Resource Allocation in Multi-Domain Networks Based on Service Level Specifications

Stefano Avallone, Salvatore D'Antonio, Marcello Esposito, Simon Pietro Romano, and Giorgio Ventre

Abstract: **The current trend toward the utilization of the Internet as a common means for the offer of heterogeneous services imposes to address the issues related to end-to-end service assurance in the inter-domain scenario. In this paper, we first present an architecture for service management in networks based on service level specifications (SLS). The architecture is designed to be independent both of the specific network technology adopted and of the high level service semantics. Then, we focus on a specific functionality of the proposed architecture: Resource allocation in the multi-domain scenario. A distributed admission control algorithm is introduced, its complexity is evaluated and a comparison with related solutions is provided.**

Index Terms: **Inter-domain operation, network management, service level specification (SLS).**

I. INTRODUCTION

Network management has represented one of the major issues since the Internet has started growing from a small research network to the world-wide infrastructure available nowadays. With the introduction of new paradigms different than the original *best effort*, such task has become harder and harder to accomplish in an effective way. Network administrators cannot refrain from taking into consideration the co-existence on the same physical architecture of a number of heterogeneous services, each imposing different requirements in terms of traffic patterns and needed network resources.

We propose in this work an architecture exploiting a component based model, where roles and interfaces are clearly defined. Such an architecture consists of a stack of three layers, each perceiving quality of service (QoS) concepts at a different level of abstraction. The highest one is characterized by the maximum degree of generality and is totally independent of the QoS paradigm adopted, of the underlying network technology and of the physical characteristics of the single nodes as well. Thus, decisions taken within the boundary of this layer are based upon a minimal set of assumptions about the transport infrastructure. This upper layer is also the place where we look after two fundamental aspects: Time as a QoS parameter and inter-domain operation. Lower layers progressively reduce the level of abstraction in order to have, eventually, a set of traffic control commands for network devices configuration.

Manuscript received April 29, 2004; approved for publication by Danny Raz, Division III Editor, July 15, 2005.

S. Avallone, S. P. Romano, and G. Ventre are with the Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, Italy, email: {stavallo, spromano, giorgio}@unina.it.

S. D'Antonio is with the ITEM Laboratory, Consorzio CINI, Italy, email: sdantonio@napoli.consorzio-cini.it.

M. Esposito is with the CRIAI, Italy, email: m.esposito@criai.it.

After introducing the overall architecture, we dig into the details of the inter-domain resource allocation process. In particular, we propose a distributed admission control algorithm which takes into account the coexistence of multiple domains along the end-to-end path connecting the service requester to the service provider.

The major obstacles encountered by the wide deployment of appropriate mechanisms for end-to-end QoS provisioning dig their roots in barely political reasons, rather than either technological or scientific ones. Hence, it is a mandatory requirement for any deployable solution to take into account the explicit wills of the various stakeholders involved in the service delivery chain. The management architecture we propose, together with the algorithm we devised for it, has been conceived at the outset to tackle exactly the above issue. More precisely,

- no global knowledge of the state of the network is requested;
- a network manager is not required to publish any information related to the inner structure of his own network, as well as to the specific policies he adopts (topology, routing, traffic engineering, charging, etc.);
- a network manager can embrace the proposed management architecture regardless of both the underlying transport technology and the protocols employed in the network (provided that he somehow ensures QoS guarantees in his own domain);
- the network providers are not compelled to converge on a unique network technology: The higher layers of the management architecture provide a uniform interface ensuring interoperability and shielding from the lower-level technological details;
- the management architecture allows for inter-provider cooperation while at the same time leaving intra-domain management solely under the responsibility of the domain's owner. In other words, each physical device does not have to be open to configuration from other entities but the owner of the domain in which the device itself is located.

The above points are recognized as key enabling factors for any provider wishing to effectively administrate his network while contributing to the QoS-aware service delivery chain. It will be shown in this paper that the architecture we propose provides full support to such enabling factors.

The paper is organized in seven sections. Section II provides some background information about the concept of service level specification (SLS). In Section III, our three-layer architecture is illustrated. Section IV focuses on the issues related to SLS-based multi-domain admission control. In particular, Section IV-E provides a complexity analysis of the proposed algorithm. Section V shows how the proposed algorithm lends itself to a distributed implementation. Section VI draws a comparison between our solution and previous work related to the same

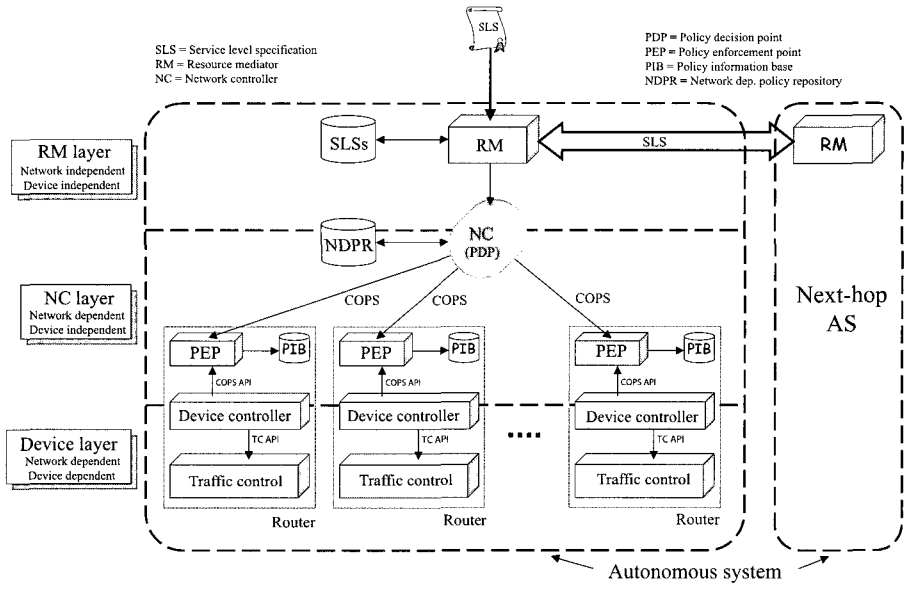


Fig. 1. The reference network architecture.

topic. Finally, Section VII contains some concluding remarks.

II. THE SLS AS SERVICE ABSTRACTION

The aim to design a network architecture capable to offer generic services requires a thorough definition of *what a service is*. The service provider and the network provider, which we consider as two neatly separated actors, have to agree on such a definition in order to correctly carry out the negotiation of a service instance. Hence, a SLS is a fundamental concept since it contains the definition of an abstract service a network can offer, at a level of generality which is as high as possible: A network provider can offer *all what can fit inside an SLS and no more than this*.

A formal definition of the concept of SLS has been recently provided [2]. The fields it contains are scope, flow identification, traffic envelope and traffic conformance, excess treatment, performance guarantees, service schedule, and reliability. Four of these fields play a major role during the service negotiation phase.

- Scope, determining the service ingress and egress points—that is, *where* the service has to be enforced;
- traffic envelope, containing a characterization of the traffic associated with the service instance—that is, *what* service has to be enforced;
- performance guarantees, specifying the guarantees associated with the service (in the form of maximum delay, jitter, packet loss, and throughput)—that is, *how* the service has to be enforced;
- service schedule, indicating the time schedule related to the service—that is, *when* the service has to be enforced.

In the following, we take for granted the existence of an SLS compliant with the four fields above. Under this assumption, the service negotiation phase can be triggered by submitting an SLS to a network provider and subsequently waiting for a response that indicates whether the related service instance has been accepted.

III. THE REFERENCE NETWORK ARCHITECTURE

As shown in Fig. 1, starting from an SLS instance (coming from a service provider not shown in the picture), we cross all of the management framework components (resource mediator—RM, network controller—NC, devices) in order to arrive at the network devices and appropriately configure them. In the following, we will focus on each of the layers of our architecture.

A. The RM Layer

The main task of the uppermost layer is to receive SLSs from network clients; hence, the upper layers are provided with an interface for SLS submission. Furthermore, this layer owns and manages the SLS repository; this is necessary not only for merely practical reasons, but also for administrative purposes. Since SLSs are intrinsically time dependent, the uppermost layer has full knowledge of what is going to happen in the future network's life. This makes the RM the best candidate to manage the time-dependent triggers raised by the SLS repository.

Multi-domain network management is another important functionality which will be herein shown as naturally pertaining to this layer. During the negotiation of an SLS spanning over multiple domains, a certain number of RMs—those belonging to the crossed domains—must be involved in the negotiation phase. Each RM in the chain is in charge of assuring that part of the service pertaining to its own domain. *How is it possible to let different RMs cooperate in order to reach end-to-end service enforcement?* An answer can be found if we think that all of the RMs are able to accept SLSs from network clients and nothing prevents an RM from being a network client of a neighboring RM. This consideration suggests the adoption of a cascade model where a multi-domain SLS can be recursively split in two parts: A single domain SLS plus a remaining part that has to be enforced elsewhere, i.e., over one or more downstream

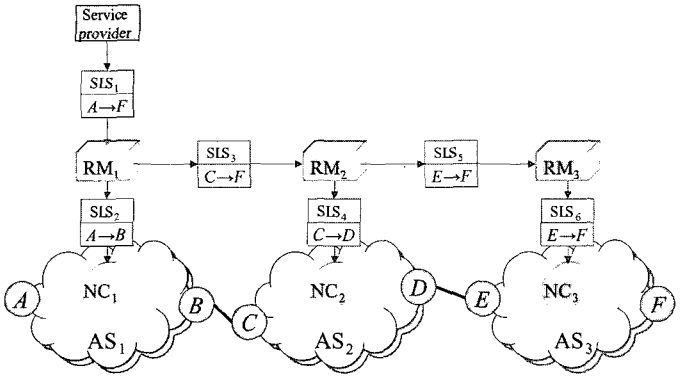


Fig. 2. An example of multi-domain SLS splitting.

domains. Under this light, *end-to-end* service configuration can be achieved by composing one or more *edge-to-edge*[1] services, each described by means of a single-domain SLS instance.

As an example, let us refer to Fig. 2. The figure shows multi-domain SLS processing in case of three different autonomous systems, each managed by a different RM-NC pair. In the picture, only the boundary routers $(A, B, \cdots, F)$ are visible. SLS_1 arrives at RM_1 from a service provider requesting a specific service that has to be instantiated between points A and F. Hence, RM_1 asks NC_1 information about the boundary router toward the destination F. Based on the received response, SLS_1 is split in two parts: SLS_2 and SLS_3. The former, related to a single domain, is delivered to NC_1 for further network-specific processing [6]; the latter is delivered to RM_2 which, in turn, recognizes it as a multi-domain SLS and starts a new splitting operation. Only local SLSs (SLS_2, SLS_4, and SLS_6) are delivered to the NCs in order to be processed by the intra-domain admission control algorithm. With respect to charging, each RM in the chain just interacts with its direct neighbors: It pays for the service it is receiving from the downstream neighbor (if any) and gets paid for the service it is offering to the upstream neighbor (either the previous RM along the chain or the service provider that triggered the overall process). Thanks to this example, we can now state that *the RM is in charge of addressing the inter-domain issues.*

Summarizing the considerations above, the main features of the RM layer are

- SLS repository management;
- sensitivity to asynchronous time events raised by the SLS repository;
- inter-domain communication.

We just remark that the RM is completely unaware of both network QoS paradigm adopted and network topology.

B. The NC Layer

The central layer of the architecture encapsulates the network-specific functionality. The NC is a logically centralized component that presides over the autonomous system to which it belongs. Its implementation is strictly dependent on the QoS network paradigm adopted. Since different networks need different NC implementations, a brand new NC must be implemented whenever a new network technology is adopted. Hence, we put outside the NC all of the network independent functions such as inter-domain and time dependency. This aspect has another important side. Different ASs are not compelled to converge to a unique QoS paradigm thanks to the compatibility at the RM layer (which completely hides the specific underlying network technology). Thus, each AS inside a network can choose its favorite QoS paradigm, regardless of the solutions implemented within other ASs.

The main functionality of the NC layer is intra-domain admission control (AC). Since here the underlying technology (DiffServ, MPLS, overprovisioning, etc.) is well known, the NC is able to verify whether a new service stemming from an SLS can be accepted without jeopardizing the allocated resources. For example, in the case of a DiffServ network, a new request will be mapped onto a set of nodes—according to the routing protocol—and onto a DiffServ class of service—depending on the guarantees needed by the service. If enough resources are available for the chosen class along the entire path identified, AC is successful. Otherwise, a negative response is returned. For MPLS networks, AC is based on the search of a suitable label switched path. In the case of a DiffServ over MPLS network the degrees of freedom are even more, and the AC has more chances to give a successful response. Finally, for a simple overprovisioned network, the AC algorithm is a dummy routine which always returns a positive answer.

Unlike the RM, from the NC standpoint, admission control represents a transaction that can be decomposed in a set of single requests to the nodes involved in the intra-domain service provisioning chain. *Only if all of the requests give a successful* response, the overall AC succeeds. Then, we might say that the NC performs an *edge-to-edge* AC by appropriately combining a set of *point-to-point* ACs. Again, this gives the flavor of how the level of abstraction progressively decreases when proceeding from the uppermost layer toward the lowest one.

Communication with the network nodes can be accomplished by means of the common open policy service (COPS) protocol [5] which introduces the policy decision point (PDP) and the policy enforcement point (PEP) entities. The PDP is a centralized entity in charge of taking decisions to be sent to the PEPs for enforcement. This paradigm perfectly fits the approach we chose: The NC acts as a PDP and sends its decisions to the network nodes acting as PEPs.

C. The Devices Layer

The lowermost layer is populated with the set of network devices, each one independent of the others. This layer performs the lowest level actions. The main service it offers is the translation of the network configuration policies, formulated at the abstract level of the QoS paradigm adopted, into configuration commands whose syntax depends on the network device implementation. For example, an MPLS network can be built of different label switched routers from different manufacturers (e.g., Cisco, Linux, etc.). Thanks to this layer, the network is still seen by the NC as a homogeneous MPLS network, regardless of the implementation differences. The policies flowing down from the NC toward the network nodes will be formulated

[1]By definition, *edge-to-edge* refers to that part of a service falling between two boundary routers within the same AS.

at the MPLS level of abstraction (e.g., "bind LSP with label X to outgoing interface Y"). The device controller (DC) component on each network element is in charge of translating the policies into configuration commands to be enforced on the traffic control modules through their API. We just point out that this kind of messages can also flow in the opposite direction; for instance, this happens in the case of data related to the monitoring of the devices. Also in this case, starting from local statistics, it is possible for the NC to evaluate the *edge-to-edge* service behavior and by RMs interaction the level of abstraction can be further increased so to reach an *end-to-end* service performance evaluation.

As far as implementation issues, a different device controller must be designed for each different QoS paradigm and for each different device model. Anyway, such DCs contain no more than mapping rules allowing the translation of policies into traffic control commands.

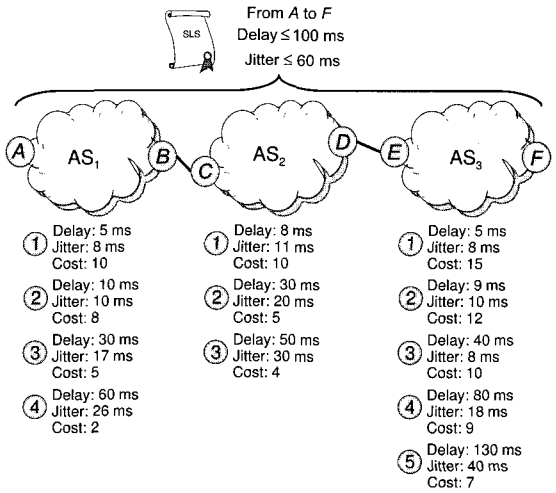


Fig. 3. A finite number of classes are available at each AS.

IV. MULTI-DOMAIN ADMISSION CONTROL

In this section, we will dig into the details of a relevant functionality of the framework: Multi-domain, SLS-based admission control. We will explain how the SLS parameters can be distributed over multiple domains ensuring both the respect of the negotiated guarantees and the minimization of the afforded costs.

A. The SLS Parameters Distribution Problem

In the previous sections, we introduced the SLS splitting process: An SLS spanning over multiple domains can be split in a set of intra-domain SLSs to be enforced in a centralized fashion. Unfortunately, the SLS conveys an end-to-end view of the service it describes. For example, the *delay* specified in a multi-domain SLS refers to the total amount of delay for each packet, from the network ingress-point to the egress-point, regardless of the number of crossed network domains. Therefore, each autonomous system adds its own contribution to the overall delay specified in the SLS. A problem comes to the fore: Since the delay is an additive parameter, *how is it possible to distribute it over the crossed domains*?

One might think to equally distribute the delay among the domains. For example, if the total delay specified by SLS_1 in Fig. 2 were equal to 100 ms from A to F, the three ASs might be asked to guarantee a delay of 33 ms each. In general, this strategy does not represent the optimal one. In fact, the ASs sell their resources at different prices. Hopefully, the cheaper the AS, the more the resources bought from it.

Let's return again to our well-known 3-domains example. A typical situation is depicted in Fig. 3. An SLS asking for an overall delay of 100 ms and a total jitter of 60 ms has to be enforced over three domains. We can imagine that a different number of service levels is available at each domain. Each service level is associated with a different edge-to-edge (i.e., AS) delay-jitter pair and a different cost. For instance, the second AS has 3 available service levels; a flow associated with the first one will be subject to a delay of 8 ms (from C to D), a jitter of 11 ms and charged a cost of 10. The problem is to find the combination of service levels that provides the requested guarantees (or even better) at the lowest cost. In our example, the optimal combination includes the fourth service level of AS_1, the second of AS_2, and the second of AS_3, as will be formally demonstrated in Section IV-C. Such a combination provides an end-to-end delay of 99 ms and a total jitter of 56 ms at a cost of 19.

This trivial example differs from a real-world scenario for at least two reasons.

- It takes into account only the delay and jitter constraints. Actually, all of the requirements imposed by an SLS have to be considered;
- we herein assumed that a finite number of service levels can be available at each AS, i.e., the offered guarantees cannot continuously vary but the allowed configurations are discretely distributed.

The former simplification will be removed in the following, as we will present an algorithm which considers $m \geq 1$ constraints. The latter seems quite realistic and will not be abandoned. In fact, in the case of DiffServ environments, the only degree of freedom is represented by the assignment of a flow to a finite number of DiffServ classes (i.e., PHBs). In MPLS, routing is a further degree of freedom, but the routes, albeit numerous, are still finite. In the following we will refer to the available service levels as *classes*.

In the light of the above considerations, the presented problem can be seen as a *multi-dimensional non-linear optimization problem*. The multi-dimensionality is due to the existence of several parameters to be simultaneously distributed (delay, jitter, packet-loss, throughput) and the non-linearity can be ascribed to the cost trend with respect to the reserved resources. Such a behavior is arbitrarily defined by each network provider according to its own policies.

B. Problem Formulation

This section is devoted to formally define the SLS splitting problem. We assume each service class be specified by m QoS measures (e.g., delay, packet loss, etc.). The overall requirements imposed by the SLS are represented by the m-

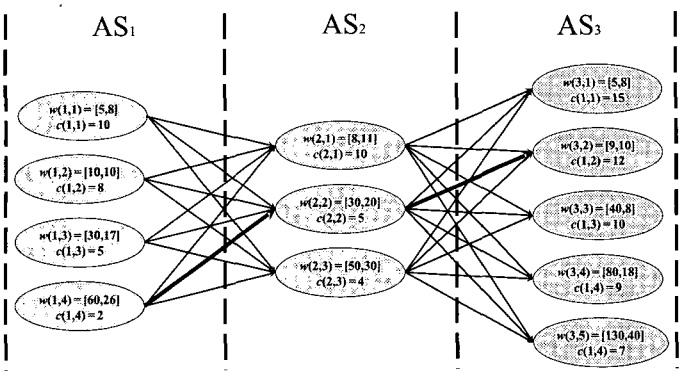


Fig. 4. The graphical problem representation in the 3-domains scenario. The bold arrows identify a particular problem solution.

dimensional constraint vector $\vec{Q} = [Q_1, \cdots, Q_m]$. We distinguish among three classes of QoS measures: Bottleneck, additive, and multiplicative. Bottleneck QoS measures are such that their value over multiple ASs is the minimum (maximum) among the values in every single AS. The most commonly used bottleneck QoS measure is the throughput. Additive (multiplicative) measures are such that their value over multiple ASs is the sum (product) of the values in every single AS. Hop count, delay and jitter are typical additive QoS measures, while 1 minus the probability of packet loss is a multiplicative measure. Bottleneck measures can be easily dealt with by discarding the service classes offering a smaller (larger) value than the minimum (maximum) required. Multiplicative measures can be converted into additive ones by taking the logarithm. Therefore, without loss of generality, we only consider additive measures.

Definition IV.1 (SLS splitting problem): Let us denote by K the number of domains to be crossed, N_i the (finite) available number of classes inside the i-th AS, $\vec{w}(i,j)$ the vector of the m additive QoS measures values guaranteed at a non-negative cost $c(i,j)$ by the j-th class inside AS_i and $\vec{Q}$ the vector of QoS constraints. The SLS splitting problem is to find, if any, a K-tuple $(j_1, j_2, \cdots, j_K)$, $j_i = 1, \cdots, N_i$, $\forall i = 1, \cdots, K$ such that

$$\begin{aligned} &\text{(i)} \quad \sum_{i=1}^{K} w_l(i, j_i) \leq Q_l, \ \forall l = 1, \cdots, m \\ &\text{(ii)} \quad (j_1, \cdots, j_K) \in \\ &\qquad \operatorname{argmin}_{(j'_1, \cdots, j'_K)\text{satisfying(i)}} C(j'_1, \cdots, j'_K) \end{aligned} \tag{1}$$

where $C(j_1, \cdots, j_K) = \sum_{i=1}^{K} c(i, j_i)$ is the overall cost of the K-tuple $(j_1, \cdots, j_K)$.

A K-tuple $(j_1, \cdots, j_K)$ is said to be feasible if it satisfies (i). The SLS splitting problem is therefore to find a feasible K-tuple having the smallest cost. The following subsection gives a formulation of the algorithm we propose to solve the problem defined above.

C. Solution Algorithm

As for every combinatorial problem, the simplest solution relies on an exhaustive search extended to the entire solution space. An iterative algorithm performing K steps can serve this purpose. The i-th step combines the $(i-1)$-tuples with the classes of AS_i to produce the candidate i-tuples. Step K returns all the possible solutions. A graphical representation of the problem may help make the way such algorithm works clearer. Fig. 4, e.g., depicts our 3-domains sample scenario. The nodes represent the available classes and the edges connect two nodes belonging to neighboring ASs. Nodes belonging to the same AS are not connected, as well as nodes belonging to non-adjacent ASs. A candidate solution is thus represented by a path connecting in some way a class inside the first AS to a class inside the last AS.

Unfortunately, the number of solutions to be evaluated grows exponentially with the number of ASs (K). Complexity can be reduced by exploring a subset of all the possible paths, provided that the optimal solution is still returned. The approach we propose achieves an efficient search space reduction while returning the optimal solution. The basic idea is to early discard not only the unfeasible I-tuples but also the *dominated* ones.

Definition IV.2 (dominated I-tuples): An I-tuple $(j_1^b, \cdots, j_I^b)$, $I \in \{1, \cdots, K\}$, is said to be dominated by an I-tuple $(j_1^a, \cdots, j_I^a)$ if and only if

$$\sum_{i=1}^{I} w_l(i, j_i^a) \leq \sum_{i=1}^{I} w_l(i, j_i^b), \ \forall l = 1, \cdots, m \tag{2}$$

$$C(j_1^a, \cdots, j_I^a) \leq C(j_1^b, \cdots, j_I^b) \tag{3}$$

where $C(j_1, \cdots, j_I) = \sum_{i=1}^{I} c(i, j_i)$ is the cost of the I-tuple $(j_1, \cdots, j_I)$.

In other words, if two I-tuples $(j_1^a, \cdots, j_I^a)$ and $(j_1^b, \cdots, j_I^b)$ are found at step I such that the former provides better QoS values at a lower cost than the latter, then we discard $(j_1^b, \cdots, j_I^b)$. We now prove that such operation does not affect the optimality of the solution returned. Two events must not occur:

- (e_1) The dominated I-tuple leads to a feasible K-tuple and the dominant I-tuple does not lead to any feasible K-tuple.
- (e_2) The dominated I-tuple leads to a feasible K-tuple with a smaller length than that of the K-tuple the dominant I-tuple leads to.

Let us denote by $(j'_{I+1}, \cdots, j'_K)$ a $(K-I)$-tuple such that $(j_1^b, \cdots, j_I^b, j'_{I+1}, \cdots, j'_K)$ is a feasible K-tuple. Using inequalities (2), it yields

$$\sum_{i=1}^{I} w_l(i, j_i^a) + \sum_{i=I+1}^{K} w_l(i, j'_i) \leq \sum_{i=1}^{I} w_l(i, j_i^b) + \sum_{i=I+1}^{K} w_l(i, j'_i) \leq Q_l, \ \forall l = 1, \cdots, m$$

i.e., $(j_1^a, \cdots, j_I^a, j'_{I+1}, \cdots, j'_K)$ is a feasible K-tuple. Thus, event (e_1) cannot occur. Analogously, it can be shown, using (3), that event (e_2) cannot occur, concluding the proof that our algorithm returns an optimal solution. Finally, we remark that no particular hypothesis needs to be made on the nature of the discrete cost function $c(i,j)$.

D. Algorithm Pseudo-Code

The pseudo-code of the algorithm we propose to solve the SLS splitting problem (Definition IV.1) is shown in Fig. 5. The inputs to our algorithm are the number of ASs (K), the constraints on the additive QoS measures ($\vec{Q}$) and the QoS values

```
SLS-SPLITTING(K, q⃗, offer)
 1  c[n] ← 0
 2  w⃗[n] ← 0⃗
 3  LIST-INSERT(S[0], n)
 4  for i ← 1 to K
 5      do if S[i − 1] is empty
 6           then return NIL
 7         for each u ∈ S[i − 1]
 8             do for each z ∈ offer[i]
 9                    do w⃗[v] ← w⃗[z] + w⃗[u]
10                       c[v] ← c[z] + c[u]
11                       id_i[v] ← id[z]
12                       if IS-FEASIBLE(v, q⃗) and
                         IS-NOT-DOMINATED(v, S[i])
13                         then LIST-INSERT(S[i], v)
14                              DOMINANCE-CHECK(v, S[i])
15  return LIST-MINIMUM-COST(S[K])
```

Fig. 5. Pseudo-code SLS-SPLITTING.

```
IS-FEASIBLE(v, q⃗)
1  for i ← 1 to m
2      do if w_i[v] > Q_i
3           then return FALSE
4  return TRUE
```

Fig. 6. Pseudo-code IS-FEASIBLE.

```
IS-NOT-DOMINATED(v, S)
1  for each u ∈ S
2      do min ← w_1[v] − w_1[u]
3         for i ← 2 to m
4             do if w_i[v] − w_i[u] < min
5                  then min ← w_i[v] − w_i[u]
6         if min ≥ 0 and c[v] ≥ c[u]
7           then return FALSE
8  return TRUE
```

Fig. 7. Pseudo-code IS-NOT-DOMINATED

associated with the service classes of all the ASs, represented as an array *offer* of K lists. The j-th element of list $offer[i]$ is an object representing the j-th service class of AS_i. Such objects have a cost field c, a vector field $\vec{w}$ containing QoS values and an integer identifier field id. As far as objects, we adopt the following pseudo-code convention (borrowed from [7]): A particular field is accessed using the field name followed by the name of its object in square brackets. For instance, if z denotes the j-th service class of AS_i then $c[z]$ represents $c(i,j)$ and $\vec{w}[z]$ represents $\vec{w}(i,j)$. Each I-tuple $(j_1 \cdots, j_I)$ determined at the I-th step ($I = 1, \cdots, K$) is represented by an object, say it v, stored in list $S[I]$. Such object has a cost field c ($c[v] = C(j_1 \cdots, j_I)$), a vector field $\vec{w}$ ($\vec{w}[v] = \sum_{i=1}^{I} \vec{w}(i, j_i)$) containing the cumulative QoS values and a vector field $\vec{id}$ whose components are the indices of the I-tuple. Finally, we assume the existence of a constant m indicating the number of QoS measures, whose scope includes all the subroutines.

The procedure SLS-SPLITTING returns NIL if no feasible K-tuple exists and an optimal solution otherwise. Lines 1–3 initialize $S[0]$ to the list containing just a null element. Then, for every step $i = 1, \cdots, K$ the list $S[i-1]$ associated with the previous step is examined. If such list is empty, it means that no

```
DOMINANCE-CHECK(v, S)
1  for each u ∈ S − {v}
2      do max ← w_1[v] − w_1[u]
3         for i ← 2 to m
4             do if w_i[v] − w_i[u] > max
5                  then max ← w_i[v] − w_i[u]
6         if max ≤ 0 and c[v] ≤ c[u]
7           then LIST-DELETE(S, u)
```

Fig. 8. Pseudo-code DOMINANCE-CHECK

feasible solution exists and the procedure returns NIL (line 6). Otherwise, each $(i-1)$-tuple of $S[i-1]$ is combined with the classes in the i-th AS to produce the candidate i-tuples (lines 7–11). Not all such elements are inserted in $S[i]$. For every new i-tuple, the subroutine IS-FEASIBLE (Fig. 6) checks whether it obeys the constraint vector $\vec{Q}$. Then, the subroutine IS-NOT-DOMINATED (Fig. 7) compares the new i-tuple with those already inserted in $S[i]$. If a dominant i-tuple is found in $S[i]$, the subroutine returns FALSE (lines 6–7) and the new i-tuple is discarded. Otherwise, the new i-tuple is inserted in $S[i]$ (line 13 of SLS-SPLITTING) and the procedure DOMINANCE-CHECK (Fig. 8) is invoked. The purpose of such procedure is to remove possible dominated i-tuples from $S[i]$. If the new i-tuple is found to dominate an element in $S[i]$, this last element is removed from $S[i]$ (lines 6–7). Thus, at any time $S[i]$ contains i-tuples that do not dominate each other. After step K, the list $S[K]$ contains the set of candidate solutions. The subroutine LIST-MINIMUM-COST returns NIL if the list is empty and the object associated with the K-tuple having the lowest cost otherwise. The returned K-tuple, if any, is an optimal solution to the SLS splitting problem.

E. Complexity Analysis

This section is to evaluate the worst-case complexity of SLS-SPLITTING. The initialization (lines 1–3) takes $O(1)$ time. In the worst-case, the outer **for** loop performs K iterations. The **for** loop of line 7 iterates over the element in $S[i-1]$. We denote by $n_{\max}$ the maximum number of elements simultaneously stored in a list at any time. Thus, the **for** loop of line 7 performs $O(n_{\max})$ iterations. The **for** loop of line 8 is repeated at most $N_{\max}$ times, where $N_{\max} = \max_{i \in \{1, \cdots, K\}} N_i$. The assignments of lines 9 to 11 takes $O(1)$ time (if we consider the number of QoS constraints as a constant), so as the subroutines IS-FEASIBLE and LIST-INSERT do. We have to compute the worst-case running time of IS-NOT-DOMINATED. The **for** loop performs $O(n_{\max})$ iterations, each of which takes $O(1)$ time. Hence, the complexity of IS-NOT-DOMINATED is $O(n_{\max})$. Analogously, the complexity of DOMINANCE-CHECK is $O(n_{\max})$. Consequently, the running time of the outer **for** loop in SLS-SPLITTING is $O(K n_{\max}^2 N_{\max})$. The running time of LIST-MINIMUM-COST is $O(n_{\max})$, as it performs a linear search on a list of at most $n_{\max}$ elements. Hence, the worst-case running time of SLS-SPLITTING is $O(K n_{\max}^2 N_{\max})$.

An upper bound to $n_{\max}$ is $\prod_{i=1}^{K} N_i$, of course. But, if QoS values and constraints are integers, we can attain a stricter upper bound. For convenience, we introduce a constraint on the cost of the candidate solutions. Let the sum of the costs

of the most expensive class at each AS be the cost constraint $Q_0 = \sum_{i=1}^{K} \max_{j\in\{1,\cdots,N_i\}} c(i,j)$. Clearly, the cost of every candidate solution must be less than or equal to Q_0. Also, consider the augmented constraint vector $\vec{Q}' = [Q_0\ \vec{Q}]$. Since an i-tuple that does not obey such constraints is discarded, there can be at most $\prod_{i=0}^{m} Q'_i$ distinct objects in list $S[i], \forall i = 1,\cdots,m$, corresponding to all the possible combinations of QoS values and cost. We want to determine the size of a largest subset of elements that do not dominate each other. Two objects u and v do not dominate each other if, for at least two different components $0 \le a \ne b \le m$ holds that $w'_a[u] < w'_a[v]$ and $w'_b[u] > w'_b[v]$, where $\vec{w}'$ is the augmented vector field $[c\ \vec{w}]$. It means that, for any couple of non-dominated objects u and v, at least two components of the $(m+1)$-dimensional vector $\vec{w}'[u]$ must be different from $\vec{w}'[v]$. We now prove by construction that the maximum size $S_{\max}$ of a largest subset of elements that do not dominate each other is

$$S_{\max} = \frac{\prod_{i=0}^{m} Q'_i}{\max\limits_{0\le i\le m} Q'_i}. \tag{4}$$

Without loss of generality, we assume for convenience that the components of $\vec{Q}'$ are ordered in non-increasing order, i.e., $Q'_0 \ge Q'_1 \ge \cdots \ge Q'_m$. Hence, (4) is equivalent to

$$S_{\max} = \prod_{i=1}^{m} Q'_i. \tag{5}$$

We start by considering the case $m = 1$. Since two elements u and v do not dominate each other if $w'_0[u] < w'_0[v]$ and $w'_1[u] > w'_1[v]$ or viceversa, we can build a largest subset of non-dominated objects starting from the element $[Q'_0\ \ 1]$ and adding new elements by decreasing the first component and increasing the other. A largest subset is therefore

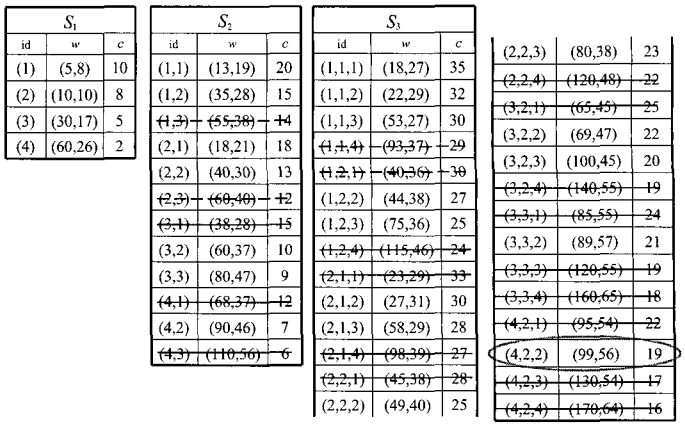
$$\begin{bmatrix} Q'_0 - q_1 \\ 1 + q_1 \end{bmatrix}, \quad q_1 = 0,\cdots,Q'_1 - 1. \tag{6}$$

The number of elements in such subset is Q'_1. Analogously, when considering the case $m = 2$, we build a largest subset starting from the subset $[Q'_0 - q_1\ \ 1+q_1\ \ 1]$, $q_1 = 0,\cdots,Q'_1 - 1$, and adding new elements by decreasing the first component (associated with the largest constraint) and increasing the last one. A largest subset is therefore

$$\begin{bmatrix} Q'_0 - q_1 - q_2 \\ 1 + q_1 \\ 1 + q_2 \end{bmatrix}, \quad \begin{matrix} q_1 = 0,\cdots,Q'_1 - 1 \\ q_2 = 0,\cdots,Q'_2 - 1. \end{matrix} \tag{7}$$

Thus, for a given value of q_1, different values of q_2 give rise to objects that do not dominate each other. The maximum size of such largest subset is $Q'_1 Q'_2$. The maximum size can be achieved when $Q'_0 - (Q'_1 - 1) - (Q'_2 - 1) \ge 1$. Iterating this procedure m times, we obtain a largest subset of $(m+1)$-sized elements that do not dominate each other

$$\begin{bmatrix} Q'_0 - \sum_{i=1}^{m} q_i \\ 1 + q_1 \\ \vdots \\ 1 + q_m \end{bmatrix}, \quad \begin{matrix} q_1 = 0,\cdots,Q'_1 - 1 \\ q_2 = 0,\cdots,Q'_2 - 1 \\ \vdots \\ q_m = 0,\cdots,Q'_m - 1. \end{matrix} \tag{8}$$



S_1		
id	w	c
(1)	(5,8)	10
(2)	(10,10)	8
(3)	(30,17)	5
(4)	(60,26)	2

S_2		
id	w	c
(1,1)	(13,19)	20
(1,2)	(35,28)	15
~~(1,3)~~	~~(55,38)~~	~~14~~
(2,1)	(18,21)	18
(2,2)	(40,30)	13
~~(2,3)~~	~~(68,48)~~	~~12~~
~~(3,1)~~	~~(38,28)~~	~~15~~
(3,2)	(60,37)	10
(3,3)	(80,47)	9
~~(4,1)~~	~~(68,37)~~	~~12~~
(4,2)	(90,46)	7
~~(4,3)~~	~~(110,56)~~	~~6~~

S_3		
id	w	c
(1,1,1)	(18,27)	35
(1,1,2)	(22,29)	32
(1,1,3)	(53,27)	30
~~(1,1,4)~~	~~(93,37)~~	~~29~~
~~(1,2,1)~~	~~(40,36)~~	~~30~~
(1,2,2)	(44,38)	27
(1,2,3)	(75,36)	25
~~(1,2,4)~~	~~(115,46)~~	~~24~~
~~(2,1,1)~~	~~(23,29)~~	~~33~~
(2,1,2)	(27,31)	30
(2,1,3)	(58,29)	28
~~(2,1,4)~~	~~(98,39)~~	~~27~~
~~(2,2,1)~~	~~(45,38)~~	~~28~~
(2,2,2)	(49,40)	25
(2,2,3)	(80,38)	23
~~(2,2,4)~~	~~(120,48)~~	~~22~~
~~(3,2,1)~~	~~(65,45)~~	~~25~~
(3,2,2)	(69,47)	22
(3,2,3)	(100,45)	20
~~(3,2,4)~~	~~(140,55)~~	~~19~~
~~(3,3,1)~~	~~(85,55)~~	~~24~~
(3,3,2)	(89,57)	21
~~(3,3,3)~~	~~(120,55)~~	~~19~~
~~(3,3,4)~~	~~(160,65)~~	~~18~~
~~(4,2,1)~~	~~(95,54)~~	~~22~~
(4,2,2)	(99,56)	19
~~(4,2,3)~~	~~(130,54)~~	~~17~~
~~(4,2,4)~~	~~(170,64)~~	~~16~~

Fig. 9. The algorithm evolution in the scenario depicted in Fig. 3.

The maximum size of such largest subset is $\prod_{i=1}^{m} Q'_i$, which can be attained if $Q'_0 \ge 1 + \sum_{i=1}^{m}(Q'_i - 1)$.

Consequently, $n_{\max} = \min\left[\prod_{i=1}^{K} N_i,\ \frac{\prod_{i=0}^{m} Q'_i}{\max_{0\le i\le m} Q'_i}\right]$. In case of finite granularity of the constraints the second bound applies and the asymptotic complexity can be expressed as $O\left(K\left(\prod_{i=1}^{m} Q_i^2\right) N_{\max}\right)$, i.e., our algorithm exhibits a pseudo-polinomial complexity.

F. Numerical Example

With reference to our 3-domains sample scenario illustrated in Fig. 3, lists $S[1]$, $S[2]$, and $S[3]$ are reported in Fig. 9 as a result of the execution of SLS-SPLITTING. $S[1]$ simply includes the classes available in AS_1. $S[2]$ is obtained by combining every element in $S[1]$ with each class in AS_2 (all of the classes in AS_2 satisfy the overall SLS constraints). We note that some combinations can be early discarded. More precisely, the elements stricken through with a continuous line have been discarded as soon as they were computed (procedure IS-NOT-DOMINATED found a dominant object already present in the list), while those stricken through with a dashed line have been discarded at a later time by procedure DOMINANCE-CHECK due to a newly inserted dominant object. For instance, element $(4,1)$ is dominated by $(3,2)$ and element $(1,3)$ is dominated by $(2,2)$. $S[3]$ is obtained by combining the surviving elements in $S[2]$ with all of the classes in AS[3]. For the sake of conciseness, the combinations stemming from class 5, which does not satisfy by itself the overall constraints, are not listed. Dominated objects are not included in $S[3]$, again, by means of IS-NOT-DOMINATED and DOMINANCE-CHECK. Among the remaining elements, the solution is found by picking the one having the lowest cost, i.e., the K-tuple $(4,2,2)$.

V. IMPLEMENTATION ISSUES

The algorithm we devised has been conceived at the outset as a distributed solution to the QoS partition problem. Notwithstanding, a centralized approach might also be employed for implementation. Such approach implies that all the information related both to classes and associated costs is available at the computing node (i.e., any of the RMs along the chain, or

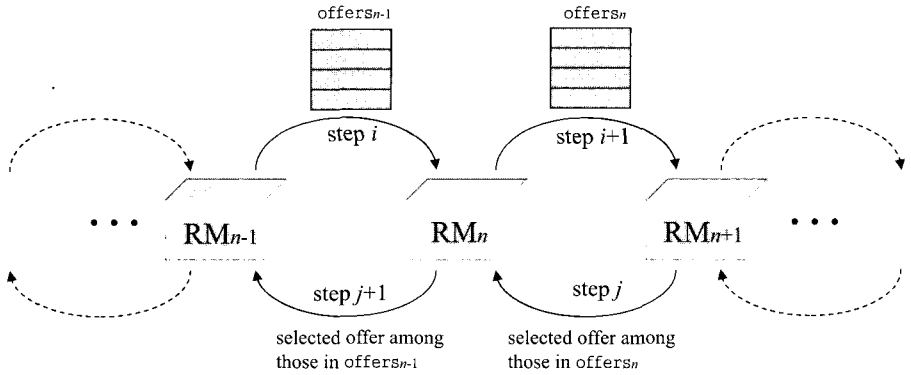


Fig. 10. The role of the intermediate RM_n along the path in the distributed approach.

even a dedicated entity). An ad-hoc protocol is needed in order to collect data that have to be processed by the path-partitioning algorithm. Also in this case, COPS seems to be a good candidate to support the transport of the data units associated with such an ad-hoc solution. This is due to COPS capability to transparently hold communication between information containers (i.e., RMs acting as PEPs) and a centralized information manager (i.e., the computing node acting as PDP).

Coming back to the distributed approach, in this case each RM along the path is responsible for carrying out all the operations pertaining to one single stage. The cascade model best describes this case. As illustrated in Fig. 10, the n-th resource mediator along the chain (RM_n) receives from RM_{n-1} the set of cumulative offers that match the constraints up to stage $n-1$. It merges its own offers with the ones in the received set and delivers the updated set of offers to RM_{n+1}.[2] Afterwards, RM_n enters a waiting state, looking for an answer flowing along the backward direction (i.e., coming from RM_{n+1}). The received answer (step j in the picture) carries information about the cumulative class of service that has been chosen for the n-th stage by the dynamic path partitioning algorithm. This cumulative class belongs to the set $\texttt{offers}_n$ obtained by RM_n after merging the offers from RM_{n-1} ($\texttt{offers}_{n-1}$) with its own. The cumulative class of service informs RM_n about both the local class that has to be configured inside the n-th domain and the cumulative class of service that has to be notified to RM_{n-1}.

Locality of information (i.e., information scope limited to a single hop) is a key point of our approach. With this algorithm no global knowledge about the different classes of service available at the various hops along the value-chain is required.

VI. RELATED WORK

Our work has many liaisons with previous research. Interdomain SLS-based network management, on one side, has been the subject of a number of interesting research studies in the last few years. The SLS specification itself has been one of the major outcomes of the Tequila European research project [8]. With respect to the specific issues related to the interdomain management of SLS-based networks, an effective architecture has been designed in the framework of the Mescal project, which actually represents the continuation of the work carried out within Tequila. Mescal [9] proposes to tackle the interdomain issue by implementing an architecture which leverages the concept of provider-based SLS (pSLS). A pSLS is used to carry information about local QoS classes between each pair of peering domains along the service delivery chain. As the pSLS crosses the involved domains, local QoS classes are transformed into so-called *extended QoS classes*. An extended QoS class indeed represents the level of QoS available at a specific domain falling along the delivery chain. It thus brings memory of the QoS experienced by the interdomain flow while crossing the previous domains. From the architectural standpoint, our approach is compliant with the Mescal solution, which seems to be general enough to cope with the signaling phase needed in order to enable information exchanging among different domains. Notwithstanding, Mescal does not focus on the specific algorithm to be employed in order to solve the QoS partition problem.

With respect to this last point, we remark that the SLS splitting problem we consider highly resembles the QoS partition problem studied by Lorenz and Orda in [4]. Such problem is to partition the QoS requirements of an application along a selected path in such a way to minimize the cost. The similarity holds if each node of the path is viewed as an autonomous system. Lorenz and Orda proposed in [4] a solution to the QoS partition problem assuming the link cost be a continuous convex function of the delay. This last point clearly marks the difference between our approach and that of Lorenz and Orda. In fact, with our solution no particular assumption holds concerning the nature of the cost function considered.

Starting from the consideration that QoS architectures like DiffServ provide for finitely many service classes, Raz and Shavitt [3] studied the case of discrete cost functions. They proposed a polynomial dynamic programming algorithm for the general case and then investigated some particular instances. The QoS partition problem with discrete cost functions is equivalent to the SLS splitting problem. Our algorithm is an improvement over the dynamic programming one proposed in [3] mainly because the former supports multiple QoS measures while the latter is restricted to just one measure. Interestingly enough, as shown in Section IV-E, the asymptotic complexity of our algorithm linearly depends on the number of domains and grows

[2] RM_{n+1} recursively performs the same operations, by interacting with RM_{n+2} (not shown in the picture).

with the square of the QoS constraints, thus showing exactly the same asymptotic behavior of the algorithm proposed by Raz and Shavitt.

VII. CONCLUSIONS

In this paper we presented an architecture for effective management of SLS-based networks. When designing our framework we took into account the following guidelines: Clear identification of roles and definition of responsibilities. Such guidelines suggested us to embrace a component based approach which brought us to the introduction of a three layer-architecture. Each layer perceives the concept of a *service* at a different level of abstraction and interacts with the neighboring layers by means of lean interfaces. We also showed how the uppermost layers can effectively interact in order to achieve end-to-end service assurance across multiple domains in an orchestrated fashion. With respect to this last point, we illustrated in some more depth an algorithmic approach to the SLS splitting issue. The algorithm we propose brings in a novel contribution since it computes an optimal solution to the partition problem also in the case where multiple QoS constraints have to be jointly satisfied. Moreover, this objective is achieved at an affordable computational complexity. Last but not least, the algorithm naturally lends itself to actual deployment in the current Internet scenario. On one hand, it is naturally prone to a distributed implementation. On the other hand, it fully complies with the current trend which sees service providers' reluctance to disclose information about both their inner structure and their business and management policies. With our approach, the only information that is propagated towards the other entities is the cost associated with the delivery of a service with specified guarantees.

ACKNOWLEDGMENTS

Research outlined in this paper has been partially supported by the European Union under the E-Next project FP6-506869 and by the Italian Ministry for Education, University and Research (MIUR) in the framework of the WEB-MINDS (wide-scale, broadband, middleware for network distributed services) project (FIRB program) and the QUASAR project (PRIN program).

REFERENCES

[1] P. Cremonese, G. Cortese, A. Diaconescu, S. D'Antonio, M. Esposito, R. Fiutem, and S. P. Romano, "Cadenus: Creation and deployment of end-user services in premium IP networks," *IEEE Commun. Mag.*, Jan. 2003.

[2] D. Goderis, M. Buchli, Y. T'joens, C. Jacquenet, G. Memenios, G. Pavlou, R. Egan, D. Griffin, P. Georgatsos, L. Georgiadis, and P. V. Heuven, "Service level specification semantics and parameters," draft-tequila-sls-02.txt, Internet Draft, Jan. 2002, expires Aug. 2002.

[3] D. Raz and Y. Shavitt, "Optimal partition of QoS requirements with discrete cost functions," *IEEE J. Select. Areas Commun.*, vol. 18, no. 12, pp. 2593–2602, Dec. 2000.

[4] D. H. Lorenz and A. Orda, "Optimal partition of QoS requirements on unicast paths and multicast trees," in *Proc. INFOCOM'99*, Mar. 1999, pp. 246–253.

[5] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, "The COPS (common ppen policy service) protocol," RFC2748, Jan. 2000.

[6] S. D'Antonio, M. Esposito, S. P. Romano, and G. Ventre, "Time aware admission control on top of time unaware network infrastructures," *Computer Commun.*, vol. 28, no. 4, pp. 405–416, 2005.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., McGraw-Hill, 2001.

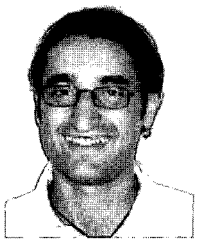
[8] E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Trimintzios, G. Pavlou, and D. Griffin, "Admission control for providing QoS in IP DiffServ networks: The TEQUILA approach," *IEEE Commun. Mag.*, vol. 41, no. 1, pp. 38–44, Jan. 2003

[9] P. Levis, H. Asgari, and P. Trimintzios, "Consideration on inter-domain QoS and traffic engineering issues through a utopian approach," in *Proc. SAPIR 2004 workshop of ICT 2004*, Aug. 2004.

Stefano Avallone received the M.S. degree in Telecommunications Engineering (2001) and the Ph.D. degree in Computer Networks (2005) from the University of Napoli "Federico II." His research interests include computer and communication networks, traffic engineering, QoS routing, and wireless mesh networks. He was a visiting researcher at the Delft University of Technology (2003–2004) and at the Georgia Institute of Technology (2005). In 2004, he was awarded a research funding from the European Doctoral School of Advanced Topics in Networking (SATIN), the instrument employed by E-NEXT (an EU FP6 Network of Excellence) to invest in education of researchers for the European Research Area.

Salvatore D'Antonio received the M.S. degree in computer engineering from the University of Napoli "Federico II." He is currently a researcher at C.I.N.I., the Italian University Consortium for Computer Science. His current research interests include network monitoring and control, quality of service (QoS) provisioning over IP networks, and e-business platforms. He is currently involved in a number of international research projects in the area of inter-domain QoS monitoring and delivery.

Marcello Esposito received the degree in Telecommunications Engineering from the University of Napoli "Federico II" in 2000. He is currently working as a researcher at C.I.N.I., the Italian University Consortium for computer science and he teaches computer programming at the University of Napoli. His research interests include QoS provisioning, interdomain network management, and content adaptation for web services.



Simon Pietro Romano received the degree in Computer Engineering from the University of Napoli "Federico II," Italy, in 1998. He obtained a Ph.D. degree in Computer Networks in 2001. He is currently an assistant professor at the Computer Science Department of the University of Napoli. His research interests primarily fall in the field of networking, with special regard to quality of service (QoS) provisioning in heterogeneous environments, network security, dynamic network management and web-based applications (electronic market-places distance learning, etc.). He is currently involved in a number of research projects, whose main scope is the design and implementation of effective solutions for the provisioning of services with quality assurance over Premium IP networks. Simon Pietro Romano is member of both the IEEE Computer Society and the ACM.

Giorgio Ventre is professor of Computer Networks in the Department of Computer Engineering and Systems of the University of Napoli "Federico II" where he is leader of the COMICS team. COMICS stands for Computers for Interaction and Communications and is a research initiative in the areas of networking and multimedia communications. After started ITEM, the first research laboratory of the Italian University Consortium for Informatics (CINI), Giorgio Ventre is now president and CEO of CRIAI, a research company active in the areas of Information Technologies. As leader of the networking research group at University of Napoli Federico II Giorgio Ventre was principal investigator for a number of national and international research projects. Currently, he is involved in the E-NEXT Network of Excellence of the VI Framework Programme of the European Union where he is leading the activities of the WG on Traffic Engineering and Monitoring. Giorgio Ventre has co-authored more than 150 publications and he is member of the IEEE Computer Society and of the ACM.