INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2003; **16**:951–973 (DOI: 10.1002/dac.635)

Two algorithms for multi-constrained optimal multicast routing

Kun-Cheng Tsai*,† and Chyouhwa Chen‡

Department of Electronic Engineering, National Taiwan University of Science and Technology, Taiwan

SUMMARY

Multimedia applications, such as video-conferencing and video-on-demand, often require quality of service (QoS) guarantees from the network, typically in the form of minimum bandwidth, maximum delay, jitter and packet loss constraints, among others. The problem of multicast routing subject to various forms of QoS constraints has been studied extensively. However, most previous efforts have focused on special situations where a single or a pair of constraints is considered. In general, routing under multiple constraints, even in the unicast case is an NP-complete problem. We present in this paper two practical and efficient algorithms, called multi-constrained QoS dependent multicast routing (M_QDMR) and (multicasting routing with multi-constrained optimal path selection (M_MCOP)), for QoS-based multicast routing under multiple constraints with cost optimization. We provide proof in the paper that our algorithms are correct. Furthermore, through extensive simulations, we illustrate the effectiveness and efficiency of our proposals and demonstrate their significant performance improvement in creating multicast trees with lower cost and higher success probability. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: routing; multicast; quality of service (QoS); Steiner tree; M_MCOP; M_QDMR

1. INTRODUCTION

As the Internet becomes more and more widely used as a communication medium, many interesting multimedia applications are being introduced. Applications such as video-conferencing, multimedia broadcasting, and distance learning, are multicast in nature and network QoS-sensitive, and can benefit greatly from QoS guarantees provided by the underlying communication network. For example, a voice over IP application typically can tolerate a maximum transfer delay of 100–150 ms. For live video distribution applications, the maximum transfer delay is typically specified as 250 ms [1]. Other QoS parameters include bandwidth, bit error rate, packet error rate, delay jitter, and so on. Users enjoy increased satisfaction when QoS guarantees are met. On the other hand, providing QoS guarantees requires extensive support from the network infrastructure. For example, the network must incur increased routing

*Correspondence to: Kun-Cheng Tsai, Department of Electronic Engineering, National Taiwan University of Science and Technology, Taiwan.
†E-mail: garytsai.tw@yahoo.com.tw
‡E-mail: cchen@et.ntust.edu.tw

protocol overhead when trying to find feasible paths around congested links. Routers must store the discovered routing state in their internal storage space, thus increasing the storage overhead when QoS sensitive applications are to be supported. Furthermore, in the case of multicasting, the routing actions must create multicast trees that allow as much link sharing as possible since each link represents an additional set of network resources, such as storage and bandwidth, used to support the application. Designing practical and efficient routing algorithms that can find multicast trees that consume the least amount of network resources is thus of much interest to the network research community. Since there may be various kinds of overhead associated with using a link, usually a single abstract *cost* is assigned to a link to denote the combined expense an application or the network must spend to use that link. The routing actions must then create multicast trees with the lowest total combined cost, while satisfying the QoS requirements imposed with high success probability

This general problem of routing subject to various forms of QoS constraints has been studied extensively [2–6]. In this paper, we study the general problem of finding optimal multicast trees subject to *multiple* QoS constraints, and present two practical and efficient algorithms for solving the problem. We focus on additive constraints only, because it is well known that non-additive ones can be easily dealt with by using a preprocessing step to prune all links that do not satisfy the non-additive QoS constraints [2, 7].

For our network model, we adopt the notations similar to those in Reference [7]. A network is represented by a directed graph $G = (V, E)$, where V denotes the set of nodes and E denotes the set of directed links, (u, v), for $u, v \in V$. A primary cost function $c(u, v)$ associates each link $(u, v) \in E$ with a value representing the cost of using the link. In addition, K non-negative additive QoS link weight parameters $w_k(u, v),\ k = 1, 2, \ldots, K$ are associated with each link; for example, the link delay associated with the link. The notation $p(u, v)$ denotes the simple path from node u to v. For path $p(u, v)$, the $w_k()$ function is extended to denote the cumulative link weight of the kth QoS link weight parameter $w_k(p(u, v)) = \sum_{(i,j)\in p(u,v)} w_k(i, j)$, and will be abbreviated as the kth QoS path weight associated with path $p(u, v)$. Since in our setting, there is only one source node, $p(v)$ is used to mean the path from source s to v. A multicast group is denoted by the set $G = \{\{s\} \cup D\} \subseteq V$, where s is a designated source node and $D = \{m_1, m_2, \ldots, m_M\}$ are M destination nodes.

Definition 1
A multicast tree $T(s, D) \subseteq E$ is a tree rooted at s and spanning all members of D. The path $p(m)$ is the set of links in T that leads from s to $m \in D$. We consider only simple paths, i.e. those without loops, in this paper.

Definition 2
Given K constraint bounds $c_k,\ k = 1, \ldots, K$, a multicast tree T is feasible if

$$w_k(p(m)) = \sum_{(u,v)\in p(m)} w_k(u, v) \leqslant c_k$$
$$\text{for all } m = 1,\ 2, \ldots, M;\ k = 1, 2, \ldots, K \tag{1}$$

Definition 3
Multi-constrained optimal multicast problem (MCOM): Given K constraint bounds c_k, $k = 1, 2, \ldots, K$, the problem is to find a feasible multicast tree T, such that its tree cost

$$Cost(T) = \sum_{(u,v)\in T} c(u, v) \tag{2}$$

is minimized over all feasible trees.

Of interest in unicast routing algorithm design is the simpler multi-constrained optimal path problem (MCOP), where the general multicast tree requirement is restricted to be a single path. This MCOP problem, even when K is restricted to 1, is known as the NP-complete restricted shortest path (RSP) problem [7]. Solutions for the general MCOP problem, for the case $K \geqslant 2$, have only recently been explored in Reference [7]. In the multicast setting, if the requirement for optimal tree cost is dropped, the MCOM problem is known as the multi-constrained multicast (MCM) problem [3]. When the number of constraints $k = 0$, the MCOM problem reduces to the classical Steiner tree (ST) problem, i.e. that of finding the minimal cost tree spanning a given subset of nodes in a graph [6], which has been shown to be NP-complete [8]. For $k = 1$, the MCOM problem has been termed the constrained Steiner tree (CST) problem, and is of much interest due to its applications in delay-constrained multicast routing algorithm design. In the following, we briefly describe related work for solving the multicast routing problem.

1.1. The single constraint Steiner tree problem

The constrained Steiner tree (CST) problem is particularly important due to its practical utility in computer networks. Solutions for the problem must find a least-cost multicast tree that satisfies a given delay upper-bound constraint [9]. Many heuristics have been proposed, such as Zhu's BSMA [4], Widyonon's CBF [10], Kompella's KPP [9], Sun's CKMB [13] and Ibrahim's QDMR [2]. Both the KPP and the CKMB algorithms are based on the famous KMB [11] heuristic with extension for the end-to-end delay constraint. The KPP is quite computation intensive, with complexity on the order of $O(\Delta n^3)$, where Δ is the given delay bound. Most of the computing time is spent to compute a closure graph. The edges in the closure graph are the cheapest constrained paths between all nodes in the original graph. The KPP then utilizes two heuristic functions to select edges in multicast tree construction. One is for minimizing the cost of tree and the other uses the quotient of cost and the residual delay. Simulation results in Reference [12] show that the cost performance of CKMB is better than of KPP, while the running time of KPP is larger than that of CKMB. The computing complexity of CKMB is $O(mn^2)$, where m is the number of destinations in the multicast group. Widyono presented another heuristic in Reference [10], which uses the constrained Bellman-Ford (CBF) algorithm. The computing complexity of this algorithm is exponential to the size of the network. Zhu *et al.* [4] presented the Bounded Shortest Multicast Algorithm, (BSMA), which starts by computing a shortest-path tree with respect to delay using Dijkstra's algorithm. The algorithm then proceeds by iteratively refining the selected tree to reduce its cost. The computing complexity of BSMA is $O(kn^3 \log(n))$, where k is the average number of the k-shortest paths constructed to obtain the delay-bounded shortest path and n is the number of nodes in the network.

Another more effective class of heuristics for the CST problem is based on the efficient Dijkstra's shortest-path algorithm. For example, the QDMR [1] algorithm, based on Shaikh

and Shin's destination-driven multicasting (DDMC) algorithm [12], has been shown to be an effective heuristic solution for solving the unconstrained multicast routing problem. The idea of DDMC is based on an observation on the Dijkstra's algorithm. The relaxation process of Dijkstra's algorithm is modified by using a new cost function, as shown in Equation (3), for determining the cost of connecting a new node $v \notin T$ via node $u \in T$ onto the tree.

$$\text{Cost}[v] = I_{\text{D}}(u)\text{Cost}[u] + C(u, v) \tag{3}$$

The indicator function $I_{\text{D}}(u)$ is 0 if u is a destination node, and is 1, otherwise. The motivation behind the cost function is the observation that multicast tree cost is different from the shortest path cost from the source node. When u is a destination node, since the cost of the path to node u in the multicast tree is already included in the final multicast tree, the cost to node v does not need to doubly include the cost to node u. This cost function definition makes nodes close to destination nodes targets to be preferentially connected onto the tree. The result is an efficient shortest path-based algorithm that can create multicast trees with minimized final tree cost. For unconstrained multicast problem, DDMC is an efficient algorithm to create low-cost multicast trees. However, the DDMC approach may generate trees in which some destination nodes are connected to the source node via exceedingly long paths. In a delay-constrained environment, the long paths often violate the stated QoS constraints if the idea of DDMC is applied without modification. QDMR [2] suitably modifies the DDMC algorithm to derive an approach that can be used in a delay-constrained environment. The QDMR algorithm comprises a construction phase, a merging phase and a pruning phase. In the construction phase, the algorithm is similar to the DDMC algorithm, but with the indicator function replaced by the ratio between the cumulative delay from the source to node u and the delay bound, when u is a destination node. In this manner, node v would be preferentially considered if node u is a destination node, and the cumulative delay to it is small. However, this initial construction phase may not find feasible path for all nodes. For each un-covered destination, QDMR simply merge the least-delay path from the source to that node onto the tree. Finally, the algorithm prunes off non-destination leaves in the multicast tree. A nice property of the QDMR algorithm is it is guaranteed to find a feasible path for each destination node if there is one, and always constructs a delay-bounded multicast tree if such a tree exists. The time complexity of QDMR is $O(e\log(n))$, where e is the number of the links in the graph. It is orders of magnitude faster than the KMB class of algorithms, yet constructs trees with comparable, and sometimes slightly better multicast tree cost.

1.2. The multiple constraint Steiner tree problem

For the MCOM problem where $K \geqslant 2$, there has not been any work. Our work represents the first attempt in solving the problem. A related proposal for solving the MCM problem for the $K \geqslant 2$ case is the MAMCRA algorithm [3], which tries to find *feasible* multi-constrained multicast trees. The algorithm involves two phases. In the first phase, the SAMCRA algorithm [14] is applied to find a set of feasible *unicast paths*. In the second phase, MAMCRA constructs the multicast tree by merging the shortest paths, found in the first phase, one by one to build the multicast tree, eliminating loops in the process. The worst-case complexity of MAMCRA is $O(kN\log(kN) + k^2mE)$ for part one of MAMCRA, and $O(Np^2)$ for part two, where $k = k_{\max}$ is the maximum number of the unicast paths maintained at each node, N is the number of nodes in the network, m is the number of the weights, E is the number of the links in the network and p is

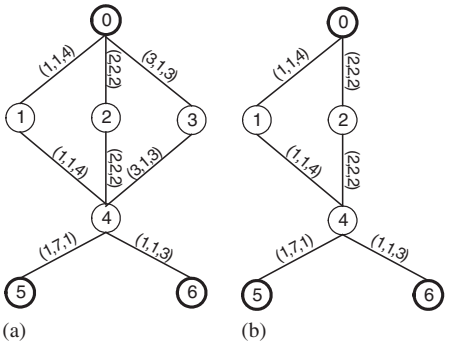

Figure 1. Example graphs illustrating MAMCRA. For each link, (c, w_1, w_2) denotes its cost and two weights. The constraint bounds are $(c_1, c_2) = (10, 10)$. (a) Original network and (b) the tree generated by MAMCRA.

the number of the destination nodes. The problem with MAMCRA is that the multicast tree constructed may not be feasible even though the unicast QoS paths are, when each unicast path is considered in isolation. We observe that it is not always possible to construct a feasible multicast tree out of a set of feasible unicast paths. The reason is that links not in the discovered optimal unicast paths to the individually considered destinations may be critical for satisfying feasibility constraints when multiple destinations are considered together. Consider the topology in Figure 1(a) as an example of the input graph. The source node is node 0, and destinations are nodes $\{5, 6\}$. The unicast paths determined by MAMCRA after the first phase are shown in Figure 1(b). Since the resulting graph is not a tree, the second phase of MAMCRA must extract a feasible multicast tree from the graph. To make the graph in Figure 1(b) into a tree, one of the two paths connecting the source to node 4 must be selected and the other one discarded. It can be easily verified that no matter which path is chosen, the resulting tree would violate the constraint for one of the two destination nodes.

A second problem with MAMCRA is that it is not adaptive to constraint *looseness*. When the expected values of the link weights are much smaller than constraint bounds c_k, the constraints are *loose*; otherwise, they are *strict*. When constraints are loose, many feasible paths exist, and it pays to focus on cost optimization, rather than feasibility checking. However, when constraints are loose, MAMCRA still maintains all feasible paths, and fails to take advantage of the situation when finding feasible multicast trees is the easiest.

However, of interest in the multi-constrained context is the non-linear cost measure for paths in MAMCRA and the related H_MCOP [7] algorithms. The path cost function is defined as follows:

$$g_\lambda(p) = \sum_{1 \leqslant k \leqslant K} \left(\frac{w_k(p)}{c_k} \right)^\lambda \tag{4}$$

where $w_k(p)$ denotes the kth path weight of path p. Properties of the non-linear length measure $g_\lambda(p)$ have been discussed in References [7, 14]. Briefly, an algorithm that searches for a path p by minimizing this cost function has a higher success probability for finding a feasible path subject to the constraints. Note that the function $g_\lambda(p)$, $\lambda = \infty$ is equivalent to the function $h(p)$ in (5) [7].

$$h(p) = \max\left\{\frac{w_k(p)}{c_k}\middle| 1 \leqslant k \leqslant K\right\} \tag{5}$$

That is, $h(p)$ is the maximum value of p's K path weights, each normalized by the individual constraint bounds. From the computational aspect, the non-linearity of the function renders invalid the Optimal Substructure principle relied on by the shortest path, and dynamic programming algorithms [15]. Therefore, efficient manner techniques, such as divide and conquer, cannot be used to find paths that minimize its cost. However, $h(P)$ can be computed incrementally starting from the first node on p, and the g_1 function, i.e. $\lambda = 1$ in Equation (4), *can* be computed efficiently using the Dijkstra's algorithm.

H_MCOP [7] tries to solve the MCOP problem by using the $g_\lambda(p)$ function to guide its heuristic search procedure. Due to the non-linearity of the function, H_MCOP computes $g_\lambda(p)$ for each path p from the source to the destination only approximately by dividing the path into two subpaths p_1 and p_2. Path p_1 starts from the source to the current node u being examined, and p_2 starts from u to the destination. Only function g_1 is evaluated for p_2 for efficiency reasons. Since g_1 must be evaluated for all the subpaths from the node being explored to the destination, possibly repeatedly, a key feature of the algorithm is its first **Reverse_Dijkstra** phase, which, for each constraint k, pre-computes and stores the kth path weight in variable $R_k(u)$ from each node u to the destination node m along the least cost path with respect to the g_1 function. In this way, $R_k(u)$ contains feasibility information, albeit not exactly, from each node u to the destination node. The **Reverse_Dijkstra** procedure is a slight modification of the Dijkstra's procedure [8]. Whereas the Dijkstra's procedure builds a directed, shortest-path out-tree to every other node rooted at the source by scanning the outgoing links when examining a node, **Reverse_Dijkstra** builds a directed shortest-path in-tree rooted at the destination by scanning the incoming links when examining a node. The second, **Look_Ahead_Dijkstra**, phase of H_MCOP is then a modified Dijkstra's forward search procedure that attempts to find the least cost path with high feasibility. With $R_k(u)$ computed, it is possible to assess approximate feasibility of a path by summing the normalized kth path weight from source s to u, together with the pre-computed $R_k(u)$. This pre-computation idea is extended to the multicast case in our algorithms. Since there are multiple destination nodes in our case, we need to maintain the equivalent of $R_k(u)$ in H_MCOP for each destination node m. We add an additional superscript and use $R_k^m[u]$ to denote the kth path weight from node u to destination m along the least g_1 cost path.

Motivated by the previous successful algorithms based on the Dijkstra's approach, we propose the M_QDMR, and M_MCOP algorithms, which are both efficient and effective for solving the MCOM problem. M_QDMR has been designed with efficiency in mind, while the focus of M_MCOP is on maximizing success probability. We provide theoretical proof that the algorithms are correct, and using extensive simulations, we show that both of our proposals can create multicast trees with much lower cost and higher success probability than MAMCRA. In addition, the computation time of M_QDMR is also substantially faster than MAMCRA.

The rest of this paper is organized as follows. Section 2 describes our algorithms. We compare the performance of M_QDMR, M_MCOP, and MAMCRA in Section 3. Finally, in Section 4 we present our conclusions and future work.

2. THE M_QDMR AND M_MCOP ALGORITHMS

The design of the algorithms is based on several key observations. First, while searching for the multicast tree, since no polynomial time algorithm can find the optimal tree to all destinations, a better strategy is to try to construct a tree to cover as many destinations as possible using an efficient heuristic algorithm, and then rely on a later patch up phase to attach yet uncovered destinations onto the existent tree, as in QDMR [2]. This idea motivates the overall architecture of both of our algorithms. Second, calculation of path weights from every node to the destination along minimal g_1 paths, as done in the first phase of H_MCOP, is a good compromise for estimating the feasibility of yet un-explored path segments [7]. Even though the computation cost for g_1 in the multicast setting is more expensive as the calculation must be performed for all destination nodes, the overall computation time still lies within acceptable bounds. Furthermore, an extension of the idea to computation of path weights of the subtree proves to be very important in the merging phase in both of our proposed algorithms, to be described shortly.

Therefore, architecturally, both of our proposed algorithms employ a two-phase design. First, a **Constrained_Dijkstra** phase is employed that starts from the source and seeks to create a partial multicast tree that covers as many destinations as possible. Second, since there may still be destination nodes not reachable via the partially constructed tree, for each uncovered destination, the strategy is to first to find a feasible path that connects the destination to the source, by applying a modified H_MCOP algorithm, and then graft the path onto the tree. Specifically, the second phase involves three procedures: **Reverse_Dijkstra**, **Look_Ahead_Dijkstra**, and **Merge** for each un-reached destination. Procedure **Reverse_Dijkstra** is first used to compute path weights from every node to the destination along minimal g_1 paths, and then procedure **Look_Ahead_Dijkstra** is used to actually search for a feasible path in a forward manner by starting from the source. After a feasible path is found, the **Merge** procedure is responsible for grafting each discovered feasible path onto the multicast tree. Overall, both M_QDMR and M_MCOP have high and comparable success probabilities when constraints are loose. M_MCOP can achieve 4–7% higher success probabilities when constraints are strict, at the expense of more computation time.

We summarize our contributions to the solution of the MCOM problem:

(1) Our algorithms are the first proposals to the general MCOM problem in the literature. From the experiments, we show that they are effective in solving the problem.
(2) The two-stage algorithm architecture is adapted from the idea first proposed in QDMR for the single constraint case, which is substantially simpler. For example, the second stage of QDMR adopts the obvious strategy of finding the shortest delay path and attaching it to the partial multicast tree. This strategy is completely inapplicable in the multiple constraint case.
(3) Application of the non-linear function $g_\lambda(p)$ for unicast routing under multiple constraints is first proposed by the authors in Reference [14], and later further explored in H_MCOP. We extend its application to the multicast case based on the summarization variable mechanism and prove the correctness of our strategy.

2.1. The M_QDMR algorithm

We now describe an overview of the phases of M_QDMR, followed by detailed pseudo-code descriptions. M_QDMR comprises of two main phases. The first phase, embodied in the

Constrained_Dijkstra procedure, seeks to create a partial multicast tree covering as many destinations as possible based on a Dijkstra-like algorithm. In **Constrained_Dijkstra**, an evaluation function applicable in the multi-constrained multicasting environment is needed. The evaluation function chosen is based on a modified $h(p)$ function as defined in Equation (5) to maximize feasibility. Let u be the node being examined, and the neighbour node v the one being relaxed in the relaxation procedure. The evaluation function is defined as follows:

$$d[v] = \begin{cases} d[u] \times h(u) + c(u,v), & \text{if } u \in D \\ d[u] + c(u,v), & \text{otherwise} \end{cases} \tag{6}$$

where $c(u,v)$ is the primary cost of link (u,v), and $h(u)$ denotes the value of $h(p(s,u))$. In the original Dijkstra's algorithm, $d[v]$ is interpreted to be the distance from source node s to node v. In Equation (6), the distance function is modified so that, when considering whether to reach node v via its neighbour u, $d[u]$ is weighted by $h(u)$ if node u is already a destination node. To understand the strategy implied by the evaluation function, consider the case when the QoS constraint consists of only a single additive constraint and $\lambda = 1$. The single additive constraint may be interpreted to be, for example, a delay constraint. In this case, $h(u) = g_\lambda(u) = w(p)/c$. If u is a destination node, and the cumulative delay up to u is small, the value of $d[v]$ is small accordingly. In other words, the further away the destination node u is from violating the delay bound, the more preferable its neighbours are considered for further exploration. Equation (6) may be viewed as an extension of the single constraint case to the multiple constraint case, only that in this case, the path weight closest to its constraint bound has the most influence. If u is not a destination node, Equation (6) reduces to normal Dijkstra's relaxation function.

Due to the heuristic nature of the procedure **Constrained_Dijkstra**, there may still be destination nodes not reachable via the partially constructed tree. The second M_QDMR phase is responsible for finding a feasible path for each uncovered node using a modified H_MCOP algorithm and then grafts the path onto the already constructed multicast tree. Even though H_MCOP can find feasible unicast paths subject to multiple QoS constraints, in a multi- constrained multicast routing problem, blindly grafting any feasible unicast path onto the partially constructed tree may violate previously established feasibility conditions. To guarantee the discovered unicast path is compatible with the already constructed portion of the multicast tree so that the path may be later merged into the tree, the H_MCOP algorithm is modified to consider an additional set of QoS constraints, in the form of summarization variables which represent the requirements imposed by the destination nodes on the already constructed multicast tree.

Finding a feasible path for a destination node in M_QDMR comprises of two steps: **Reverse_Dijkstra**, and **Look_Ahead_Dijkstra**. The **Reverse_Dijkstra** procedure finds the set of least cost paths with respect to g_1 from all other nodes u to m, and stores the results of the computation in the variable $R_k^m[u]$. The variable contains the kth QoS *path weight* along the least g_1 cost path from u to m, and is called the *look-ahead* variable, that enables the *look-ahead condition* to be examined in the relaxation process in **Look_Ahead_Dijkstra**, to be described next.

Look_Ahead_Dijkstra is a Dijkstra procedure that starts from the source s and searches for a feasible unicast path for a single destination node. Let the destination node being searched for be m, the current node being examined be u, the neighbour of u being relaxed be v. Let $G_k[u]$ denote the kth QoS *path weight* for path $p(s,u)$, and p be the path leading from the source to m by concatenating $p(s,u)$, link (u,v), and the minimal g_1 cost path from v to m. Using the look-ahead variable R_k^m, the following equation, called the *look-ahead condition*, serves two purposes

in the estimation of the feasibility of the path p.

$$G_k[u] + w_k(u,v) + R_k^m[v] \leqslant c_k, \quad \forall k = 1,2,\ldots,K \tag{7}$$

First, if the look-ahead condition is satisfied by the path $p(s,u)$ being examined, the path p is feasible according to the definition in Equation (1). If the look-ahead condition is not satisfied, it simply means $p(s,u)$ cannot be extended to reach the destination m via link (u,v) and the minimal g_1 path from v to m. However, there may still be other alternative paths for $p(s,u)$ to be extended to reach the destination, in which case, the second use of the look-ahead condition is in the approximate evaluation of the $h(p)$ function to estimate the feasibility of a path p using the following expression:

$$h(p) = \max\left\{ \frac{G_k[u] + w_k(u,v) + R_k^m[v]}{c_k} \middle| 1 \leqslant k \leqslant K \right\} \tag{8}$$

As discussed earlier, a path with a smaller $h(p)$ value is preferred since it has a higher probability of being a feasible path. Therefore, when the current neighbour node v has been visited before, Equation (8) is applied in **Look_Ahead_Dijkstra** to decide whether the previously explored path or the current path $p(s,u)$ is preferred as the best path to reach v.

An important aspect of **Look_Ahead_Dijkstra** is that the search must take care to ensure the path being searched does not violate the feasibility established in the already constructed multicast tree since the path must be merged with the already constructed tree. Consider the partially constructed multicast subtree T_u rooted at node u. A search effort planning to include u as an intermediate node must take into account of the fact that each destination node m already under T_u imposes the additional set of constraints because the path $p(u,m)$ require path weight $\sum_{(i,j)\in p(u,m)} w_k(i,j)$ for each of the k constraints. The combined feasibility requirements for all destination nodes under u are formalized using the constraint summarization variable $R'_k[u]$,[§] for each constraint k.

$$R'_k[u] = \max\left\{ \sum_{(i,j)\in p(u,m)} w_k(i,j) \mid m \in T_u \right\} \tag{9}$$

The variable $R'_k[u]$ stores the largest path weight for constraint k among paths to all destinations on the subtree T_u. $R'_k[u]$ is the extra constraints imposed at u by destinations under it. In the search process, only nodes u satisfying the following condition will be explored.

$$G_k[u] + R'_k[u] \leqslant c_k, \quad \text{for all } k = 1,2,\ldots,K \tag{10}$$

The inequality will be termed the *summarization condition* in the rest of the paper. Note that $R'_k[u]$ refers to the summarized path weight on the actual multicast tree, while $R_k^m[u]$ denotes the kth QoS *path weight* along the least g_1 cost path from u to m.

An extremely important property of $R'_k[u]$, which we prove formally later, is that a unicast path segment $\tilde{p}(s,u)$ discovered by **Look_Ahead_Dijkstra** can always replace the segment $p(s,u)$ in the partially constructed multicast tree. To elaborate, if a path $\tilde{p}(s,u)$ is returned by **Look_Ahead_Dijkstra** for m, $\tilde{p}(s,u)$ has the important property that, for every node u on $\tilde{p}(s,u)$, if u is in the original multicast tree, the path segment $\tilde{p}(s,u)$ can replace the original path $p(s,u)$

[§] Readers should note the difference between $R'_k[u]$ and $R_k^m[u]$. The latter refers to the path weight from u to m along least g_1 cost path, and is un-related to the summarization variable being defined here.

from s to u in the multicast tree without violating the QoS constraints. This property is utilized by the **Merge** procedure and makes the procedure remarkably simple.

In the following, we provide more details of the algorithm by describing the pseudo code of M_QDMR. The main procedure is shown in Figure 2. The main M_QDMR phases are: a **Constrained_Dijkstra** phase (lines 1–3), and a merging phase (lines 5–11).

The **Constrained_Dijkstra** phase performs a Dijkstra search to construct an initial partial multicast tree. The modified relaxation procedure is shown in Figure 3. The algorithm uses the definition of the distance function $d[v]$ as specified in (6) as the distance function for s to reach the destination nodes via u. Initially, we set $d[u] = \infty,\ \forall u \in V$. We maintain several variables in all phases of **Constrained_Dijkstra**. $G_k[u]$ denotes the path weight from source s to node u for constraint k. The predecessor of u on the partially constructed multicast tree is stored in the variable $\pi[u]$. During **Constrained_Dijkstra**, since there may be intermediate results stored in $\pi[u]$ that do not ultimately lead to nodes in the destination set, the **Prune_Leaves** procedure on line 3 removes those unfruitful nodes and branches from the partial multicast tree.

An important step in M_QDMR is line 4 in Figure 2. We set $c(u, v)$ to 0 for all on tree links on the partially constructed multicast tree. This step turns all on-tree nodes into *virtual* source nodes, having costs of zero between each other. This step makes all neighbouring nodes to the

M_QDMR ($G, s, D, R_k, k = 1,2,..., K$)
1. $T = \phi$
2. **Constrained_Dijkstra** ($G, s, D, R_k, k = 1,2,..., K$);
3. **Prune_Leaves** (T);// Prune non-destination leaves
4. Set $c(u,v) = 0$ for all links on T ;
   // Merging phase
5. for each destination m, $m \notin T$ {
6. **Reverse_Dijkstra** (G , m);
7. **Look_Ahead_Dijkstra** (G , T , s , m);
8. if ($G_k[m] \le c_k$, for all $k = 1,2,..., K$) {
9. **Merge** (T , π , $\tilde{\pi}$, m);
10. **Prune_Leaves** (T); }
11. else return failure; }
12. return T ;

Figure 2. The pseudo code of M_QDMR.

Constrained_Dijkstra Relax (u) // for M_QDMR
1. Let t be a temporary node
2. if $u \in D$ then $d[t] = d[u] \times h[u] + c(u,v)$
3. else $d[t] = d[u] + c(u,v)$
4. $G_k[t] = G_k[u] + w_k(u,v)$, $k = 1,2,..., K$
5. if ($d[t] < d[v]$ and $G_k[t] \le c_k$, $\forall k = 1,2,..., K$) {
6. $d[v] = d[t]$; $\pi[v] = u$;
   $G_k[v] = G_k[t], k = 1,2,..., K$; }

Figure 3. Relaxation procedure for **Constrained_Dijkstra** of M_QDMR.

Reverse_Dijkstra _Relax (m , u , v)

1. $temp = \sum_{k=1}^{K} \frac{R_k^m\ [v] + w_k\ (u,v)}{c_k}$; // $temp = g_1\ (p(v,m))$
2. if $r^m\ [u] > temp$ {
3. $r^m\ [u] = temp$;
4. $R_k^m\ [u] = R_k^m\ [v] + w_k\ (u,v)$, $k = 1,2,...,K$; }

Figure 4. Relaxation procedure for **Reverse_Dijkstra** based on minimizing $g_\lambda(p)$, $\lambda = 1$.

multicast tree preferable and is effective in minimize the final tree cost [2], in the same spirit of the DDMC algorithm. In lines 5–11, the algorithm performs the merging phase, which, for each destination node m not yet attached to the multicast tree, the algorithm searches for a feasible unicast path, and merges it into the partially constructed multicast tree. This phase further comprises of three main procedures: **Reverse_Dijkstra**, **Look_Ahead_Dijkstra**, and **Merge**.

The relaxation procedure for **Reverse_Dijkstra** is shown in Figure 4. Its aim is to compute the variable $R_k^m[u]$. The variable $r^m[u]$ stores the g_1 function value of the g_1 least cost path from node u to m. The procedure starts from the destination and works in directions opposite to those pointed to by the links. In the code, the current node being examined is v, with $R_k^m[v]$ already computed, and the link being relaxed is $(u, v) \in E$.

The pseudo-code for the relaxation procedure of **Look_Ahead_Dijkstra** is shown in Figure 5. The non-linear function $h(p)$ is used to increase the probability of finding a feasible path, whose value is stored in $g[u]$. The predecessor of u in the path $\tilde{p}(s, u)$ discovered by **Look_Ahead_Dijkstra** is stored in the variable $\tilde{\pi}[u]$. The procedure is conceptually similar to the procedure with the same name in Reference [7] since we are dealing with finding an optimal unicast path. On line 4, $h(p)$ is computed according to Equation (8). Line 5 computes the path weight from s to v.

The differences are line 2 in **Look_Ahead_Dijkstra**, which handles on-tree links, and lines 1–5 in **Prefer_the_best**, which deals with checking the validity of the summarization variable $R'_k[u]$. In **Prefer_the_best**, the procedure first checks whether either of the nodes satisfies the summarization condition. If neither satisfies the summarization condition, the search fails. Lines 6 and 7 deal with the case when both nodes satisfy the summarization condition, and one or both nodes satisfy the look-ahead condition. The procedure selects the node with lower cost to explore next, and thus optimizes for tree cost when multiple paths are feasible. Lines 8 and 9 deal with the case when neither node satisfies the look- ahead condition. In this case, the node with the lower $h(p)$ function value is preferred. Note that the $R'_k[u]$ needs to be computed only once before **Look_Ahead_Dijkstra** procedure, and the pseudo code is not shown in the paper due to its simplicity. Initially, we set $g[u] = \infty$, $\forall u \in V$. The variables $G_k[u]$, $k = 1, \ldots, K$, serve the same purpose as in the previous step, and $\tilde{\pi}[u]$ stores the returned feasible path.

The **Merge** procedure is shown in Figure 6, which, for every node u in the returned path $\tilde{p}(s, u)$ that is also in the multicast tree, simply replaces the predecessor of u in the multicast tree by that in the returned path. The correctness of **Merge** is formally established later.

2.2. The M_MCOP Algorithm

Since M_QDMR is a heuristic algorithm, it is not always possible to find feasible multicast trees when they exist. Even though the algorithm achieves excellent performance in constructing

Look_Ahead_Dijkstra_Relax (m , u , v)
1. Let t be a temporary node
2. if $(u,v) \in T$ then $d[t] = d[u]$;
3. else $d[t] = d[u] + c(u,v)$;
4. $g[t] = \max\left\{ \dfrac{G_k[u] + w_k(u,v) + R_k^m[v]}{c_k} \middle| 1 \le k \le K \right\}$;
5. $G_k[t] = G_k[u] + w_k(u,v)$, $k = 1,2,...,K$;
6. if (**Prefer_the_best** (t , v) = t) {
7. $d[v] = d[t]$; $g[v] = g[t]$; $\tilde{\pi}[v] = u$
8. $G_k[v] = G_k[t]$, $k = 1,2,...,K$ }

Prefer_the_best (t, v)
1. $t.feasible = \forall k \;\; G_k[t] + R'_k[t] \le c_k$? *true* : *false*;
2. $v.feasible = \forall k \;\; G_k[v] + R'_k[v] \le c_k$? *true* : *false*;
3. if ($t.feasible$ and $!v.feasible$) return t;
4. else if ($!t.feasible$ and $v.feasible$) return v;
5. else if ($!t.feasible$ and $!v.feasible$) return failure
6. if($d[t] < d[v]$ and $G_k[t] + R_k^m[v] < c_k, \forall k = 1,2,...,K$)
return t;
7. if($d[v] < d[t]$ and $G_k[v] + R_k^m[v] < c_k, \forall k = 1,2,...,K$)
return v;
8. if $g[t] < g[v]$ return t
9. else return v;

Figure 5. Relaxation procedure for **Look_Ahead_Dijkstra**.

Merge (T , π , $\tilde{\pi}$, m)
1. $u = m$; $p = \tilde{\pi}[u]$;
2. while ($u \ne s$){
3. $\pi[u] = p$;
4. $u = p$; $p = \tilde{\pi}[u]$; }

Figure 6. **Merge** procedure.

feasible multicast trees in general, when constraints are strict, the algorithm achieves only around 85% success probability, determined from empirical experiments to be discussed later. We note that in M_QDMR, the look-ahead condition is utilized only in the second stage of the algorithm, when a feasible path is to be found for a destination node not reached from the multicast tree constructed by the first stage **Constrained_Dijkstra** procedure. Since look-ahead condition is based on the g_λ function, which provably increases the probability of finding feasible paths in a multi-constraint environment, it seems natural to take advantage of that function when the first stage of the search effort fails to find a feasible path for a destination. A logical question to ask is whether it is fruitful to incorporate path feasibility information into the

first stage of our algorithmic architecture. In this section, we do just that and describe a refined version of M_QDMR, named M_MCOP, which is designed to further improve upon the success probability of M_QDMR in finding feasible multicast trees when constraints are more strict.

The M_MCOP algorithm incorporates g_λ path feasibility information into the **Constrained_Dijkstra** procedure as well. The first stage of our architecture, responsible for constructing a preliminary multicast tree, now comprises of two steps, as illustrated in the pseudo-code in Figure 7.

M_MCOP invokes the **Reverse_Dijkstra** procedure for every destination node to calculate the kth path weight $R_k^m[u]$ along minimal g_1 paths before the **Constrained_Dijkstra** procedure. Note that, for any destination node m, for any given $\lambda \geqslant 1$, if $g_\lambda(p) > K$, then $w_k(p) > c_k$ for all k's. Hence there is no feasible path for m in the graph [7]. Therefore, the algorithm checks whether $r^m[sK$ to determine if a feasible path exists at the end of this phase. Even though the computation of the path weights along minimal g_1 paths for all destinations is much more costly than in the unicast setting, we shall see later that the cost is still quite acceptable for the benefit.

M_MCOP also makes use of $R_k^m[u]$ this look-ahead information in its **Constrained_Dijkstra** procedure. The modified relaxation procedure for **Constrained_Dijkstra** phase is shown in Figure 8. The **Constrained_Dijkstra** phase performs the same forward Dijkstra's search to construct an initial multicast tree. However, the definition of the distance function $d[v]$ is changed to include the additional information gained from $R_k^m[u]$. Let u be the parent of $d[v]$ in the relaxation procedure, the revised definition of $d[v]$ is shown in (11).

$$d[v] = \begin{cases} d[u] \times h[u] + \dfrac{c(u,v)}{Count}, & \text{if } u \in D \\ d[u] + \dfrac{c(u,v)}{Count}, & \text{otherwise} \end{cases} \tag{11}$$

The variable, *Count*, contains an estimated number of *feasible* destinations reachable from node u through the link (u, v).

```
M_MCOP ( G, s, D, R_k, k = 1,2,..., K )
1. T = φ
2. for each destination m {
3.    Reverse_Dijkstra ( G , m );
4.    if r^m [s] > K , return failure; }
5. Constrained_Dijkstra ( G, s, D, R_k, k = 1,2,..., K );
6. Prune_Leaves ( T ); // Prune non-destination leaves
7. Set c(u,v) = 0 for all links on T ;
   // Merging phase
8. for each destination m, m ∉ T {
9.       Look_Ahead_Dijkstra ( G , T , s , m );
10.      if ( G_k [m] ≤ c_k , for all k = 1,2,..., K ) {
11.            Merge ( T , π , π̃ , m );
12.            Prune_Leaves ( T ); }
13.      else return failure; }
14. return T ;
```

Figure 7. The pseudo code of M_MCOP.

Constrained_Dijkstra Relax (u , v) //for M_MCOP
1. Let t be a temporary node
2. $Count = 0$; // number of reachable destinations
3. for each $m \in D, m \notin T$ {
4. if $G_k[u] + w_k(u,v) + R_k^m[v] \le c_k$, $\forall\ k = 1,2,...,K$
5. $Count = Count + 1$; }
6. if $Count > 0$ {
7. if $u \in D$ $d[t] = d[u] \times h[u] + \dfrac{c(u,v)}{Count}$;
8. else $d[t] = d[u] + \dfrac{c(u,v)}{Count}$; }
9. else $d[t] = \infty$;
10. $G_k[t] = G_k[u] + w_k(u,v)$, $k = 1,2,...,K$;
11. if($d[t] < d[v]$ and $G_k[t] + R_k^m[v] \le c_k$,
$\forall\ k = 1,2,...,K$) {
12. $d[v] = d[t]$; $\pi[v] = u$;
$G_k[v] = G_k[t]$, $k = 1,2,...,K$;}

Figure 8. Relaxation procedure for **Constrained_Dijkstra** of M_MCOP.

We now discuss in more detail the pseudo-code of the M_MCOP algorithm as shown in Figure 7. Lines 2–4 are the extra steps performed in M_MCOP in comparison with M_QDMR.

The computation of *Count* is performed using the look-ahead condition, as shown in lines 3–5 in Figure 8. Since the cost of the link is shared by all destinations under (u, v), the second term in (11) represents link cost $c(u, v)$ normalized by the number of potential destinations reachable via *Count*.

In the code in Figure 8, line 11 of the procedure is modified to include a look-ahead condition check. Owing to the efficacy of the early examination of the look-ahead condition, the number of uncovered destination nodes will decrease after the new version of **Constrained_Dijkstra**, leaving fewer nodes for **Look_Ahead_Dijkstra** to deal with. Therefore the success probability for finding feasible multicast trees is improved in M_MCOP.

2.3. An example

Figure 9 contains an example illustrating the operation of our algorithms. Starting from the same example, Figure 9(a) shows the original network. The constraint bounds are $(c_1, c_2) = (10, 10)$. The source node is node 0, and destinations are nodes $\{5, 6\}$.

Figure 9(b) shows the tree generated by M_QDMR and M_MCOP after the **Constrained_Dijkstra** phase. The tree does not cover node 6 because it violates the stated constraint $(c_1, c_2) = (10, 10)$. Without considering $R'_k[u]$, **Look_Ahead_Dijkstra** would have found path $\{(0, 2), (2, 4), (4, 6)\}$ to be the lowest cost feasible path. If the path is merged with the current tree, the tree shown in Figure 9(c) would have constructed. The tree cannot be made feasible by loop elimination. With $R'_k[u]$ taken into consideration, the feasible path $\{(0, 3), (3, 4), (4, 6)\}$ in Figure 9(d) is found. When it is merged with the tree in Figure 9(b), the tree in Figure 9(e) is obtained.

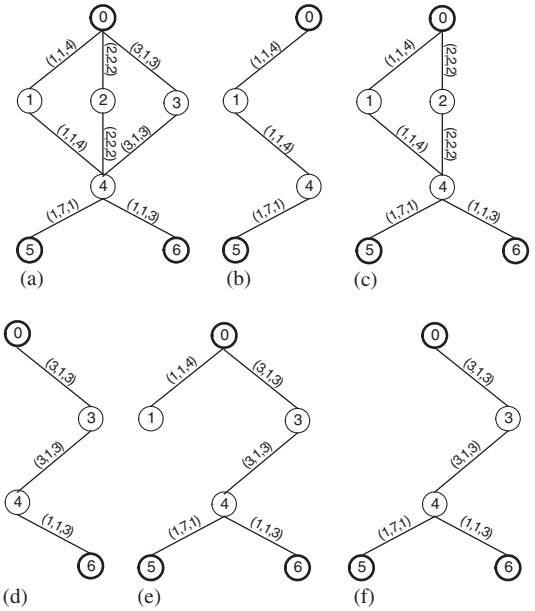

Figure 9. Example graphs illustrating M_QDMR and M_MCOP. For each link, (c, w_1, w_2) denotes its cost and two weights: (a) Original network, (b) the tree generated in the **Constrained_Dijkstra**, (c) the tree generated by M_MCOP if **Look_Ahead_Dijkstra** without R'_k, (d) the path that has been found by **Look_Ahead_Dijkstra** with the assistance of R'_k, (e) the tree generated by Merge procedure, and (f) the final tree generated by M_MCOP.

The **Merge** procedure simply replaces the predecessor of node 4 in the original tree (node 1) by that in the returned path (node 3) from the second stage. Pruning off the non-destination node 1 in the tree, the final multicast tree in Figure 9(f) is obtained.

2.4. Time complexity of M_QDMR and M_MCOP

If N denotes the number of the nodes, and E is the number of links in the graph $G(N, E)$; M denotes the number of destination nodes in the multicast group, and K is the number of the QoS constraints. The complexity of standard Dijkstra's algorithm is $O(N \log N + E)$ when using an adjacency list representation for the graph and using the set of nodes as a priority queue with Fibonacci Heap. The complexity of $O(N \log N)$ is for N times inserting and extracting minimum of Fibonacci Heap. The complexity of $O(E)$ is for e times relaxation of Dijkstra's algorithm. In M_QDMR, the complexity of **Constrained_Dijkstra** phase is $O(N \log N + EK)$ because the deference between standard Dijkstra and **Constrained_Dijkstra** is that our algorithm needs to compute $G_k[t] = G_k[u] + w_k(u, v)$ for $k = 1, \ldots, K$ and checks them with K QoS constraints in the relaxation phase. Besides, the M_QDMR has to do M times **Reverse_Dijkstra** and **Look_Ahead_Dijkstra** in the worst case. The complexity of each **Reverse_Dijkstra** is $O(N \log N + 2EK)$, where the complexity of $O(2EK)$ is for computing the values of $r^m[u]$ and $R_k^m[u]$. The complexity of each **Look_Ahead_Dijkstra** is $O(N \log N + 6EK)$, where the complexity

of $O(2EK)$ is for computing

$$g[t] = \max\left\{\frac{G_k[u] + w_k(u,v) + R_k^m[v]}{c_k}\middle| 1 \leqslant k \leqslant K\right\}$$

and $G_k[t]$ for $k = 1, 2, \ldots, K$; the complexity of $O(4EK)$ is for the **Prefer_the_best** procedure. The total complexity of M_QDMR algorithm is $O(M(2N \log N + 9EK))$ if the size of multicast group is far greater than one. Example of such applications include content-distribution network Akamai [16] and CNN news broadcasting.

The M_MCOP spends the extra time complexity $O(MEK)$ in computing the estimated number of the reachable destinations *Count* in the **Constrained_Dijkstra** procedure. So, the complexity of M_MCOP is $O(M(2N \log N + 10EK))$. In the simulation section, empirically, the computation of **Reverse_Dijkstra** and related modifications in M_MCOP results in substantial computational overhead increase. However, the extra computation required is relative to the extremely efficient Dijkstra's class algorithms, and is still quite acceptable in practical situations.

2.5. *Proof of correctness*

Theorem 1
If a feasible unicast path subject to the QoS constraints exists in the network, then there is at least one path p such that $g_\lambda(p) \leqslant K$, for any given $\lambda \geqslant 1$.

Proof
If there is a feasible path p in the network, then $(w_k(P)/c_k)^\lambda \leqslant 1$ for any a given $\lambda \geqslant 1$ in function (4) because $w_k(p) \leqslant c_k$ for all $k = 1, 2, \ldots, K$. Therefore $g_\lambda(p) \leqslant K$ if path p is feasible. In other words, no feasible path exists if $g_\lambda(p) > K$, establishing the validity of line 4 of M_MCOP.

Lemma 1
Let T denote the partially constructed tree after the **Constrained_Dijkstra** procedure. Given a destination node m, let $\tilde{p}$ be the returned path by **Look_Ahead_Dijkstra** procedure. Let node u be the node on $\tilde{p}$ closest to m that is also on the partially constructed multicast tree. Denote the subtree rooted at u on the partially constructed multicast tree by T_u. Then the multicast tree $T' = \tilde{p} \cup T_u$, formed by the union of links on path segment $\tilde{p}$ and those in T_u, is a tree, and all destinations in T' satisfy the original system constraints.

Proof
We first note that $\tilde{p}$ and T_u both satisfy the original system constraints. Consider any destination node m' in T_u and the path $p(u, m')$ in T_u form u to m'. During the **Look_Ahead_Dijkstra** procedure, the summarization variable $R'_k[u]$ is computed for T_u. Since $\sum_{(i,j)\in p(u,m')} w_k(i,j) \leqslant R'_k[u]$ according to Equation (9), and on line 1 and 2 in **Prefer_the_best** procedure, we made sure $G_k[u] + R'_k[u] \leqslant c_k$ for $k = 1, 2, \ldots, K$, therefore the path $\tilde{p}(s,u) \cup p(u,m')$ does not violate system constraints. Since this condition holds for all destinations in T_u, the lemma holds. To see that T' is a tree, we note that attaching a tree to a path by overlaying the root of the tree to any node on the path results in a tree, if the rest of the nodes in the tree and the path are disjoint. Therefore the fact that T' is a tree is obvious, since u is the closest node on $\tilde{p}$ that is also on T_u. No nodes are shared by $\tilde{p}(u,m)$ and T_u except u.

Theorem 2
Given $(T, \pi, \tilde{\pi}, m)$, **Merge** procedure creates a multicast tree that includes all destinations in the partially constructed multicast tree T, and satisfies the system constraints.

Proof
We first note that detaching a subtree from a tree T does not destroy the property of T being a tree. The **Merge** procedure loops from the lowest node in $\tilde{p}$ upward, ending at the source node s. For node u, $\pi[u]$ and $\tilde{\pi}[u]$ store the predecessor of u in the partially constructed multicast tree T and the path $\tilde{p}$ found by **Look_Ahead_Dijkstra**, respectively. The effect of line 3 of **Merge** is simply changing $\pi[u]$ to that of $\tilde{\pi}[u]$. If $u \in T$, line 3 effectively overlays the subtree rooted at T_u with the path $\tilde{p}$. Since the rest of the nodes in T_u and those in the path $\tilde{p}(u, m)$ are Disjoint, after the first loop, the created tree $T' = \tilde{p} \cup T_u$ and T are both trees. Furthermore, by Lemma 1, T' satisfies the system constraints. This step is performed so that the results of the final merged multicast tree are store in $\pi[u]$. By induction, each loop of the **Merge** procedure creates a tree by removing a portion of T and attaching it to $\tilde{p}$. Finally, we note that the procedure finishes by including the remaining subtree rooted at the source node. Thus, the **Merge** procedure creates a multicast tree that includes all destinations in the original partially constructed multicast tree.

Theorem 3
The tree generated by M_QDMR and M_MCOP is loop-free.

Proof
The tree generated in **Constrained_Dijkstra** phase is loop-free, since we simply apply the standard Dijkstra's algorithm with a different cost function. The algorithm visits each node in the network at most once. The **Look_Ahead_Dijkstra** procedure returns a unicast path. The **Merge** phase does not introduce loops into the tree because it simply replaces the predecessor of nodes in the constructed tree by that in the returned path of **Look_Ahead_Dijkstra** procedure.

3. PERFORMANCE EVALUATION

Many previous simulation studies adopt the Waxman's graphs [6] as their network topologies. However, one problem with the Waxman's graphs is that as the number of nodes increases, the mean degree of a node increases as well. This is not the case in real networks. We apply the network model proposed by Doar and Leslie [5] in our simulations since it creates topologies more similar to those in real networks. The edges were added to the graph using the probability function, as shown in (12).

$$P_e(u, v) = \frac{k\bar{e}}{|G|} \beta \exp \frac{-d(u, v)}{\alpha L} \tag{12}$$

In (12), the factor k is the scale factor related to the mean distance between two nodes to ensure that the mean degree of each node remains constant, $\bar{e}$ is the mean degree of a node, $d(u, v)$ is the distance between nodes u and v, and L is the maximum possible distance between any pair of nodes. The divisor $|G|$ is the number of nodes in the graph G. The parameters α and β, in the range $0 < \alpha$, $\beta \leqslant 1$, can be used to control various properties of the generated networks. A smaller value of α increases the density of shorter links relative to longer ones, while a larger

value of β gives nodes with a higher average degree. In our simulations, 100 nodes are distributed across a Cartesian co-ordinate plane with co-ordinates between $(0, 0)$ and $(100, 100)$ length unit. Therefore, the maximum possible distance L between any pair of nodes is 141.42 units in length. The parameters of the network topologies are set as follows: $\bar{e} = 4$, $k = 40$, $\alpha = 0.25$, and $\beta = 0.2$. Each link in the network is associated with one primary cost and four constraint weights. We investigated network topologies in which the link weights and primary costs exhibit various types of correlation between them and among themselves. Due to space limitations, in the following we present results for the no correlation case. Results for other types of correlations are presented in Reference [17]. In the no correlation case, the primary cost and link weight values are uniformly distributed random values between 0 and L. In this case, the average link weight values are $0.5L$. As for the constraint bounds, we introduce a notion of strictness of constraints on feasible multicast trees.

Definition 4
The average constraint strictness c, abbreviated as constraint strictness, is an input parameter associated with each problem instance. The constraint bounds for each constraint in the problem instance are drawn uniformly from the range $[0.9, 1.1] \times c \times L$.

Even though the link weights and constraint bounds are, strictly speaking, not necessarily required to be related to L, doing so links the notion of constraint strictness to the number of hops in the network naturally. Recall that when link weights are not correlated, the average link weight values are $0.5L$. Therefore, a constraint strictness of about $c = 1.7$ is very strict, since it gives a constraint bound of around $1.7L$, meaning that if a destination node is 4 hops away from the source node, almost no feasible multicast trees exist.

In additional to strictness of the constraints, results in all experiments are presented against densities of the destinations in the network. Let M be the number of destinations in the multicast group, and N be the number of nodes in the network, the destination node density is defined as M/N. This measure is an important indicator of how effective an algorithm is in creating low cost multicast trees. When destination node density is low, it is not easy to build trees that have much link sharing among the destination nodes. As destination density increases, however, it should be progressively easier to find multicast trees that use shared links to reach the destinations. This phenomenon is the source of multicasting gain. The results are now reported in the following figures, which are the averages of 400 runs.

3.1. Success probability analysis

In Figure 10, we show the results of the success probabilities versus destination node densities. The constraint strictness c_k are set to 2 for $k = 1, 2, \ldots, 4$. As delineated in the figure, the success probabilities of M_MCOP, and M_QDMR are significantly better than that of MAMCRA. As the destination node density increases, the success probabilities of all algorithms decrease because there are less and less feasible multicast trees in the network.

In Figure 11, we show the success probabilities versus constraint strictness between 1.7 and 2.6 with 50% destination node density. In contrast to Figure 10, the larger the constraint strictness is, the more successful the algorithms are because there are more and more feasible multicast trees in the network. The M_MCOP has the highest success probability for finding feasible multicast trees because we make use of more path feasibility information in

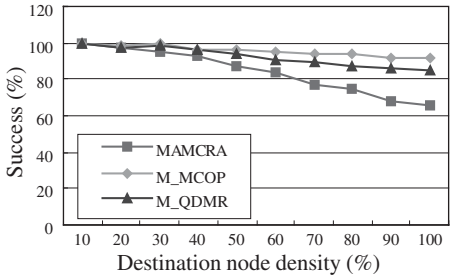

Figure 10. The success probability versus destination node densities in percentage relative to total nodes with constraint strictness equal to 2.

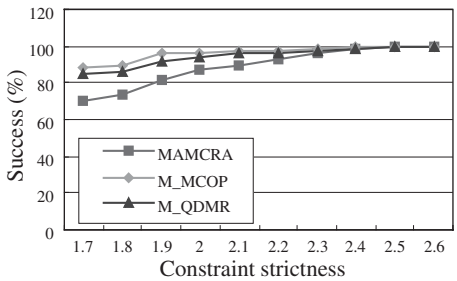

Figure 11. The success probability versus constraint strictness with 50% destination node density.

Constrained_Dijkstra to construct the multicast tree. The results of all simulations confirm that no matter what the constraint strictness is, the success probability of M_MCOP is the highest.

3.2. Multicast tree cost and computation time

The results of cost and computation time performance are shown in Figures 12–14, relative to the unconstrained Dijkstra's shortest-path algorithm, which represents the base 'simulcast' case when multicast is simulated using unicast connections. The tree cost for the base case is defined as the summation of individual unicast paths $\sum_{t \in D} \cos t(p(s,t))$. In the figures, solid lines denote multicast tree cost performance using the vertical axis on the left as their scale. Dotted lines denote computation time behavior of the algorithms using the vertical axis on the right as their scale.

In the figures, we show the results alongside the line denoted by $\hat{m}^{0.8}/m$, which denote the value of the function $M^{0.8}/M$, which is derived from the power law investigated by Chuang and Sirbu [18]. The power law function $L(M)$ gives an explicit formula for the average tree cost of a multicast tree to reach M randomly chosen network locations from any given source. Let M represent the multicast group size, L_u denote the average length of unicast routing paths, and L_M denote the total length of the multicast distribution tree that is the summation of edge costs of

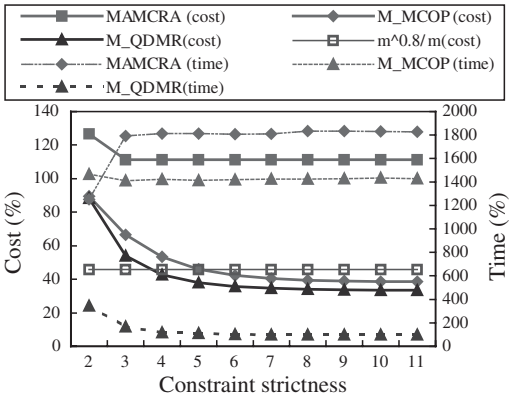

Figure 12. The cost and computing time versus constraint strictness with 50% destination node density.

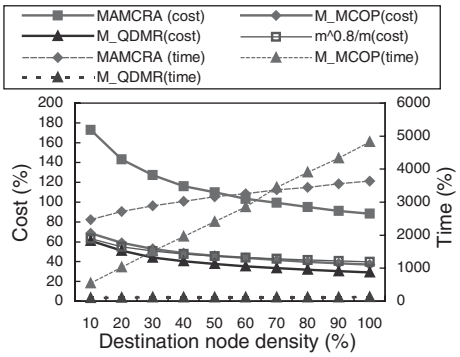

Figure 13. The cost and computing time ratios versus destination node density with constraint strictness equal to 5.

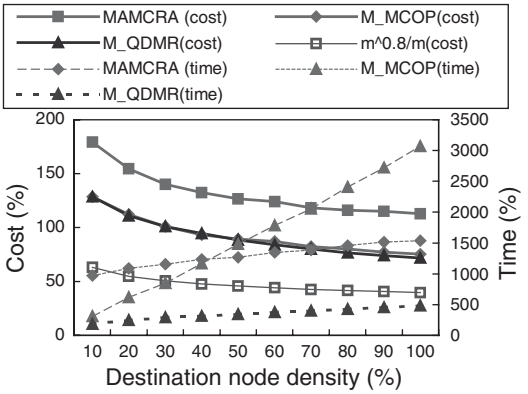

Figure 14. The cost and computing time ratios versus destination node density with constraint strictness equal to 2.

all links that make up the tree, power law is re-stated as follows:

$$\frac{L_M}{L_u} \propto M^{0.8} \tag{13}$$

Therefore, the tree cost ratio according to the power law would be calculated as follows:

$$\frac{L_M}{\sum_{t \in D} \cos t(p(s,t))} \propto \frac{L_u \times M^{0.8}}{M \times L_u} = \frac{M^{0.8}}{M} \tag{14}$$

In Equation (14), since the average cost of each shortest path is L_u, $\sum_{t \in D} \cos t(p(s,t))$ is reasonably approximated by $M \times L_u$. Therefore, $M^{0.8}/M$ is a reasonable approximation of the tree cost ratio as predicted by the power law.

From Figure 12 it is clear M_QDMR achieves the best tree cost performance (solid lines) across the spectrum of constraint strictness. When constraints are loose and destination node density is 50%, tree cost performance of M_QDMR is around 33% higher than that of Dijkstra's algorithm, with running time comparative to Dijkstra's. When constraints are strict, there are more un-reached destination nodes after the **Constrained_Dijkstra** procedure. For each unreached destination node, the algorithm has to do **Reverse_Dijkstra** and **Look_Ahead_Dijkstra** once. Therefore M_QDMR's computation time increases to about three times that of Dijkstra's. The computation time of M_MCOP is more expensive than that of M_QDMR because the algorithm has to execute **Reverse_Dijkstra** procedure for every destination nodes in advance to achieve higher success probabilities. It is worth noting M_QDMR requires progressively less computation time as constraints become loose, yet MAMCRA requires more and more time because it has to maintain more and more feasible paths. Notice the similarity of tree cost performance M_QDMR and M_MCOP to that exhibited by the power law curve in Figure 12.

The performance results of the algorithms with respect to destination node density are shown in Figures 13 and 14. Figure 13 shows the results for the case when constraints are loose, with a constraint strictness of 5. The performance results of M_QDMR and M_MCOP agree extremely well with the findings of the power law [18, 19] because the constraints are loose. For M_MCOP, even though it always constructs lower cost trees than MAMCRA, its computation time exceeds that of MAMCRA when the destination node density exceeds 70%, as computation time of M_MCOP increases linearly according to the number of destination nodes in **Reverse_Dijkstra**.

In Figure 14, the constraint strictness is fixed at 2, representing the case when constraints are strict. The algorithms construct trees with higher costs relative to those by Dijkstra's as destination node densities become smaller. However, we note that the trees constructed by the unconstrained Dijkstra's algorithm usually are not feasible in this case. Finally, we note that in all simulations in this case, MAMCRA does not exhibit any multicasting gain, since the tree cost of the multicast tree constructed is always more than that of the simulcast case, indicating there is almost no sharing of network links in the trees even when nodes are densely populated in the network. On the other hand, M_QDMR and M_MCOP exhibit the ability to achieve multicasting gain when link sharing opportunities arise as node densities increases.

Finally, we note that the in terms of absolute computation time, the various algorithms are quite efficient. For example, it takes M_MCOP only 21 milliseconds to run for the most time consuming case when destination node density is 100. They can be incorporated into general routing protocols easily.

4. CONCLUSIONS AND FUTURE WORK

Multicast routing with cost optimization subject to multiple constraints is an important and difficult problem. Most of existing QoS-based multicast routing algorithms deal with the problem under only one additive constraint, and they do not appear to have ready extensions to the multi-constraint case. The only algorithm known to us for this problem, MAMCRA, suffers from the problems of excessive computation overhead, and low success probability. Based on a number of key observations motivated by previous work, including MAMCRA, QDMR, and H_MCOP, we propose the M_QDMR and M_MCOP algorithms that appear to be both efficient and effective. M_QDMR is optimized for network resource efficiency, while M_MCOP is optimized for high success probabilities in finding feasible multicast trees. Both can construct multicast trees with substantially lower tree costs and higher success probabilities than previous algorithms. As future work, first, we intend to extend the non-linear function $g_\lambda(p)$ with respect to a tree and investigate its properties for use in a multicast setting. Second, we plan to investigate other search strategies in the **Constrained_Dijkstra** algorithm since it has the most dramatic impact on the quality of constructed multicast trees. We are also developing algorithms that are adaptive to the strictness of the constraints. In addition, we are interested in making M_MCOP work in a distributed manner so that it may be incorporated into a multicast routing protocol.

REFERENCES

1. Ferguson P, Huston G. *Quality of Service*. Wiley: New York, 1998.
2. Ibrahim M, Liang G. QDMR: an efficient QoS dependent multicast routing algorithm. *Real-Time Technology and Applications Symposium, 1999*. *Proceedings of the Fifth IEEE*, 1999; 213–222.
3. Kuipers F, Van Mieghem P. MAMCRA: constrained-based multicast routing algorithm. *Computer Communications* 2002; **25**:802–811.
4. Qing Z, Parsa M, Garcia-Luna-Aceves JJ. A source-based algorithm for delay-constrained minimum-cost multicasting. *INFOCOM '95,* Boston, MA, 1995; 377–385.
5. Doar M, Leslie I. How bad is naive multicast routing? In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April, 1993; 82–89.
6. Waxman BM. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications* 1998; **6**(9):1617–1622.
7. Turgay K, Marwan K. Multi-constrained optimal path selection. *Proceedings of the IEEE Infocom Conference*, Anchorage, Alaska USA, April, 2001.
8. Ahuja RK, Magnanti TL, Orlin JB. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Inc: Englewood Cliffs, NJ, 1993.
9. Kompella VP, Pasquale JC, Polyzos GC. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking* 1993; **1**(3):286–292.
10. Widyonon R. The design and evaluation of routing algorithms for real-time channels technical report. *ICSI* TR-94-024, *International Computer Science Institute*, U.C. Berkeley, June 1994.
11. Kou L, Markowsky G, Berman L. A fast algorithm for Steiner trees. *Acta Informatica* 1981; **15**:141–145.
12. Sun Q, Langendoerfer H. An efficient delay-constrained multicast routing algorithm. *Journal of High-Speed Networks* 1998; **7**(1):43–55.
13. Anees S, Kang GS. Destination-driven routing for low-cost multicast. *IEEE Journal on Selected Areas in Communications* 1997; **15**(3):373–381.
14. Mieghem PV, Neve HD, Kuipers F. Hop-by-hop quality of service routing. *Computer Networks* 2001; **37**(3–4): 407–423.
15. Horowitz E, Sahni S. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, MD, 1978.
16. Akamai Technologies, Inc. http://www.akamai.com/, 2003.
17. Tsai K-C, Chen C. *Effective multi-constrained optimal multicast routing technical report*. Department of Electronic Engineering, National Taiwan University of Education and Technology, Taiwan, 2002.

18. Phillips G, Shenker S, Tangmunarunkit H. Scaling of multicast trees: comments on the Chuang-Sirbu scaling law. *Proceedings of the ACM SIGCOMM*, September 1999.
19. Chuang J, Sirbu M. Pricing multicast communication: a cost-based approach. *Proceedings of the INET' 98*, Geneva, Switzerland, 1998.

AUTHORS' BIOGRAPHIES

Kun-Cheng Tsai received the MS degrees in Electrical Engineering from Chung Hua College, Taiwan in 1995. During 1997–2002, he stayed in Multimedia and Communications Research Laboratory of University of Science and Technology in Taiwan. He is now studying Real-time multimedia communication and wireless network systems.

Chyouhwa Chen is an associate professor at Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taiwan. He received his PhD degree from State University of New York at Stony Brook. His research interests include computer network, traffic management, multimedia communications, and issues in multicast routing.

