Distributed Service Deployment over Programmable Networks

Robert Haas[1], Patrick Droz[1] and Burkhard Stiller[2]

[1] *IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland, {rha,dro}@zurich.ibm.com*
[2] *Computer Engineering and Networks Laboratory, ETHZ, 8092 Zürich, Switzerland, stiller@tik.ee.ethz.ch*

This paper presents the elements of a framework enabling QoS-aware service deployment over programmable heterogeneous networks. For ease of service deployment, it is expected that network management tools will require the network itself to participate in this task so as to be able to scale to very large numbers of network elements, with widely varying programmability levels. Three categories of service deployment are considered, namely services spanning along paths in the network, services involving only selected nodes, and combinations of both. The underlying hierarchical structure and the representation of capabilities used by the mechanism to deploy new services into a network automatically are presented. A formal description of the mechanism based on a Gather-Compute-Scatter paradigm is introduced and then illustrated by examples of all three service-deployment categories.

Keywords: Automated service deployment, quality of service, service provisioning, programmable networks.

1 Introduction

Ideally, new services should be deployed in a network so as to make optimal use of its capabilities. Given that both the number of services as well as the possible combinations of distributed network capabilities to support a given service are expected to dramatically increase in the near future, a manual deployment becomes less and less adequate. Moreover, network elements in general offer an increasing spectrum of capabilities, some with dedicated specialized functions, others with programmable behaviors, in soft- or hardware, thereby multiplying the number of deployment alternatives.

In the context of large heterogeneous and programmable networks, it seems therefore contradictory to deploy services in a scalable as well as optimized manner. This contribution shows how this can be solved.

In order to enable interoperability and faster creation of services, standardized Application Programming Interfaces (APIs) for network elements are being defined. While such APIs definitely ease the deployment of services within a particular network element, they do not define how to organize the deployment at a network level, where connectivity and individual capabilities are heterogeneously distributed.

The framework presented here allows the distributed capture of the dynamic capabilities of a network to support a new service and organizes the deployment based on specific service-allocation policies, thereby addressing the programmability and heterogeneity aspects of the network. This contribution is, to our knowledge, the first to recognize and propose solutions to this particular problem.

The motivation of our approach can be viewed as 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 any operator or management platform alone could.

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 briefly reviews some of its key elements such as the hierarchical architecture and the capabilities representation. Section 4 introduces a formalism for hierarchically-distributed computations, and presents the details of the mechanism, illustrated with examples of three possible service-deployment categories. Section 5 contains a brief summary of this contribution as well as an outlook.

2 Related Work

Few activities focus on the automated deployment of services over large heterogeneous programmable networks. Hierarchical architectures have been used in routing protocols, service discovery, network management, and, more recently, sensor networks [Kum01], but have not yet been considered in the context of deploying services. Let us briefly review the main activities of related work.

In [CKV$^+$99], the need for an automated design, creation, and deployment of network architectures is presented, and a high-level methodology to spawn virtual network architectures based on the Genesis profiling system that relies on distributed object technology and centralized profile databases is proposed. In [BS99], a framework to isolate between services deployed in different virtual active networks (VANs) is presented, whereas the creation of a VAN essentially remains a manual task.

Dynamic composition and deployment of services in the context of end-to-end application sessions are addressed in [NWX00, NPB01, CTW01]. This applies for instance to the setup of a network path for a multimedia session based on the availability and cost of image transcoders and compression service components active in intermediate network nodes.

Hierarchical structures are used in IP and ATM networks to aggregate and propagate routing information. While IP networks only aggregate routing information, with two to three levels of hierarchy, ATM-PNNI [AF96] also summarizes bandwidth and delay characteristics to allow QoS routing. In [IS99], a complex node representation that captures the relevant characteristics of the nodes at the lower hierarchy levels is proposed. Likewise, distributed network management [YGY91, LS01, SQK00, QK99] uses hierarchical structures to better scale with large numbers of nodes and complex management tasks such as distributed monitoring with mid-level managers. In [XNW01, CZH$^+$99], the discovery process of distributed services over wide-area networks exploits a hierarchy of service brokers to propagate end-hosts queries and to cache advertisements of installed media gateway services.

In [KS00], the need for a distributed and programmable management platform for the future Internet is presented, and simple navigation patterns for mobile agents are described. The underlying hierarchical structure in our contribution can be viewed as another, more sophisticated navigation pattern used by mobile agents to perform the specific task of scalable service deployment.

To accelerate the deployment of network protocols, efforts have begun focusing on the standardization of interfaces in networking equipment, either in the form of control protocols for label switches (IETF GSMP [DHSW00]), IP routers (IETF FoRCES [F01]), and media gateways (IETF MEGACO [GRR00]), or more generic APIs such as those described in [Pet99, Bis00, CF00, dKTG00]. It is expected that in a heterogeneous network a variety of solutions are likely to coexist.

3 Service-Deployment Framework

Large-scale service deployment over programmable networks requires an automated installation mechanism. To provide this, a number of new elements need to be defined. Here, we present the key elements of the service-deployment framework. Services have to specify their 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. The successive steps of the algorithm to deploy services are executed along this hierarchical architecture, and result in the installation of services in the network according to specific allocation policies.

Moreover, by remaining service agnostic, i.e. by 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.

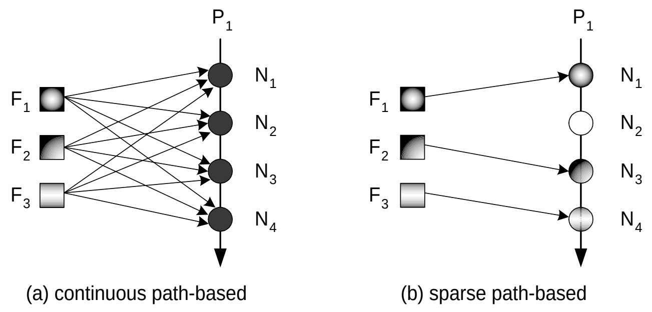

Fig. 1: Types of path-based deployment.

3.1 Categories of Services

Services are assumed to be decomposable into sets of functions to be executed by individual nodes. We distinguish three categories of service deployment:

- *path-based*, between a set of source(s) and destination(s), which is further divided into two types:
  - *continuous*, for which a set of functions must be present in **each** node on the path, and
  - *sparse path-based*, or discontinuous, for which a set of functions must be present in the **set of nodes** on the path (see Fig. 1). Note that the functions in the set might have ordering rules, i.e. F_1, F_2 and F_3 must be executed in this sequence,
- *node-based*, for which only selected nodes need to be activated, and no source/destination pairs are specified, but rather domains, and
- *path- and node-based* deployment, for which extremities of paths need specific capabilities, differing from the in-between nodes.

The service deployment category is reflected in the *service-specific allocation policy*. In addition, we classify control-plane services based on explicit or implicit addressing. Implicit-addressing-based signaling does not require a node to use the addresses of its peers that run the same service to correctly execute that service. Examples are in-band signaling such as IETF differentiated services (diff-serv) [BBC+98], and out-of-band soft-state signaling such as IETF RSVP [BZB+97] (note that RSVP messages are not included in data packets, hence the out-of-band signaling). 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 in which routing update messages are explicitly addressed to the peer routers.

All three deployment categories of control services are supported by the framework proposed here. Moreover, both implict- and explicit-addressing services are supported by means of an appropriate advertisement procedure (implicit-addressing services need to be advertised so that data packets requiring such services are routed over an appropriate path).

For a service to be deployed according to its requirements, i.e. into the appropriate network nodes that have the corresponding capabilities, a common description for node capabilities and service requirements needs to be defined. The representation of node capabilities can, for instance, be expressed in XML, as an extension to the IETF host-resource MIB [GW93]: it includes a description of the type of APIs to access, configure and operate the resources in the node, either base ([Bis00]) or higher-level resources such as OS-resident services, as well as the utilization of these resources. A dual representation for services is used to match against node capabilities. This representation includes the required node capabilities to accommodate the service. The actual evaluation process of node capabilities against service requirements exceeds 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 the number of simultaneous forwarding tables supported (required for VPN support), hardware-level programmability (required for fast packet handling), and software-level programmability. Figure 2 shows a short example of possible NP capabilities. Using XML for such

```
<Network_Processor>
  <base_capabilities>
    <API_supported> CPIX, 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>
        <line_rate> 100% </line_rate>
        <table_size> 100k </table_size>
        <number_of_tables> 1 </number_of_tables>
      </forwarding>
    </general>
    <resource_usage>
      // current usage for the defined capabilities
    </resource_usage>
  </base_capabilities>
  <diff_serv>    // absent if NP does not provide
                 // explicit support for diff-serv
    <API_supported> PIN_1520 </API_supported>
      <general>
        <classifier>
          <fields> 6 </fields>
          // etc
        </classifier>
        // etc
      </general>
      <resource_usage>
        // current usage for the defined capabilities
      </resource_usage>
  </diff_serv>
  // etc
</Network_Processor>
```

Fig. 2: XML representation of the capabilities of a network processor.

a representation rather than a MIB-like structure is interesting because owing to its self-contained structure XML is easily extendable.

3.2 Service-Deployment Hierarchy

Compared to existing hierarchies used for routing or network management, the service-deployment hierarchy extends the summarization (or aggregation) techniques to treat more generic information than only IP- or ATM-addressing and QoS. In addition, whereas network management mostly performs collection and aggregation of data upwards, the service-deployment hierarchy is used both ways: to collect data and to execute the deployment of a service based on the data collected. Note that 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, and a fourth level of hierarchy is often hidden within distributed routers (clusters) that appear as a single node with a single IP address to the outside. Extending the service hierarchy into such nodes can help automate the placement of functions in clusters.

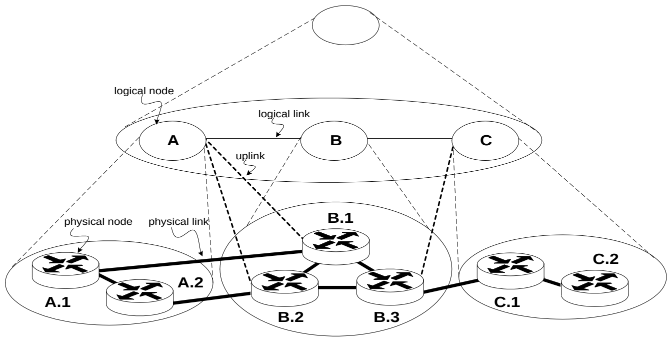

Fig. 3: A sample three-layer hierarchy.

Physical network topology is the main factor in creating a hierarchy, at least in fixed networks. A spanning-tree is built by successively grouping nodes at each hierarchy level. Figure 3 shows a simplified example of a seven-node network on top of which a three-layer hierarchy has been built. Nodes $B.1$, $B.2$, and $B.3$ are grouped together and represented by logical node B at the next level of the hierarchy. Routing across a hierarchical network requires the use of uplinks [AF96], as represented for node B in dashed thick lines $(B.1-A)$, $(B.2-A)$, and $(B.3-C)$ [uplinks $(A.1-B)$, $(A.2-B)$, and $(C.1-B)$ are not shown]. For instance, when routing a virtual circuit from a source in A to a destination in C, uplink $(B.3-C)$ shows that $B.3$ is the only possible border node towards C, whereas in the opposite direction, uplinks $(B.1-A)$ and $(B.2-A)$ show that $B.1$ and $B.2$ are the only possible border nodes towards A. For certain types of services, proper deployment requires that nodes along entire paths are enabled with the service. In such cases, the use of uplinks is necessary, as it will be illustrated later.

It is possible to construct many different hierarchies out of a given fixed physical topology, although administrative constraints such as trust, 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 [ADS00]. 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 nodes needed 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 topology changes, unnecessary redeployment of services might result.

In fixed networks, routing and network management hierarchies generally remain stable, whereas ad-hoc networks require special methods to dynamically adapt a hierarchy to varying connectivity [CJS99]. In fixed networks, changes in the hierarchy occur only when groups are partitioned because of links going down, or when a new node enters or quits a given group, such as a mobile network joining or leaving a fixed network. Partitions can be avoided by employing an appropriate network design, for instance by having multiple links interconnecting groups. The evaluation of possible hierarchies is left for further research.

3.3 Service-Deployment Mechanism

The service-deployment mechanism is divided into five steps. Figure 4 shows these steps and the resulting deployment procedures when all or only some of the steps are executed. Using only the last two steps in Fig. 4 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 all five steps have been executed will an *automatic deployment with custom metrics and automatic configuration* result. A more detailed description of the mechanism is presented in [HDS01b].

4 Hierarchical QoS-aware Service Deployment

Now that we described the key elements of the service-deployment framework, this section concentrates on the formalization of the service-deployment mechanism using an approach similar to [CMZB00], albeit

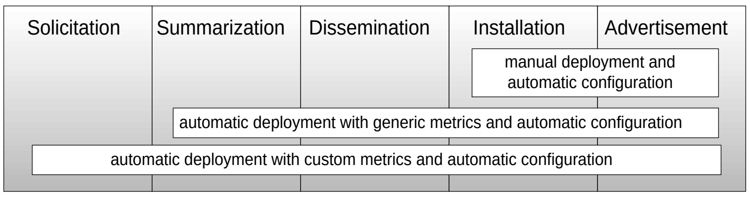
Solicitation
Summarization
Dissemination
Installation
Advertisement
manual deployment and automatic configuration
automatic deployment with generic metrics and automatic configuration
automatic deployment with custom metrics and automatic configuration

Fig. 4: The five steps of the service-deployment mechanism.

with specific enhancements. We then illustrate how it is implemented for service deployment of the three categories (*path-based*, *node-based*, and *path- and node-based* deployment), using examples of an in-band-signaling implicit-addressing service (called diff-serv++), an in-band-signaling explicit-addressing service (distributed web cache), and an out-of-band-signaling service (VPN, with both implicit- and explicit-addressing). In the path-based category, after the detailed example of the continuous type, the complexity of the sparse-type variant is presented.

4.1 Hierarchical Iterative Gather-Compute-Scatter (HIGCS) Algorithm

The mechanism is divided into a sequence of iterations each consisting of gather, compute, and scatter phases. The sets of destinations and origins relevant for the messages exchanged in the scatter and gather phases, respectively, are obtained from the compute phase. Note that these iterations can occur indistinctively in logical nodes of the hierarchy as well as in nodes of the underlying physical topology.

Service-deployment HIGCS messages exchanged during these iterations follow the tree-like topology of the service hierarchy. The logical node of a group is merely responsible for communicating with its underlying peer group members of the scatter set. The logical node neither has to monitor all members of its group nor perform all computations centrally. But without loss of generality, we assume in the following descriptions that the scatter set is always determined by a central computation performed by the logical node rather than distributing this computation among group members. In addition, here we consider it more appropriate to focus on the distributed computations taking place along the hierarchy dimension itself, than on those within groups.

From this point on, logical nodes are considered to be the location of computation taking place on behalf of the group members. Note that a logical node is nothing but a process executing in some underlying physical node.

Each node executes iterative computations based on a tuple $\{(G_i, C_i, S_i) | 0 \leq i \leq (k-1)\}$, where k is the total number of iterations. The gather set G_i is defined here as the set of (logical or physical) nodes from which messages are expected. The compute phase C_i executes once the *iMsg* messages have been received from all nodes in G_i. The scatter set S_i denotes the (logical or physical) nodes to which *oMsg* messages are sent once the compute phase C_i completes. $oMsg_n$ messages can differ depending on their destination node n in the S_i set. Similarly to [CMZB00], node attributes are assumed to be available during the compute phase. This includes, for instance, the hierarchy level at which the node is located.

The generic signaling message format used by the service-deployment mechanism is defined in Table 1.

4.2 Service-Deployment Computations

To perform service deployment, iterations are associated with the steps as described in 3.3. For that purpose, we define the following general behaviors for C_i:

- C_0 selects the set of underlying nodes S_0 that have to be solicited (null set if executed on a physical node),
- C_1 summarizes information gathered from the set of underlying nodes G_1 (on a physical node, this information is created), and places it in *sMetric*,
- C_2 selects the set of nodes S_2 where the service is to be deployed (on a physical node, this is the node itself),
- C_3 summarizes the results from the deployment on the set of nodes G_3 (on a physical node, this information is obtained locally after the service has been installed), and places it in *iMetric*.

Tab. 1: The HIGCS service-deployment message format and the HIGCS diff-serv++ message.

Parameter	Generic value
serviceId	Instance of service deployed
hierarchyId	Identifier of service hierarchy
srcId	Source of message
destId	Destination
iterationId	Current iteration
G_i	Gather set for iteration i
C_i	Compute function for iteration i
S_i	Scatter set for iteration i
servSpec	Service specification
sMetric	Solicited metric
iMetric	Installed metric
...	Other service-specific info

Parameter	Diff-serv++ value
serviceId	diff-serv++ ID
hierarchyId	unique ID
srcId	B
G_0	null
C_0	`DS-solicit`
C_1	`DS-summarize`
C_2	`DS-disseminate`
C_3	`DS-install`
servSpec	diff-serv++ specification
iMetric	DS-iMetric
sMetric	DS-sMetric
ends	A, C

Clearly, these functions will be implemented differently for each service to be deployed, as we shall describe next. The service-specific allocation policy is contained in the implementation of the C_1 and C_2 functions.

4.3 Continuous Path-Based Deployment: Differentiated Service

Table 1 shows on the right-hand side the message used in the deployment of a hypothetical diff-serv++ service on all nodes of a path between two customer sites (represented by the A and C top-level nodes in Fig. 5).

The computations executed for this example are shown in Table 2. For the sake of simplicity, the straightforward update of other fields in the *oMsg* (such as *srcId*) and fields that do not change compared to the *iMsg* message are not shown.

Tab. 2: The diff-serv++ C_0, C_1, C_2, and C_3 functions.

C_0		DS-solicit
S_0	$\leftarrow$	`SelectAllNodesBetween(`*iMsg.ends*`)`
$oMsg_n.ends$	$\leftarrow$	`SelectNeighborNodes(`$n \mid n \in S_0$`)`
G_1	$\leftarrow$	S_0
C_1		DS-summarize
S_1	$\leftarrow$	`GetLogicalNode()`
oMsg.sMetric	$\leftarrow$	**if** `IsLogicalNode()` **then** `SummarizeMetrics(`$iMsg_j.sMetric$, $\forall j \in G_1$`)` **else** `CreateMetric(`*iMsg.servSpec*`)`
G_2	$\leftarrow$	S_1
C_2		DS-disseminate
S_2	$\leftarrow$	**if** `IsLogicalNode()` **then** `SelectNodesOnShortestPath(`*iMsg.ends*`)` **else** `null`
$oMsg_n.ends$	$\leftarrow$	`SelectNeighborNodes(`$n \mid n \in S_2$`)`
G_3	$\leftarrow$	S_2
C_3		DS-install
oMsg.iMetric	$\leftarrow$	**if** `IsLogicalNode()` **then** `SummarizeInstalledMetrics(`$iMsg_j.iMetric$, $\forall j \in G_3$`)` **else** `InstallService(`*iMsg.servSpec*`)`
S_3	$\leftarrow$	`GetLogicalNode()`

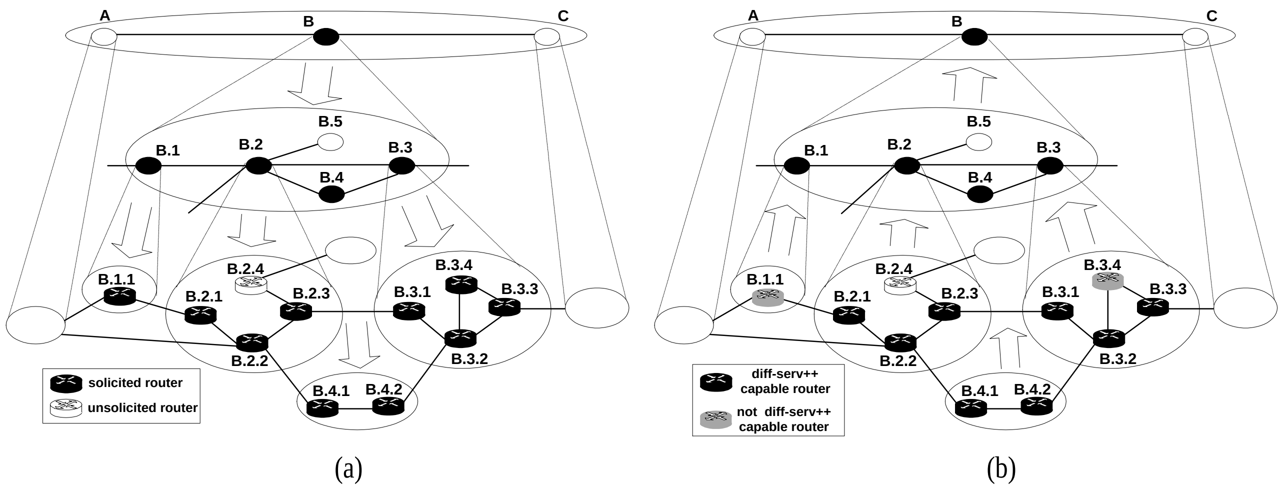

Fig. 5: (a) Solicitation step and (b) Summarization step.

DS-solicit

Figure 5 (a) illustrates the behavior of the `SelectAllNodesBetween()` and `SelectNeighborNodes()` functions used during the C_0 computation, i.e. the first iteration in the HIGCS algorithm. When it is executed at node B, `SelectAllNodesBetween(`A`, `C`)` returns the nodes of the scatter set $B.1$ through $B.4$, but not $B.5$ as it is not on a path (without loop) between A and C. For each node returned by this function, the *ends* are updated using `SelectNeighborNodes()`, and a specific *oMsg* message is forwarded to that node. For instance, `SelectNeighborNodes(`$B.2$`)` will return $\{A, B.1, B.3, B.4\}$, as these are the direct neighbor nodes on a path between A and C when considering the uplinks (see Section 3.2). Uplinks (not shown) exist between $(B.1 - A)$, $(B.2 - A)$, and $(B.3 - C)$. The gather set G_1 for the following iteration is set to the scatter set of the current iteration, i.e., responses are expected from each node of S_0 before executing C_1.

This procedure is repeated until the messages reach physical nodes: there, S_0 and G_1 are null, and the next iteration begins. Nodes involved during the first iteration are shown in black in Fig. 5 (a).

DS-summarize

During this iteration, C_1 either computes a metric for a physical node or summarizes such metrics for a logical node. `CreateMetrics()` evaluates node capabilities against the specific diff-serv++ service requirements *servSpec*. Routers capable of supporting diff-serv++ are shown in black in Fig. 5 (b). `SummarizeMetrics()` computes a transition matrix [IS99] T with the shortest path composed of diff-serv++ capable routers only. The scatter set S_1 is defined by `GetLogicalNode()` as the logical node associated with the current node, for instance, node B for node $B.1$. S_1 defines the recipient for the *oMsg* message. The gather set G_2 for the following iteration is set to the same value as S_1.

Transition matrices $T_{B.1}$, $T_{B.2}$, $T_{B.3}$, $T_{B.4}$, and finally T_B, are obtained successively at each level of the hierarchy. Each element $m_{i,j}$ of these matrices is defined as the shortest number of diff-serv++ capable hops between border nodes i and j. Given the numbering of nodes chosen here and matrix T_B, $m_{i,j}$ corresponds to the path cost between nodes $B.(i+1)$ and $B.(j+1)$.

$$T_{B.2} = T_{B.3} = \begin{pmatrix} 1 & \dots & \\ 2 & 1 & \dots \\ 3 & 2 & 1 \end{pmatrix}; T_B = \begin{pmatrix} 0 & \dots & \\ 0 & 1 & \dots \\ 0 & 5 & 1 \end{pmatrix}$$

As matrices are symmetric in this example, only one half is shown. As can be seen from $T_{B.2}$, $B.2.4$ is simply ignored because this border node is not diff-serv++ capable.

Note that *oMsg.sMetric* contains only the result of the summarization. But until all iterations have completed, each node keeps a state of the nodes composing the shortest paths whose costs are advertised in T. This avoids recomputing these paths in the next step.

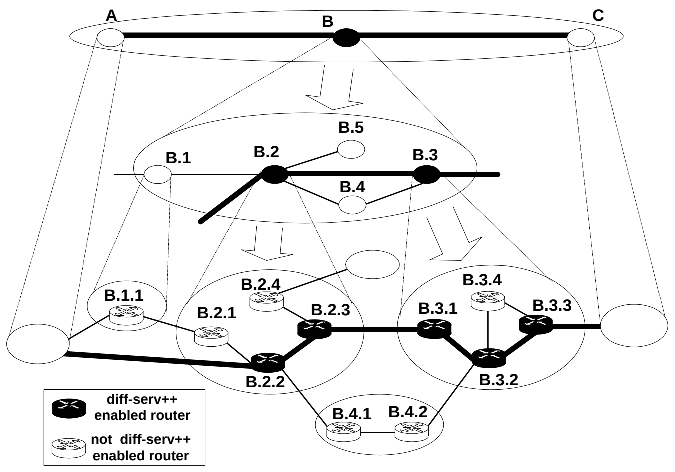

Fig. 6: Dissemination step.

Minimum link bandwidth requirements for the service could be added, and used to compute the appropriate T. Similarly to flow routing, many different algorithms can be used to compute the best paths, such as widest path first, where also the processing capability of nodes along the path could be considered. Every service can specify its own preferences in C_1. The complexity of path routing with multiple metrics can be high or even become NP-complete, especially if additive metrics are used. Therefore, metrics definition always has to be balanced against the complexity in its summarization that could arise if it is not chosen appropriately.

The next iteration begins when all *oMsg* messages have been collected by B (i.e., coming from every node in G_1), as `GetLogicalNode` returns null at B.

DS-disseminate

Inspecting matrix T_B reveals that $m_{2,1}$ is different from zero, i.e., that the transit network is capable of supporting the diff-serv++ service between the two customer sites A and C. Therefore, the dissemination takes place, as shown in Fig. 6, where `SelectNodesOnShortestPath()` selects the shortest path at each level of the hierarchy based on the transition matrices T provided by the lower level. The edges of the shortest path are shown as thicker lines at each level.

This step directs the deployment of a service. More specifically, during this third iteration of the mechanism, the service-deployment message is forwarded by a logical node only to those nodes in which the service needs to be deployed (defined as S_2), rather than executing a complete flooding.

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 left for further study.

When these messages reach the physical nodes, S_2 and G_3 are null, and the next iteration begins.

Note that nodes that are part of the set S_0 but not of S_2 will not complete their iterations. These nodes can either time out and clear the state associated with the iterations already executed, or the state can be cleared explicitly by sending a specific dissemination message to them.

DS-install

Physical nodes first execute this iteration and install the service. `InstallService()` for instance returns a boolean, transmitted in *oMsg.iMetric* to the node set in S_3. Logical nodes execute `Summarize-InstalledMetrics()`, which can for instance perform a boolean `AND` from underlying results. Node B eventually obtains confirmation that the diff-serv++ has been successfully installed. Routers with service installed are shown in black in Fig. 6.

4.4 Sparse Path-Based Deployment

Instead of installing the same diff-serv++ function in each node along the path, a sparse-type service requires various functions (such as filtering, compression, transcoding, and encryption) to be installed in only certain nodes of a path, as shown in Fig. 1.

Assuming that a set of functions $F = \{F_1, F_2, F_3\}$ has to be executed in this order between customer sites A and C of the previous example, then shortest-path transition matrices that account for the cost of traversing the logical nodes and executing such functions can be computed. Each solicited logical node below B computes seven transition matrices, forming the set $M(F_1, F_2, F_3)$:

- $T^{\emptyset}$ (none of the functions are accounted),
- $T^{\{F_1\}}$ (only F_1 is accounted), and
- $T^{\{F_1,F_2\}}$ (both F_1 and F_2 are accounted), and
- $T^{\{F_1,F_2,F_3\}}$, $T^{\{F_2\}}$, $T^{\{F_2,F_3\}}$, and $T^{\{F_3\}}$.

Other combinations, such as $T^{\{F_1,F_3\}}$ are dismissed as they would break the ordering rule. Note that node B only needs to compute $T_B^{\{F_1,F_2,F_3\}}$. Ultimately, all three functions will be installed on a path between the two customer sites, for instance with F_1 executed in $B.2$, while F_2 and F_3 are executed in $B.3$. [CTW01] presents an algorithm that computes shortest paths when a sequence of functions need to be executed along the path, although only in the simpler form of non-hierarchical networks, and with total ordering of functions.

We define l as the number of such matrices T:

$$l \quad \triangleq \quad |M(F_1, F_2, \ldots, F_n)| \quad .$$

In general, assuming total ordering of functions, $l_{\text{total order}}$ can be calculated as follows:

$$l_{\text{total order}} \quad = \quad 1 + \sum_{i=0}^{n-1} (n-i) \quad = \quad 1 + \frac{n(n+1)}{2}$$

where n is the number of functions in the set F.

On the other hand, if there is no ordering rule at all, then the number $l_{\text{no order}}$ of matrices is:

$$l_{\text{no order}} \quad = \quad 1 + \sum_{i=1}^{n} \frac{n!}{(n-i)!} \quad = \quad 1 + n! \cdot \sum_{i=0}^{n-1} \frac{1}{i!} \quad < \quad 1 + \mathbf{e} \cdot n! \qquad \text{as} \qquad \mathbf{e} = \sum_{i=0}^{\infty} \frac{1}{i!} \cong 2.718 \quad .$$

Therefore, as opposed to continuous path-based deployment, scalability of sparse path-based deployment depends heavily on the number n of functions composing the service as well as the number of combinations allowed by the ordering rules. The number of transition matrices computed by each logical node is in the order of $O(n!)$ in the worst case, whereas it is $O(n^2)$ in the best case.

4.5 Node-Based Deployment: Caching Service

We consider here the deployment of a distributed web-caching mechanism. For each group of (physical) nodes, the one having the best processing capability to support the service will be set up with it. Table 3 shows the C_0, C_1, and C_2 functions used for that purpose.

In this example, nodes in the network are polled against the web-cache service requirements (outcome of `CreateMetric(`*iMsg.servSpec*`)`). Out of each group, the node with the best metric is chosen (defined as `MAX(`$iMsg_j.sMetric$`,`$\forall j \in G_1$`)`). The remainder of the summarization proceeds so that finally the metric of the worst node (defined as `MIN(`$iMsg_j.sMetric$`,`$\forall j \in G_1$`)`) is known as indicator of how the entire network can support the service. In the dissemination step, `SelectBestProcessingNode()` is used in the logical nodes of the hierarchy layer just above the physical layer (where `IsFirstLogicalNode()` returns true) to pick the best node in that group.

Tab. 3: The web cache C_0, C_1, and C_2 functions.

C_0		CS-solicit
S_0	$\leftarrow$	`SelectAllNodesInGroup()`
G_1	$\leftarrow$	S_0
C_1		CS-summarize
oMsg.sMetric	$\leftarrow$	**if** `IsPhysicalNode()` **then** `CreateMetric(`*iMsg.servSpec*`)` **else if** `IsFirstLogicalNode()` **then** `MAX(`$iMsg_j.sMetric$`,` $\forall j \in G_1$`)` **else** `MIN(`$iMsg_j.sMetric$`,` $\forall j \in G_1$`)`
S_1	$\leftarrow$	`GetLogicalNode()`
G_2	$\leftarrow$	S_1
C_2		CS-disseminate
S_2	$\leftarrow$	**if** `IsPhysicalNode()` **then** `null` **else if** `IsFirstLogicalNode()` **then** `SelectBestProcessingNode()` **else** `SelectAllNodesInGroup()`
G_3	$\leftarrow$	S_2

Clearly, depending on the type of service to be installed, it might not be necessary to have an indicator of how well the service can be supported at the highest hierarchy level. By modifying the above example as shown in Fig. 7, the web-cache service is installed without executing all iterations: in any group of nodes the service is directly installed on the best node provided it has at least a processing capability of *minValue*. Otherwise, the service is not installed. Again, only the C_0, C_1, and C_2 functions are shown in Table 4. An indicator *iMetric* of how well the service is installed could be computed by a function C_3, counting for instance the number of groups in which the service has been installed successfully.

Leaving the G_i set undefined prevents the C_i computation from taking place, whereas setting it to null will immediately trigger the C_i computation. As C_0 sets G_3, this will cause C_1 and C_2 to be skipped in those nodes for which the network capabilities need not be known when installing the service. These nodes are represented by dashed lines in the pyramid in Fig. 7 (b).

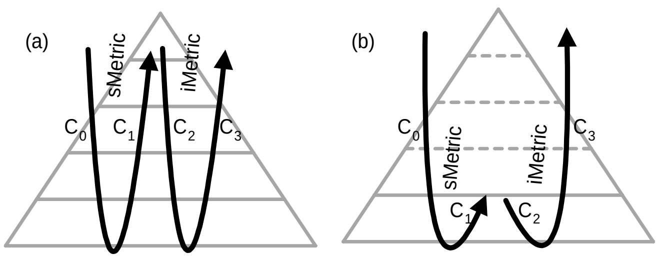

Fig. 7: (a) Caching Service and (b) Direct Caching Service service-deployment flows.

Allowing iterations to be skipped requires that care be taken so that no inconsistency occurs: a logical node should not receive messages that belong to different iterations. The example above ensures that this does not happen by having all nodes at the same level skip the same iteration. Resolving possible expected inconsistencies, as occur for instance in more sophisticated deployment scenarios, is left for further research.

4.6 Path- and Node-Based Deployment: VPN

As an example of a service deployed both along paths and at selected nodes with differing requirements, we consider the deployment of a VPN. Encryption capabilities are required at the VPN endpoints, and QoS treatment of packets is ensured by RSVP-enabled nodes between those endpoints. This service is based on

Tab. 4: The web cache C_0, C_1, and C_2 functions for a direct install.

C_0		DCS-solicit
S_0	$\leftarrow$	`SelectAllNodesInGroup()`
G_1	$\leftarrow$	S_0
G_3	$\leftarrow$	S_0
C_1		DCS-summarize
$oMsg.sMetric$	$\leftarrow$	**if** `IsPhysicalNode()` **then**
		`CreateMetric(`$iMsg.servSpec$`)`
S_1	$\leftarrow$	**if** `IsFirstLogicalNode()` **then**
		`null`
		else `GetLogicalNode()`
G_2	$\leftarrow$	**if** `(IsFirstLogicalNode() AND`
		$(\texttt{MAX}(iMsg_j.sMetric,\ \forall j \in G_1) \geq iMsg.minValue))$
		then
		`null`
		else if `NOT(IsFirstLogicalNode())` **then**
		S_1
G_3	$\leftarrow$	**if** `IsFirstLogicalNode() AND`
		$\texttt{NOT}(\texttt{MAX}(iMsg_j.sMetric,\ \forall j \in G_1) \geq iMsg.minValue))$
		then
		`null`
C_2		DCS-disseminate
S_2	$\leftarrow$	**if** `IsFirstLogicalNode()` **then**
		`SelectBestProcessingNode()`
		else
		`null`
G_3	$\leftarrow$	S_2

implicit addressing for the RSVP part and on explicit addressing for the encryption endpoints that have to be configured with each other addresses.

Here we concentrate only on the metric used by the C_1 function, and how to summarize it. The C_2 function then uses this metric to pick the best VPN configuration, i.e. a lowest-cost spanning tree, successively at each level of the hierarchy, using search methods such as presented in [Isa00].

The metric used is an extended transition matrix defined as follows:

$$T = (\ M\ ;\ \ P\ ).$$

Elements $m_{i,j}$ indicate the shortest number of RSVP-capable hops between border nodes i and j. Elements $p_{i,j}$ indicate the shortest path from a node in domain D_j that fulfills the requirements of the VPN-endpoint service specification to border node i. The VPN interconnects n domains D_n. Logical nodes represent these domains.

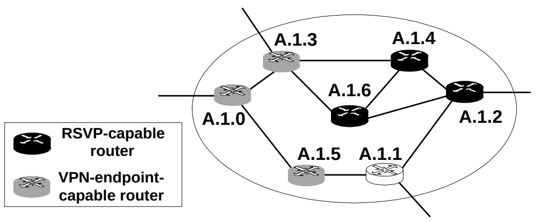

Fig. 8: A group of nodes solicited for the VPN service.

Figure 8 shows a group of nodes with their capabilities. For simplicity, it is assumed that all VPN-endpoint-capable nodes are also RSVP-capable, but not vice-versa. The extended transition matrix for this group is [assume $m_{i,j}$ corresponds to $(A.1.i - A.1.j)$]

$$T_{A.1} = \begin{pmatrix} 1 & \dots & & & 1 \\ 0 & 0 & \dots & & 0 \\ 4 & 0 & 1 & \dots & 3 \\ 2 & 0 & 3 & 1 & 1 \end{pmatrix}.$$

If we assume that logical node $A.1$ represents one domain and that logical node $A.2$ (not shown) represents another domain, then the P matrix of the extended transition matrix for logical node A, composed of $A.1$ and $A.2$, will have two columns $\mathbf{p}_{:1}$ and $\mathbf{p}_{:2}$, one for each domain. When summarizing such extended matrices, a new column is appended for each domain considered.

4.7 Deployment Reliability and Auto-Configuration of Services

Installation can fail for multiple reasons, such as stale metrics. The *iMetric* can be set to `False` in the diff-serv++ example to notify higher logical nodes of the failure. This event will trigger local repairs that try to find a replacement path in the same group, and escalate the failure notification to the next higher logical node if this is not feasible (this behavior has been deliberately omitted from the description of the C_3 function in Fig. 2).

Once installed, a service needs to be configured appropriately. It 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. When routing needs to be aware of the installed service (i.e., implicit-addressing services such as presented in 4.3), then having the service-deployment hierarchy aligned with the routing hierarchy will significantly ease the advertisement procedure. In [HDB00] the various automatic discovery techniques for the configuration of explicit-addressing services are evaluated in the context of IP and ATM networks. Compared to centralized directory services, PAR (PNNI Augmented Routing [AF99]) can be used to advertise such services, as it is more robust and scalable. Examples of service advertisement procedures to automatically configure a hierarchical IP network are given in [PDH00].

Once the service is installed and running, failures could still occur. The mechanism presented here does not maintain state in nodes after completion of the deployment procedure. If nodes participating in the service are unable to detect that an error has occurred (e.g., implicit-addressing services), then a special monitoring function needs to be installed: the C_3 function for instance installs a hard state (a state that remains after the final iteration of the deployment procedure) in each physical or logical node where it executes, and initiates a monitoring procedure that lasts for the duration of the service. This monitoring procedure executes computations similar to C_3 at regular time intervals, or when specific events occur.

5 Conclusion

New mechanisms that leverage the openness and programmability starting to appear in network equipment are key to managing the intelligent network infrastructure. The automated deployment of services in open programmable heterogenous networks is one such mechanism.

This paper has introduced a formalization called HIGCS of the steps proposed in [HDS01b] to execute a generic automated deployment of services. A new taxonomy for describing the deployment needs of a service is introduced and illustrated with examples of (continuous and sparse) path-based, node-based, and path- and node-based deployment. In the case of sparse path-based deployment, the lower and upper bounds of the number of necessary information elements (namely transition matrices for the chosen example) have been presented. It was shown that, contrary to continuous path-based deployment, a careful selection of the number of functions composing the service and of the number of combinations allowed by the ordering rules is required if a sparse path-based deployment is to remain scalable.

The backbone of the mechanism, namely the hierarchical structure, allows commands to be progressively refined and information to be aggregated. As each service can independently define its metrics and allocation policy in the service-deployment messages that travel along the hierarchy, it allows deployment to be tailored to the specific needs of each service.

The resulting automated deployment avoids time-consuming and error-prone manual operations. When diversity of services increases and networks grow in size and heterogeneity, the deployment mechanism can still capture the essential data for deploying any particular service, without losing information relevant to that specific service.

Current work includes the simulation of service deployment over large-scale networks using mobile agents, implementing the HIGCS distributed computation paradigm. In addition, specific information aggregation methods that include economical factors are being investigated [HDS01a]. We also try to refine the representation of relevant node capabilities to allow simple matchmaking against service requirements.

References

[ADS00] B. Awerbuch, Y. Du, and Y. Shavitt. *The Effect of Network Hierarchy Structure on Performance of ATM PNNI Hierarchical Routing*. *Computer Comm.*, 23(10), May 2000.

[AF96] ATM-Forum. *P-NNI V1.0*. af-pnni-0055.000, March 1996.

[AF99] ATM-Forum. *PNNI Augmented Routing (PAR) Version 1.0*. af-ra-0104, January 1999.

[BBC$^+$98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services*. IETF RFC 2475, December 1998.

[Bis00] J. Biswas et al. *Application Programming Interfaces for Networks*. Technical report, IEEE PIN1520 Working Group, 2000. www.ieee-pin.org.

[BS99] M. Brunner and R. Stadler. *Virtual Active Networks - Safe and Flexible Environments for Customer-Managed Services*. In *Proc. 10th IFIP/IEEE Int'l Workshop on Distributed Systems (DSOM'99)*, Zurich, Switzerland, October 1999.

[BZB$^+$97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource Reservation Protocol (RSVP)– Version 1 Functional Specification*. IETF RFC 2205, September 1997.

[CF00] CPIX-Forum. *Common Programming Interfaces*, 2000. www.cpixforum.org.

[CJS99] W. Chen, N. Jain, and S. Singh. *ANMP: Ad Hoc Network Management Protocol*. *IEEE J. Sel. Areas in Commun.*, 17(8), August 1999.

[CKV$^+$99] A. Campbell, M. Kounavis, D. Villela, J. Vicente, K. Miki, H. D. Meer, and K. Kalaichelvan. *Spawning Networks*. *IEEE Network Magazine*, July/August 1999.

[CMZB00] Y. Chae, S. Merugu, E. Zegura, and S. Bhattacharjee. *Exposing the Network: Support for Topology-Sensitive Applications*. In *OPENARCH 2000*, Tel Aviv, Israel, March 2000.

[CTW01] S. Choi, J. Turner, and T. Wolf. *Configuring Sessions in Programmable Networks*. In *Infocom 2001*, Anchorage, Alaska, April 2001.

[CZH$^+$99] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. *An Architecture for a Secure Service Discovery Service*. In *Mobile Computing and Networking*, pages 24–35, 1999.

[DHSW00] A. Doria, F. Hellstrand, K. Sundell, and T. Worster. *General Switch Management Protocol V3*. IETF draft <draft-ietf-gsmp-05.txt>, January 2000.

[dKTG00] J. de Keijzer, D. Tait, and R. Goedman. *JAIN: A New Approach to Services in Communication Networks*. *IEEE Commun. Magazine*, January 2000.

[F01] IETF ForCES Working Group. *Requirements for Separation of IP Control and Forwarding*. IETF draft in preparation, 2001.

[GRR00] N. Greene, M. Ramalho, and B. Rosen. *Media Gateway Control Protocol Architecture and Requirements*. IETF RFC 2805, April 2000.

[GW93] P. Grillo and S. Waldbusser. *Host Resources MIB*. IETF RFC 1514, September 1993.

[HDB00] R. Haas, P. Droz, and D. Bauer. *PNNI Augmented Routing (PAR) and Proxy-PAR*. *Computer Networks*, 34(3):399–418, September 2000.

[HDS01a] R. Haas, P. Droz, and B. Stiller. *Cost- and Quality-of-Service-Aware Network-Service Deployment*. In *ICQT 2001*, Vienna, Austria, September 2001.

[HDS01b] R. Haas, P. Droz, and B. Stiller. *A Hierarchical Mechanism for the Scalable Deployment of Services over Large Programmable and Heterogeneous Networks*. In *ICC 2001*, Helsinki, Finland, June 2001.

[IS99] I. Iliadis and P. Scotton. *Transition Matrix Generation for Complex Node Representation*. In *Proc. IEEE ATM Workshop'99*, pages 489–500, Kochi, Japan, May 1999.

[Isa00] R. Isaacs. *Lightweight, Dynamic and Programmable Virtual Networks*. In *OPENARCH 2000*, Tel Aviv, Israel, March 2000.

[KS00] R. Kawamura and R. Stadler. *Active Distributed Management for IP Networks*. *IEEE Communications Magazine*, 38(4):114–120, April 2000.

[Kum01] S. Kumar. *Self-organizing hierarchies and their application to the Internet and emerging pervasive computing applications*. PhD thesis, USC, April 2001.

[LS01] K. Lim and R. Stadler. *A Navigation Pattern for Scalable Internet Management*. In *IFIP/IEEE International Symposium on Integrated Network Management (IM'01)*, Seattle,Washington, May 2001.

[NPB01] A. Nakao, L. Peterson, and A. Bavier. *Constructing End-to-End Paths for Playing Media Objects*. In *OpenArch 2001*, Anchorage, Alaska, March 2001.

[NWX00] K. Nahrstedt, D. Wichadakul, and D. Xu. *Distributed QoS Compilation and Runtime Instantiation*. In *Proceedings of the IEEE/IFIP International Workshop on QoS (IWQoS'2000)*, Pittsburgh, June 2000.

[PDH00] T. Przygienda, P. Droz, and R. Haas. *OSPF over ATM and Proxy-PAR*. RFC 2844, IETF, May 2000.

[Pet99] L. Peterson. *NodeOS Interface Specification*. Technical report, DARPA Active Networks Program, June 1999.

[QK99] J. Quittek and C. Kappler. *Practical Experiences with Script MIB Applications*. *The Simple Times*, 7(2), November 1999. www.simple-times.org.

[SQK00] J. Schonwalder, J. Quittek, and C. Kappler. *Building Distributed Management Applications with the IETF Script MIB*. *IEEE J. Sel. Areas Commun., Special Issue on Network Management and Operations*, 18(5), May 2000.

[XNW01] D. Xu, K. Nahrstedt, and D. Wichadakul. *QoS-Aware Discovery of Wide-Area Distributed Services*. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 2001.

[YGY91] Y. Yemini, G. Goldszmidt, and S. Yemini. *Network Management by Delegation*. In *Proc. Int'l Symp. on Integrated Network Management*, pages 95–107, April 1991.