Unicast and Multicast Qos Routing with Soft Constraint Logic Programming

STEFANO BISTARELLI
University of Chieti-Pescara, Institute for Informatics and Telematics
UGO MONTANARI
University of Pisa
FRANCESCA ROSSI
University of Padova
and
FRANCESCO SANTINI
IMT Lucca - Institute for Advanced Studies, Institute for Informatics and Telematics

We present a formal model to represent and solve the unicast/multicast routing problem in networks with *Quality of Service* (QoS) requirements. To attain this, first we translate the network adapting it to a weighted graph (unicast) or *and-or* graph (multicast), where the weight on a connector corresponds to the multidimensional cost of sending a packet on the related network link: each component of the weights vector represents a different QoS metric value (e.g. bandwidth, delay, packet loss). The second step consists in writing this graph as a program in *Soft Constraint Logic Programming*: the engine of this framework is then able to find the best paths/trees by optimizing their costs and solving the constraints imposed on them (e.g. $delay \leq 40msec$), thus finding a solution to QoS routing problem. Soft constraints, and related *c-semiring* structures, prove to be a convenient tool to manage QoS costs and their compositions along the routes.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*; F.4.1 [**Mathematical Logic And Formal Languages**]: Mathematical Logic—*Logic and constraint programming*

General Terms: Languages, Measurement, Theory

Additional Key Words and Phrases: Quality of service, network routing, soft constraint logic programming

Author's address: Stefano Bistarelli, Dipartimento di Scienze, Università di Chieti-Pescara, Viale Pindaro 42, Pescara, 65127, Italy. `bista@unich.it.` - Institute for Informatics and Telematics, Via G. Moruzzi 1, 56100 Pisa, Italy. `stefano.bistarelli@iit.cnr.it.`
Ugo Montanari, Dipartimento di Informatica, Università di Pisa, Largo Bruno Pontecorvo 3, Pisa, 56127, Italy. `ugo@di.unipi.it`
Francesca Rossi, Dipartimento di Matematica Pura e Applicata, Università di Padova, Via Trieste 63 35121 Padova, Italy. `frossi@math.unipd.it`
Francesco Santini, IMT - Institute for Advanced Studies, Piazza San Ponziano 6, 55100 Lucca, Italy. `f.santini@imtlucca.it.` - Institute for Informatics and Telematics, Via G. Moruzzi 1, 56100 Pisa, Italy. `francesco.santini@iit.cnr.it.`

1. INTRODUCTION

Towards the second half of the nineties, Internet Engineering Task Force (IETF) and the research community have proposed many service models and mechanisms [Xiao and Ni 1999] to meet the demand for network *Quality of Service* (QoS). The reason is that traditional networks cannot recognize a priority associated with data, because they handle network traffic with the *best effort* principles. According to this treatment, the network does not provide any guarantees that data is delivered or that a user is assisted with a guaranteed QoS level or a certain priority (due to congestions). In best effort networks, all users obtain exactly the same treatment.

However nowadays, networked applications, such as Enterprise Resource Planning (ERP), data mining, distance learning, resource discovery, e-commerce, and distribution of multimedia-content, stock quotes, and news, are bandwidth hungry, need a certain "timeliness" (i.e. events occurring at a suitable and opportune time) and are also mission critical.

For all these reasons, the routing problem has naturally been extended to include and to guarantee QoS requirements [Younis and Fahmy 2003; Xiao and Ni 1999], and consequently is usually abbreviated to *QoS routing*. As defined in [Crawley et al. 1998], QoS is "a set of service requirements to be met by the network while transporting a flow", where a flow is "a packet stream from source to a destination (unicast or multicast) with an associated Quality of Service (QoS)". To be implemented and subsequently satisfied, service requirements have to be expressed in some measurable QoS metrics, such as bandwidth, number of hops, delay, jitter, cost and loss probability of packets.

This paper combines and extends the two works presented in [Bistarelli et al. 2002] and [Bistarelli et al.]. First, we detail the modelling procedure to represent and solve plain Shortest Path (SP) [Cormen et al. 1990] problems with *Soft Constraint Logic Programming* (see Section 2). We consider several versions of SP problems, from the classical one to the *multi-criteria* case (i.e. many costs to be optimized), from *partially-ordered* problems to those that are based on modalities associated to the use of the arcs (i.e. *modality-based*), and we show how to model and solve them via SCLP programs. The basic idea is that the paths represent network routes, edge costs represent QoS metric values, and our aim is to guarantee the requested QoS on the found unicast routes, by satisfying the QoS constraints and optimizing the cost of the route at the same time. The different criteria can be, for example, maximizing the global bandwidth and minimizing the delay that can be experienced on a end-to-end communication.

Then, extending the unicast solution, we suggest a formal model to represent and solve the multicast routing problem in multicast networks (i.e. networks supporting the multicast delivery schema) that need QoS support. To attain this, we draw the network adapting it to a weighted *and-or* graph [Martelli and Montanari 1978], where the weight on a connector corresponds to the cost of sending a packet on the network link modelled by that connector. Then, we translate the hypergraph in a SCLP program and we show how the semantic of this program computes the best tree in the corresponding *and-or* graph. We apply this result to find, from a given source node in the network, the multicast distribution tree having the minimum cost and reaching all the destination nodes of the multicast communication. The

costs of the connectors can be described as vectors (multidimensional costs), each component representing a different QoS metric value. We show also how modalities can be added to multicast problems, and how the computational complexity of this framework can reduced. Therefore, in this paper we present a complete formal model to represent and solve the unicast/multicast QoS routing problem.

As a quick reminder, SCLP programs are logic programs where each ground atom can be seen as an instantiated soft constraint [Bistarelli et al. 1995; 1997b] and it can be associated with an element taken from a set. Formally, this set is a *c-semiring* [Bistarelli 2004] (or simply, semiring), that is, a set plus two operations, $+$ and $\times$, which basically say how to combine constraints and how to compare them. The presence of these two operations allows to replace the usual boolean algebra for logic programming with a more general algebra where *logical and* and *logical or* are replaced by the two semiring operations. In this way, the underlying logic programming engine provides a natural tool to specify and solve combinatorial problems, while the soft constraint machinery provides greater expressivity and flexibility.

In our solution, we use SCLP for some important features of this framework: the first is that SCLP is a declarative programming environment and, thus, is relatively easy to specify a lot of different problems, ranging from paths to trees. The second reason is that the semiring structure is a very flexible and parametric tool where to represent several and different cost models, with respect to QoS metrics; obviously, the same SCLP programming environment and operational semantic engine can be used with all these different semirings. Finally, since QoS routing problem can be in general NP-Complete (Sec. 3), SCLP promises to be suitable tool, due to its ability for solving combinatorial problems.

1.1 Structure of the paper

The remainder of this paper is organized as follows. In Section 2 we describe the SCLP framework, while in Section 3 we complete the background by introducing the multicast/unicast QoS routing: we show that the problem of defining a route that has to be optimized and is subject to constraints concerning QoS metrics, is, in general, a NP-Complete problem. Then, we report some of the solutions, mostly through heuristics, given in the real world. Section 4 proposes how to model and solve the unicast QoS routing with SCLP, considering also problems with multidimensional costs (i.e. multi-criteria problems) and based on modalities of use associated with the links of the network (i.e. modality-based problems). Section 5 outlines a similar framework, based on hypergraph and SCLP, for the management of the multicast QoS routing: we show how to translate a network in a corresponding *and-or* graph and then we compute the best distribution tree by using SCLP. Even in this case we extend the model to include problems with modalities and, afterwards, we suggest how to mitigate the computational complexity of the model, with the adoption of *tabling* techniques. Section 6 gives some important considerations about semirings that improve the model when the costs of the network links are multidimensional and partially ordered: this is the common case, since an effective measurement of QoS will necessarily involve a collection of measures. At last, Section 7 ends the paper with the final conclusions and ideas about future work.

2. SOFT CONSTRAINT LOGIC PROGRAMMING

The SCLP framework [Bistarelli 2004; Bistarelli et al. 1997a; Georget and Codognet 1998], is based on the notion of *c-semiring* introduced in [Bistarelli et al. 1995; 1997b]. A semiring S is a tuple $\langle A, +, \times, 0, 1\rangle$ where A is a set with two special elements ($0, 1 \in A$) and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of A and thus is commutative, associative, idempotent, it is closed and 0 is its unit element and 1 is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, 1 is its unit element, and 0 is its absorbing element (for the exhaustive definition, please refer to [Bistarelli et al. 1997b]).

The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; we say that $a \leq_S b$ if b represents a value *better* than a. Other properties related to the two operations are that $+$ and $\times$ are monotone on $\leq_S$, 0 is its minimum and **1** its maximum, $\langle A, \leq_S\rangle$ is a complete lattice and $+$ is its lub. Finally, if $\times$ is idempotent, then $+$ distributes over $\times$, $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times$ its glb.

Semiring-based Constraint Satisfaction Problems (SCSPs) [Bistarelli 2004] are constraint problems where each variable instantiation is associated to an element of a c-semiring A (to be interpreted as a cost, level of preference, ...), and constraints are combined via the $\times$ operation and compared via the $\leq_S$ ordering. Varying the set A and the meaning of the $+$ and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness, probability, and optimization. Notice also that the cartesian product of two c-semirings is a c-semiring [Bistarelli et al. 1997b], and this can be fruitfully used to describe multi-criteria constraint satisfaction and optimization problems.

Constraint Logic Programming (CLP) [Jaffar and Maher 1994] extends Logic Programming by replacing term equalities with constraints and unification with constraint solving. The SCLP framework extends the classical CLP formalism in order to be able to handle also SCSP [Bistarelli et al. 1995; 1997b] problems. In passing from CLP to SCLP languages, we replace classical constraints with the more general SCSP constraints where we are able to assign a *level of preference* to each instantiated constraint (i.e. a ground atom). To do this, we also modify the notions of interpretation, model, model intersection, and others, since we have to take into account the semiring operations and not the usual CLP operations.

The fact that we have to combine several refutation paths when we have a partial order among the elements of the semiring (instead of a total one), can be fruitfully used in the context of this paper when we have an graph/hyoergraph problems with incomparable costs associated to the edges/connectors. In fact, in the case of a partial order, the solution of the problem of finding the best path/tree should consist of all those paths/trees whose cost is not "dominated" by others.

A simple example of an SCLP program over the semiring $\langle \mathbb{N}, min, +, +\infty, 0\rangle$, where $\mathbb{N}$ is the set of non-negative integers and $D = \{a, b, c\}$, is represented in Table I. The choice of this semiring allows us to represent constraint optimization problems where the semiring elements are the costs for the instantiated atoms. To better understand this Table, we briefly recall the SCLP syntax: a program is a set of clauses and each clause is composed by a head and a body. The head is just an

Table I. A simple example of an SCLP program.

```
s(X)   :- p(X,Y).
p(a,b) :- q(a).
p(a,c) :- r(a).
q(a)   :- t(a).
t(a)   :- 2.
r(a)   :- 3.
```

atom, and the body is either a collection of atoms, or a value of the semiring, or a special symbol ($\Box$) to denote that it is empty. Clauses where the body is empty or it is just a semiring element are called facts and define predicates which represent constraints. When the body is empty, we interpret it as having the best semiring element (that is, 1).

The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ (in Table I) is that $r(a)$ costs 3 units. Thus the set $\mathbb{N}$ contains all possible costs, and the choice of the two operations *min* and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimum cost overall. Given a goal like $s(x)$ to this program, the operational semantics collects both a substitution for x (in this case, $x = a$) and also a semiring value (in this case, 2) which represents the minimum cost among the costs for all derivations for $s(x)$. To find one of these solutions, it starts from the goal and uses the clauses as usual in logic programming, except that at each step two items are accumulated and combined with the current state: a substitution and a semiring value (both provided by the used clause). The combination of these two items with what is contained in the current goal is done via the usual combination of substitutions (for the substitution part) and via the multiplicative operation of the semiring (for the semiring value part), which in this example is $+$. Thus, in the example of goal $s(X)$, we get two possible solutions, both with substitution $X = a$ but with two different semiring values: 2 and 3. Then, the combination of such two solutions via the *min* operation give us the semiring value 2.

3. BACKGROUND ON QOS ROUTING

With *Constraint-Based Routing* (CBR) we refer to a class of routing algorithms that base path selection decisions on a set of requirements or constraints, in addition to destination criteria. These constraints may be imposed by administrative policies (i.e. *policy routing*), or by QoS requirements (i.e. QoS routing, as already cited in Section 1), and so they can be classified in two classes with different characteristics. The aim of CBR is to reduce the manual configuration and intervention required for attaining traffic engineering objectives [Rosen et al. 2001]; for this reason, CBR enhances the classical routing paradigm with special properties, such as being resource reservation-aware and demand-driven.

The routing associated with administration decisions is referred to as *policy routing* (or policy-based routing), and it is used to select paths that conform to administrative rules and *Service Level Agreements* (SLAs) stipulated among service providers and clients. In this way, routing decisions can be based not only on the destination location, but also on other factors such as applications or protocols

used, size of packets, or identity of the communicating entities. Policy constraints can help improving the global security of the network: constraints can be used to guarantee agreed service provisioning and safety from malicious users attempting to steal the resources not included in their contracts. Finally, the policy routing problem can be viewed as a resource allocation problem that includes business decisions.

QoS routing attempts to simultaneously satisfy multiple QoS requirements requested by real-time applications: the requirements are usually expressed using metrics as, e.g. delay and bandwidth. Policy routing (or policy-based routing) is instead used to select paths that conform to imposed administrative rules. In this way, routing decisions can be based not only on the destination location, but also on factors such as used applications and protocols, size of packets, or identity of both source and destination end systems of the flow. Policy constraints can improve the global security of network infrastructure and are able to realize business related decisions.

Traditionally, QoS metrics can be organized into three distinct classes, depending on how they are combined along a path: they can be *i) additive*, *ii) multiplicative* or *iii) concave* [Wang and Crowcroft 1996]. They are defined as follows: with $n_1, n_2, n_3 \ldots, n_i$,
n_j representing network nodes, let $m(n_1, n_2)$ be a metric value for the link connecting n_1 and n_2). For any path $P = (n, n_2, \ldots, n_i, n_j)$, the metric corresponding is:

—*Additive*, if $m(P) = m(n_1, n_2) + m(n_2, n_3) + ... + m(n_i, n_j)$ The additive metric of a path is the sum of the metric for all the links constituting the path. Some examples are delay, jitter (the delay variation on a network path), cost and hop-count.

—*Multiplicative*, if $m(P) = m(n_1, n_2) \times m(n_2, n_3) \cdots \times m(n_i, n_j)$ Multiplicative metric of a path consists in the multiplication of the metric values for all the links constituting the path. Example is reliability or loss probability.

—*Concave*, if $m(P) = \max / \min\{m(n_1, n_2), m(n_2, n_3), ..., m(n_i, n_j)\}$. The concave metric of a path is the maximum or the minimum of the metric values over all the links in the path. The classical example is bandwidth, meaning that the bandwidth of a path is determined by the link with the minimum available bandwidth, i.e. the bottleneck of the path.

Even if usually the metric classes are introduced for paths, most of times they can be suitable also for trees: consider, for example, if we need to find a global cost of the tree by summing up all the weights on the tree edges (i.e. additive), or if we want to maximize the bandwidth of bottleneck link (i.e. concave).

Given a node generating packets, we can classify network data delivery schemas into three main classes: *i) unicast*, when data is delivered from one sender to one specific recipient, providing one-to-one delivery, *ii) broadcast*, when data is instead delivered to all hosts, providing one-to-all delivery, and finally, *iii) multicast*, when data is delivered to all the selected hosts that have expressed interest; thus, this last method provides one-to-many delivery. Since our intention is to provide a model for reasoning on QoS routing (both unicast and multicast), we will concentrate on *i)* and *iii)*.

In the next two Sections we describe the background about unicast and multicast QoS delivery-schemas (respectively in Section 3.2 and Section 3.3). We list and discuss some of the related problems and proposed solutions in this field. In the following of the paper we will mainly focus over QoS routing, even if we will propose some considerations also for policy routing. In Section 3.1 we start by explaining why tractability is the primary challenge with QoS routing.

3.1 Two NP-Complete Problems

By composing together the notions of QoS routing and QoS metrics, we can deduce a very important outcome if we use multiple metrics to model and try to solve QoS routing: in general, this problem is a NP-Complete problem. A typical scenario involves resources that are independent and allowed to take real or unbounded integer values [Kompella and Awduche 2001]. For example, it could be necessary to find a route with the objective of cost minimization (i.e. a *quantitative* constraint) and subject to a path delay $\leq$ 40msec (i.e. a *boolean* constraint) at the same time, therefore we would have the set of constraints $C = (delay \leq 40, \min(cost))$. In such scenarios, satisfying two boolean constraints, or a boolean constraint and a quantitative (optimization) constraint is NP-Complete [Younis and Fahmy 2003]. If all resources except one take bounded integer values, or if resources are dependent, then the problems can be solved in polynomial time [Chen and Nahrstedt 1998]. Most of the proposed algorithms in this area apply heuristics to reduce the complexity, as we will see in Sections 3.2 and 3.3.

Multicast and unicast QoS routing can be reconducted to two well-known and more general problems: respectively, *Multi-Constrained Path* (MCP) [Korkmaz and Krunz 2001] and *Steiner Tree* (ST) [Winter 1987] problems, which, together with their variants and associated problems, we will define in the following of this Section.

Definition 3.1. *A Multi-Constrained Path can be found in a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Each link $(i, j) \in E$ is associated with k non-negative additive weights $w_k(i, j)$ for $k = 1, 2, \ldots, K$. Given K constraints c_k, $k = 1, 2, \ldots, K$, the problem is to find a path p, from source node s to destination node t, that simultaneously satisfy all the K constraints:*

$$\Sigma_{(i,j)\in p}\, w_k(i, j) \leqslant c_k, \quad for\ k = 1, 2, \ldots, K$$

Note that the metric composition function (Σ in Def. 3.1) can be a generic f, if we are using also other non-additive metrics; f can represent the $\times$ operation of the semiring structure (see Sections 4 and 5). The NP-Completeness conclusion can be reached also when we consider multiplicative metrics instead of only additive ones, or even mixing them [Wang and Crowcroft 1996] together. Constraints on concave measures can be treated, for instance, by omitting all links (and possibly disconnected nodes) which do not satisfy the requested constraints. This procedure is usually referred as *topology filtering*.

There may be multiple different paths in the graph $G(N, E)$ that satisfy the same set of constraints. Such paths are said to be feasible. According to Definition 3.1, any of these paths is a solution to the MCP problem. However, often it might be desirable to retrieve an optimal path, according to some criteria, and respecting also the bounds imposed by the constraints. This more difficult problem is known as the

Multi-Constrained Optimal Path (MCOP) problem. Clearly, since the paths must be optimized according to some costs criteria, MCOP intersects the SP problem:

DEFINITION 3.2. *A Shortest Path problem can be represented as a directed graph $G = (N, E)$, where each arc $e \in E$ from node p to node q ($p, q \in N$) has associated a label representing the cost of the arc from p to q: the problem is to find a path between two vertices such that the sum of the weights of its constituent edges is minimized.*

In Section 4 we model SP problems with SCLP and we extend them by also adding constraints on the path, as required by QoS routing.

The MCP problem is a NP-Complete problem. The authors of [Garey and Johnson 1979] were the first to list the MCP problem with a number of metrics $m = 2$ as being NP-complete, but they did not provide a proof. Wang and Crowcroft have provided this proof for $m \geq 2$ in [Wang and Crowcroft 1996] and [Wang 1999], which basically consisted of reducing the MCP problem for $m = 2$ to an instance of the partition problem, a well-known NP-complete problem [Garey and Johnson 1979]. However, simulations performed in (for example) [Mieghem et al. 2001; Kuipers et al. 2004; Younis and Fahmy 2003] show that QoS routing may not be intractable in some of the possible cases.

Now we define the ST problem:

DEFINITION 3.3. *Given an undirected distance graph $G = (V, E, d)$ and a set S, where V is the set of vertices in G, E is the set of edges in G, $\mathcal{C}$ is a distance function which maps E into the set of nonnegative numbers and $S \subseteq V$ is a subset of the vertices of V, the Steiner Tree problem is to find a tree of G that spans S with minimal total distance on its edges:*

$$\sum_{\forall e \in S} \mathcal{C}(e) \quad \textit{is minimized}$$

If the $S = V$, the ST problem reduces to the *Minimum Spanning Tree* (MST) problem. A solution to the MST problem have to span all the nides in the graph and minimize the total weight of the tree at the same time; in Prim algorithm [Cormen et al. 1990], at each step the solution tree is augmented with an edge that contributes with the minimum amount possible to the total cost of the tree, thus the algorithm is greedy. ST has been extended to *Constrained Steiner Tree* (CST), to include constraints concerning the weights of the links; for example, if we want that the sum of the metric values for each path p from the source s to each leaf $s \in S$, is less than a chosen limit Δ, the constraint is:

$$\sum_{\forall p \in Path(s,v)} \mathcal{D}(e) \; < \; \Delta$$

Where $\mathcal{D}$ is another function mapping the edges to a different cost (e.g. measuring the link delay). Thus, a CST minimizes the global cost of the tree and satisfy these constraints. ST and CST are NP-Complete problems [Winter 1987] since the second can be reduced to the first one.

As can be seen, the problems related to multicast inherit both the difficulty of multiple constrained metrics, and the difficulty to reach multiple end-nodes at the same time.

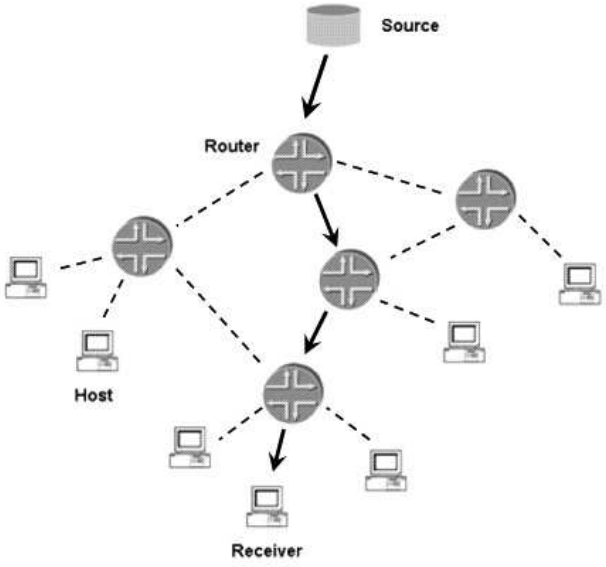

Fig. 1. An example of a unicast distribution between the source and the receiver. Oriented arcs highlight the path, while dashed lines correspond to links not traversed by the flow.

3.2 Unicast Routing with QoS Extensions

In Fig. 1 we show an example of a unicast communication between the *source*, generating data, and the only one *receiver* (i.e. the destination of the communication): the thick oriented lines highlight the direction of the packet flow, while dashed lines correspond to links not traversed.

Now we present some of the unicast QoS routing proposals, each of them oriented at optimizing only a small subset of the possible QoS metrics or using heuristics, since, as presented in Section 3.1, the problem is in general NP-Complete. For example, several solutions have been proposed to bandwidth-bounded routing: an interesting approach proposed in [Ma and Steenkiste 1997] exploits the dependencies among resources, e.g. available bandwidth, delay, and buffer space, to simplify the problem; then, a modified version Bellman-Ford algorithm can be used. One approach to satisfy both bandwidth and delay bounds is to first prune all links not satisfying the bandwidth requirement. Dijkstras shortest path algorithm is then applied to find a feasible path, if any, satisfying the delay requirements [Wang and Crowcroft 1996]. The problem of optimizing both the bandwidth and the delay can be either solved as a *widest shortest path* problem or a *shortest widest path* problem, depending if the algorithm gives higher priority to selecting paths with minimum hop counts (i.e. *widest shortest path*), or to selecting paths with maximum bandwidth (i.e. *shortest widest path*) [Wang and Crowcroft 1996]. The objective of multi-constrained routing is to simultaneously satisfy a set of constraints, as described in [Ma and Steenkiste 1997; Korkmaz and Krunz 2001]. In [Korkmaz and Krunz 2001] is proposed a heuristic approach for the multi-constrained optimal path problem (defined a *H_MCOP*), which optimizes a non-linear function (for fea-

sibility) and a primary function (for optimality). There are also solutions for bandwidth and cost bounded routing, which typically map the cost or the bandwidth to a bounded integer value, and then solve the problem in polynomial time using an extended version of Bellman-Ford or Dijkstra algorithms [Chen and Nahrstedt 1998].

3.3 Multicast Routing with QoS extensions

Multicast is an important bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to multiple receivers (as shown in Fig. 2). Therefore, while saving resources, multicast is well suited to concurrently distribute contents on behalf of applications asking for a certain timeliness of delivery: thus, also multicast routing has naturally been extended to guarantee QoS requirements [Wang and Hou 2000]. In its simplest implementation, multicast can be provided using multiple unicast transmissions, but with this solution, the same packet can traverse the same link multiple times. For this reason, the network must provide this service natively.

A multicast address is also called a multicast group address, with which the routers can locate and send packets to all the members in the group. A group member is a host that expresses interest in receiving packets sent to a specific group address. A group member is also sometimes called a *receiver* or a *listener*. A multicast source is a host that sends packets with the destination address set to a multicast group. To deliver data only to interested parties, routers in the network build a *multicast* (or *distribution*) *tree* (Figure 2). Each subnetwork that contains at least one interested listener is a leaf of the tree. Where the tree branches, routers replicate the data and send a single packet down each branch. No link ever carries a duplicate flow of packets, since packets are replicated in the network only at the point where paths diverge, reducing the global traffic.

Multicast problem has been studied with several algorithms and variants, such as *Shortest-Path Tree* (SPT), MST, ST, CST (see Section 3), and other miscellaneous trees [Wang and Hou 2000]. Algorithms based on SPT (e.g. Dijkstra or Bellman-Ford [Cormen et al. 1990]) aim to minimize the sum of the weights on the links from the source to each receiver, and if all the link cost one unit, the resulting tree is the least-hop one.

Multicast QoS routing is generally more complex than unicast QoS routing, and for this reason less proposals have been elaborated in this area [Younis and Fahmy 2003]. With respect to unicast, the additional complexity stems from the need to support shared and heterogeneous reservation styles (towards distinct group members) and global admission control of the distribution flow. Some of the approaches use a Steiner tree formulation [Berman et al. 1979] or extend existing algorithm to optimize the delay (i.e. MOSPF [Moy 1998] is the multicast version of the classical OSPF), while the *Delay Variation Multicast Algorithm* (DVMA) [Rouskas and Baldine 1997] computes a multicast tree with both bounded delay and bounded jitter. Also, delay-bounded and cost-optimized multicast routing can be formulated as a Steiner tree: an example approach is *QoS-aware Multicast Routing Protocol* [Chen et al. 2000] (QMRP). Other multicast QoS routing algorithms and related problems (entailing stability, robustness and scalability) are presented in [Younis and Fahmy 2003].

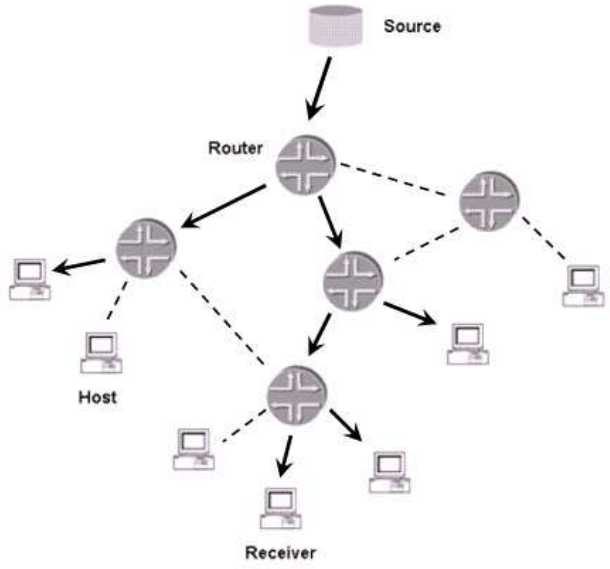

Fig. 2. An example of a multicast tree built over a network: oriented arcs highlight the tree (direction is down stream), while dashed lines correspond to links not traversed by the flow.

4. FINDING UNICAST QOS ROUTES WITH SCLP PROGRAMS

In this Section we will show how to represent and solve unicast QoS routing with SCLP. At the beginning the problem will be treated only from the cost optimization view, i.e. as a SP problem, while in the last part we propose an example on how to add constraints on the path (i.e. solving the MCOP problem seen in Sec. 3.2). Section 4.1 translates SP problems as SCLP programs, while in Section 4.2 the same model is extended for multi-criteria optimizations, thus featuring vectors of costs on the edges, and not a single value. Section 4.3 describes the case where each arc also stores information about the *modality* to be used to traverse the arc. At last, in Section 4.4 we add constraints on the QoS metrics, in order to fully obtain a model for constrained paths.

4.1 From SP Problems to SCLP Programs

We suppose to work with a graph $G = (N, E)$, where each oriented arc $e \in E$ from node p to node q ($p, q \in N$) has associated a label representing the cost of the arc from p to q, as the example in Fig 3. This graph can be easily used to represent a network, if nodes are associated to network devices (routers and hosts) and arcs to network links. From any SP problem we can build an SCLP program as follows.

For each arc we have two clauses: one describes the arc and the other one its cost. More precisely, the head of the first clause represents the starting node, and its body contains both the final node and a predicate, say c, representing the cost of the arc. Then, the second clause is a fact associating to predicate c its cost (which is a semiring element). Even if in this Section the concept of cost is quite general, we recall that with this fact we represent the QoS metric values on the arc (see

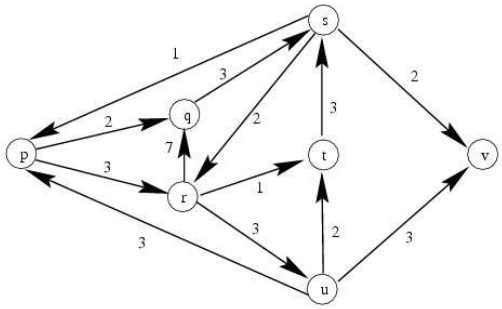

Fig. 3. An SP problem.

Table II. The SCLP program representing the SP problem in Figure 3.

`p :-` c_{pq} `, q.`	c_{pq} `:- 2.`
`p :-` c_{pr} `, r.`	c_{pr} `:- 3.`
`q :-` c_{qs} `, s.`	c_{qs} `:- 3.`
`r :-` c_{rq} `, q.`	c_{rq} `:- 7.`
`r :-` c_{rt} `, t.`	c_{rt} `:- 1.`
`r :-` c_{ru} `, u.`	c_{ru} `:- 3.`
`s :-` c_{sp} `, p.`	c_{sp} `:- 1.`
`s :-` c_{sr} `, r.`	c_{sr} `:- 2.`
`s :-` c_{sv} `, v.`	c_{sv} `:- 2.`
`t :-` c_{ts} `, s.`	c_{ts} `:- 3.`
`u :-` c_{up} `, p.`	c_{up} `:- 3.`
`u :-` c_{ut} `, t.`	c_{ut} `:- 2.`
`u :-` c_{uv} `, v.`	c_{uv} `:- 3.`
`v :- 0.`	

Section 3). For example, if we consider the arc from p to q with cost 2, we have the clause

`p :-` c_{pq} `, q.`

and the fact

c_{pq} `:- 2.`

Finally, we must code that we want v to be the final node of the path. This is done by adding a clause of the form `v :- 0.` Note also that any node can be required to be the final one, not just those nodes without outgoing arcs (like v is in this example). The whole program corresponding to the SP problem in Figure 3 can be seen in Table II.

To represent the classical version of SP problems, we consider SCLP programs over the semiring $S = \langle \mathbb{N}, \min, +, +\infty, 0 \rangle$, which is an appropriated framework to represent constraint problems where one wants to minimize the sum of the costs of the solutions. For example, we can imagine that the cost on the arcs represents to us the average delay experienced on the related link (measured in tens milli-seconds). To compute a solution of the SP problem it is enough to perform a query in the SCLP framework; for example, if we want to compute the cost of the path from

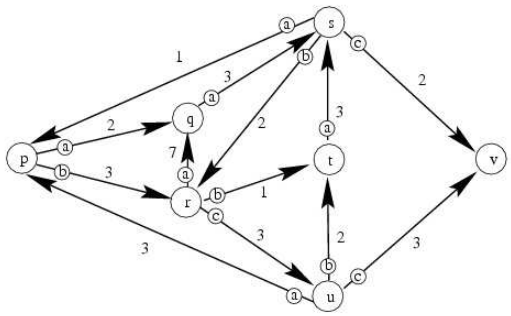

Fig. 4. An SP problem with labeled arcs.

r to v we have to perform the query `:- r`. For this query, we obtain the value 6, that represents the cost of the best path(s) from r to v, optimizing in this way the total delay experienced on the route from r to v. Clearly, different semirings can be chosen to represent the composition properties of the different metrics, as we will see better in Section 4.2 by proposing bandwidth as the second metric describing the link costs.

Notice that to represent classical SP problems in SCLP, we do not need any variable. Thus the resulting program is propositional. However, this program, while giving us the cost of the shortest paths, does not give us any information about the arcs which form such paths. This information could be obtained by providing each predicate with an argument, which represents the arc chosen at each step.

Figure 4 shows the same SP problem of Figure 3 where the arcs outgoing each node have been labeled with different labels to distinguish them. Such labels can then be coded into the corresponding SCLP program to "remember" the arcs traversed during the path corresponding to a solution. For example, clause

`p :-` c_{pq}`, q.`

would be rewritten as

`p(a) :-` c_{pq}`, q(X).`

Here constant a represents one of the arcs going out of p: the one which goes to q. If all clauses are rewritten similarly, then the answer to a goal like `:- r(X)` will be both a semiring value (in our case 6) and a substitution for X. This substitution will identify the first arc of a shortest path from r to v. For example, if we have $X = b$, it means that the first arc is the one that goes from r to t. To find a complete shortest path, we just need to compare the semiring values associated with each instantiated goal, starting from r and following the path. For example, in our case (of the goal $\exists X.r(X)$) we have that the answer to the goal will be $X = c$ with semiring value 6. Thus we know that a shortest path from r to v can start with the arc from r to u. To find the following arc of this path, we compare the semiring values of $u(a)$, $u(b)$, and $u(c)$. The result is that $u(c)$ has the smallest value, which is 3. Thus the second arc of the shortest path we are constructing is the one from u to v. The path is now finished because we reached v which is our final destination.

Notice that a shortest path could be found even if variables are not allowed in the program, but more work is needed. In fact, instead of comparing different instantiations of a predicate, we need to compare the values associated with the predicates that represent nodes reachable by alternative arcs starting from a certain node, and sum them to the cost of such arcs. For example, instead of comparing the values of $p(a)$ and $p(b)$ (Figure 4), we have to compare the values of $q + 2$ and of $r + 3$ (Figure 3).

A third alternative to compute a shortest path, and not only its cost, is to use lists: by replacing each clause of the form

`p :-` c_{xy}`, q.`

with the clause

`p([a|T]) :-` c_{xy}`, q(T).`

during the computation we also build the list containing all arcs which constitute the corresponding path. Thus, by giving the goal `:- p(L).`, we would get both the cost of a shortest path and also the shortest path itself, represented by the list L.

An alternative representation, probably more familiar for CLP-ers, of SP problems in SCLP is one where there are facts of the form

```
c(p,q) :- 2.
⋮
c(u,v) :- 3.
```

to model the graph, and the two clauses

```
path(X,Y) :- c(X,Y).
path(X,Y) :- c(X,Z), path(Z,Y).
```

to model paths of length one or more. In this representation the goal to be given to find the cost of the shortest path from p to v is `:- path(p,v).` This representation is obviously more compact than the one in Table II, and has equivalent results and properties. However, in next Sections we will continue using the simpler representation, used in Table II, where all the clauses have at most one predicate in the body. The possibility of representing SP problems with SCLP programs containing only such a kind of clauses is important, since it will allow us to use efficient algorithms to compute the semantics of such programs (see [Bistarelli et al. 2002] for more details).

4.2 Partially-Ordered SP Problems

Sometimes, the costs of the arcs are not elements of a totally ordered set. A typical example is obtained when we consider multi-criteria SP problems. Consider for example the multi-criteria SP problem shown in Figure 5: each arc has associated a pair that represent the weight of the arc in terms of *cost* of use and average *delay* (i.e. two possible QoS metrics); thus, the values are in the $\langle cost, delay\rangle$ form. Given any node p, we want to find a path from p to v (if it exists) that minimizes both criteria. In this example, there may be cases in which the labels of two arcs are not compatible, like $\langle 5, 20\rangle$ and $\langle 7, 15\rangle$, since the cost is better in the first pair, while the delay is lower in the second one. In general, when we have a partially ordered set of costs, it may be possible to have several paths, all of which are not *dominated* by others, but which have different incomparable costs (see also Section 6).

We can translate this SP problem in in Figure 5 into the corresponding SCLP

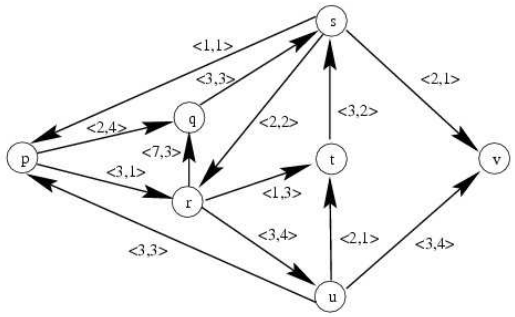

Fig. 5. A multi-criteria SP problem.

Table III. The SCLP program representing the multi-criteria SP problem in Figure 5.

`p :-` c_{pq} `, q.`	c_{pq} `:- < 2,4 >.`
`p :-` c_{pr} `, r.`	c_{pr} `:- < 3,1 >.`
`q :-` c_{qs} `, s.`	c_{qs} `:- < 3,3 >.`
`r :-` c_{rq} `, q.`	c_{rq} `:- < 7,3 >.`
`r :-` c_{rt} `, t.`	c_{rt} `:- < 1,3 >.`
`r :-` c_{ru} `, u.`	c_{ru} `:- < 3,4 >.`
`s :-` c_{sp} `, p.`	c_{sp} `:- < 1,1 >.`
`s :-` c_{sr} `, r.`	c_{sr} `:- < 2,2 >.`
`s :-` c_{sv} `, v.`	c_{sv} `:- < 2,1 >.`
`t :-` c_{ts} `, s.`	c_{ts} `:- < 3,2 >.`
`u :-` c_{up} `, p.`	c_{up} `:- < 3,3 >.`
`u :-` c_{ut} `, t.`	c_{ut} `:- < 2,1 >.`
`u :-` c_{uv} `, v.`	c_{uv} `:- < 3,4 >.`
`v :- < 0,0 >.`	

program in Table III. This program works over the semiring

$$\langle \mathbb{N}^2, \min', +', \langle +\infty, +\infty \rangle, \langle 0, 0 \rangle \rangle,$$

where min' and $+'$ are classical min and $+$, suitably extended to pairs. In practice, this semiring is obtained by putting together, via the Cartesian product, two instances of the semiring $\langle \mathbb{N}, \min, +, +\infty, 0 \rangle$ (we recall that the Cartesian product of two c-semirings is a c-semiring as well [Bistarelli et al. 1997b]). One of the two instances is used to deal with the cost criteria, the other one is for the delay criteria. By working on the combined semiring, we can deal with both criteria simultaneously: the partial order will tell us when a $\langle cost, delay \rangle$ pair is preferable to another one, and also when they are not comparable.

To give an idea of another practical application of partially-ordered SP problems, just think of network routing problems where we need to optimize according to the following criteria: minimize the delay, minimize the cost, minimize the number of arcs traversed, and maximize the bandwidth. The first three criteria correspond to the same semiring, which is $\langle \mathbb{N}, min, +, +\infty, 0 \rangle$, while the fourth criteria can be characterized by the semiring $\langle \mathcal{B}, max, min, 0, +\infty \rangle$, where $\mathcal{B}$ is the set of the possible bandwidth values (in Section 5.1 we will better investigate these semirings).

In this example, we have to work on a semiring which is obtained by vectorizing all these four semirings. Each of the semirings is totally ordered but the resulting semiring, whose elements are four-tuples, is partially ordered.

4.3 Modality-based SP Problems

Until now we have considered situations in which an arc is labeled by its cost, be it one element or a tuple of elements as in the multi-criteria case. However, sometimes it may be useful to associate with each arc also information about the *modality* to be used to traverse the arc.

For example, interpreting the arcs of a graph as links between cities, we may want to model the fact that we can cover such an arc by *car*, or by *train*, or by *plane*. Another example of a modality could be the *time of the day* in which we cover the arc, like *morning*, *afternoon*, and *night*. One more example, this time strictly related to topic of this paper, could be represented by the modalities associated with the network link, e.g. *wired*, *wireless* or *VPN*, if there is the opportunity to establish a *Virtual Private Network* on it. Therefore the modalities can be used to manage policies for the routing (i.e. for policy routing). In all these examples, the cost of an arc may depend on its modality.

An important thing to notice is that a path could be made of arcs which not necessarily are all covered with the same modality. For example, the network connection between two distant buildings of the same company can be made of many hops, some of which are covered with the wireless modality and others with wired one. Moreover, it can be that different arcs have different sets of modalities. For example, from node n_0 to node n_1 we can use both the wired or wireless connection, and from node n_1 to node n_2 we can use only a VPN. Thus modalities cannot be simply treated by selecting a subset of arcs (all those with the same modality).

An example of an SP problem with three modalities representing a network with cryptographic service on the links (c) (both wired or wireless), wired/no-crypt (w), and wireless/no-crypt (l) can be seen in Figure 6. Here the problem is to find a shortest path from any node to v (our final destination), and to know both its delay and also the modalities of its arcs. This SP problem can be modeled via the SCLP program in Table IV. In this program, the variables represent the modalities. If we ask the query `:-p(c).`, it means that we want to know the smallest delay for a route from p to v using the the links with the cryptographic service. The result of this query in our example is $p(c) = 9$ (using the path $p - r - u - v$).

Notice that the formulation shown in Figure IV puts some possibly undesired constraints on the shortest path to be found. In fact, by using the same variable in all the predicates of a rule, we make sure that the same modality (in our case the same transport mean) is used throughout the whole path. If instead we want to allow different modalities in different arcs of the path, then we just need to change the rules by putting a new variable on the last predicate of each rule. For example, the rule in Tab. IV

`p(X) :-` c_{pq}`(X), q(X).`

would become

`p(X) :-` c_{pq}`(X), q(Y).`

Now we can use a modality for the arc from p to q, and another one for the next

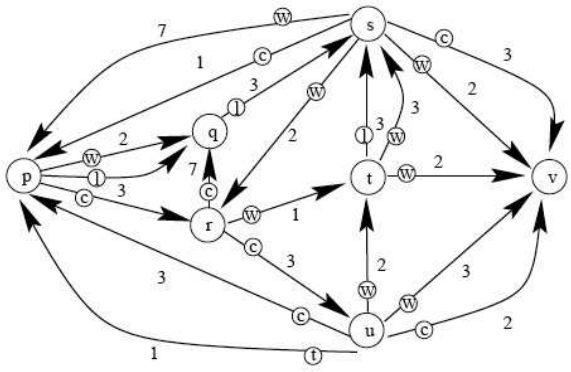

Fig. 6. An SP problem with modalities.

Table IV. The SCLP program representing the SP problem with modalities in Fig. 6.

p(X) :- c_{pq}(X), q(X).	c_{pq}(w) :- 2.
p(X) :- c_{pr}(X), r(X).	c_{pq}(l) :- 3.
q(X) :- c_{qs}(X), s(X).	c_{pr}(c) :- 3.
r(X) :- c_{rq}(X), q(X).	c_{qs}(l) :- 3.
r(X) :- c_{rt}(X), t(X).	c_{rq}(c) :- 7.
r(X) :- c_{ru}(X), u(X).	c_{rt}(w) :- 1.
s(X) :- c_{sp}(X), p(X).	c_{ru}(c) :- 3.
s(X) :- c_{sr}(X), r(X).	c_{sp}(c) :- 1.
s(X) :- c_{sv}(X), v(X).	c_{sp}(w) :- 7.
t(X) :- c_{ts}(X), s(X).	c_{sr}(w) :- 2.
u(X) :- c_{up}(X), p(X).	c_{sv}(w) :- 2.
u(X) :- c_{ut}(X), t(X).	c_{sv}(c) :- 3.
u(X) :- c_{uv}(X), v(X).	c_{ts}(l) :- 3.
v(X) :- 0.	c_{ts}(w) :- 3.
	c_{up}(c) :- 3.
	c_{up}(w) :- 1.
	c_{ut}(w) :- 2.
	c_{uv}(w) :- 3.
	c_{uv}(c) :- 2.

arc. In this new program, asking the query `:-p(c).` means that we want to know the smallest delay for a trip from p to v using the cryptographic service in the first arc.

The same methods used in the previous sections to find a shortest path, or a non-dominated path in the case of a partial order, can be used in this kind of SCLP programs as well. Thus we can put additional variables in the predicates to represents alternative arcs outgoing the corresponding nodes, and we can shift to the semiring containing sets of costs to find a non-dominated path. In particular, a clause like

p(X) :- c_{pq}(X), q(Y).

would be rewritten as

`p(X,a) :-` c_{pq}`(X), q(Y,Z).`

4.4 Adding constraints to SP problems

As seen in Section 3.1 a MCOP is much more difficult to solve than a SP problem, that is NP-Complete. So far we considered only variants of SP problems (partially-ordered or modality-based), but our aim is to provide a complete model for the unicast QoS routing. Thus, besides achieving cost optimization, we need also to consider constraints on the QoS metrics.

In our example we consider again the multi-criteria graph in Fig. 5: each arc has associated a pair that can represent the weight of the arc in terms of *cost* of use and average *delay*. However, in this case our goal is to minimize the cost and to guarantee a final average delay less than or equal to 8 ($80msec$), thus we want to add the constraint $delay \leq 8$.

We chose to represent constrained paths with a program in *CIAO Prolog* [Bueno et al. 1997], a system that offers a complete Prolog system supporting ISO-Prolog, but, at the same time its modular design allows both restricting and extending the basic language. Thus, it allows both to work with subsets of Prolog and to work with programming extensions implementing functions, higher-order (with predicate abstractions), constraints, objects, concurrency, parallel and distributed computations, sockets, interfaces to other programming languages (C, Java, Tcl/Tk), relational databases and many more.

CIAO Prolog has also a fuzzy extension, but since it does not completely conform to the semantic of SCLP defined in [Bistarelli et al. 1997a] (due to interpolation in the interval of the fuzzy set), we decided to use the CIAO operators among constraints (as $<$ and $\leq$), and to model the $\times$ operator of the c-semiring with them. For this reason, we inserted the cost of the edges in the head of the clauses, differently from SCLP clauses which have the cost in the body of the clause.

In Tab. V is shown the CIAO program that represents the graph in Figure 5: here the edges (i.e. all the *Edges* facts in Table V) are in the form:

$$edge(Source\_Node, Destination\_Node, [Link\_Cost, Link\_Delay])$$

Moreover, we can see the two clauses that describe the structure of paths: *Rule 1* and *Rule 2* respectively represent the base (or termination) case, where a path is simply an edge, and the recursive case, needed to add one edge to the path. To avoid infinite recursion, and thus the program crashing, we need to deal with graph loops by considering the list of the already visited nodes, in order to prevent the search from visiting them twice. Moreover, we inserted a variable in the head of the *path* clauses to remember, at the end, all the visited nodes of the path: this list will store the nodes following the correct ordering of the visit. Finally, the last variable of the clause head is used to retrieve only the paths with a total delay equal or less than the passed value. Thus, the *path* clause-heads are in the form:

$$path(Source\_Node, Destination\_Node, Path\_Nodes, Already\_Visisted\_Nodes,$$

$$[Path\_Cost, Path\_Delay], Path\_Max\_Delay)$$

Table V. The CIAO program representing all the paths of Fig. 5, with $delay \leq 8$

```
:- module(path,_,_).
:- use_module(library(lists)).

Edges
edge(p, q, [2,4]).
edge(p, r, [3,1]).
edge(q, s, [3,3]).
edge(r, q, [7,3]).
edge(r, t, [1,3]).
edge(r, u, [3,4]).
edge(s, p, [1,1]).
edge(s, r, [2,2]).
edge(s, v, [2,1]).
edge(t, s, [3,2]).
edge(u, p, [3,3]).
edge(u, t, [2,1]).
edge(u, v, [3,4]).

1)
path(X,Y,[X,Y],_,[C,D],L):-
        edge(X,Y,[C,D]),
        D =< L.

2)
path(X,Y,[X|T],V,[C,D],L):-
        edge(X,Z,[C1,D1]),
        nocontainsx(V,Z),
        path(Z,Y,T,[Z|V],[C2,D2]),
        times([C1,D1], [C2,D2], [C,D]),
        D =< L.

Aggregator
times([C1,D1], [C2,D2], [C3,D3]) :-
        C3 = C1+C2,
        D3 = D1+D2.
```

```
Ciao-Prolog 1.10 #5: Fri Aug 6 19:01:54 2004
?- path(p,v,P,[p],[C,D],8).

C = 2+(3+2),
D = 4+(3+1),
P = [p,q,s,v] ? .

C = 3+(7+(3+2)),
D = 1+(3+(3+1)),
P = [p,r,q,s,v] ? .

C = 3+(1+(3+2)),
D = 1+(3+(2+1)),
P = [p,r,t,s,v] ? .

no
?-
```

Fig. 7. The CIAO output for the program in Tab. V: three paths are found with $delay \leq 8$.

The *Aggregator* clause mimics the $\times$ operation of the semiring (i.e. $+$ extended to pairs, as in Sec. 4.2), and therefore it composes the global costs of the edges together, edge costs with costs, and edge delays with delays.

All the found paths with a $delay \leq 8$, and the relative query $path(p, v, P, [p], [C, D], 8)$ are shown in Fig. 7. The p source node of the path, must be included in the list of the visited nodes from the beginning. Figure 7 corresponds to the output of the CIAO program in Tab. V, and for each of the three found paths it shows the variable P, which stores the sequence of the nodes in the path, and the C - D pair, which corresponds to the total cost of the path in terms of $\langle cost, delay \rangle$.

5. EXTENDING THE MODEL TO DEAL WITH MULTICAST QOS ROUTING

Now we extend the framework given in Section 4 in order to manage also the multicast delivery schema. The first step is represented by the use of hypergraphs instead of simple graphs, since we need a method to connect one node to multiple destinations at the same time (i.e. when the same packet must be routed on different links). Section 5.1 presents a possible transformation procedure from networks

to *and-or* graphs, showing also how to find a cost for the hyperarcs and related semirings. In Section 5.2 we describe the SCLP programs representing and solving the multicast QoS routing. In Section 5.3 we associate modalities to hyperarcs, as we did in Section 4.3 for paths, and, at last, Section 5.4 proposes some suggestions on reusing the same subcomputation results to reduce the computational complexity of the problem, when working on large networks (with *tabling* techniques).

5.1 From networks to hypergraphs

In this Section we explain a method to translate the representation of a multicast network with QoS requirements (Figure 9*a*) into a corresponding weighted *and-or* graph [Martelli and Montanari 1978] (Figure 9*b*). This procedure can be split in three distinct steps, respectively focusing on the representation of *i)* network nodes, *ii)* network links and *iii)* link costs in terms of QoS metrics.

An *and-or* graph [Martelli and Montanari 1978] is defined essentially as a hypergraph. Namely, instead of arcs connecting pairs of nodes there are hyperarcs connecting an n-tuple of nodes ($n = 1, 2, 3, \ldots$). Hyperarcs are called *connectors* and they must be considered as directed from their first node to all others. Formally an *and-or* graph is a pair $G = (N, C)$, where N is a set of *nodes* and C is a set of connectors

$$C \subseteq N \times \bigcup_{i=0}^{k} N^i.$$

Note that the definition allows 0-connectors, i.e. connectors with one input and no output node. 0-connectors are represented as a line ending with a square (Figure 9*b*). In the following of the explanation we will also use the concept of *and* tree [Martelli and Montanari 1978]: given an *and-or* graph G, an *and* tree H is a *solution tree of* G *with start node* n_r, if there is a function g mapping nodes of H into nodes of G such that:

—the root of H is mapped in n_r.
—if $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$ is a connector of H, then $(g(n_{i_0}), g(n_{i_1}), \ldots, g(n_{i_k}))$ is a connector of G.

Informally, a solution tree of an *and-or* graph is analogous to a path of an ordinary graph: it can be obtained by selecting exactly one outgoing connector for each node.

Each of the network nodes can be easily cast in the corresponding *and-or* graphs as a single graph node: thus, each node in the graph can represent an interconnecting device (e.g. a router), or a node acting as the source of a multicast communication (injecting packets in the network), or, finally, a receiver belonging to a multicast group and participating in the communication. In Section 5.2, when we will look for the best tree solution, the root of the best *and* tree will be mapped to the node representing the source of the multicast communication; in the same way, receivers will be modelled by the leaves of the resulting *and* tree. When we translate a receiver, we add an outgoing 0-connector to model the end-point of the communication, and whose cost will be explained below. Suppose that $\{n_0, n_1, \ldots, n_9\}$ in Fig. 9*a* are the identifiers of the network nodes.

To model the links, we examine the forward star (*f-star*) of each node in the network (i.e. the set of arcs outgoing from a node): we consider the links as

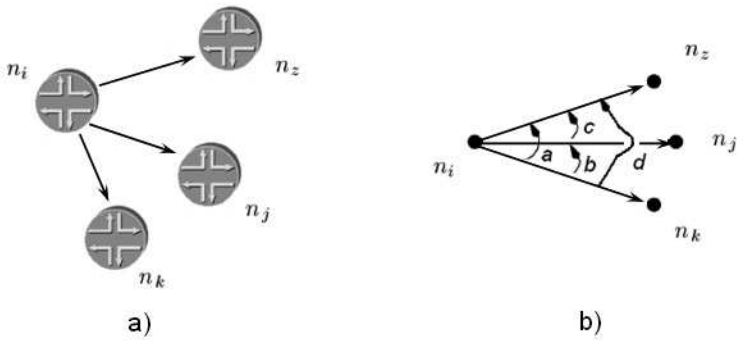

Fig. 8. *a)* the f-star of n_i network-node and *b)* its representation with connectors.

oriented, since the cost of sending packets from node n_i to n_j can be different from the cost of sending from n_j to n_i (one non-oriented link can be easily replaced by two oriented ones). Supposing that the f-star of node n_i includes the arcs (n_i, n_j), (n_i, n_k) and (n_i, n_z), we translate this f-star by constructing one connector directed from n_i to each of the subsets of destination nodes $\{j, k, z\}$ (Figure 8), for a possible maximal number of $2^{|N|}-1$ subsets (where $|N|$ is the cardinality of the set of node in the graph), i.e. excluding the emptyset; in Sec. 5.4 we will see how to minimize this exponential growth. Thus, all the resulting connectors with n_i as the input node are (n_i, n_j), (n_i, n_k), (n_i, n_z), (n_i, n_k, n_j), (n_i, n_k, n_z), (n_i, n_j, n_z) and (n_i, n_j, n_k, n_z). In the connectors tuple-ordering of the nodes, the input node is at the first position and the output nodes (when more than one) follow the orientation of the related arrow in Figure 8.

To simplify Fig. 8*b*, the arcs linking directly two nodes represent 1-connectors (n_i, n_j), (n_i, n_k) and (n_i, n_z), while curved oriented lines represent n-connectors (with $n > 1$), where the set of their output nodes corresponds to the output nodes of the traversed arcs. With respect to n_i, in Fig. 8 we have a curved line labelled with a that corresponds to (n_i, n_k, n_j, n_z), b to (n_i, n_k, n_j), c to (n_i, n_j, n_z), and, at last, d to (n_i, n_k, n_z). To have a clear figure, the network links in Fig. 9*a* are oriented "towards" the receivers, thus we put only the corresponding connectors in Fig. 9*b*.

In the example we propose here, we are interested in QoS link-state information concerning only bandwidth and delay. Therefore, each link of the network can be labeled with a 2-dimensional cost, for example the pair $\langle 7, 3\rangle$ tells us that the maximum bandwidth on that specific link is 70Mbps and the maximum delay is 30msec. In general, we could have a cost expressed with a n-dimensional vector, where n is the number of metrics to be taken in account while computing the best distribution tree. Since we want to maintain this link state information even in the *and-or* graph, we label the corresponding connector with the same tuple of values (Figure 9).

In the case when a connector represent more than one network link (i.e. a n-connector with $n \geq 2$), its cost is decided by assembling the costs of the these links with the composition operation $\circ$, which takes as many n-dimensional vectors as operands, as the number of links represented by the connector. Naturally, we can

instantiate this operation for the particular types of costs adopted to express QoS: for the example given in this Section, the result of $\circ$ is the minimum bandwidth and the highest delay, ergo, the worst QoS metric values:

$$\circ(\langle b_1, d_1\rangle, \langle b_2, d_2\rangle, \ldots, \langle b_n, d_n\rangle) \longrightarrow \langle \min(b_1, b_2, \ldots, b_n), \max(d_1, d_2, \ldots, d_n)\rangle$$

The cost of the connector (n_1, n_3, n_4) in Fig. 9*b* will be $\langle 7, 3\rangle$, since the costs of connectors (n_1, n_3) and (n_1, n_4) are respectively $\langle 7, 2\rangle$ and $\langle 10, 3\rangle$:

$$\circ(\langle 7, 2\rangle, \langle 10, 3\rangle) = \langle 7, 3\rangle$$

To simplify Fig. 9*b*, we inserted only the costs for the 1-connectors, but the costs for the other connectors can be easily computed with the $\circ$ operation, and are all reported in Tab. VI.

So far, we are able to translate an entire network with QoS requirements in a corresponding *and-or* weighted graph, but still we need some algebraic framework to model our preferences for the links to use in the best tree. For this reason, we use the semiring structure (Sec. 2). An exhaustive explanation of the semiring framework approach for shortest-distance problems is presented in [Mohri 2002].

For example, if we are interested in maximizing the bandwidth of the distribution tree, we can use the semiring $S_{Bandwidth} = \langle \mathcal{B} \cup \{0, +\infty\}, \max, \min, 0, +\infty\rangle$ (otherwise, we could be interested in minimizing the global bandwidth with $\langle \mathcal{B} \cup \{0, +\infty\}, \max, \min, +\infty, 0\rangle$. We can use $S_{Delay} = \langle \mathcal{D} \cup \{0, +\infty\}, \min, \max, + \infty, 0\rangle$ for the delay, if we need to minimize the maximum delay that can be experienced on a single link. With this result and the depth of the final tree, we can compute an upper bound for the end-to-end delay. Elements of $\mathcal{B}$ (i.e. the set of bandwidth values) and $\mathcal{D}$ (i.e. the set of delay values) can be obtained by collecting information about the network configuration, the current traffic state and technical information about the links. Since the composition of c-semirings is still a c-semiring [Bistarelli et al. 1997b],

$$S_{Network} = \langle \langle \mathcal{B} \cup \{0, +\infty\}, \mathcal{D} \cup \{0, +\infty\}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$$

where $+'$ and $\times'$ correspond to the vectorization of the $+$ and $\times$ operations in the two c-semirings: given $b_1, b_2 \in \mathcal{B} \cup \{0, +\infty\}$ and $d_1, d_2 \in \mathcal{D} \cup \{0, +\infty\}$,

$$\langle b_1, d_1\rangle +' \langle b_2, d_2\rangle = \langle \max(b_1, b_2), \min(d_1, d_2)\rangle$$

$$\langle b_1, d_1\rangle \times' \langle b_2, d_2\rangle = \langle \min(b_1, b_2), \max(d_1, d_2)\rangle$$

Clearly, the problem of finding best distribution tree is multi-criteria, since both bandwidth and delay must be optimized. We consider the criteria as independent among them, otherwise they can be rephrased to a single criteria. Thus, the multidimensional costs of the connectors are not elements of a totally ordered set, and it may be possible to obtain several trees, all of which are not *dominated* by others, but which have different incomparable costs.

For each receiver node, the cost of its outgoing 0-connector will be always included in every tree reaching it. As a remind, a 0-connector has only one input node but no destination nodes. If we consider a receiver as a plain node, we can set the cost as the 1 element of the adopted c-semiring (1 is the unit element for $\times$), since the cost to reach this node is already completely described by the other connectors of the

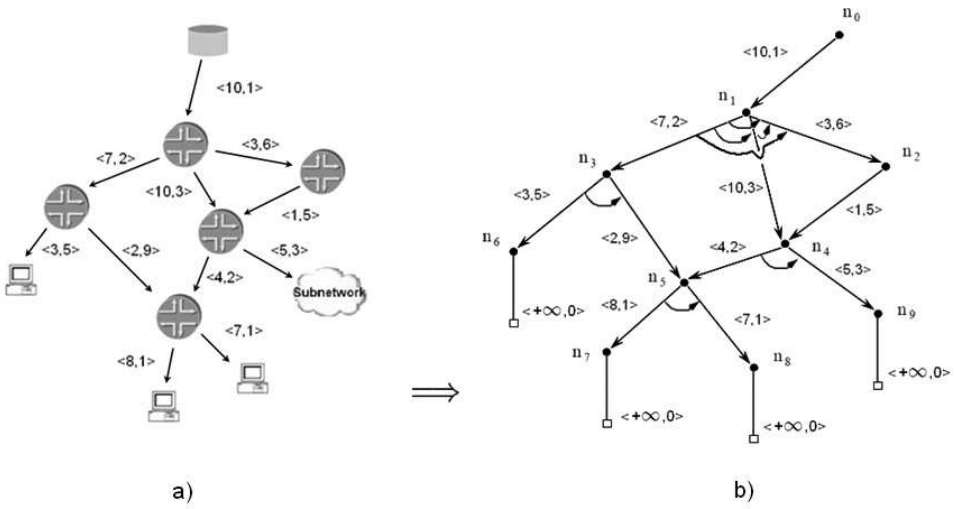

Fig. 9. A network example and the corresponding *and-or* graph representation.

tree branch ending in this node: practically, we associate the highest possible QoS values to this 0-connector, in this case infinite bandwidth and null delay. Otherwise we can imagine a receiver as a more complex subnetwork, and thus we can set the cost of the 0-connector as the cost needed to finally reach a node in that subnetwork, in case we do not want, or cannot, show the topology of the subnetwork, e.g. for security reasons.

5.2 *And-or* graphs using SCLP

In this Section, we represent an *and-or* graph with a program in SCLP. Using this framework, we can easily solve the multi-criteria example concerning the multicast QoS network in Fig. 9*b*.

As already proposed in Sec. 4, to represent the connectors in SCLP we can write clauses like $c(n_i, [n_j, n_k]) : -\langle 10, 3\rangle$, stating that the graph has connector from n_0 to nodes n_j and n_k with a bandwidth cost of 100Mbps and a delay of 30msec. Other SCLP clauses can properly describe the structure of the tree we desire to search over the graph.

For the same reasons exposed in Section 4.4, we chose to represent an *and-or* graph with a program in *CIAO Prolog* [Bueno et al. 1997]. As an example, from the weighted *and-or* graph problem in Fig. 9*b* we can build the corresponding CIAO program of Table VI as follows. First of all, we can describe the connectors of the graph with facts like

$$connector(Source\_Node, [List\_of\_Destination\_Nodes],$$

$$[Link\_Bandwidth, Link\_Delay])$$

e.g. the fact $connector(n_0, [n_2, n_3, n_4], [3, 6])$ represents the connector of the graph $(n_0, n_1, n_2, n_3, n_4)$ with bandwidth 30Mbps and delay 60msec. The set of the connector facts is highlighted as *Connectors* in Table VI. In despite of what

we declared in Section 5.1, here we choose a different ordering for the nodes in the connector tuples when we have to write the program clauses: the input node is again (as in the previous Sections) at the first position of the list representing the connector in the clause, but in this Section we decide to lexicographically order the output nodes (i.e. n_0 precedes n_1, n_1 precedes n_2 and so on). This decision is dictated by the resulting simplification in writing the program and the queries, since the ordering of the nodes can be easily remembered.

The *Leaves* of Table VI represent the terminations for the Prolog rules, and their cost is the cost of the associated 0-connector (100 represents ∞ bandwidth).

The *Aggregators* rules, *times* in Table VI, mimic the $\times$ operation of the c-semiring proposed in Section 5.1:

$$S_{Network} = \langle \langle \mathcal{B} \cup \{0, +\infty\}, \mathcal{D} \cup \{0, +\infty\} \rangle, +', \times', \langle 0, +\infty \rangle, \langle +\infty, 0 \rangle \rangle$$

where $\times'$ is equal to $\langle \min, \max \rangle$, and $+'$ is equal to $\langle \max, \min \rangle$.

At last, the rules *1-2-3-4* of Table VI describe the structure of the trees we want to find over the graph. *Rule 1* represents a tree made of only one leaf node, *Rule 2* outlines a tree made of a connector plus a list of sub-trees with root nodes in the list of the destination nodes of the connector, *Rule 3* is the termination for *Rule 4*, and *Rule 4* is needed to manage the junction of the disjoint sub-trees with roots in the list $[X|Xs]$. When we compose connectors and trees (*Rule 2* and *Rule 4*), we use the *Aggregators* to compose their costs together.

To make the program in Table VI as readable as possible, we omitted two predicates: the *sort* predicate, needed to order the elements inside the list of destination-nodes of connectors and trees (otherwise, the query $tree(n_0, [n_6, n_7$ $, n_8, n_9], [B, D])$ and $tree(n_0, [n_9, n_7, n_8, n_6], [B, D])$ would produce different results), and the *intersection* predicate to check that multiple occurrences of the same node do not appear in the same list of destination nodes, if reachable with different connectors (otherwise, for example, the tree $n_0, [n_7, n_7, n8, n9]$ would be a valid result).

To solve the *and-or* graph problem it is enough to perform a query in Prolog language: for example, if we want to compute the cost of all the trees rooted at n_0 and having as leaves the nodes representing all the receivers (i.e. $\{n_6, n_7, n_8, n_9\}$), we have to perform the query $tree(n_0, [n_6, n_7, n_8, n_9], [B, D])$, where B and D variables will be instantiated with the bandwidth and delay costs of the found trees. One of the outputs of the CIAO program for this query corresponds to the cost of the tree in Fig. 10, i.e. $\langle 2, 5 \rangle$, since $\times'$ computes the *minimum bandwidth - maximum delay* of the connectors.

A global cost can be given to *and* trees: recursively, to every subtree of H with root node n_{i_0}, a cost c_{i_0} is given as follows:

—If n_{i_0} is a leaf, then its cost is the associated constant.

—If n_{i_0} is the input node of a connector $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$, then its cost is $c_{i_0} = f_r(c_{i_1}, \ldots, c_{i_k})$ where f_r is the function cost associated with the connector, and $c_{i_1}, \ldots, c_{i_k}$ are the costs of the subtrees rooted at nodes $n_{i_1}, \ldots, n_{i_k}$.

The final cost of the tree obtained with the CIAO program is equivalent to the one that can be computed by using $\times'$ to define the f_r cost function. Starting from

Table VI. The CIAO program representing all the AND trees over the weighted *and-or* graph problem in Fig. 9*b*.

```
:- module(netModel,_,_).
:- use_module(library(lists)).

% Aggregators
times([B1, D1], [B2, D2], [B1, D1]) :-
      B1 =< B2,
      D1 >= D2.
times([B1, D1], [B2, D2], [B1, D2]) :-
      B1 =< B2,
      D1 < D2.
times([B1, D1], [B2, D2], [B2, D1]) :-
      B1 > B2,
      D1 >= D2.
times([B1, D1], [B2, D2], [B2, D2]) :-
      B1 > B2,
      D1 < D2.

% Leaves
leaf([n0], [100, 0]).
leaf([n1], [100, 0]).
leaf([n2], [100, 0]).
leaf([n3], [100, 0]).
leaf([n4], [100, 0]).
leaf([n5], [100, 0]).
leaf([n6], [100, 0]).
leaf([n7], [100, 0]).
leaf([n8], [100, 0]).
leaf([n9], [2, 3]).

% Connectors
connector(n0, [n1], [10, 1]).
connector(n1, [n2], [3, 6]).
connector(n1, [n3], [7, 2]).
connector(n1, [n4], [10, 3]).
connector(n1, [n3,n4], [7, 3]).
connector(n1, [n2,n3], [3, 6]).
connector(n1, [n2,n4], [3, 6]).
connector(n1, [n2,n3,n4], [3, 6]).
connector(n2, [n4], [1, 5]).
connector(n3, [n5], [2, 9]).
connector(n3, [n6], [3, 5]).
connector(n3, [n5,n6], [2, 9]).
connector(n4, [n5], [4, 2]).
connector(n4, [n9], [5, 3]).
connector(n4, [n5,n9], [4, 3]).
connector(n5, [n7], [8, 1]).
connector(n5, [n8], [7, 1]).
connector(n5, [n7,n8], [7, 1]).

% 1)
tree(X, [X], [B, D]):-
    leaf([X], [B, D]).

% 2)
tree(X, Z, [B, D]):-
    connector(X, W, [B1, D1]),
    treeList(W, Z, [B2, D2]),
    times([B1, D1], [B2, D2], [B, D]).

% 3)
treeList([], [], [100, 0]).

% 4)
treeList([X|Xs], Z, [B, D]):-
    tree(X, Z1, [B1, D1]),
    append(Z1, Z2, Z),
    treeList(Xs, Z2, [B2, D2]),
    times([B1, D1], [B2, D2], [B, D]).
```

the n_0^i source node and the connector (n_0^i, n_1^i) with cost $\langle 10, 1\rangle$, the total cost of the tree c_{n_0} is

$$c_0 = f_r(c_1) = \langle 10, 1\rangle \times' c_1$$

Clearly, this framework can be used to solve the unicast problem as well, if the asked query include only one destination node, e.g. $tree(n_0, [n_6], [B, D])$.

5.3 Modality-based Steiner Tree Problems

In this Section, as we provide in Sec. 4.3 for plain paths, we improve the tree search by including the possibility of considering some modalities associated with the use of the hyperarcs.

Even in this case the justification is easy, since sometimes it may be useful to

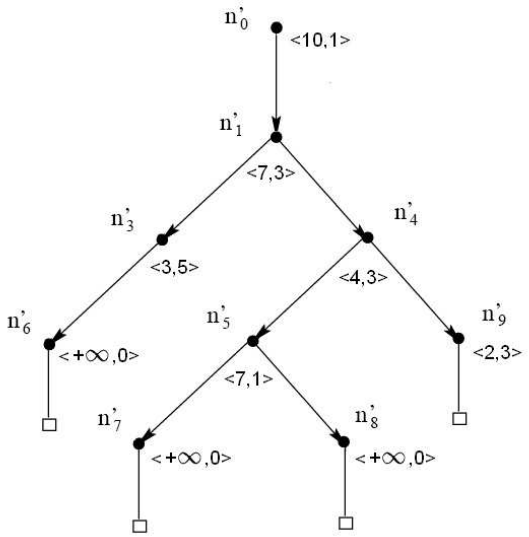

Fig. 10. One of the multicast distribution tree that can be found with the program in Table VI.

associate with each hyperarc also the information about the modality to be used to traverse that specific hyperarc. In this Section, we show an example using only two of the three modalities of Sec. 4.3: wired link with no encryption service (w), and wireless link with no encryption service (l). Other classes could collect slices of day time in which network links are preferred to be used (e.g. to better support the peaks of traffic), or label special conditions of use, e.g. to support "night back-up" or "black-out" events.

In Fig. 11 we show an example on how to pass from a network to a corresponding hypergraph with modalities (from Fig. 11a to Fig. 11b): the modality associated with a connector is found by using the union operator (i.e. $\cup$) on the sets of modalities associated with each of the links represented by that connector. In the example of Fig. 11, 0-connector have emptyset as label, since in this case we do not need any further information to finally reach a receiver; however, in general 0-connector labels may contain the same modalities as the other n-connector labels, e.g. when they represent the internal structure of a sub network, as (n_9) in Fig. 9b. For example, if the connection from n_0 to n_1 is wired, and the connection from n_0 to n_2 is wireless, the connector (n_0, n_1, n_2) will be labelled with the $\{w, l\}$ modality set. Thus, connectors are now represented in the following way:

$$connector(Source\_Node, [List\_of\_Destination\_Nodes],$$

$$[Link\_Bandwidth, Link\_Delay], [List\_of\_Modalities])$$

The query for the tree search must now be performed by including also the set of modalities that the tree connectors need to support: if the set of modalities associated with a connector is a subset of the modalities asked in the query, then that connector can be used to build the tree. This can be accomplished by using, for

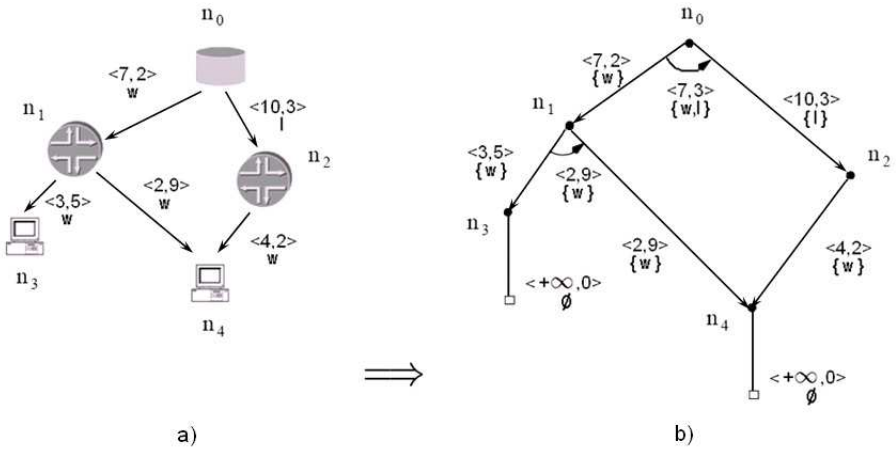

Fig. 11. *a)* A network with modalities associated to the links, and *b)* the corresponding hypergraph.

example, the CIAO *difference* predicate between the two lists (sets) of modalities, or the *sublist* property.

For example, when asking for $tree(n_0, [n_3, n_4], [B, D], [w])$, the $(n_0, n1, n2)$ connector cannot be used because its label is $\{w, l\}$ and we do not want to use wireless links. To include also that specific connector in the search, we have to ask the query $tree(n_0, [n_3, n_4], [B, D], [w, l])$. clearly, the final 0-connectors are always included in trees because they have an emptyset label.

5.4 Reducing the complexity

As shown in Section 5.1, the representation of the f-star of node in the multicast model can be composed by a total of $O(2^n)$ connectors, thus in the worst case it is exponential in the number of graph nodes. This drawback, which is vigorously perceived in strongly connected networks, and together with considering a real case network linking hundreds of nodes, would heavily impact on the time-response performance during a practical application of our model. Therefore, it is necessary to elaborate some improvements to reduce the complexity of the tree search, for example by visiting as few branches of the SCLP tree as possible (thus, restricting the solution space to be explored).

In logic programming, the basic idea behind *tabling* (or *memoing*) is that the calls to tabled predicates are stored in a searchable structure together with their proven instances: subsequent identical calls can use the stored answers without repeating the computation. This collection of tabled subgoals paired with their answers, generally referred to as *call table* and *answer table* respectively, is consulted whenever a new call, C, to a tabled predicate is issued. If C is similar to a tabled subgoal S, then the set of answers, A, associated with S may be used to satisfy C. In such instances, C is resolved against the answers in A, and hence we refer to C as a *consumer* of A (or S). If there is no such S, then C is entered into the call table and is resolved against program clauses. As each answer is derived

during this process, it is inserted into the answer table entry associated with C if it contains information not already in A. Furthermore, left recursion need not lead to non-termination because identical subgoals are not evaluated, and thus the possible infinite loops are avoided.

The same considerations stand also for tabling with constraints, apart from the fact that a subgoal call is associated with a set of constraints, and an answer to a set of answer constraints. The basic idea is to try to avoid recomputing a subgoal G if has been called in similar environment, i.e. the constraint set related to G. Moreover, some general operations on constraints, as *call abstraction*, *projection* and *entailment checking*, are needed to properly manage the tabling and control the growth of the tables (a critical aspect for this technique). These operation are usually executed in the given order: call abstraction avoids the duplication of information in the tables by restricting to a small set of call instantiation patterns. The projection of the answer on the non-local variables of the call is used because local variables are meaningless outside of the call; moreover, projection can be used to obtain a finite number of answers by discarding the constraints on local and unreachable variables. Entailment is the last step, since some of the answers computed for a predicate could be redundant and so need not to be saved: if the store already contains a more general answer, the new obtained answer can be immediately forgotten.

Tabling improves the computability power of Prolog systems and for this reason many programming frameworks have been extended in this direction. Due to the power of this extension, many efforts have been made to include it also in CLP, thus leading to the *Tabled Constraint Logic Programming* (TCLP) framework. In [Cui and Warren 2000] the authors present a TCLP framework for constraint solvers written using attributed variables; however, when programming with attributed variables, the user have to take care of of many implementation issues such as constraint store representation and scheduling strategies. A more recent work [Schrijvers and Warren 2004] explains how to port *Constraint Handling Rules* (CHR) to XSB (acronym of *eXtended Stony Brook*), and in particular its focus is on technical issues related to the integration of CHR with tabled resolution. CHR is a high-level natural formalism to specify constraint solvers and propagation algorithm. This could be the promising framework where to solve QoS routing problems, since soft constraints have already been successfully ported to the CHR system [Bistarelli et al. 2002]. As far as we know, all these TCLP systems are difficult to use at the present moment, due to the attributed variables choice, or due to the lack of an implementation.

One more consideration that can be taken into account while trying to reduce the complexity, is that large networks, as Internet, are already partitioned into different *Autonomous System* (AS), or however, into subnetworks. An AS is a collection of networks and routers under the control of one entity (or sometimes more) that presents a common routing policy to the Internet. AS can be classified by observing the types of traffic traversing them. A *multihomed* AS maintains connections to more than one other AS; however, it would not allow traffic from one AS to pass through on its way to another AS. A *stub* AS is only connected to a single AS. A *transit* AS provides connections through itself to the networks

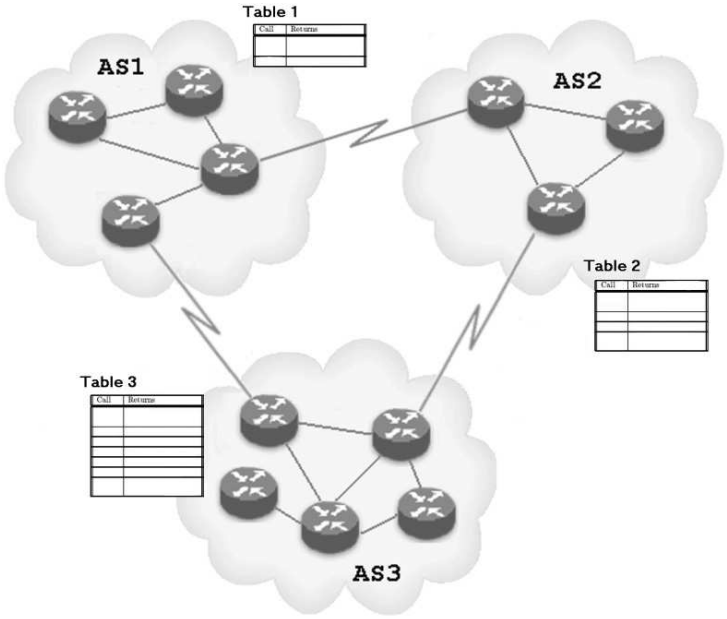

Fig. 12. A network subdivided in Autonomous Systems; each AS can store in its border routers a table with the goals related to that specific AS.

connected to it. Considering Fig. 12, network *AS1* can use the transit *AS3* to connect to network *AS2*. An *AS number* (or ASN) uniquely identifies each AS on the internet (i.e. *AS1*, *AS2* and *AS3*).

As shown in Fig. 12, in each AS (or subnetwork in general) we can find a table with the QoS routing goals concerning the destinations (routers and hosts) within its bounds, by using tabling techniques. At this point, these tables helps to find the routes that span multiple ASs and the search procedure is considerably speeded up: the routes internal to each AS can be composed together by simply using the links connecting the border routers. For example, consider when a sender in *AS1* needs to start a multicast communication towards some receivers in *AS2* and *AS3*: the routers inside *AS1* can use *Table 1* to find the routes from the source to the border routers of *AS1* (i.e. it can communicate with other ASs). Then, the border routers in *AS2* and *AS3* respectively use *Table 2* and *Table 3* to find the second and final part of the route towards the receivers inside their AS. The procedure of finding such a goal table for a single AS is much less time consuming than finding it for the whole not-partitioned network. Clearly, the fundamental premise to obtain a substantial benefit from this technique is to have strongly-connected subnetworks and few "bridges" among them.

6. A LAST REFINEMENT ON SEMIRINGS FOR PARTIALLY-ORDERED PROBLEMS

As seen in Sections 4.2 and 5.1, the costs on the connectors can be represented by vectors of costs, representing the QoS metric values of the network links. But since we can have a partial order, two such pairs may possibly be incomparable, and this may lead to a strange situation while computing the semantics of a given goal. Considering the example in Section 4.2 and the related program in Table III, if we want to compute the cost and delay of the best path from p to v, by giving the query `:- p.`, the answer in this case is the value $\langle 7, 7\rangle$. While the semiring value obtained in totally ordered SCLP programs represents the cost of one of the shortest paths, here it is possible that there are no routes with this cost: the obtained semiring value is in fact the greatest lower bound (w.r.t. both cost and delay) of the costs of all the paths from p to v. This behavior comes from the fact that, if different refutations for the same goal have different semiring values, the SCLP framework combines them via the $+$ operator of the semiring (which, in the case of our example, is the min' operator of Section 4.2). If the semiring is partially ordered, it may be that $a + b$ is different from both a and b. On the contrary, if we have a total order $a + b$ is always either a or b.

This problem of course is not satisfactory, because usually one does not want to find the greatest lower bound of the costs of all paths from the given node to the destination node, but rather prefers to have one of the non-dominated paths. To solve this problem, we can add variables to the SCLP program, as we did in the previous section, and also change the semiring. In fact, we now need a semiring which allows us to associate with the source node the set of the costs of all non-dominated path from there to the destination node. In other words, starting from the semiring $S = \langle A, +, \times, 0, 1\rangle$ (which, we recall, in the example of Sec. 4.2 is $\langle \mathbb{N}^2, min', +', \langle +\infty, +\infty\rangle, \langle 0, 0\rangle\rangle$), we now have to work with the semiring $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$, where:

—$P^H(A)$ is the Hoare Power Domain [Smyth 1978] of A, that is, $P^H(A) = \{S \subseteq A \mid x \in S, y \leq_S x \text{ implies } y \in S\}$. In words, $P^H(A)$ is the set of all subsets of A which are downward closed under the ordering $\leq_S$. It is easy to show that such sets are isomorphic to those containing just the non-dominated values. Thus in the following we will use this more compact representation for efficiency purposes. In this compact representation, each element of $P^H(A)$ will represent the costs of all non-dominated paths from a node to the destination node;

—the top element of the semiring is the set A (its compact form is $\{1\}$, which in our example is $\{\langle 0, 0\rangle\}$);

—the bottom element is the empty set;

—the additive operation $\uplus$ is the *formal union* [Smyth 1978] that takes two sets and obtains their union;

—the multiplicative operation $\times^*$ takes two sets and produces another set obtained by multiplying (using the multiplicative operation $\times$ of the original semiring, in our case $+$') each element of the first set with each element of the second one;

—the partial order of this semiring is as follows: $a \leq_{P^H(S)} b$ iff $a \uplus b = b$, that is for each element of a, there is an element in b which dominates it (in the partial

order $\leq_S$ of the original semiring).

From the theoretical results in [Smyth 1978], adapted to consider c-semirings, we can prove that $P^H(S)$ and its more compact form are indeed isomorphic. Moreover, we can also prove that given a c-semiring S, the structure $P^H(S)$ is a c-semiring as well.

THEOREM 6.1. *Given a c-semiring* $S = \langle A, +, \times, 0, 1\rangle$*, the structure* $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$ *obtained using the* Power domain of Hoare *operator is a c-semiring.*

The proof easily follows from the properties of the $\times$ operator in the c-semiring S and from the properties (commutativity, associativity, and idempotence) of the formal union $\uplus$ in $P^H(S)$.

Note that in this theorem we do not need any assumption over the c-semiring S. Thus the construction of $P^H(S)$ can be done for any c-semiring S. Notice also that, if S is totally ordered, the c-semiring $P^H(S)$ does not give any additional information w.r.t. S. In fact, if we consider together the empty set (with the meaning that there are no paths) and the set containing only the bottom of A (with the meaning that there exists a path whose cost is 0), it is possible to build an isomorphism between S and $P^H(S)$ by mapping each element p (a set) of $P^H(A)$ onto the element a of A such that $a \in p$ and a dominates all elements in the set p.

The only change we need to make to the program with variables, in order to work with this new semiring, is that costs now have to be represented as singleton sets. For example, clause c_{pq} `:- < 2, 4 >.` will become c_{pq} `:- {< 2, 4 >}.`

Still considering the example in Sec. 4.2, Let us now see what happens in our example if we move to this new semiring. First we give a goal like `:- p(X).`. As the answer, we get a set of pairs, representing the costs of all non-dominated paths from p to v. All these costs are non-comparable in the partial order, thus the user is requested to make a choice. However, this choice could identify a single cost or also a set of them. In this second case, it means that the user does not want to commit to a single path from the beginning and rather prefers to maintain some alternatives. The choice of one cost of a specific non-dominated path will thus be delayed until later. If instead the user wants to commit to one specific cost at the beginning, say $\langle c_1, c_2\rangle$, he/she then proceeds to find a path which costs exactly $\langle c_1, c_2\rangle$. By comparing the answers for all goals of the form $p(a)$, where a represents one of the arcs out of p, we can see which arc can start a path with cost $\langle c_1, c_2\rangle$. In fact, such an arc will be labeled $\langle c_{1a}, c_{2a}\rangle$ and will lead to a node with an associated set of costs $\langle c_3, c_4\rangle$ such that $\langle c_3, c_4\rangle \times \langle c_{1a}, c_{2a}\rangle = \langle c_1, c_2\rangle$. Suppose it is the arc from p to q, which has cost $\langle 7, 3\rangle$. Now we do the same with goals of the form $q(a)$, to see which is the next arc to follow. However, we now have to look for the presence of a pair $\langle c_3, c_4\rangle$ such that $\langle c_3, c_4\rangle \times \langle 7, 3\rangle = \langle c_1, c_2\rangle$.

Notice that each time we look for the next arc, we choose just one alternative and we disregard the others. If we used a fixpoint (or any bottom-up) semantics to compute the answer of the initial goal :- p(X)., then all the other answers we need for this method have been already computed, thus the method does not require any additional computational effort to find a non-dominated path.

Notice also that the sets of costs associated with each node are non-dominated.

Thus the size of these sets in the worst case is the size of the maximal "horizontal slice" of the partial order: if we can have at most $\mathbb{N}$ non-dominated element in the partial order, then each of such sets will have size at most $\mathbb{N}$. Of course in the worst case $\mathbb{N}$ could be the size of the whole semiring (when the partial order is completely "flat").

Most classical methods to handle multi-criteria SP problems find the shortest paths by considering each criteria separately, while our method deals with all criteria at once. This allows to obtain optimal solutions which are not generated by looking at each single criteria. In fact, some optimal solutions could be non-optimal in each of the single criteria, but still are incomparable in the overall ordering. Thus we offer the user a greater set of non-comparable optimal solutions. For example, by using a cost-delay multi-criteria scenario, the optimal solution w.r.t. cost could be 10 euro (with a delay of 100msec), while the optimal solution w.r.t. delay could be 10msec (with a cost of 100 euro). By considering both criteria together, we could also obtain the solution with 20 euro and 20msec!

Note that the considerations on partially-ordered problems of this Section clearly state for the multicast tree example in Section 5.1 as well. In this case, $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$ uses the semiring for bandwidth-delay multi criteria: $S = \langle\langle \mathcal{B} \cup \{0, +\infty\}, \mathcal{D} \cup \{0, +\infty\}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$, where $\mathcal{B}$ is the set of bandwidth values, $\mathcal{D}$ is the set of delay values, $+'$ is $\langle \max, \min\rangle$ and $\times'$ is $\langle \min, \max\rangle$. Therefore, $\times^*$ uses $\langle \min, \max\rangle$ $(\times')$ to compose two sets, and $\uplus$ the ordering $\leq_S$ defined by $\langle \max, \min\rangle$ $(+')$.

Finally, this method is applicable not only to the multi-criteria case, but to any partial order, giving us a general way to find a non-dominated path in a partially-ordered SP problem. It is important to notice here the flexibility of the semiring approach, which allows us to use the same syntax and computational engine, but on a different semiring, to compute different objects.

7. CONCLUSIONS

We have described a method to represent and solve the unicast/multicast QoS routing problem with the combination of graph/hypergraph and SCLP programming: *i)* the best path found in this way corresponds to the best unicast route distributing (for example) multimedia content from the source to the only receiver; *ii)* the same considerations are also valid for the best tree found over an *and-or* graph, since it corresponds to the best multicast distribution tree towards all the receivers. The best path/tree optimizes objectives regarding QoS performance, e.g. minimizing the global bandwidth consumption or reducing the delay, and can satisfy constraints on these metric values at the same time. The structure of a *c-semiring* defines the algebraic framework to model the costs of the links, and the SCLP framework describes and solves the SCSP problem in a declarative fashion. Since several distinct criteria must be all optimized (the costs of the arcs may include multiple QoS metric values), the best route problem belongs to the multi-criteria problem class, i.e. it can result in a partially-ordered problem. Moreover we have seen also how to deal with modality-based problems, relating them to preferences connected to policy routing rules. Therefore, the model proposed in this paper can be used to reason upon (and solve!) CBR, that is, in general, a NP-Complete problem. Note

also that, even if the paper is focused on routing problems, the same framework can be generally adopted with every problem related to MCOP or ST.

For what about regarding the related works, in [de Nicola et al. 2003] and [Hirsch and Tuosto 2005] the authors adopt a hypergraph model in joint with semirings too, but the minimal path between two nodes (thus, not over an entire tree) is computed via a graphical calculous instead of SCLP. At the moment, all these frameworks are not comparable from the computational performance point of view, since we have not yet collected a reasonable amount of timing results (for large networks in particular), and, so far the systems in [de Nicola et al. 2003] and [Hirsch and Tuosto 2005] have not yet been implemented.

In the future, we plan to enrich this framework by using *Soft Concurrent Constraint Programming* (SCCP) [Bistarelli et al. 2006] to handle the interactions among the routing devices and the receivers, and, consequently, we would like to introduce new "soft" operations (e.g. a *retract* of a constraint) to enable the release of the resources reserved by the receivers of the communication, introducing a non-monotonic evolution of the constraint store (which is not allowed in classical SCCP).

A second step should consist in further expanding the SCCP framework with some simple primitives which allows to specify timing constraints. According to our opinion, time critical aspects are essential to the management of QoS, and, in general, when modelling the possible interactions among distributed or concurrent systems. These entities must continuously react to the inputs coming from the environment and act in an appropriate manner.

A further extension to our QoS framework could be the introduction of probabilistic metrics as the weight of the graph-links: we could consider this value as the probability of packet loss on that connection, or the probability of a connection existence between two nodes in the network. In this case, the global probability of existence of a path between two nodes p and v depends on the probability of all the possible different paths connecting p and v in the graph. The problem is represented by the composition of the different probabilities of these paths, which cannot be easily modelled with a semiring.

Future research could address also the remodelling of the best tree due to the continuous network-state changes, including the requests of multicast group members to dynamically join in and leave from the communication, or the updates of the QoS metric values on the links, since a network must be efficiently used to transport multiple flows at the same time.

REFERENCES

Berman, L., Kou, L., and Markowsky, G. 1979. A fast algorithm for steiner trees. *Acta Informatica 15,* 2 (December), 141–145.

Bistarelli, S. 2004. *Semirings for Soft Constraint Solving and Programming (Lecture Notes In Computer Science).* SpringerVerlag.

Bistarelli, S., Frühwirth, T., and Marte, M. 2002. Soft constraint propagation and solving in chrs. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing.* ACM Press, New York, NY, USA, 1–5.

Bistarelli, S., Montanari, U., and Rossi, F. 1995. Constraint Solving over Semirings. In *Proc. IJCAI95.* Morgan Kaufman, 624–630.

Bistarelli, S., Montanari, U., and Rossi, F. 1997a. Semiring-based Constraint Logic Programming. In *Proc. IJCAI97*. Morgan Kaufman, 352–357.

Bistarelli, S., Montanari, U., and Rossi, F. 2002. Soft constraint logic programming and generalized shortest path problems. *Journal of Heuristics 8,* 1, 25–41.

Bistarelli, S., Montanari, U., and Rossi, F. 2006. Soft concurrent constraint programming. *ACM Trans. Comput. Logic 7,* 3, 563–589.

Bistarelli, S., Montanari, U., and Rossi, F. March 1997b. Semiring-based Constraint Solving and Optimization. *Journal of the ACM 44,* 2, 201–236.

Bistarelli, S., Montanari, U., Santini, F., and Rossi, F. Modelling multicast qos routing by using best-tree search in and-or graphs and soft constraint logic programming. To appear in Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL2007).

Bueno, F., Cabeza, D., Carro, M., Hermenegildo, M., Lopez, P., and Puebla, G. 1997. The ciao prolog system. reference manual. The CIAO System Documentation Series–TR CLIP3/97.1.

Chen, S. and Nahrstedt, K. 1998. An overview of quality of service routing for next-generation high-speed networks: Problems and solutions. *IEEE Network 12,* 6 (November/December), 64–79.

Chen, S., Nahrstedt, K., and Shavitt, Y. 2000. A QoS-aware multicast routing protocol. In *INFOCOM (3)*. 1594–1603.

Cormen, T. T., Leiserson, C. E., and Rivest, R. L. 1990. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA.

Crawley, E., Nair, R., Rajagopalan, B., and Sandick, H. 1998. RFC 2386: A framework for QoS-based routing in the Internet. Informational.

Cui, B. and Warren, D. S. 2000. A system for tabled constraint logic programming. In *CL '00: Proceedings of the First International Conference on Computational Logic*. Springer-Verlag, London, UK, 478–492.

de Nicola, R., Ferrari, G. L., Montanari, U., Pugliese, R., and Tuosto, E. 2003. A formal basis for reasoning on programmable qos. In *Verification: Theory and Practice*, N. Dershowitz, Ed. Lecture Notes in Computer Science, vol. 2772. Springer, 436–479.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Georget, Y. and Codognet, P. 1998. Compiling semiring-based constraints with clp (fd, s). In *CP '98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, London, UK, 205–219.

Hirsch, D. and Tuosto, E. 2005. Shreq: Coordinating application level qos. In *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*. IEEE Computer Society, Washington, DC, USA, 425–434.

Jaffar, J. and Maher, M. J. 1994. Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, 503–581.

Kompella, K. and Awduche, D. 2001. Notes on path computation in constraint-based routing. Internet Draft.

Korkmaz, T. and Krunz, M. 2001. Multi-constrained optimal path selection. In *INFOCOM*. 834–843.

Kuipers, F. A., Korkmaz, T., Krunz, M., and Mieghem, P. V. 2004. Performance evaluation of constraint-based path selection algorithms. *IEEE Network 18,* 5, 16–23.

Ma, Q. and Steenkiste, P. 1997. Quality of service routing for traffic with performance guarantees.

Martelli, A. and Montanari, U. 1978. Optimizing decision trees through heuristically guided search. *Commun. ACM 21,* 12, 1025–1039.

Mieghem, P. V., Neve, H. D., and Kuipers, F. A. 2001. Hop-by-hop quality of service routing. *Computer Networks 37,* 3/4, 407–423.

Mohri, M. 2002. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb. 7,* 3, 321–350.

Moy, J. 1998. RFC 2328: OSPF version 2. Standard.

Rosen, E., Viswanathan, A., and Callon, R. 2001. Multiprotocol Label Switching Architecture.

Rouskas, G. N. and Baldine, I. 1997. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal of Selected Areas in Communications 15,* 3, 346–356.

Schrijvers, T. and Warren, D. S. 2004. Constraint handling rules and tabled execution. In *ICLP*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer, 120–136.

Smyth, M. B. 1978. Power domains. *Journal of Computer and System Sciences 16,* 1 (Feb.), 23–36.

Wang, B. and Hou, J. 2000. Multicast routing and its QoS extension: problems, algorithms, and protocols. *IEEE Network 14*.

Wang, Z. 1999. On the complexity of quality of service routing. *Inf. Process. Lett. 69,* 3, 111–114.

Wang, Z. and Crowcroft, J. 1996. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications 14,* 7, 1228–1234.

Winter, P. 1987. Steiner problem in networks: a survey. *Netw. 17,* 2, 129–167.

Xiao, X. and Ni, L. M. 1999. Internet qos: A big picture. *IEEE Network 13,* 2 (March), 8–18.

Younis, O. and Fahmy, S. 2003. Constraint-based routing in the internet: Basic principles and recent research. *IEEE Communications Surveys and Tutorials 5,* 1, 2–13.