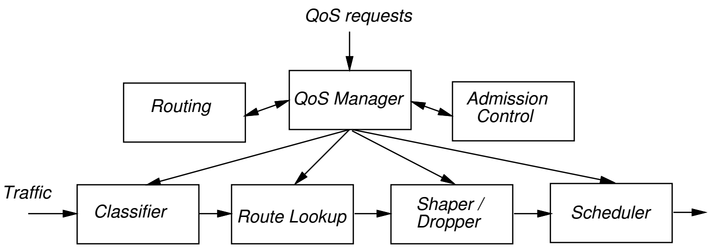

Fig. 6. A general architecture of a QoS network.

Figure 7 shows the architecture of a general scheduler and a traffic shaper. The scheduler isolates different traffic flows by allocating them fractions of the link's resources (bandwidth, buffer space). The traffic of a flow is shaped before entering the scheduler to compete for resources. The shaper shown is called "token bucket shaper." Tokens accumulate (fill the bucket) at some specified rate. A packet is allowed to enter the scheduler only if there is one or more tokens to drain. The depth of the bucket allows for the flow to burst (i.e. packets can enter back-to-back), but then the rate at which data enters so as to contend for resources is bounded by the rate at which tokens are generated. This bounded traffic specification makes it possible to test whether a new flow could be supported by considering its worst-case behavior.

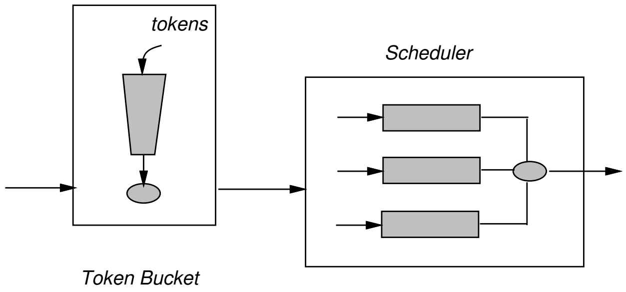

Fig. 7. A general scheduler with a traffic shaper.

As for routing (multicast) traffic, the routing protocol should choose a path that satisfies the QoS requirements of (all) the receiver(s). Thus, in multicasting, the type of multicast tree that should be constructed depends on the state of the network. Consider the example in Figure 8, where the numbers shown reflect the maximum link delays in the presence of a three-participant application, two of which send and receive at nodes A and C. Assume the application's QoS requirements are maximum end-to-end delay of 13 from the sender to any receiver, and jitter (or maximum difference between individual end-to-end delays) of 7. A shared tree would violate the application's delay and jitter requirements since the maximum end-to-end delay is 20 and the jitter is 10. Thus, under this network state, source-based trees should be constructed. Figure 9 shows the network in a different state, where a shared tree should be constructed.

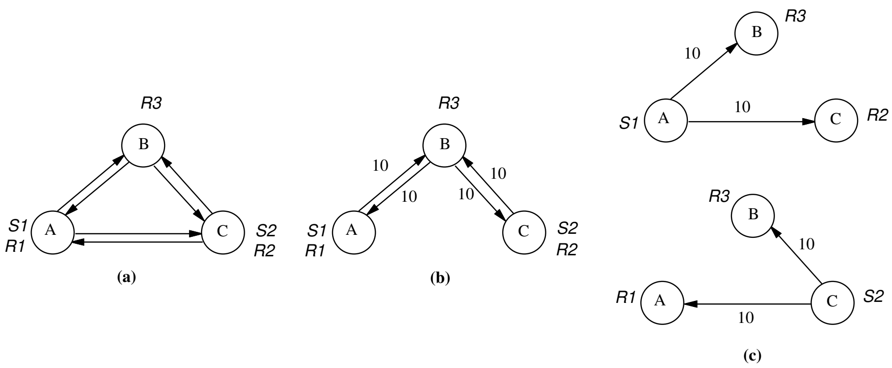

Fig. 8. (a) Original network, (b) shared tree, (c) source-based trees. Example where source-based trees should be constructed.

Clearly, to be QoS-sensitive, protocols have to account for the various dynamics at all levels: application, host, and network. Protocols have to be aware of the current participants in the application. Who communicates with

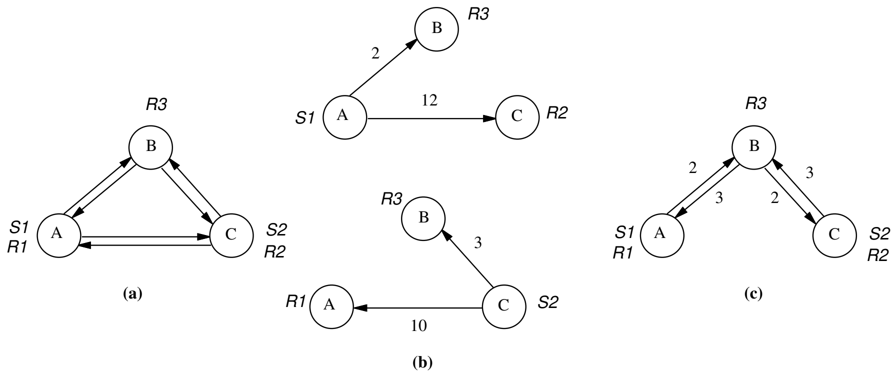

Fig. 9. (a) Original network, (b) source-based trees, (c) shared tree. Example where a shared tree should be constructed.

whom? Is a participant aware of the quality of her transmission so she can adapt to varying qualities? How is data generated during this transmission? What are the delivery requirements? How much resources are available at hosts (or end-systems) and at the network (switches/routers)? How does the layout (topology) of the network look like now? By accounting for all these dynamics, we would have a system that is capable of satisfying a wide variety of QoS requirements for diverse applications at all times, while operating efficiently.

Going back to multicast routing as an example of network control, the system may migrate from one multicast tree to another in response to changes at the application, host or network level. Figure 10 shows an example, where if the multicast protocol builds a single shared tree, then 5 links are used, and only $S2$'s packet needs to be replicated. Also, if, say, each source transmits at x bits/sec, then to guarantee rate for $R2$ to receive from both senders, we need to reserve $2x$ bits/sec on link $(S2, R2)$. If the available bandwidth on $(S2, R2)$ were less than $2x$, then a QoS network would not admit this application. However, if the multicast routing protocol adapts its tree construction mechanism and switches to source-based trees, this application could be admitted since the traffic is now better distributed over the network.

Thus, it is sometimes beneficial to switch to building a new type of tree. Building source-based trees in the previous configuration had allowed for admitting the application, although at the expense of more replication and more routing state information as we have to maintain 2 trees (one for each source) as opposed to 1 tree. This previous example also contradicts the common view that a minimum-cost shared tree reduces the amount of bandwidth consumed compared to sender-based trees, especially when we have multiple senders as the path from some sender to a receiver may turn out to be very long. Thus, a QoS system must employ a more intelligent tree construction strategy that adapts the shape of the tree dynamically, where we can trade-

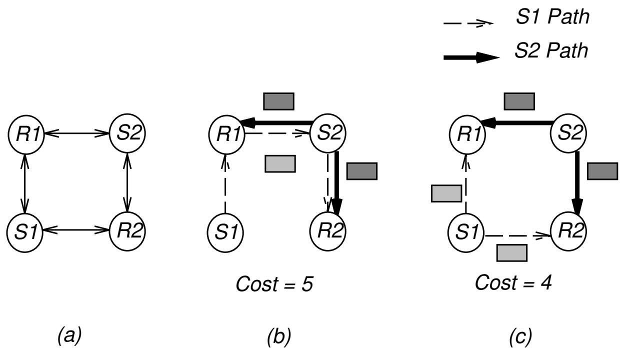

Fig. 10. (a) Original network, (b) shared tree, (c) source-based trees. Example illustrates benefits of adaptive multicast tree construction.

off between revenue from QoS support and cost of overhead to provide this support. We present an example of such strategy later.

4 Integrated QoS Architecture

The objective is to build an architecture that allows for protocols to adapt their behavior so as to account for various parameters and dynamics. The architecture should facilitate the exchange of information between the applications and network; the application should express an acceptable QoS region and be able to adapt its behavior to one that matches the level of QoS that the network currently delivers. The network should of course communicate that QoS level to the application. The goal is to efficiently utilize the network (or maximize its revenue) under the QoS constraints imposed by existing applications.

Figure 11 shows a generic integrated QoS architecture. In this architecture, applications express their QoS requirements in application-specific terms to an application-specific QoS manager. This manager understands the various attributes of a specific type of applications, and maps the application-dependent requirements into application-independent and implementation-independent requirements. A host QoS manager maps these in turn into implementation-dependent requirements so that necessary resources are allocated to the application by the operation system and network subsystem at the hosts as well as by router QoS managers within the network. QoS managers communicate with their peers or directly with their neighbor managers to coordinate the allocation of resources. Router QoS managers control the allocation of paths within the network by communicating with routing managers, which may use different types of routing protocols to locally or globally build paths that are capable of satisfying the QoS requirements of applications.

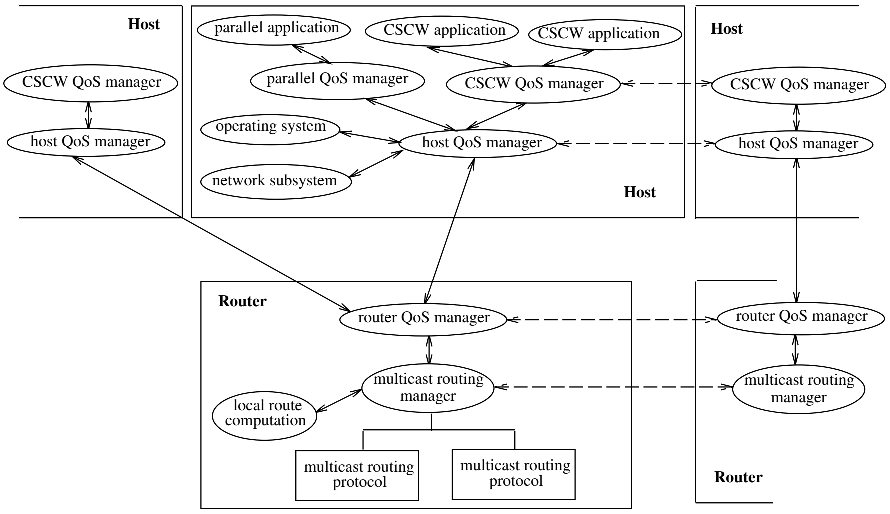

Fig. 11. Integrated QoS architecture.

Such architecture allows for supporting QoS between the endpoints of communication (*i.e.* the applications) in the presence of various dynamics. The QoS is controlled in a manner sensitive to the specifics of the application so that the system can successfully and efficiently deliver targeted QoS. The applications and network are allowed to exchange information so as to adapt mutually to system dynamics. In the following subsections, we elaborate on the delivery of targeted (application-oriented) QoS and the application's capability to be aware of (and adaptive to) system state, and also as an example of network control, we elaborate on building multicast routing trees that are sensitive to QoS.

We finally discuss mechanisms that should be in place in order to scale to large networks and to provide stability and reliability in the presence of changes in the state of the system.

4.1 Application-oriented QoS Mappings

An end-to-end QoS architecture has to deal with mapping the logical view of the applications to a physical allocation of resources. This involves application-specific QoS managers that take as input abstract QoS requests, such as application input graph with various tasks and interdependencies between them. Through a protocol to discover the state and location of physical resources (this knowledge is typically outdated), the QoS manager produces as output the needed physical resources that are likely to satisfy the application's QoS in an efficient manner. See Figure 12. This in turn involves host QoS managers that would communicate with other control entities at hosts and within the network using some signaling protocol to finally allocate the physical resources.

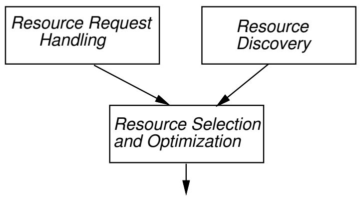

Fig. 12. Application-oriented QoS mapping.

Application-oriented Resource Selection: Given a (typically outdated) view of the system, which physical resources to select to satisfy an application-level QoS request? The answer to this question depends on the nature of the application. For example, consider an application that is real-time in nature and requires the scheduling of real-time tasks on a set of hosts. A real-time task needs to be reserved some amount of CPU cycles to meet a deadline. The host that is selected may end up rejecting the assigned task if the host finds that it does not currently have enough cycles (capacity). For this application, the major objective is to minimize task rejection rate. A traditional load distribution scheme is "load balancing." The goal here is to equalize the load over candidate hosts. However, as shown in Figure 13, this load-balancing strategy may result in capacity fragmentation and so higher rejection. Thus, although load balancing is adequate for providing best-effort QoS (optimizing *average* measures), it is not adequate for providing guaranteed QoS (optimizing *real-time* measures). A more appropriate scheme is *load profiling* [20]. The idea here is to have a more diverse profile of available capacity on the candidate hosts, so we increase the likelihood of finding a feasible host for future requests.

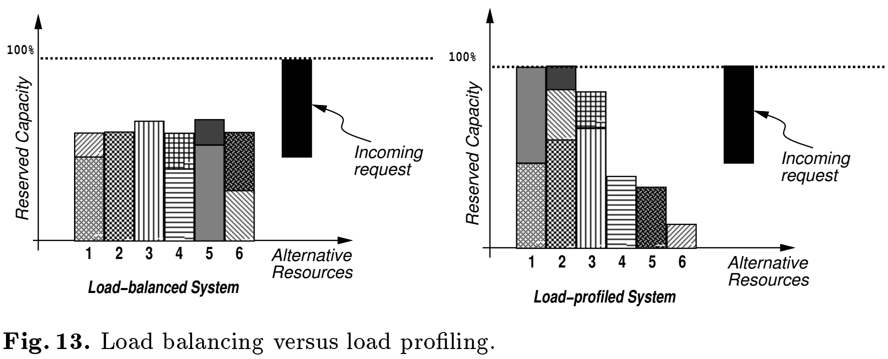

Fig. 13. Load balancing versus load profiling.

Figure 14 shows how after choosing the least-loaded host (with idle capacity of 15) for a class-1 task, we can then only accept 4 consecutive class-2 tasks. On the other hand, if we choose the most-loaded host (with idle capacity of 11), we can then accept 5 consecutive class-2 tasks, as we won't have fragmented capacity in the system. Choosing the most-loaded candidate is a "load packing" strategy, which is only asymptotically optimal for large systems and accurate feedback about system state. In a distributed system with delayed (inaccurate) feedback, a strategy that has the same effect but operates probabilistically is less sensitive to the inaccuracies in feedback information and is more appropriate. We call it "load profiling" strategy.

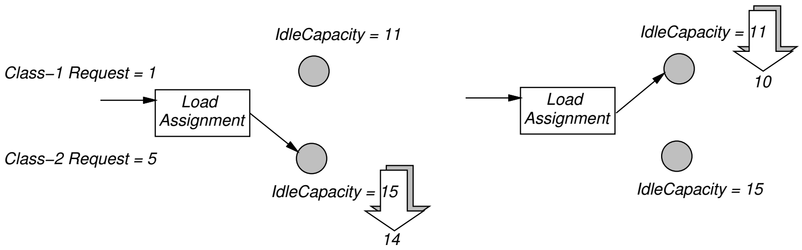

Fig. 14. Example illustrates difference between (a) load balancing and (b) load packing/profiling.

The main idea behind load profiling is illustrated in Figure 15, where the probabilities of selecting each candidate resource are adjusted so we would bring the distribution of QoS requests as close as possible to the distribution of available capacity. This is the well-known supply-demand matching problem.

The gain from load profiling (due to reducing fragmentation) is more significant when we have large requests, which is especially the case when a request represents the aggregate of many micro-requests. This gain over load balancing is also more pronounced as the system becomes more loaded. Extended models that consider the lifetimes of tasks, the costs of migration of tasks, etc. require more careful and more complicated analysis. So in summary, resource selection is an important and difficult problem: how to select resources so as to optimize some application-oriented measure(s) subject to QoS constraints and possibly other constraints on the type of resources, interdependencies between tasks to be assigned, etc. This is a multi-constrained optimization problem that needs fast heuristics that can produce high-quality solutions.

4.2 Network-aware Applications

To make resource selection easier, applications should specify a range of acceptable QoS if possible. This has many other benefits: most importantly is

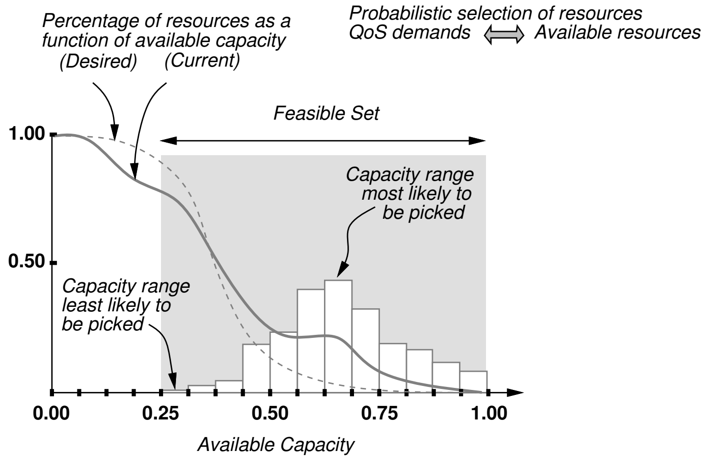

Fig. 15. Maintaining a resource availability profile that matches the characteristics of QoS requests.

that if the requested QoS can not be delivered by the network in a strictly guaranteed manner, then the application can adapt to the currently delivered QoS in a controlled way. For example, the application could send fewer data that have a chance of making it through the network and so the application gets a consistent (although of lesser quality) service. To do this, application-specific QoS managers need to maintain for applications QoS measures of interest to them, and inform applications about the current QoS operating point so that applications adapt accordingly. Applications could also maintain quality by compensating for QoS violations. For example, knowing the loss rate at the receiver, a video source could adjust its FEC (Forward Error Correction) error recovery scheme to compensate for errors. QoS managers could also try to hide QoS violations from applications, for example, by reducing latency through caching or prefetching requested data, by overlapping communication and computation, by migrating processes to where the data resides, etc.

4.3 QoS Multicast Tree Construction

Another important component of an end-to-end QoS architecture is efficient network services that are sensitive to the QoS requested by applications. One important service is multicast routing. A major goal here is to build a delivery multicast tree with minimum cost and which satisfies QoS delivery constraints. Again, this is a multi-constrained optimization problem and we need good and fast heuristics.

An example heuristic is called QDMR (QoS Dependent Multicast Routing) [15]. A nice feature of this heuristic is that it constructs a low-cost tree using a greedy strategy that augments the partially constructed tree with nodes of minimum cost. However, since this can lead to paths that violate QoS delay bounds, the tree construction policy is adapted on the fly to give up some cost savings so as to increase the likelihood of satisfying the QoS delay bound. Figure 16 illustrates the idea.

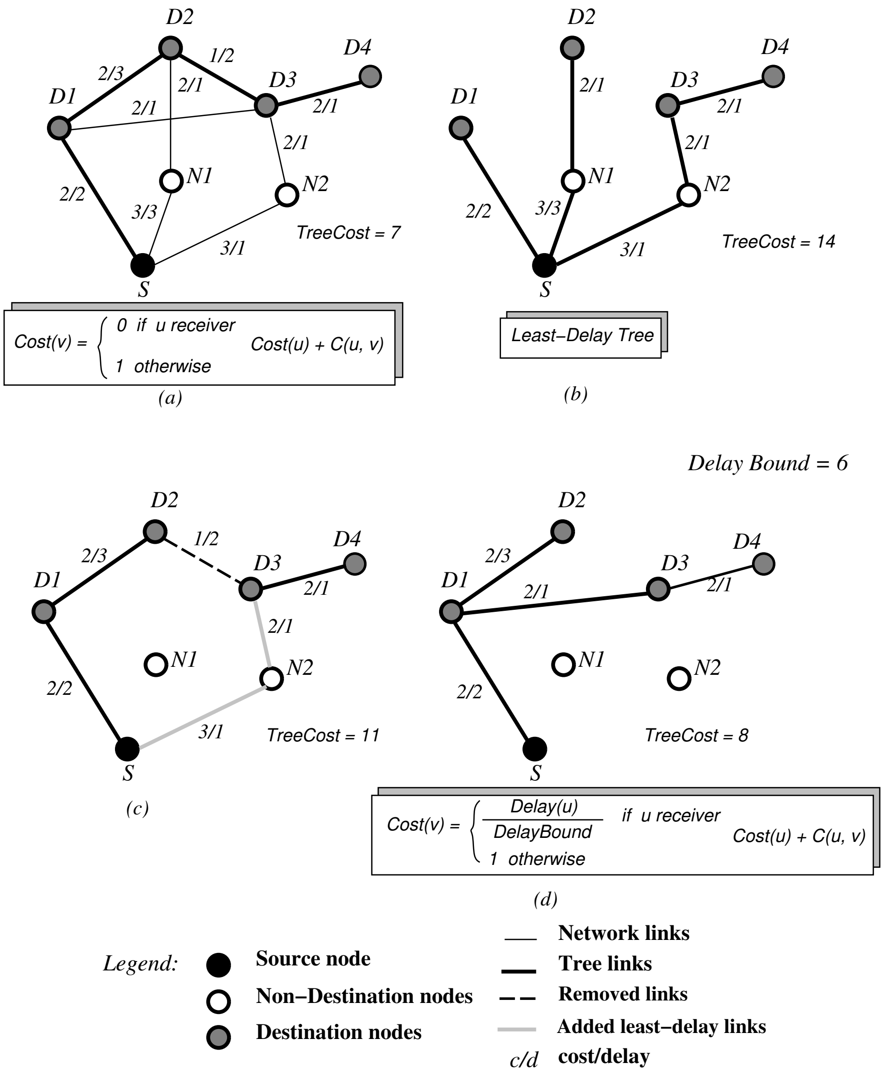

Fig. 16. QoS-aware multicast tree construction.

Figure 16(a) shows a low-cost tree construction policy. The cost of a new node v, denoted by $Cost(v)$, is defined in terms of $Cost(u)$, the cost of node u that is already on the tree, and $C(u, v)$, the cost of the link from u to v. $Cost(u)$ does *not* contribute to $Cost(v)$ if u is a receiver. The idea is to give priority to tree paths going through destination nodes so they are extended to add new nodes (as they would likely have lower cost). By leveraging the cost of reaching a destination to reach other destinations, the total cost of the tree is lowered. This, however, may violate the requested delay bound (which is the case for destination nodes $D3$ and $D4$). In Figure 16(b), the (sender-based) tree of least-delay paths is shown, which would satisfy the delay bound if this were indeed feasible. However, the tree cost is high.

In Figure 16(c), we modify the tree in (a) by replacing the infeasible paths for destinations $D3$ and $D4$ by the corresponding least-delay paths so as to obtain a tree that satisfies the delay bound. However, QDMR can generate a lower cost feasible tree as shown in Figure 16(d). The cost of a new node to be added to the current tree depends on how far we are from violating the delay bound. Paths through destination nodes are no longer given priority as we get closer to violating the delay bound. This makes the tree "bushier" and the delay bound is satisfied at a lower cost.

4.4 Scalability

Scalability is another important aspect of an end-to-end QoS architecture, especially for large wide-area systems. One main goal is to reduce the view a QoS manager has about the state of the system, and based on which it schedules resources (hosts, paths, etc.). This manager could be a sender or a receiver or an agent on behalf of the application, depending on where resource selection/allocation is done. A key to scalability is to separate the way the view is collected from how resources are selected. One way is to have pre-defined classes of applications and collect class statistics to attach to the view, as opposed to statistics about individual applications, for example, the total capacity used by a class rather than the individual capacities used by each application.

Another approach is to have special control entities, called *view-servers* [1], where each view-server maintains only a small view of its surrounding area, as opposed to a full view of the whole system. If a larger area is needed, more than one view-server could be queried and their views merged. Figure 17 illustrates the idea of view-servers.

Another more traditional approach is area-based, where nodes are grouped into level-1 areas, level-1 areas are grouped into level-2 areas, etc. See Figure 18. This is, for example, the scaling approach used in PNNI ATM routing [13]. The idea is that a node has only a detailed view of its own area, and less detailed (summarized or aggregated) views of remote areas, *i.e.* summarized view of level-1 areas in the same level-2 area, of level-2 areas in the same level-3 area, and so on. Different schemes could be used to aggregate an

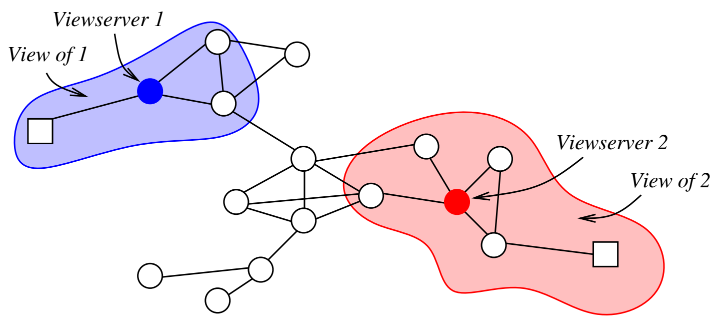

Fig. 17. Viewserver hierarchy.

area. For example, an area may be represented by a fully connected logical graph connecting all B border nodes (those nodes connecting the area to other areas), or by a logical star graph with a virtual node in the center, or by a logical single node. The accuracy of the view, which is presented to the resource selection process, decreases with more aggressive aggregation at the benefit of lesser overhead.

4.5 Robustness

Reliability is another important aspect of an end-to-end QoS architecture. This involves replication of important control entities to survive their failures. It involves avoiding oscillations between alternative configurations that would arise because of the performance interdependencies among the various applications. This could happen if we blindly honor QoS requests, violating existing QoS promises which are then reinstated by again violating other promises. Also, robustness involves reliable switchover to new configurations. For example, switching to a new multicast tree may require keeping transmission over the old tree until the new tree is fully established and the new QoS can be reliably delivered.

5 Conclusion

This article surveys some of the grand challenges in building integrated end-to-end QoS architectures. As we have seen, this involves a plethora of issues in finding fast and good heuristics, defining secure interfaces between different components, investigating the interactions between these components horizontally and vertically, etc. Another important issue is how to develop such a complex software in an easy and reusable way. One recent approach is aspect-oriented programming [18], which differs from the traditional object-oriented in that different aspects of the application or protocol, such as communication, core behavior, structure, etc. are not tangled together, which makes maintenance much easier.

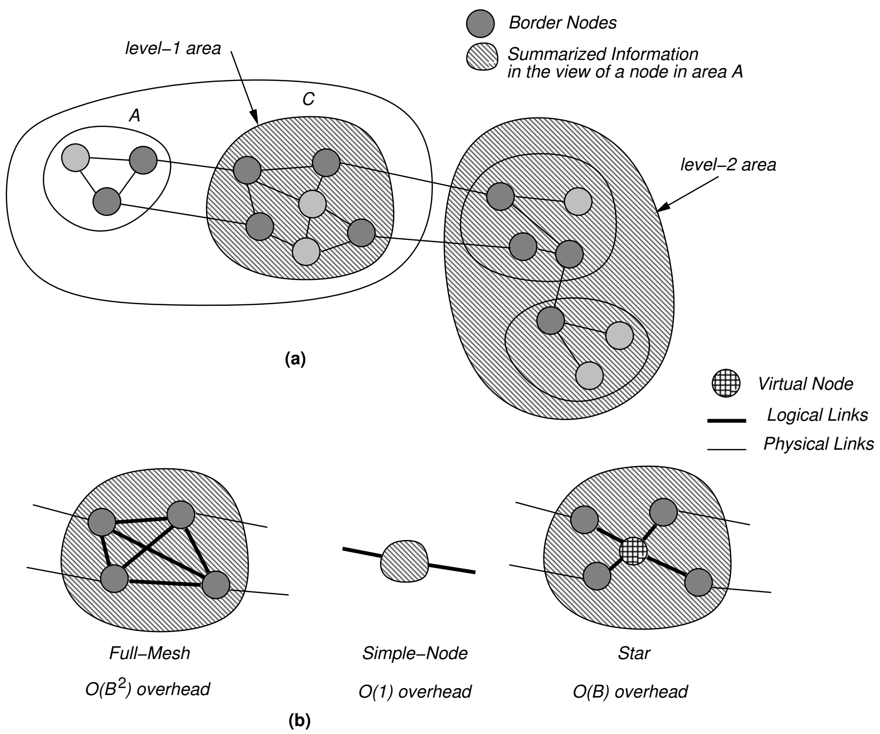

Fig. 18. (a) Area hierarchy, (b) aggregation of area C.

Research and development efforts to build an end-to-end QoS global system are necessarily multi-disciplinary. For such system to become reality, solutions to different problems have to be integrated into a flexible QoS architecture, and the overall performance and cost be evaluated. Initiatives, such as Internet2 [17] and NGI (Next Generation Internet) [16], are providing the infrastructure to deploy and test such advanced architectures.

References

1. C. Alaettinoglu, I. Matta, and A.U. Shankar. A Scalable Virtual Circuit Routing Scheme for ATM Networks. In Proc. *International Conference on Computer Communications and Networks - ICCCN '95*, pages 630–637, Las Vegas, Nevada, September 1995.
2. C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *ACM/Springer Verlag Multimedia Systems, Special Issue on QoS Architecture*, May 1998.
3. A. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees. In Proc. *SIGCOMM '93*, San Francisco, California, September 1993.
4. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.

5. J-C. Bolot. Adaptive Applications Tutorial. ADAPTS BOF, IETF Meeting, Washington DC, December 1997.
6. B. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: An Overview. Internet Draft, October 1993.
7. B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Internet Draft, March 1996.
8. S. Deering, D. Estrin, D. Farrinacci, V. Jacobson, C. Liu, and L. Wei. Protocol Independent Multicast (PIM): Protocol Specification. Internet Draft, 1995.
9. S. Fischer, A. Hafid, G. Bochmann, and H. de Meer. Cooperative QoS Management for Multimedia Applications. In Proc. *Fourth IEEE International Conference on Multimedia Computing and Systems (ICMCS'97)*, pages 303–310, June 1997.
10. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. Supercomputing Applications*, 11(2):115–128, 1997.
11. M.R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
12. A. Grimshaw, W. Wulf, and Legion Team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.
13. ATM Forum PNNI Subworking Group. Private Network-Network Specification Interface v1.0 (PNNI 1.0). Technical report, March 1996.
14. ATM Forum Traffic Management Working Group. ATM Forum Traffic Management Specification v4.0. Technical report, 1996.
15. L. Guo and I. Matta. QDMR: An Efficient QoS Dependent Multicast Routing Algorithm. Technical Report NU-CCS-98-05, College of Computer Science, Northeastern University, Boston, MA 02115, August 1998. To appear in *IEEE RTAS '99*.
16. NGI (Next Generation Internet). http://www.ngi.gov.
17. Internet2. http://www.internet2.edu/.
18. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Proc. *European Conference on Object-Oriented Programming*, pages 220–242. Springer Verlag, 1997.
19. J. Kurose. Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks. *ACM Computer Communication Review*, January 1993.
20. I. Matta and A. Bestavros. A Load Profiling Approach to Routing Guaranteed Bandwidth Flows. In Proc. *IEEE INFOCOM*, March 1998. Extended version in *European Transactions on Telecommunications - Special Issue on Architectures, Protocols and Quality of Service for the Internet of the Future*, February-March 1999.
21. I. Matta and M. Eltoweissy. A Scalable QoS Routing Architecture for Real-Time CSCW Applications. In Proc. *Fourth IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, June 1998.
22. I. Matta, M. Eltoweissy, and K. Lieberherr. From CSCW Applications to Multicast Routing: An Integrated QoS Architecture. In Proc. *IEEE International Conference on Communications (ICC'98)*, June 1998.
23. J. Moy. Multicast Extensions to OSPF. Internet draft, Network Working Group, September 1992.
24. K. Nahrstedt and J. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.

25. P. Steenkiste, A. Fisher, and H. Zhang. Darwin: Resource Management in Application-Aware Networks. Technical Report CMU-CS-97-195, Carnegie Mellon University, December 1997.
26. H. Topcuoglu, S. Hariri, D. Kim, Y. Kim, X. Bing, B. Ye, I. Ra, and J. Valente. The Design and Evaluation of a Virtual Distributed Computing Environment. *Cluster Computing*, 1(1):81–93, 1998.
27. D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. Request for Comments RFC-1175, November 1988.
28. J. Zinky, D. Bakken, and R. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1):55–73, 1997.