RZ 3300 (# 93346) 12/11/00
Electrical Engineering 19 pages

Research Report

A Service Deployment Framework for Programmable Networks

Robert Haas, Patrick Droz

IBM Research
Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon
Switzerland

Burkhard Stiller

Computer Engineering and Networks Laboratory
ETH Zürich
CH-8092 Zürich
Switzerland

A Service Deployment Framework for Programmable Networks

Robert Haas, Patrick Droz

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Burkhard Stiller

Computer Engineering and Networks Laboratory, ETH Zürich, CH-8092 Zürich, Switzerland

Abstract

This paper presents the elements of a framework enabling scalable service deployment over programmable heterogeneous networks. For ease of service deployment it is expected that network management tools will require participation of the network itself in order to be able to scale to a very large number of network elements with widely varying programmability levels. The type of services considered is very broad, ranging from routing services involving selected nodes of a network to end-to-end services spanning the entire network. The underlying hierarchical structure and the representation of capabilities used by the mechanism to deploy new services into a network automatically, are presented. The algorithmic structure of the mechanism described is then illustrated by several examples showing its applicability and complexity.

Keywords: Automated service deployment, programmable networks, network management.

1 Introduction

Services are the key differentiators for internet service providers. To enable interoperability and fast creation of new services, standardized application programming interfaces (APIs) with their corresponding programmable network elements are beginning to appear. In an environment of networks having a large number of nodes that need to be enabled with new services and that have widely varying capabilities and resources, it is necessary to define and provide a way to organize the deployment of new services. Here, we present a framework containing a new mechanism that captures the capabilities of a network to support a new service and organizes the deployment based on specific service policies, thereby addressing the programmability and heterogeneity aspects of the network.

The motivation of our approach is similar to that of quality-of-service (QoS)-aware routing protocols. Such protocols replace lengthy and error-prone manual steps of provisioning resources within a network to guarantee a certain level of QoS. By handling the information related to the available QoS in the network internally, routing protocols can perform this task more efficiently than the operator or the management platform alone.

This paper is structured as follows: Section 2 reviews related activities. Section 3 first classifies the types of services supported by the framework presented here, then its key elements such as the capabilities representation, the hierarchical architecture, and the software modules architecture, and finally the five steps of the deployment mechanism, illustrated with an example. Section 4 shows a variety of examples where the framework can be applied. Section 5 discusses issues related to the use of the mechanism, and Section 6 contains a brief summary of this contribution and the outlook for this work.

2 Related Work

There are only few known activities that focus on the automated deployment of services over large heterogeneous programmable networks. Hierarchical architecture has been used in routing protocols and network management, but not yet considered in the context of deploying services. Let us briefly review the main activities of related work.

In Ref. [1], the need for an automated design, creation, and deployment of network architecture is presented, and a high-level methodology to spawn virtual network architecture, based on the Genesis profiling system that relies on distributed object technology and centralized profile databases, is proposed. In Ref. [2], a framework to isolate services deployed in different virtual active networks (VANs) is presented, whereas the creation of a VAN remains essentially a manual task. A method to automatically provision a virtual private network (VPN) based on a service level agreement (SLA) specification is proposed in Ref. [3]. Active networks consider a packet-centric service-deployment paradigm that is not necessarily suitable over a heterogeneous network where predictable and coordinated deployment of services is required. Heterogeneity is considered in Ref. [4], where an abstraction for links traversing non-active hops is used.

Hierarchical structures are used in IP and ATM networks to aggregate and propagate routing information. IP networks only aggregate routing information with two to three levels of hierarchy,whereas ATM-PNNI [5] also summarizes bandwidth and delay characteristics to allow QoS routing. In Ref. [6], a complex node representation that captures the relevant node characteristics at the lower levels of the hierarchy is proposed. Distributed network management [7-12] also uses hierarchical structures to better scale with large numbers of nodes and complex management tasks such as distributed monitoring with mid-level managers.

To accelerate the deployment of network protocols in the future, efforts have begun focusing on the standardization of interfaces in networking equipment, either in the form of control protocols for label switches (IETF GSMP [13]) or media gateways (IETF MEGACO [14]), or more generic APIs such as those presented in Refs. [15-18]. It is expected that in a heterogeneous network a variety of solutions are likely to coexist.

3 Service-Deployment Framework

Service deployment over large-scale programmable networks requires an automated mechanism. To provide this, a number of new elements need to be defined. In this section, we present the key elements of the service-deployment framework. Services have to specify their network needs, whereas elements in the network have to present their capabilities in a compatible and uniform way. To handle the vast amount of data and processing this entails, a hierarchical architecture is introduced that consists of distributed modules. The successive steps of the algorithm to deploy services are performed by these modules, and result in the installation of services in the network according to specific allocation policies.

Moreover, by remaining service agnostic i.e., not being targeted at a particular type of service, this framework will provide adequate support for future services. Similarly, future equipment providing enhanced or newer functionalities will be supported.

3.1 Supported Services

So-called control-plane services allow the proper transfer of payloads through a network, and include services that provide routing, QoS, and security. We distinguish control-plane services based on explicit or implicit addressing. Implicit-addressing signaling does not require a node to use the addresses of its peers, running the same service, to correctly execute that service. Examples are in-band signaling such as IETF differentiated services (diff-serv) [19], and out-of-band soft-state signaling such as IETF RSVP [20]. For instance, RSVP PATH messages are forwarded just like any other data packet, regardless of whether the next hop is RSVP capable, and this can lead to weak QoS guarantees.

Conversely, explicit-addressing examples include most routing protocols, where routing update messages are explicitly addressed to the peer routers.

The proposed mechanism addresses the deployment of control services in both categories. The subtle difference lies in the way implicit-addressing services needs to be advertised so that data packets requiring such services are routed over a proper path. Clearly, it is expected that not all services are deployed over all possible paths. Depending on the service, they can be required at each hop on a path, at the edges of a path, at selected nodes in the network, etc. How the service needs to be installed to function properly is indicated in the *service-specific allocation policy* (SSAP).

3.2 Service-Deployment Hierarchy

The service-deployment hierarchy is a key element in the proposed framework, and provides reliability and efficiency, similar to hierarchies used in routing or network management tasks. Compared to these existing hierarchies, it extends the summarization (or aggregation) techniques to treat more generic information than only IP- or ATM-addressing and QoS, usually handled by hierarchical routing protocols. Also, whereas network management mostly performs collection and aggregation of

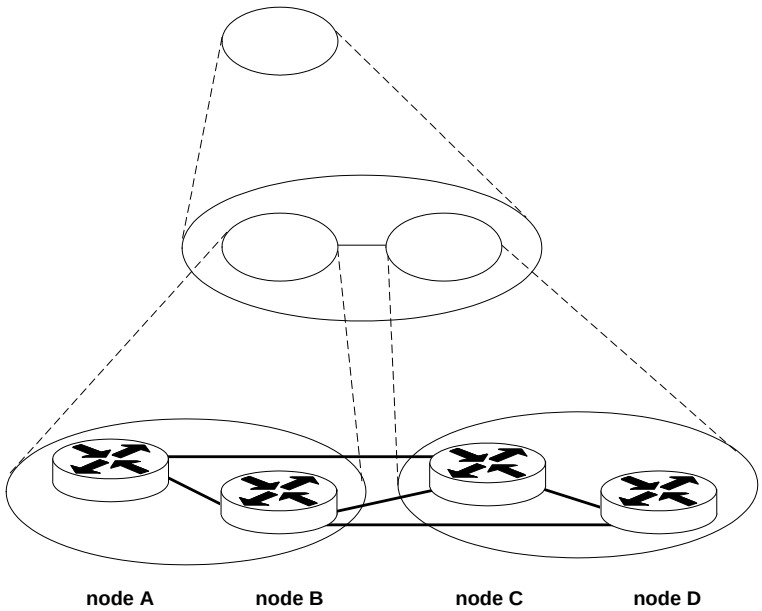

Figure 1: A sample three-layers hierarchy.

data upwards, the service-deployment hierarchy is used both ways: to collect data and to execute the deployment of a service based on the collected data. Conceptually, the mechanism presented here resembles a routing protocol, the difference being that services rather than packets and connections are routed (or deployed) over the network.

Physical network topology is the main factor in creating a hierarchy, at least in fixed networks. Nodes in the network are partitioned into groups of interconnected nodes, each group being assigned a peer group leader (denoted as group leader in the remainder of the paper to avoid confusion with PNNI PGL [5]). A group of nodes appears as a single node at the next level of the hierarchy in the form of a logical group node. The process of partitioning is repeated at each level of the hierarchy. This leads to a spanning-tree-like architecture, where the use of scopes limits the flooding of information to sub-trees. Figure 1 shows a simplified example of a four-node network on top of which a three-layer hierarchy has been built. Nodes A and B belong to the same peer group, where node B is the group leader. A logical node representing this peer group is represented at the next level of the hierarchy. Out of the two resulting logical nodes at that level, one becomes the group leader, and a logical node is created that represents the entire network.

Out of a given physical topology, it is possible to construct many different hierarchies, although administrative constraints such as addressing and wiring can affect how groups are formed. Ultimately, a possible hierarchy has to be evaluated in terms of performance, cost, and stability. The signaling delay for connection setup is an indicator of the performance of a possible routing hierarchy [21]. In the context of service deployment, performance is indicated by the delay until a new service has been deployed. Cost is calculated in terms of overhead processing in all the nodes to set up and maintain such a hierarchy. Finally, stability of the hierarchy under various network conditions is important, especially when it is used to deploy and maintain services in a network. If the hierarchy is overly sensitive to changes in the topology, it might cause unnecessary redeployment of services.

Routing and network management hierarchies remain mostly stable in fixed networks, while ad-hoc networks require special methods to dynamically adapt a hierarchy to varying connectivity [22]. In fixed networks, changes in the hierarchy occur only when peer groups are partitioned because

of links going down, or when a new node integrates or quits a given peer group, such as a mobile network joining or leaving a fixed network. Partitions can be avoided with an appropriate network design, for instance by having multiple links interconnecting groups.

The most suitable service-deployment hierarchy can depend on the type of service being installed, in which case performance and cost will have to be weighted against each other. The evaluation of possible hierarchies for service deployment remains for further research, although it will be shown in Step 5 of Section 3.5 that certain implicit-addressing services should use a service-deployment hierarchy aligned with the routing hierarchy.

3.3 Capabilities Representation

This representation captures information needed in the deployment phase of a new service to decide whether a certain service is compatible with the resources it will require both in terms of how it interfaces with them and in terms of the quantity of resources needed. This representation, therefore, includes

- a description of the base resources [16] and their utilization,
- a description of the higher-level resources, such as OS-resident services, and their utilization if applicable, and
- a reference to the type of APIs to access, configure, and operate the above resources.

The IETF host-resource MIB [23] describes certain features such as processor, memory, and storage for a host computer. It can be extended to include all the necessary elements listed above.

The mechanism presented in Section 3.5 is relatively independent of the specific representation of capabilities. The service to be deployed has to express its requirements in a way that is compatible with the representation of capabilities used in the nodes. The exact mechanism of the negotiation that takes place to determine whether a node is capable of running a certain service is beyond the scope of this paper.

Network processors (NPs) are one of the key building blocks for a programmable network infrastructure. They have widely varying capabilities, such as number of simultaneous forwarding tables supported (required for VPN support), hardware-level programmability (required for fast packet handling), and software-level programmability. To obtain the maximum benefit from these features for a given service, the NPs have to show their features using an appropriate representation of capabilities. Using XML [24] for such a representation rather than a MIB-like structure is interesting because XML is easily extendable, as its structure is self-contained. Figure 2 shows a simple example of NP capabilities using XML.

3.4 Service-Deployment Architecture

Before we describe the mechanism executing the actual service deployment, let us look at the general software architecture of the service-deployment framework. Despite the fact that the architecture relies on a hierarchical structure i.e., that the possible number of levels is, therefore, theoretically unlimited there are only two types of service-deployment components: a base- (or physical) level component, and a logical-level component. Figure 3 shows the corresponding software architecture for the hierarchy described in Fig. 1, in which both types of components can be found. Communication between all instances of these components can be either internal or remote.

```
<Network_Processor>
  <base_capabilities>
    <API_supported> PIN_1520, MIB </API_supported>
    <PIN_1520_specifics>
      <version> 1.3 </version>
    </PIN_1520_specifics>
    <general>
      <processing>
        <speed> 500 MHz </speed>
      </processing>
      <scheduling>
        <total_bandwidth> 100 Mbit/s </total_bandwidth>
        <type> WFQ </type>
        <max_queues> 1000 </max_queues>
      </scheduling>
      <buffers_management>
        <total_buffer_size> 1 MB </total_buffer_size>
        <max_buffer_pools> 16 </max_buffer_pools>
        <buffer_sharing> yes </buffer_sharing>
        <RED> yes </RED>
      </buffers_management>
      <forwarding>
        <type> hardware </type>
        <programmable> yes </programmable>
        <fields> source destination address port </fields>
        <rate> 100% </rate>
        <table_size> 100k </table_size>
        <number_of_tables> 1 </number_of_tables>
      </forwarding>
    </general>
    <ressource_usage>
      // current usage for the defined capabilities
    </ressource_usage>
  </base_capabilities>
  <diff_serv>    // absent if NP does not provide
                 // explicit support for diff-serv
    <API_supported> standard_MIB </API_supported>
      <general>
        <classifier>
          <fields> 6 </fields>
          // etc
        </classifier>
        // etc
      </general>
      <ressource_usage>
        // current usage for the defined capabilities
      </ressource_usage>
  </diff_serv>
  // etc
</Network_Processor>
```

Figure 2: XML representation of the capabilities of a network processor

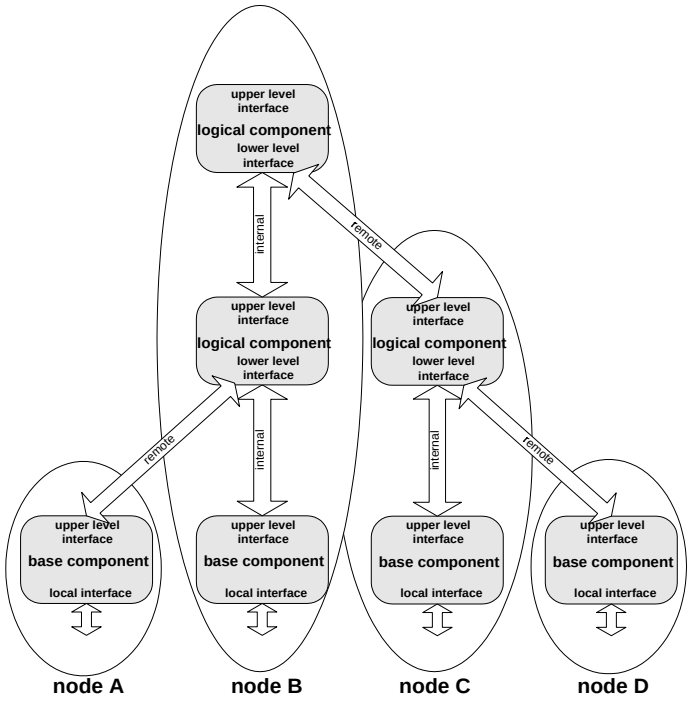

Figure 3: The service deployment architecture.

The base-node as well as the logical-node component responsible for the service-deployment function consist of three modules for polling, deployment, and advertisement. The modules of the base-node and the logical-node component differ in their internal logic and certain interfaces, whereas the upper-level interfaces remain the same.

The responsibilities for each of these three modules are defined as follows:

- the *polling module* polls the network to discover if and how the network nodes are capable of supporting a new service (note that this does not exclude notifications),
- the *deployment module* sets up new services in the network, and
- the *advertisement module* propagates information about installed services throughout the network.

The three modules of a base-level component restrict their actions to the local physical node, while modules of a logical-level component execute on a broader scope, corresponding to the level in the hierarchy where each component is instantiated. The architecture considers both stateless and stateful approaches, where stateful can be either a hard or soft state [25]. Clearly, a stateless approach can incur additional delays for a service to react to changes in the network topology or local capabilities, whereas a stateful approach is more demanding in terms of processing and storage requirements, hence limiting scalability. The inner architecture of both components is described in more detail in the following two subsections.

3.4.1 Base Component

Figure 5 shows the three modules of a base-level component. For each new service to be deployed, an entry is created in the polling module. This entry contains two items: a specification of what

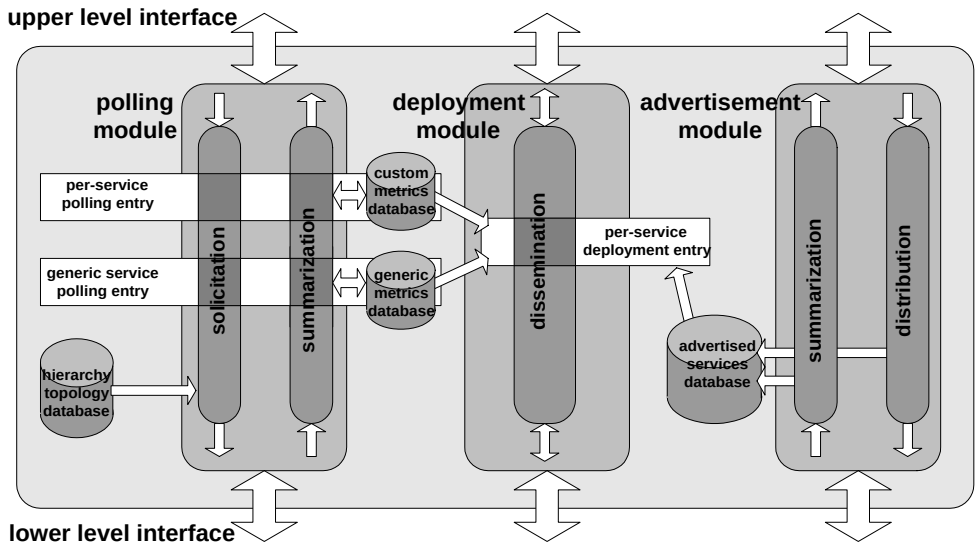

Figure 4: The service-deployment component of a logical node.

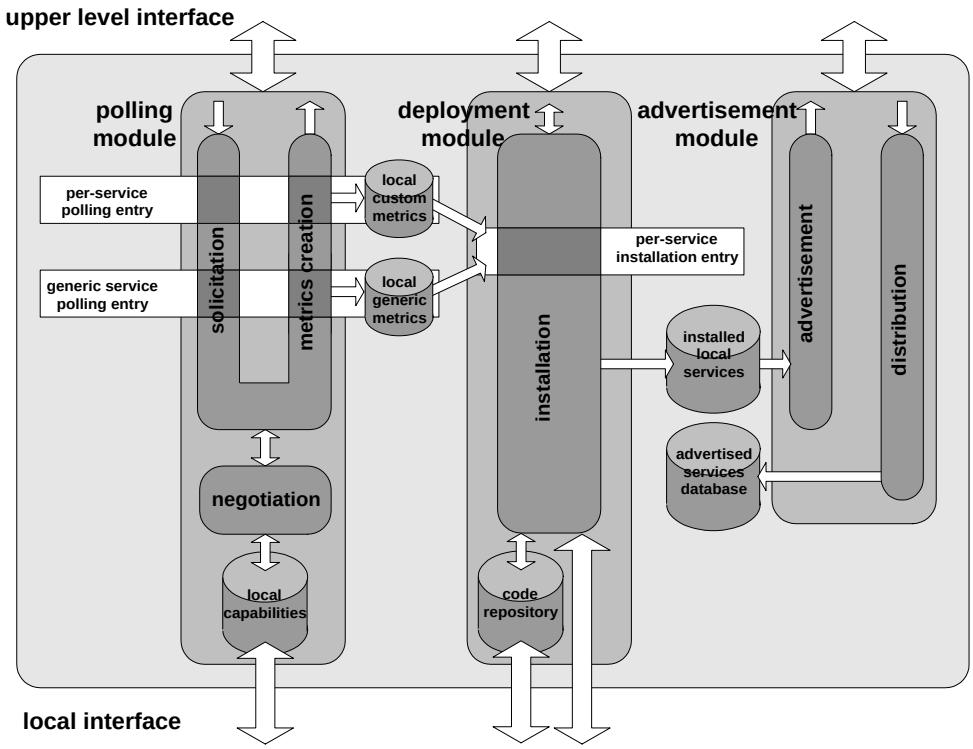

Figure 5: The service-deployment component of a base (physical) node.

the service requires from the node (represented as the intersection between the *per-service polling entry* and the *solicitation* submodule) and a procedure to create appropriate metrics resulting from the comparison between requirements of the service and actual capabilities of the node (intersection with the *metrics creation* submodule). The entry also contains storage for the computed metrics (*local custom metrics database*). A generic entry is also shown in the figure. The resulting metrics found in this entry are available by default, and can be used to deploy certain services that only require such default metrics. The negotiation submodule performs the comparison between local capabilities and the specification of the service's requirements. Note that no advance reservation of resources occurs, as the service might not be deployed at all, in the node under consideration. The upper-level interface of the polling module consists of receiving solicitation messages, and sending resulting metrics. The local interface is used to obtain the representation of capabilities from the local node.

In a stateful approach, the entry in the polling module is created after the first solicitation message has been received, and kept as long as needed. In the soft-state case, further solicitation messages for the same service need to refresh the entry, and can be more condensed because it is no longer necessary to specify the service requirements and procedure to create metrics. In both soft- and hard-state cases, and for each service the network is polled for, a state has to be installed in all nodes to which the solicitation is directed. Therefore, a node can immediately react to changes impacting a particular service, even if that service has not been installed in that node.

A stateless polling module is more adequate for networks with many services, and the entry is in this case only present while the solicitation message is being processed. On the other hand, such messages need to be exchanged regularly to obtain up-to-date information. Note that in both cases resulting metrics are saved in local databases before they are sent through the upper-level interface.

The deployment module performs the installation of a service. An *per-service installation entry* is created for each new service installed. If the installation cannot take place because metrics used by the service (either custom or generic metrics) have changed during the deployment procedure, a crank-back operation has to be initiated [5]. Otherwise, code is obtained from local or remote repositories, and the service is installed and stored in the database of local installed services. The upper-level interface of this module receives deployment commands and sends crank-back messages if necessary. The local interfaces provide access to external code repositories and, most importantly, the APIs necessary to install a service in the local node.

This module handles entries in a stateful manner. If conditions change so that a service is re-deployed or removed, the entry can either be removed automatically after a time-out (soft-state) or required to be removed or modified explicitly (hard-state).

The advertisement module exchanges advertisement messages through its upper-level interface. This module propagates information about installed local services (from the *installed local services database*), and stores received advertisements into the *advertised services database*. It does not need to maintain any state besides the information contained in these two databases.

3.4.2 Logical Component

Figure 4 shows the three modules of a logical-node component. Unlike the polling module of the base-level component, the *per-service polling state entry* here contains an indication of the nodes to poll (intersection with the *solicitation* submodule), and the rules to summarize the metrics obtained from the components in the underlying level of the hierarchy (intersection with *summarization* submodule). To forward solicitation messages down, the solicitation submodule accesses the *hierarchy topology database*, where the structure of the hierarchy is stored. For each per-service polling state

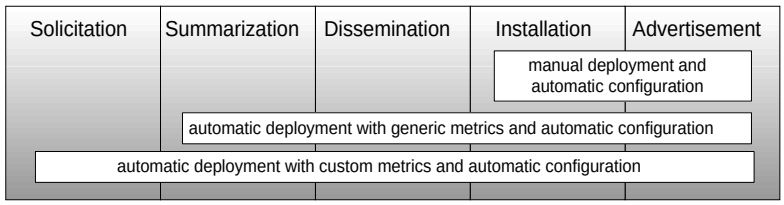

Figure 6: The service deployment mechanism divided into five steps.

entry, the summarization submodule stores metrics into databases. These are the metrics received from the nodes in the next lower level of the hierarchy. The resulting summarized metrics are also stored in the appropriate databases. The upper and lower level interfaces allow the exchange of solicitation messages as well as metrics.

The deployment module mainly executes the dissemination of deployment commands through the hierarchy downwards. For each service being deployed there is a *per-service deployment entry*, which contains the SSAP indicating how the service has to be deployed. This module decides whom to forward the dissemination message to, based on the SSAP and the data, either custom or generic metrics contained in the database of the corresponding polling entry. For this purpose, a shortest-path selection algorithm can be used or any other method specified in the per-service deployment entry. To keep a service operational, the dissemination module also obtains information from the advertised services database. The upper and lower level interfaces allow the exchange of dissemination commands and crank-back messages.

The advertisement module exchanges service advertisements, and performs summarization of advertisements coming from the lower level of the hierarchy. The upper and lower interfaces enable the exchange of such advertisements.

Similarly to their counterparts in the base component, modules of the logical component use either stateless or stateful approaches.

3.5 Service-Deployment Mechanism

Figure 6 shows the five steps comprising the mechanism and the resulting deployment procedures when all or only some of the steps are executed. The mechanism is robust and allows a dynamic redeployment of a service based on changes in network capabilities or topology. This is similar to how flows of packets are rerouted when changes in the routing table occur.

Using only the two last steps in Fig. 6 leads to a *manual deployment and automatic configuration* of a service. This is how services are generally deployed in networks today. With the intermediate solution, the result is an *automatic deployment with generic metrics and automatic configuration*. Only when the entire five steps have been executed does it lead to an *automatic deployment with custom metrics and automatic configuration*.

The first step (*Solicitation*) consists of declaring the requirements of a new service to the network. The nodes composing the network are asked to compare these requirements with their current capabilities, and return an answer in the form of a metric. Note that if the requirements of a particular service are known in advance, which generally is not the case in programmable networks, or have a generic form, then this solicitation step can simply be skipped. The answer returned by the network nodes needs to be summarized (*Summarization*), otherwise the number of responses would make the system unscalable. The polling module performs these two steps similar to a query-reply operation.

Once the responses have been summarized up to the top level of the hierarchy, the deployment is

initiated by choosing the nodes appropriate to support the service at each level, based on the SSAP (*Dissemination*), and the service is installed in the physical nodes selected (*Installation*). These two steps are executed by the deployment module.

To allow configuration, nodes then advertise their installed services, and dynamically learn from other nodes advertising the same service (*Advertisement*). This allows, for instance, explicit-addressing control services such as routing protocols to autoconfigure the neighbor routers. The advertisement module executes this step.

Note that the complete service life-cycle requires a removal step. However, here only the constructive service deployment is illustrated using a hypothetical deployment of a diff-serv++ service in a transit network.

Step 1: Solicitation

This step is essentially a directed broadcast of the service requirements to the nodes of the network. In certain cases, some portions of the network can be ignored by the solicitation step: if the edges or end points between which the service needs to be deployed are defined, the solicitation step can direct the broadcast to all nodes between these edges or end points.

The goal of this step is to obtain a piece of information specific for the particular service that can easily be summarized. This information is a metric or a set of metrics. As mentioned, certain services require a set of custom metrics and/or custom summarization rules, and those metrics are derived from the comparison of the services requirements with the actual capabilities within each network node. Other services require only need a set of generic metrics and summarization rules i.e., the network knows the metrics and their summarization.

It is clear that services sharing a common set of custom metrics but having different summarization rules will not be able to share any of the results obtained from the solicitation step.

When generic metrics are advertised by default in the network, then the solicitation and the resulting summarization step can be saved, using the corresponding generic summarization rules. This is interesting for services that need to be set up rapidly.

Values of the set of custom metrics result from the negotiation taking place when a node needs to compare its capabilities against the requirements of a certain service. The values can either simply indicate whether the service is supported (boolean), at what cost it can be supported (quantitative), or how well it is supported (qualitative).

The result of the solicitation step for our diff-serv++ example is shown in Fig. 7. Routers capable of supporting the service are shown in black (boolean metric).

Step 2: Summarization

Such sets of metrics need to be summarized prior to transmission to the top level of the hierarchy. Basically, this is how the mechanism is kept scalable. By making the summarization rules customizable, the summarized view can be made accurate enough for the deployment mechanism to work correctly. This step is repeated at regular time intervals (or when specific events take place), so as to cope with changes.

Figure 7 shows the transition matrices [6] obtained successively at each level of the hierarchy i.e., $M_{1.1}$, $M_{1.2}$, ..., and M_1. Edges are numbered so that element $m_{i,j}$ is defined as the number of hops on the shortest path between edges i and j that provides the required diff-serv++ service at each hop along the path. As the matrix is symmetric, only one half of it is shown.

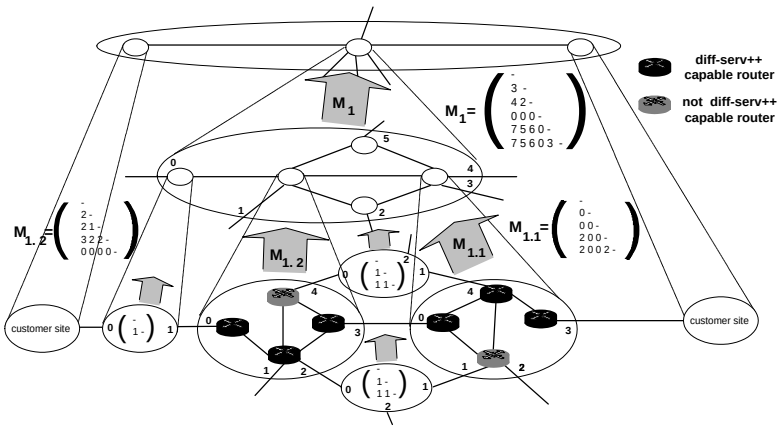

Figure 7: Summarization step.

Step 3: Dissemination

The SSAP is used in this step to direct the deployment of the particular service. It contains topology-oriented actions, such as which nodes need to be enabled with the service. More specifically, a dissemination command to deploy the service is launched from the top-most node, and travels down the hierarchy, together with the associated SSAP. On each level, the group leader forwards the command only to those nodes in which the service needs to be deployed rather than executing a complete flooding. The group leader does not propagate the command to the nodes underneath itself if it is not selected to deploy the service.

As nodes that deployed a service keep state, the service can be redeployed automatically when significant changes occur. The SSAP defines what is considered a significant change. To avoid redeployment of a service whenever a routing change occurs, certain heuristics can be used during the dissemination of the command so that the service is deployed not only on a certain path but on a set of paths instead. These heuristics are contained in the SSAP.

Inspecting the top matrix M_1 in Fig. 7 in our diff-serv++ example shows that $m_{4,0}$ is different from zero, meaning that the transit network is capable of supporting the diff-serv++ service between the two customer sites. Therefore, the dissemination takes place, as shown in Fig. 8, where the shortest path is selected at each level of the hierarchy based on the transition matrices provided by the lower level. The nodes on the path selected then forward the command downwards, and the process repeats itself. The edges of the shortest path are shown as thicker lines at each level.

Step 4: Installation

The command ultimately reaches the bottom level of the hierarchy, where the physical nodes are. Here, the service is installed independently on each node. The installation and a part of the configuration that take place at this moment are automated: for instance, a link to the code located in a repository is provided in the SSAP together with configuration parameters. Conversely, if the code required for the service is reasonably small, it can be contained in the dissemination messages. In certain cases, the necessary code may already be present in the nodes, albeit neither loaded nor configured.

Work related to automating the installation of code, more specifically the use of script MIBs to automate installation and configuration of a newer version of a protocol, has for instance been described in Ref. [12].

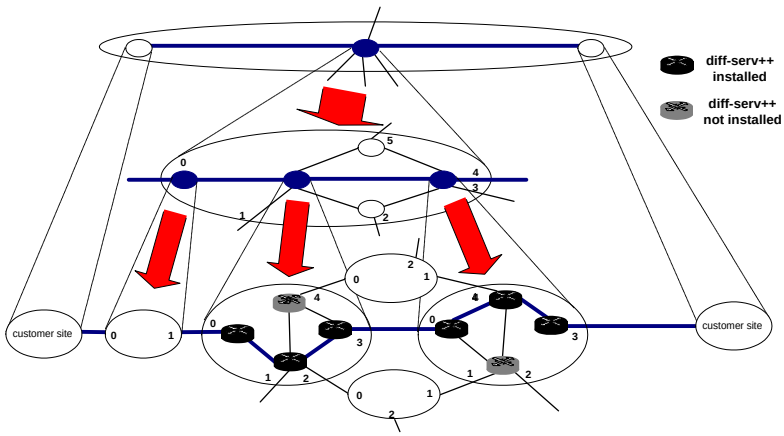

Figure 8: Dissemination step.

In general, the specific code destined to each individual node to run a certain service in the network can vary depending on the capabilities proper to each node. As the solicitation step has already verified that it is possible to run the service, this step now installs the proper code.

In our example, the diff-serv++ code is loaded into each router selected.

Step 5: Advertisement

Once installed, a service will either have to discover neighbors with which it exchanges control messages (explicit addressing) or be advertised so that data packets are routed towards their destination over nodes enabled with the service (implicit addressing). This configuration problem is not specific to our mechanism. So far it has mainly been addressed by manual operations. For instance, BGP routers [26] need to be manually configured with the IP addresses of their peers.

In our example of an implicit-addressing service, routers advertise their installed diff-serv++ service, and this information is again summarized and distributed. Here, the transition matrices advertised in this step are identical to those created in the summarization step, as there is only one path between the two customer sites that can support the diff-serv++ service. Because routing needs to be aware of such services (we assume the nontrivial case where the service is not installed in all nodes), this summarized information has to be combined with routing to provide the appropriate routes. Note that having the service-deployment hierarchy aligned with the routing hierarchy will significantly ease the advertisement procedure of implicit-addressing services.

In Ref. [27], we compared the various automatic discovery techniques for explicit-addressing control services, and proposed to use PAR (PNNI Augmented Routing [28]) to advertise such services within a ATM-PNNI environment, as PAR is more robust and scalable than centralized directory services.

3.5.1 Alternative for Controlling the Deployment

Until now we assumed that the software components executing the service-deployment procedures are located in the same nodes where services are to be loaded (see Fig. 3). However, by using a proxy function, it is possible to integrate nodes into the framework that are not capable of running the base-level component themselves (such as attached servers, illustrated in the example in Section 4.2). Also, logical-level components are not necessarily restricted to the nodes in which services are to be installed. This is similar to the case of distributed routers where route computation is centralized

within a processing module and forwarding takes place in line cards, the entire distributed router being regarded as a peer group itself. It is also similar to mid-level network managers running in distinct boxes (management stations) from the agents themselves.

3.5.2 Extended Transition Matrices for Non-Transit Service Deployment

Metrics can take the form of a transition matrix i.e., the metric is representative of the underlying group of nodes only when traversing them from edge to edge. If a particular service is deployed to a destination contained in that group, the transition matrix does not show whether this inside destination supports the service.

This problem cannot be solved in a QoS-routing environment because summarization hides the internal bandwidth and delay characteristics of a path from an edge node to an inside node.

In the service-deployment case, metrics are created on-demand, so the problem can be solved as follows: by artificially augmenting the topology with a virtual outside link connected to the inside node, this inside node (or more precisely this newly attached virtual outside link) will appear in all transition matrices built by the higher layers. The information on this node will, therefore, not be hidden by the process of summarization. The SSAP can specify between which nodes the service has to be deployed, hence the virtual outside links are added at the appropriate nodes. The example in Section 4.3 illustrates the use of virtual outside links.

Therefore, the mechanism proposed is not limited to installing services over transit networks (i.e., the nodes between which the service is deployed lie outside the transit network), but also in the edge networks that contain these nodes.

4 Examples

4.1 Hierarchical IP Routing

In this example, we show how the service-deployment hierarchy can benefit from an existing routing hierarchy such as PNNI. We illustrate this via the automated deployment of an IP routing service having three hierarchy levels. This service is an explicit-addressing out-of-band control service. It is deployed over an existing ATM-PNNI network composed of ATM switches, where IP routers are connected. The IP routing hierarchy will match the service hierarchy chosen in this case (which maps exactly to the PNNI hierarchy), therefore, automatically installing BGP and OSPF [29] services at the appropriate layers. Note that from the viewpoint of IP the ATM network is considered as non-broadcast multiple access (NMBA) [29], therefore, each router could possibly peer with all other routers. Such a full-mesh is clearly not desirable because of scalability concerns.

Figure 9 shows the service hierarchy, matching the PNNI routing hierarchy. It consists of a base, a middle and a top level. We skip the solicitation and summarization steps, and concentrate on the dissemination step, in which a part of the auto-configuration of the service is performed. For instance, based on the topology connecting $A.1$, $A.2$, $A.3$, and $A.4$, the SSAP can choose to use $A.1$ as the OSPF backbone area. Then, in each OSPF area, routers interconnect with other routers in the same area, based on topology optimization methods such as described in Ref. [30]. The advertisement from all OSPF routers is scoped to their respective groups (their scope is limited to the base level), with the exception of the border-area routers of the backbone area, which need to be advertised into all other areas (their scope is limited to the middle level) [31, 32].

The routers chosen to run the BGP service will be advertised up to the top level, so that they can be seen by their peers in the B and C groups, as well as internally in A.

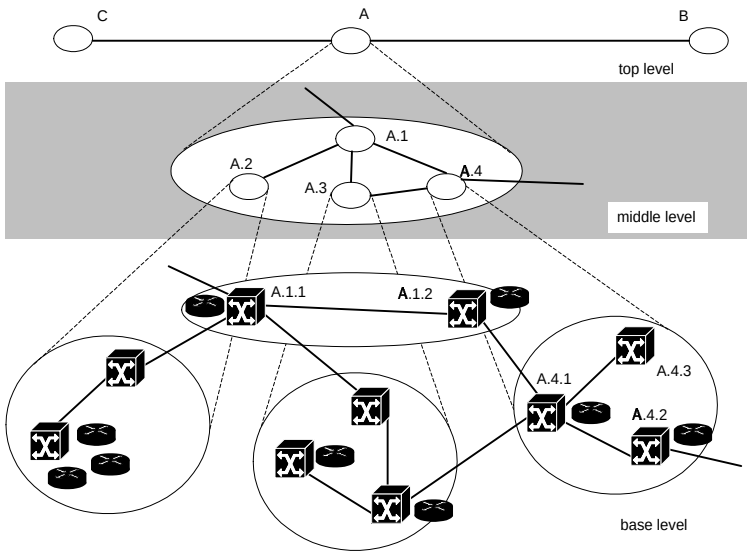

Figure 9: The service hierarchy.

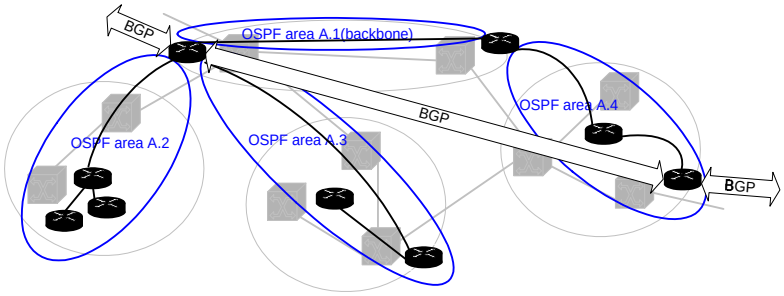

Figure 10: A possible resulting IP routing hierarchy.

Figure 10 shows a possible resulting IP routing hierarchy, with the underlying ATM network in gray. The autonomous system is composed of four OSPF areas and two BGP speakers.

4.2 Transparent Hierarchical Proxy Cache

An example of an explicit-addressing in-band service deployed at selected nodes only is a hierarchical transparent web proxy cache. Being transparent, HTTP clients need not be configured with the address of their proxy cache. Instead, the routers perform layer-4 switching and redirect HTTP requests to the proxy caches. To improve response time, two layers of caches can be used: the first-level cache contains the pages requested most often by the local user group, whereas the second-level cache contains the pages most often requested by all user groups.

We assume that an adequate routing and addressing functionality is already in place in the network before deployment of the service takes place. During the solicitation step, the specific requirements to support the service are sent to the nodes. Here, they include minimum processing capacity and storage on the directly attached servers, and capability of the routers to perform layer-4 switching. In Fig. 11, the node representing $A.1$, the backbone area, will receive a solicitation with the requirements corresponding to a second-level cache, whereas the nodes representing $A.2$, $A.3$, and $A.4$ will receive the requirements for a first-level cache. The resulting metrics, which can take the form of a boolean, are then sent up the hierarchy and summarized for each group. The summarization

procedure is a logical-OR of the results from each individual router. This is done for each node, $A.1$ through $A.4$. These metrics can then be summarized again, using a logical-AND of the previous results, so that node A will be informed whether the desired service can be deployed over the current infrastructure. Note that this example shows custom summarization rules that vary at each level of the hierarchy. In Fig. 11, the routers and servers selected by the deployment procedure are shown in black.

The system is transparent to the users, hence, they do not need to receive advertisements from this service. However, the first-level caches have to forward cache misses to the second-level cache. Therefore, the second-level cache originates advertisements carrying its address. These advertisements are scoped within the autonomous system, and directed to the first-level caches only, rather than being flooded throughout the network.

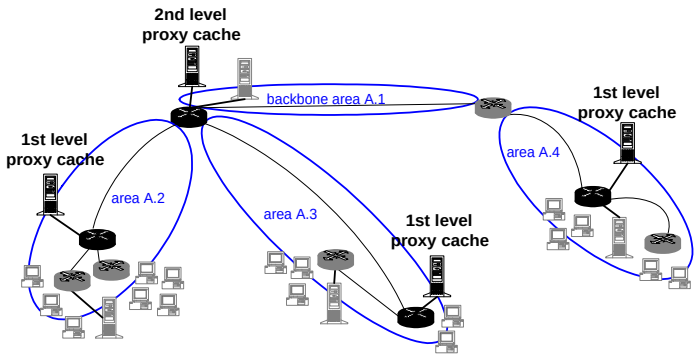

Figure 11: Deployment of a transparent two-level proxy cache.

4.3 Virtual Private Network (VPN)

Here, we illustrate the use of virtual outside links in an explicit-addressing service example. The diff-serv++ example described earlier applied to transit networks, and therefore no such virtual outside links were required. The VPN interconnects subnetworks within the same network, and requires certain encryption capabilities at the end points as well as a certain QoS. The desired QoS is signaled by a protocol such as RSVP, and the nodes along the VPN therefore have to support it. We assume that the end points of the VPN are set statically, as shown in Fig. 12 by the letters A, B, and C.

During the solicitation step, the SSAP includes addresses of the desired end points for the VPN so that the appropriate virtual outside links (in thick dashed lines) are created for B and C during the summarization step. Otherwise, the top-level view of the hierarchy would be totally unaware whether it is possible to interconnect A, B, and C with RSVP-capable routers. The transition matrices shown are similar to those in the diff-serv++ example. In matrix M_1, which is built out of $M_{1.1}$ and $M_{1.2}$, $m_{2,0} = 3$, which is the cost of the VPN between A and B. Similarly, the cost between B and C is $m_{3,2} = 4$, and between A and C $m_{3,0} = 6$. The dissemination phase can then begin, by choosing to deploy the VPN between A and B, and B and C, so as to minimize the overall cost.

4.4 Active Networking

When offering support for active networks, a certain execution environment (EE) [15] has to be put in place in the nodes of the network. This task can be handled by the present algorithm. Thanks to

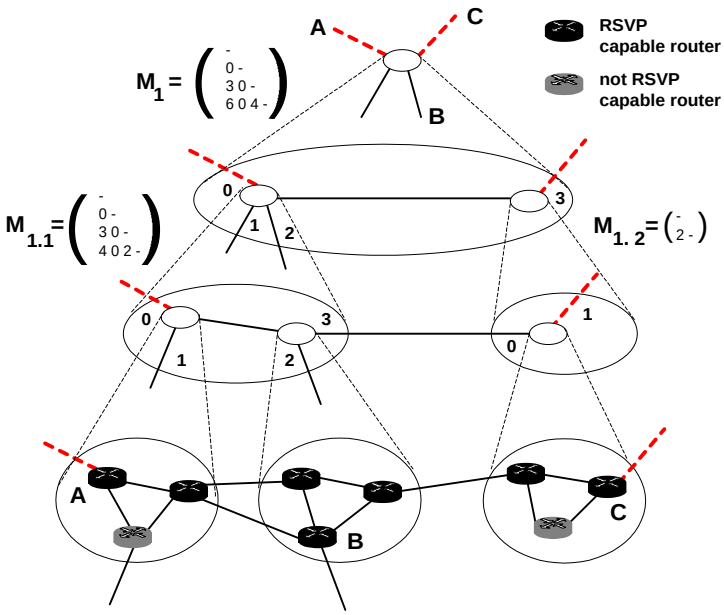

Figure 12: VPN deployment using virtual outside links.

the advertising step, it is possible to route active packets based not only on their final destination, but also on the type of EE they require at each node on the path leading to that destination.

5 Applicability

Various examples have suggested that a service hierarchy is required to keep the deployment scalable when targeting large numbers of nodes. Although the number of hierarchy levels is not limited by the mechanism, the resources in the network to maintain this hierarchy are bounded. ATM and IP networks commonly use three levels of hierarchy to aggregate routes. A fourth level of hierarchy already exists, although it is hidden from the routing protocols: distributed routers (clusters) appear as a single node with a single IP address from the outside. Extending the service hierarchy into such nodes can help automate the placement of functions within clusters.

We did not take link metrics explicitly into account when performing the service-deployment mechanism (only a hop count was used for building the transition matrices in the diff-serv++ example). The complexity of shortest-path routing with multiple metrics can be high or even become NP-complete, especially if multiple metrics, additive as well as restrictive, are used. Although the mechanism presented here allows a dynamic definition of the metrics to be used, this has to be balanced against the complexity that can arise if these metrics are not chosen appropriately.

By using generic metrics and generic summarization rules, we have shown that the solicitation step can be avoided if no virtual outside links are needed. If the network collects and summarizes these metrics by default, deployment of a new service can start immediately with the dissemination step. If custom metrics and/or summarization rules are required, the solicitation step can be performed using a directed broadcast rather than a complete broadcast, provided the region(s) in the network where the service will have to be deployed can be identified.

6 Summary and Outlook

Organizing the automated deployment of services is a key step towards an intelligent network infrastructure. Openness and programmability are beginning to appear in network equipment and, therefore, it is necessary to provide generic mechanisms that can enable the deployment of any type of service. In this paper, we have introduced the key elements of the framework and proposed a novel mechanism that addresses the deployment of a wide range of services within programmable and heterogeneous networks. The mechanism polls all capabilities available in the network and compares them with the requirements for a specific service, resulting in the creation of appropriate metrics. Therefore, the deployment of the service can be based on a summarized view of these metrics without loss of relevant information for any specific service. The deployment task is then distributed throughout the hierarchy, following a service-specific allocation policy. Thus an automated deployment and configuration is achieved, avoiding time-consuming and error-prone manual operations. Moreover, the mechanism is robust and capable of adapting to dynamic changes in the network, ensuring that a service remains available once deployed, most importantly, when networks grow in size and heterogeneity, it can still capture the essential data for deploying any particular service, while retaining its scalability thanks to the use of summarization and dissemination across the service-deployment hierarchy.

In addition, it allows experimentation involving a wide variety of new services outside the typical simplified testbed networks. On-going investigations include the relationship between the requirements for a generic service-deployment hierarchy and the various kinds of existing hierarchies, such as routing, management, or domain name service (DNS). The needs for a per-service specific service deployment hierarchy are being evaluated against the cost of putting such hierarchies in place on-demand.

References

[1] Campbell, A.T., Kounavis, M.E., Villela, D.A., Vicente, J., Miki, K., De Meer, H.G., and Kalaichelvan, K.S., "*Spawning Networks*", IEEE Network Magazine July/August 1999.

[2] Brunner, M., and Stadler, R., "*Virtual Active Networks - Safe and Flexible Environments for Customer-Managed Services*", Proc. 10th IFIP/IEEE Int'l Workshop on Distributed Systems (DSOM'99), Zurich, Switzerland, October 1999.

[3] Isaacs, R., "*Lightweight, Dynamic and Programmable Virtual Networks*", OPENARCH 2000, Tel Aviv, Israel, March 2000.

[4] Sivakumar, R., Han, S., and Bharghavan, V., "*A Scalable Architecture for Active Networks*", OPENARCH 2000, Tel Aviv, Israel, March 2000.

[5] ATM Forum, "*P-NNI V1.0*", af-pnni-0055.000, March 1996.

[6] Iliadis, I., and Scotton, P., "*Transition matrix generation for complex node representation*", Proc. IEEE ATM Workshop'99, pp. 489-500, Kochi, Japan, May 1999.

[7] Yemini, Y., Goldszmidt, G., and Yemini, S., "*Network Management by Delegation*", Proc. Int'l Symp. on Integrated Network Management, pp. 95-107, April 1991.

[8] George, Jude A., and Schlecht, Leslie E., "*The NAS Hierarchical Network Management System*", in Integrated Network Management III, H.-G. Hegering and Y. Yemini Ed., Elsevier Science Publishers B. V. Amsterdam, 1993.

[9] Stamatelopoulos, F., Roussopoulos, N., and Maglaris, B., "*Using a DBMS for Hierarchical Network Management*", Int'l Conf. on Networld & Interop, Las Vegas, NV, March 1995.

[10] Xin, T., and Xiao, D., "*An Approach for Hierarchical Network Management Architecture Based on SNMPv2*", Smartnet'99, The Fifth IFIP Conf. on Intelligence in Networks, Thailand, November 1999.

[11] Schonwalder, J., Quittek, J., and Kappler, C., "*Building Distributed Management Applications with the IETF Script MIB*", IEEE Sel. Areas Comm., Special Issue on Network Management and Operations, Volume 18, Number 5, May 2000.

[12] Quittek, J., and Kappler, C., "*Practical Experiences with Script MIB Applications*", The Simple Times 7(2), www.simple-times.org, November 1999.

[13] Doria, A., Hellstrand, F., Sundell, K., and Worster, T., "*General Switch Management Protocol V3*", IETF draft <draft-ietf-gsmp-05.txt>, January 2000.

[14] Greene, N., Ramalho, M., and Rosen, B., "*Media Gateway Control Protocol Architecture and Requirements*", Internet RFC 2805, April 2000.

[15] DARPA Active Networks program, "*NodeOS interface specification*", Peterson, L., Ed., June 1999.

[16] Biswas, J. et al. "*Application Programming Interfaces for Networks*", IEEE PIN1520 Working Group Draft White Paper, www.ieee-pin.org.

[17] CPIX Forum, "*Common Programming Interfaces*", www.cpixforum.org.

[18] de Keijzer, J., Tait, D., and Goedman, R., "*JAIN: A New Approach to Services in Communication Networks*", IEEE Communications Magazine, January 2000.

[19] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., "*An Architecture for Differentiated Services*", Internet RFC 2475, December 1998.

[20] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S., "*Resource Reservation Protocol (RSVP)– Version 1 Functional Specification*", Internet RFC 2205, September 1997.

[21] Awerbuch, B., Du, Y., and Shavitt, Y., "*The Effect of Network Hierarchy Structure on Performance of ATM PNNI Hierarchical Routing*", Computer Comm. 23(10), May 2000.

[22] Chen, W., Jain, N., and Singh, S., "*ANMP: Ad Hoc Network Management Protocol*", IEEE Sel. Areas in Comm., 17(8), August 1999.

[23] Grillo, P., and Waldbusser, S.,"*Host Resources MIB*", Internet RFC 1514, September 1993.

[24] World Wide Web Consortium (W3C), "*Extensible Markup Language (XML) 1.0*", W3C Recommendation, February 1998.

[25] Raman, S., and McCanne, S., "*A Model, Analysis, and Protocol Framework for Soft State-based Communication*", Proc. ACM SIGCOMM '99, Cambridge, MA, September 1999.

[26] Rekhter, Y., and Li, T., "*A Border Gateway Protocol 4 (BGP-4)*", Internet RFC 1771, March 1995.

[27] Haas, R., Droz, P., and Bauer, D., "*PNNI Augmented Routing (PAR) and Proxy-PAR*", Computer Networks 34(3), September 2000 (in press).

[28] ATM Forum, "*PNNI Augmented Routing (PAR) Version 1.0*", af-ra-0104, January 1999.

[29] Moy, J., "*OSPF Version 2*", Internet RFC 2178, July 1997.

[30] Frelechoux, L., Osborne, M., and Haas, R., "*Topology Optimization of IP over ATM*", Proc. 1st IEEE European Conf. on Universal Multiservices Networks (ECUMN 2000), October 2000, Colmar, France.

[31] Przygienda, T., Droz, P., and Haas, R., "*OSPF over ATM and Proxy-PAR*", Internet RFC 2844, May 2000.

[32] Droz, P., and Przygienda, T., "*Proxy-PAR*", Internet RFC 2843, May 2000.