

QoS Routing in Hypercube Multicomputers *

JIE WU and FEI DAI

*Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{jie,fdai}@cse.fau.edu*

Received (received date)

Revised (revised date)

Communicated by (Name of Editor)

ABSTRACT

In this paper, we present a coding method to capture QoS information in hypercube multicomputers. The coding method is based on a localized algorithm where each node interacts with its neighbors to gather QoS information. Specifically, each node maintains a QoS vector where the k -th element represents the guaranteed QoS performance to a destination that is k hops away. The localized algorithm exhibits desirable properties of self-stabilizing, self-optimizing, and self-healing. Simulation results show that this coding method provides a good approximation of the minimum QoS value to a k -hop destination and, at the same time, uses a relatively small number of packets to propagate a change in link state (QoS value) compared with the classical distributed Bellman-Ford method.

Keywords: Approximation, hypercubes, link state, localized algorithms, QoS.

1 Introduction

Nowadays, more and more computer applications demand for quality of services (QoS) in communication subsystems [8,9]. In general, the quality of traffic that flows through a communication subsystem depends on *topology*, *routing*, *flow control*, and *switching*. Recently, a few routers with QoS provisioning have been proposed [2]. Most of the proposals focus on switching and/or flow control based on either the hardware solution [4] or the software solution [1]. In this paper, we focus on QoS routing based on a given regular topology. Although the focus is in the hypercube topology, we believe that the result obtained in this paper can be potentially extended to other multicomputers and cluster networks with regular topologies.

QoS in routing typically deals with constraints on the selected path. Assume $m(u, v)$ is the performance metric for the link (u, v) connecting host u to host v and a path $(u, u_1, u_2, \dots, u_k, v)$ is a sequence of links connecting u to v . Three types of constraints on the path are given in [5]:

*This work was supported in part by NSF grants CCR 9900646 and ANI 0073736.

1. *Additive constraints:* A constraint is additive if $m(u, v) = m(u, u_1) + m(u_1, u_2) + \dots + m(u_k, v)$. For example, the end-to-end delay $delay(u, v)$ is an additive constraint which is equivalent to the summation of delays at each link.
2. *Multiplicative constraints:* A constraint is multiplicative if $m(u, v) = m(u, u_1) \times m(u_1, u_2) \times \dots \times m(u_k, v)$. The probability $prob(u, v)$ for a packet to reach v from u is the product of individual link probabilities.
3. *Concave constraints:* A constraint is concave if $m(u, v) = \min\{m(u, u_1), m(u_1, u_2), \dots, m(u_k, v)\}$. The bandwidth $band(u, v)$ available along path from u to v is the minimum bandwidth among the links on the path.

Here we focus on QoS with concave constraints. The objective is to find a path that meets the given QoS requirement. Consider $m(u, v)$ as the link state of (u, v) , each node can collect global link state through global distribution. Then each source node can easily determine a path to a given destination that meets the QoS requirement. However, link state is a dynamic element and it needs to be flooded to the whole network whenever there is a change in link state. In this paper, we propose a coding scheme to approximate the minimum QoS value of paths to destinations in k hops. The approximation is based on a conservative approach by taking the advantage of both the unique topological properties of hypercubes and concave QoS constraints. Specifically, each node u is associated with a *QoS vector* $B(u) = (b_1, b_2, \dots, b_n)$, where $b_k (k = 1, 2, \dots, n)$ corresponds to the minimum QoS value to any node in k hops. This approach can also be considered as an extension to Wu's safety level [6] and safety vector [7] models for fault-tolerant routing in hypercube multicomputers. Note that in fault-tolerant routing, there are two levels of QoS for each link: up (1) and down (0).

The localized algorithm exhibits desirable properties of self-stabilizing, self-optimizing, and self-healing. Specifically, when the QoS value (such as bandwidth) of a link changes, the $B(u)$ of the affected node u reaches a new vector $B'(u)$ in a self-organized way (also called self-stabilizing) through interaction between u and its neighbors. The convergence from $B(u)$ to $B'(u)$ is done in an optimized way (self-optimization). In addition, when a link fails, the network can be self-reconfigured (self-healing) by adjusting the QoS values of the affected nodes. Simulation results in this paper show that this coding method provides a good approximation of the minimum QoS value to a k -hop destination and, at the same time, uses a relatively small number of packets to propagate a change of link state (QoS value) compared with the classical Bellman-Ford method. The latter feature is also called *localized maintenance*.

The rest of this paper is organized as follows: Section 2 presents the coding scheme using QoS vectors and a distributed QoS routing algorithm based on single QoS vector for each node. Section 3 proposes an extension to the single QoS vector scheme called *double QoS vector* scheme, which achieves higher guaranteed bandwidth through longer paths. Section 4 compares the performance of the single QoS vector algorithm and the distributed Bellman-Ford algorithm through simulation. Section 5 concludes this paper. Note that although in this paper we focus only

on QoS routing, in real applications QoS routing should be accompanied and to cooperate with QoS traffic control.

2 Model

Let u be a node in an n -dimensional hypercube (n -cube) and it is represented by n bits $u_1u_2\dots u_n$. There are 2^n n -bit combinations assigned to 2^n nodes in an n -cube with each node being assigned a distinctive address. Two nodes are neighbors if and only if their addresses differ in one bit position. u^i denotes the neighbor of u along dimension i . A dimension is called a *preferred dimension* with respect to destination v if the neighbor along the dimension is closer to v than the current node u ; otherwise, it is called a *spare dimension*. If u and v are separated by k hops, then there are exactly k *preferred neighbors* (i.e., neighbors along preferred dimensions) of u to v and $n - k$ *spare neighbors*. $B(u) = (b_1, b_2, \dots, b_n)$ is a QoS vector associated with u , where b_k is called the k -hop QoS element of u . b_k^i is called the k -hop QoS element of u 's neighbor along dimension i . $m(u, u^i)$ is the QoS value associated with the adjacent link of u along dimension i and such information is called *link state* information. Normally, link state information is discretized into multiple levels correspond to levels of QoS. Without loss of generality, 0 is the lowest level of QoS value. $rank_k(B)$ selects the k -th smallest element from a given vector B .

Definition 1 Elements of $B(u)$ are defined as follows:

$$b_1 = rank_1(m(u, u^1), m(u, u^2), \dots, m(u, u^n))$$

and

$$b_k = rank_k(\min\{m(u, u^1), b_{k-1}^1\}, \min\{m(u, u^2), b_{k-1}^2\}, \dots, \min\{m(u, u^n), b_{k-1}^n\})$$

for $k = 2, 3, \dots, n$.

In $B(u)$, b_k corresponds to the minimum QoS value to any node in k hops, assuming only preferred neighbors are eligible (i.e., only optimal paths are considered). $rank_k$ is used because u has exactly k preferred neighbors to any k -hop destination and it corresponds to the minimum of maximum QoS value among any k selected neighbors. Since each b_k is defined based on b_{k-1} 's of neighbors, b_k can be easily determined through an iteration process similar to the one used in determining safety vectors [7].

Knowing the specific location of the destination, we can polish the coded QoS value as follows: Let $op(b_k)$ be the minimum QoS value to a given destination in k hops, assuming that only preferred neighbors are eligible. Let $sub_op(b_k)$ be the minimum QoS value to a given node in k hops, assuming that exactly one detour is used. $sub_op(b_k)$ corresponds to the minimum QoS value of a sub-optimal path. A sub-optimal path consists of all except one preferred dimensions and, hence, its length is 2 more than the optimal one. The neighbors' status can be determined

through the relative location of the source and destination by an exclusive-OR operation on the source and destination addresses. In the result of the exclusive-OR operation, the dimension that has 0 corresponds to a spare dimension and the one that has 1 corresponds to a preferred dimension. Assume that PD is the set of preferred dimensions and SD is the set of spare dimensions, $op(b_k)$ and $sub_op(b_k)$ can be approximated based on b_{k-1} 's and b_{k+1} 's of neighbors, respectively.

$$op(b_k) = \max\{\min\{m(u, u^i), b_{k-1}^i\}, i \in PD\}$$

If $\max\{\min\{m(u, u^i), b_{k-1}^i\}\} < \max\{\min\{m(u, u^j), b_{k+1}^j\}\}$, where $i \in PD$ and $j \in SD$, then the sub-optimal path has a higher QoS value:

$$sub_op(b_k) = \max\{\min\{m(u, u^j), b_{k+1}^j\}, j \in SD\}$$

Note that the above expression gives only a restricted implementation of $sub_op(b_k)$: a spare neighbor is selected in the first hop from the source, and those selected in the remaining ones are preferred neighbors. Let $op_dim(b_k)$ and $sub_op_dim(b_k)$ be the corresponding dimensions for $op(b_k)$ and $sub_op(b_k)$, respectively. The route is selected as follows:

Routing process (to a destination that is k hops away from the current node)

(the current node is the source node):

- The guaranteed QoS value of a minimal path is $op(b_k)$ and the corresponding dimension $op_dim(b_k)$ is selected as the forwarding dimension if $op(b_k)$ meets the given QoS requirement; otherwise,
- if $sub_op(b_k)$ meets the given QoS requirement, the guaranteed QoS value of a sub-minimal path is $sub_op(b_k)$ and the corresponding dimension $sub_op_dim(b_k)$ is selected as the forwarding dimension; otherwise, the routing process fails (to find a QoS route).

(the current node is an intermediate node u):

- Always select a preferred dimension with the highest $\min\{m(u, u^i), b_{k-1}^i\}$, where $i \in PD$.

The approximation can be further enhanced if each node has 2-hop information. In this case, both b_1 and b_2 have accurate QoS information. In order to calculate b_k ($k > 2$), the approximation starts at b_2 . L levels of QoS (also called *bandwidth levels*) are used to represent value in QoS vectors. Figure 1 shows a sample 3-cube where QoS values are represented by 10 levels (from integer 0 to 9). The QoS associated with links are as follows: 00* (representing a link connecting 000 and 001) : 7, 01* : 1, 10* : 4, 11* : 8, 0*0 : 2, 0*1 : 1, 1*0 : 5, 1*1 : 9, *00 : 6, *01 : 7, *10 : 4, and *11 : 2. Table 1 shows the QoS estimates based on 1-hop neighborhood information while Table 2 shows the ones based on 2-hop neighborhood information.

Note that when $L = 2$, the corresponding systems represent hypercubes with faulty links with 0 representing a faulty link and 1 as a healthy link.

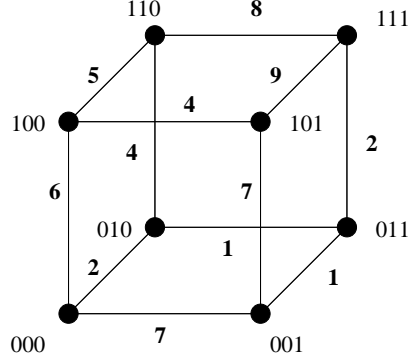


Fig. 1: A sample 3-cube.

	000	001	010	011	100	101	110	111
b_1	2	1	1	1	4	4	4	2
b_2	1	2	2	1	4	2	2	4
b_3	4	2	2	2	2	4	4	2

Table 1: QoS estimates (1-hop information).

3 Extensions

In the previous section, when the given QoS requirement cannot be satisfied by a minimal path, a sub-minimal path may be used to meet the QoS requirement, with the constraints that (1) there is only one detour, and (2) only the source node can select a spare dimension. While constraint (1) is important to avoid loop, constraint (2) can be relaxed to allow a detour at an intermediate hop; rather than just the first hop from the source. The following extension uses an extra QoS vector $D(u) = (d_1, d_2, \dots, d_n)$ for each node u , where d_k is called the k -hop QoS element with at most one detour. Similarly, d_k^i is called the k -hop QoS element with at most one detour of u 's neighbor along dimension i .

Definition 2 Elements of $D(u)$ are defined as follows: The QoS element with a immediate detour is

$$imm_k = \max\{\min\{m(u, u^1), b_{k+1}^1\}, \min\{m(u, u^2), b_{k+1}^2\}, \dots, \min\{m(u, u^n), b_{k+1}^n\}\}$$

for $k = 1, 2, 3, \dots, n-1$, and $imm_n = 0$. The QoS element with a possible deferred detour is

$$def_k = \text{rank}_k(\min\{m(u, u^1), d_{k-1}^1\}, \min\{m(u, u^2), d_{k-1}^2\}, \dots, \min\{m(u, u^n), d_{k-1}^n\})$$

for $k = 2, 3, \dots, n$, and $def_1 = 0$. For $k = 1, 2, 3, \dots, n$, the QoS element with at most one detour is

$$d_k = \max\{b_k, imm_k, def_k\}$$

	000	001	010	011	100	101	110	111
b_1	-	-	-	-	-	-	-	-
b_2	1	2	2	1	4	2	4	4
b_3	4	2	4	2	4	4	4	4

Table 2: QoS estimates (2-hop information).

In $D(u)$, d_k corresponds to the minimum QoS value to any node in k hops, assuming at most one detour is allowed in the entire path. imm_k and def_k are intermediate results and are not visible to u 's neighbors. Note that d_k can be determined once b_{k+1} 's of neighbors are determined in the iteration process. Also $d_n = \max\{b_{n-1}, def_{n-1}\}$ and, hence, only one additional round is needed to compute d_n . Given the specific location of the destination, set of preferred dimensions PD and set of spare dimensions SD , we can compute

$$\begin{aligned}
op(d_k) &= \max\{\min\{m(u, u^i), b_{k-1}^i\}, i \in PD\} \\
imm_sub_op(d_k) &= \max\{\min\{m(u, u^j), b_{k+1}^j\}, j \in SD\} \\
def_sub_op(d_k) &= \max\{\min\{m(u, u^i), d_{k-1}^i\}, i \in PD\}
\end{aligned}$$

Let $op_dim(d_k)$, $imm_sub_op_dim(d_k)$ and $def_sub_op_dim(d_k)$ be the corresponding dimensions for $op(d_k)$, $imm_sub_op(d_k)$ and $def_sub_op(d_k)$, respectively. Note that $imm_sub_op(d_k)$ takes a detour immediately at the source while $def_sub_op(d_k)$ may take a detour at a later hop. In the following extended routing process, each routing packet carries one bit *detour flag* which is initially set to be *false*:

Extended routing process (to a destination that is k hops away from the current node)

(when detour flag is *false*):

- If $op(d_k)$ meets the given QoS requirement, the guaranteed QoS value of a optimal path is $op(d_k)$ and $op_dim(d_k)$ is selected as the forwarding dimension; otherwise,
- if $def_sub_op(d_k)$ meets the requirement, the guaranteed QoS value of a optimal or sub-optimal path is $def_sub_op(d_k)$ and $def_sub_op_dim(d_k)$ is selected as the forwarding dimension; otherwise,
- if $imm_sub_op(d_k)$ meets the requirement, the guaranteed QoS value of a sub-optimal path is $imm_sub_op(d_k)$, $imm_sub_op_dim(d_k)$ is selected as the forwarding dimension, and the detour flag is set to *true*; otherwise, the routing process fails (to find a QoS route).

(when the detour flag is *true*):

- Always select a preferred dimension with the highest $\min\{m(u, u^i), b_{k-1}^i\}$, where $i \in PD$.
-

	000	001	010	011	100	101	110	111
d_1	4	2	2	4	4	4	4	2
d_2	2	4	4	2	4	2	2	4
d_3	4	4	2	2	2	4	4	2

Table 3: Extended QoS estimates (1-hop information).

Note that once the source node passes the QoS requirement, the routing process will not fail at an intermediate hop, assuming that the QoS values remain unchanged during the routing process. Table 3 shows the extended QoS estimates with at most one detour based on 1-hop neighborhood information of Figure 1.

4 Simulation

Simulations are conducted to compare the proposed QoS vector scheme with the classical distributed Bellman-Ford scheme, in terms of performance and overhead. The distributed Bellman-Ford scheme maintains at each node a complete routing table that includes id for each destination, the maximum available bandwidth, and the corresponding next hop information. The performance of a QoS routing scheme is its ability to satisfy a QoS routing request. The overhead includes the memory to store the QoS information and the control packets used to propagate such information over the network. Currently only the basic single QoS vector scheme with optimal and sub-optimal routing is simulated. The extended double QoS vector scheme that keep two QoS vectors $B(u)$ and $D(u)$ will be simulated as part of future work.

The hypercube used in the simulation are generated with the following bandwidth model: For a given hypercube dimension n , we build hypercubes with 2^n nodes and $n2^{n-1}$ bidirectional links. The bandwidth level of each link is an integer number between 0 and 99 (i.e., $L=100$). The residual (available) bandwidth of each link is equal to the maximum bandwidth minus the reserved bandwidth, which is the total bandwidth reserved by a set of independent QoS flows. We assume that the *central limit theorem* applies in this case and treat each bandwidth level as a random variable with Gaussian (normal) distribution, which is emulated by computing the average value of 12 random variables with uniform distribution (the convolution method [3]). Each simulation is repeated long enough to achieve a 90% confident interval of $\pm 10\%$.

The first set of simulations evaluates the performance of the single QoS vector scheme; that is, the maximum bandwidth can be guaranteed to a QoS routing request. For the single QoS vector scheme, we compute the QoS vector of each node, which comprises the estimated minimum bandwidth to any node in k hops ($k = 1, 2, \dots, n$), based on 1-hop or 2-hop neighborhood information. For the distributed Bellman-Ford scheme, we compute the maximum bandwidth (Max), the minimum bandwidth (Min) and the average bandwidth (Avg) to any node in k hops. Figure 2 shows the results on the 5-cube and the 10-cube. For destinations in k hops, the difference among Max_k , Min_k , and Avg_k is relatively small and

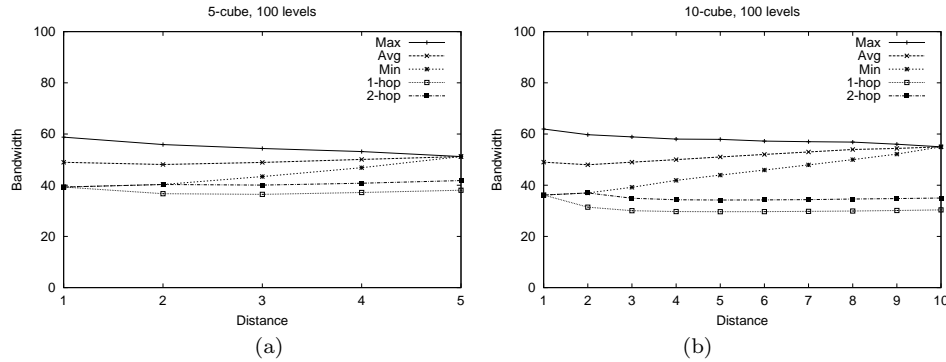


Fig. 2: The bandwidth guaranteed by the QoS vector scheme and the distributed Bellman-Ford scheme in (a) the 5-cube and (b) the 10-cube.

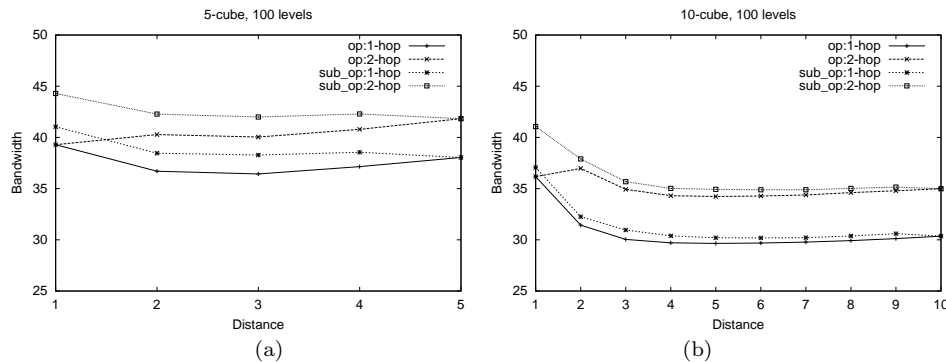


Fig. 3: The bandwidth guaranteed for the minimal paths and the sub-minimal paths by the QoS vector scheme in (a) the 5-cube and (b) the 10-cube.

becomes smaller as k increases. There is only one node that is n hops away, and therefore, $Max_n = Min_n = Avg_n$. Min is the upper bound for each QoS element, $b_1 = Min_1$ when 1-hop information is used; $b_2 = Min_2$ when 2-hop information is used. b_k is usually smaller than Min_k because the coded QoS information becomes inaccurate through approximation at each stage of propagation. As k increases, the difference becomes larger because the QoS information is less accurate for distant destinations. The situation is worse for the 10-cube, where more details are omitted by the QoS information coding scheme. However, this difference in performance is relatively small, and using 2-hop neighborhood information improves the situation.

Figure 3 shows the improvement achieved by allowing a detour at the source node. The average bandwidth guaranteed by a sub-optimal path ($sub_op : 1-hop$ and $sub_op : 2-hop$) is only slightly better than that guaranteed by an optimal path ($op : 1-hop$ and $op : 2-hop$). However, this improvement can be combined with the 2-hop information method, and is achieved without increasing the message complexity. Note that $op : 2-hop$ is better than $sub_op : 1-hop$ for $k \geq 2$, but is

Dimension	Bellman-Ford	QoS Vector
3	12.9	11.1
4	26.7	19.5
5	54.9	29.2
6	111.3	42.0
7	228.4	53.6
8	460.4	69.8
9	929.3	78.1
10	1867.2	81.7

Table 4: Average number of control packets used to propagate a change in link state.

slightly worse than *sub_op : 1-hop* for $k = 1$. That is because *op : 2-hop* cannot take the advantage of 2-hop information when the source and destination are directly connected. It is equivalent to *op : 1-hop* in this case.

Compared with the distributed Bellman-Ford scheme (each routing table contains 2^n entries), the single QoS vector scheme uses much less memory (each QoS vector contains n entries). Table 4 shows the same exponential-to-linear ratio for the message cost. Here we compare the average number of control packets caused by a single link state information change. After the computation of the bandwidth vectors (i.e., routing table in the distributed Bellman-Ford scheme and QoS vector in the single QoS vector scheme) converges in an n -cube, a link (u, v) is randomly selected to change its bandwidth level, and the bandwidth vectors of hosts u and v are re-computed. If the new bandwidth vector is different from the old one in any host, u or v , its neighbors are notified with n control packets containing the changes in the bandwidth vector. Any node receiving the control packet also needs to update its bandwidth vector and may send more control packets. This process repeats until the computation converges again. As the number of dimensions increases from 3 to 10, the average number of control packets per link state change increases exponentially in the distributed Bellman-Ford scheme; while the same metric increases linearly in the single QoS vector scheme. That is, the single QoS vector scheme has significantly less communication cost. We expect the same amount of control packets for the double QoS vector scheme with a better QoS estimation. However, its effectiveness still needs to be verified through simulation.

5 Conclusion

In this paper, we have proposed an approximation scheme to capture the minimum QoS value to support routing in hypercube multicomputers. The approximation method takes the advantage of the unique topological properties of hypercubes and concave QoS constraints. In addition, the calculation of the approximation is localized. This approximation also exhibits the desirable property of localized maintenance when the link state (QoS value) changes. Preliminary simulation results are promising and show desirable cost-effectiveness compared with the classical distributed Bellman-Ford method. In-depth simulation, under the double QoS vector scheme and/or more sophisticated QoS constraints, will be our future work.

References

1. A. A. Chien and J. H. Kim. Approaches to quality of service in high-performance networks. In *Proceedings of the Parallel Computer Routing and Communications Workshop*, pages 98–108. Lecture Notes in Computer Science v.1417, Springer-Verlag, July 1997.
2. H. Eberle and E. Qertli. Switcherland: A QoS communication architecture for workstation clusters. In *Proceedings of the International Symposium on Computer Architecture*, pages 98–108, June 1998.
3. R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, Inc, 1991.
4. Myricom Inc. M3M-PCI64B network interface card. <http://www.myri.com/myrinet/PCI64/m3m-pci64b.html>.
5. B. Wang and C.-J. Hou. A survey on multicast routing and its QoS extension: Problems, algorithms, and protocols. *IEEE Network Magazine*, 14(1):22–36, Jan./Feb. 2000.
6. J. Wu. Reliable unicasting in faulty hypercubes using safety levels. *IEEE Transactions on Computers*, 46(2):241–247, Feb. 1997.
7. J. Wu. Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):321–324, Apr. 1998.
8. K. H. Yum, E. J. Kim, and C. R. Das. QoS provisioning in clusters: An investigation of router and NIC design. In *Proceedings of the International Symposium on Computer Architecture*, pages 120–129, June 2001.
9. K. H. Yum, A. S. Vaidya, C. R. Das, and A. Sivasubramaniam. Investigating QoS support for traffic mixes with the MediaWorm router. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, pages 97–106, Jan. 2000.