Autonomic service deployment in networks

by R. Haas
P. Droz
B. Stiller

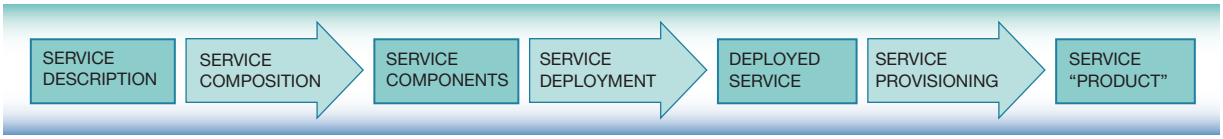
Networks have been growing dramatically in size and functionality in past years. Internet Protocol network nodes not only forward datagrams using longest-prefix matching of the destination address, but also execute functions based on dynamic policies such as proxy-caching, encryption, tunneling, and firewalling. More recently, programmable behaviors have begun to appear in network elements, allowing experimentation with even more sophisticated services. This paper presents an autonomic approach to network service deployment that scales to large heterogeneous networks. Topological categories of service deployment are introduced. A two-phase deployment mechanism that is split into hierarchically distributed and central computations is presented and illustrated with examples of actual services in a programmable network environment, together with their deployment algorithms and simulation results. Autonomic service deployment allows the distributed and complex capabilities present in network elements to be leveraged more efficiently when installing new services than is possible in traditional centralized network management-based approaches. As a result, installation is faster and use of functional resources is more optimized.

A network manager faces a daunting task today when designing, configuring, and provisioning a complete service for customers, and when trying to obtain the most use of the specific capabilities available in sophisticated network elements such as programmable routers, encryption and transcoding gateways, traffic shapers and purifiers, and distributed caches, just to name a few. However, it would not be profitable to add more capabilities to a network, for instance, in the form of network processors,[1] unless they can be exploited efficiently when installing and running a service.

If we consider an environment of networks with large numbers of nodes that have widely varying capabilities and resources and that need to be enabled with new services, it is necessary to define and provide a way to organize the deployment of new services at both the network and the node levels. The framework presented here addresses both levels globally, as well as the interactions taking place between them.

Activities that focus on the deployment of services over heterogeneous programmable networks are still very few and do not focus on those aspects that are exacerbated in large networks. Policy-based networking allows a high-level policy to be transformed into lower-level network-node configurations.[2] Such mechanisms depend on an efficient resource discovery and enablement, as presented here. Dynamic composition and deployment of services in the context of end-to-end application sessions are addressed in Refer-

Figure 1 Network service operations



ences 3, 4, and 5. 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. Active networks[6] achieve self-controlled deployment of services in a network by embedding service execution code into data packets so that the service remains dedicated to that flow of packets. This method is particularly suitable for environments with many network nodes that support the necessary execution environment and for short-lived flows that require an *ad hoc* deployment of a service exclusively along the path through which the flows have been routed. Particularly well-suited for large-scale problems are hierarchical architectures that have been used in the context of routing protocols and network management but not yet considered for deploying services.

To accelerate the deployment of network services, at least at the node level, efforts have begun focusing on the standardization of interfaces within networking equipment, either in the form of control protocols for label switches (Internet Engineering Task Force [IETF] General Switch Management Protocol [GSMP][7]), Internet Protocol (IP) routers (IETF ForCES[8]), and media gateways (IETF MEGACO[9]), or more generic application programming interfaces (APIs) such as those described in References 10 and 11. Therefore, it is expected that in a network a variety of solutions are likely to coexist.

Although the work presented here specifically addresses network services, the deployment of higher-level services such as Web services, for which the network can be viewed as a black box, indirectly benefits from the underlying network service-deployment framework.

The next section of this paper first presents the network-level and node-level service-deployment phases, then classifies the types of services supported by the framework presented here, and finally reviews the key elements such as the representation of capabilities and the hierarchical architecture. The third section focuses on network-level deployment. It presents a formalism for hierarchically distributed computations, illustrated with examples and algorithms. Simulation results of the network-level deployment are presented in the fourth section.

Service-deployment framework

Service deployment denotes the set of tasks required to provide a new service dynamically in a partially or fully programmable network. A service is an assembly of components that have to be identified and placed appropriately in a network. Service provisioning is the task that operates on a service already deployed in order to provide a product of that service. For instance, encrypted flows are a product of the Virtual Private Network (VPN) service, and the VPN service is a product of its components present in the network nodes, performing encryption or decryption at the edges and quality-of-service (QoS) in the intermediate nodes, as shown in Figure 1. Whereas service composition defines the components required by a service and how to compose them, service deployment performs the actual mapping of these components into the network.

Clearly, providing tailored services means that new components have to be placed adequately in the network. We argue that an *autonomic* approach is the only scalable solution to service deployment, given the heterogeneity and size of today's networks as well as the variety of different services and the frequency at which such services have to be deployed. Autonomic means that the network itself orchestrates the deployment process, and the interaction with the network manager is limited to specifying the service according to customer needs.

More specifically, this framework splits service deployment into two successive phases, namely, *macro* and *micro deployment*. As shown in Figure 2, each phase covers a certain scope of the network, and the border between these scopes can be adjusted. In the

Figure 2 Macro and micro deployment

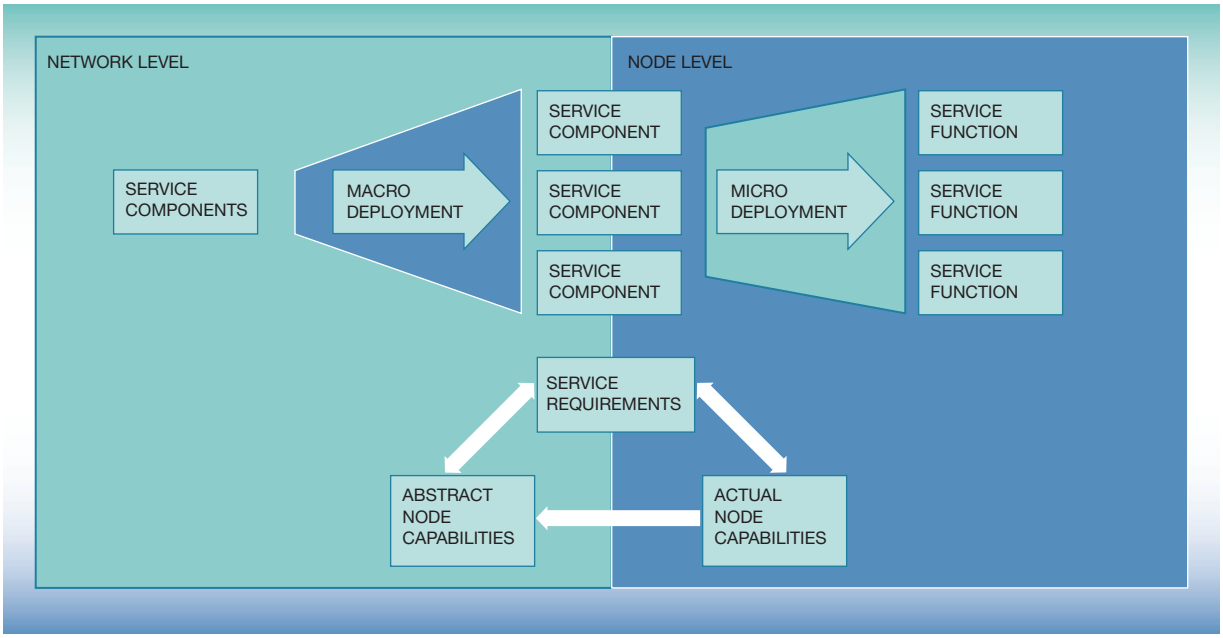


Table 1 Relevant characteristics of macro and micro deployment

Macro Deployment	Micro Deployment
Network level:	Node level:
Distribute hierarchically	Distribute centrally (control point)
Use abstract representation of node capabilities	Use actual representation of node capabilities
Minimize nodes' resources for matchmaking, get Faster processing Solicit more nodes	Make use of specific node capabilities when installing the service, for Fine-grained optimization

following discussion, we choose to place the border at the distributed-router level. Therefore, the macro deployment has a network-wide scope, whereas the micro deployment has a node-local scope. This choice does not preclude other scenarios in which the border is set instead at the local-area-network level, for instance.

For macro or network-level deployment, a sequence of five steps is executed in a hierarchically distributed manner, as described in more detail in the next section. For micro or node-level deployment, a centralized resource co-allocation method is used that benefits from information gathered during the network-level phase in order to place functions optimally. A service component could need resources of different types to be allocated, one for each service function constituting the service, hence the co-allocation problem. The main characteristics of both phases are summarized in Table 1.

Categories of services. Services are assumed to be decomposable into sets of components to be executed by individual nodes. We distinguish the following topological categories of service deployment and provide examples of current network services:

- *Path-based*, between a set of source(s) and destination(s), which is further divided into two types:
  - *Continuous*, for which the same component must be present in *each* node on the path, for instance, application-specific queuing (such as IETF Differentiated Services, or diffserv) that has to be enabled on all nodes of a path
  - *Sparse path-based*, or discontinuous, for which a set of components must be present in a *set of nodes* on the path. This type can be, for instance, a mul-

timedia transcoding and compression service, with one node on the path performing transcoding while another node performs compression.

- *Fence-based*, orthogonal to path-based, for which nodes along a path (possibly a loop) must act on the traffic crossing them, such as a firewall spanning multiple access routers
- *Node-based*, for which only selected nodes need to be activated, and no source or destination pairs are specified, but rather domains, such as a transparent Web cache acting for a group of end stations
- *Combinations* of the above, such as a path-and-node-based VPN service with encryption at the endpoints and QoS support in the intermediate nodes

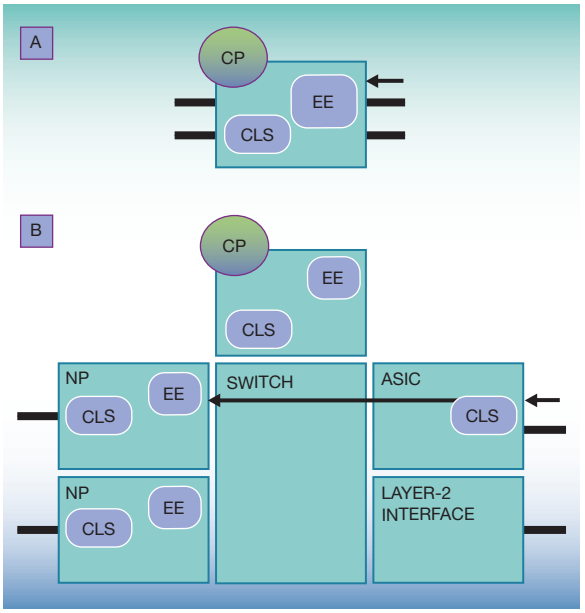
Whenever necessary, service redundancy can be achieved by deploying the service along multiple distinct paths or nodes.

Description of service requirements and node capabilities. Topological properties help to classify services from a network-level connectivity point of view, but a larger set of properties is required to fully describe service requirements as well as network or node capabilities. These properties are the following:

- Topology—description of the connectivity, which can be modified dynamically such as in wavelength-switching networks or with hot-pluggable node modules
- Functionality—description of functions, which can be static, configurable, or even programmable
- Performance—measure of resources, such as bandwidth or delay in networks and CPU speed in nodes
- Cost—administrative measure for using the above resources, relevant when the economical dimension must be taken into account during deployment, addressed in Reference 12.

The representation of node capabilities can, for instance, be expressed in XML (Extensible Markup Language) together with the appropriate Schema, as an extension to the IETF MIBs (Management Information Bases) such as described in References 13 and 14. It includes a description of the type of APIs to access, configure, and operate the resources in the node, either base[10] or higher-level resources such as operating-system-resident services, as well as the utilization of these resources. A similar representation for services is matched 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. The information resulting from this evaluation is a metric or a set of metrics that are used to direct the deployment of the service onto the most appropriate nodes, as is illustrated later in this paper. When this information is transported toward the top of the service-deployment hierarchy, during the network-level deployment phase, it can be aggregated differently, depending on service needs. The service-deployment framework proposed lets services themselves define the information they require to execute their deployment and how it should be aggregated. This defining can be viewed as an application of active code for the network control plane.

Figure 3 (A) Abstract capabilities, (B) actual capabilities for a programmable distributed router



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 assists (required for fast packet handling), and software-level programmability (required for flexible packet handling).

Figure 3 is a simple illustration of the abstract and actual representations of capabilities, as described in Figure 2. Part A of Figure 3 shows the abstracted relevant capabilities of the distributed router described in Part B, namely, a control point (CP), an execution environment (EE), and a packet-classifier mechanism (CLS). If a tunneling service that requires encapsulation and decapsulation of data packets to be performed at line speed has to be deployed, the macro deployment phase will select an appropriate endpoint for the tunnel, based on the abstract view of Part A. Once it is known that this particular node is the most appropriate one and which interface is to be used for this service (indicated by an arrow in Part A), micro deployment is able to proceed and allocate the necessary functions, such as the EE to perform the encapsulation and decapsulation, and the CLS to select the relevant packets to be tunneled. For instance, the CLS capability in the application-specific integrated circuit (ASIC) of the interface selected and the EE of an NP can be used, depending on availability.

Figure 4 shows a short example of possible NP capabilities. Using XML for such a representation rather than an MIB-like structure is interesting because XML is easily extendable, thanks to its self-contained structure.

Service-deployment hierarchy. Compared to existing hierarchies 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 (asynchronous transfer mode) addressing and QoS. Whereas network management mostly performs collection and aggregation of data upwards, the service-deployment hierarchy is used in both directions: to collect data resulting from the service requirements versus node capabilities evaluation 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: two levels for intranetwork routing, such as IP OSPF (Open Shortest-Path First) areas and Autonomous Systems, and a third level for internetwork routing, such as BGP (Border-Gateway Protocol). The hierarchy may extend downwards in network nodes such as distributed routers (clusters) that to the outside appear as a single node with a single IP address. Placing the border between macro and micro deployment within such nodes can bring the advantages of macro deployment to automating the placement of functions in large clusters (see Table 1).

The 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 5 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,[15] as represented for node B in thick dashed 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 from a source in A to a destination in C, uplink (B.3 − C) shows that B.3 is the only possible border node toward 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 toward A. For certain types of services, proper deployment requires that nodes along entire paths are enabled with a certain service component. In such cases, the use of uplinks is necessary.

Network-level service deployment

This section concentrates on the network-level deployment procedure and its formalization. We then illustrate how it is implemented for service deployment of the path-based and path-and-node-based categories, together with specific algorithms used in the deployment steps.

Deployment procedure. From a high-level perspective, the network-level service-deployment procedure can be broken down into five steps. Figure 6 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 Figure 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, for example, skipping the first step, the result is an *automatic deployment with generic metrics and automatic configuration*. Only when all five steps are executed will an *automatic deployment with custom metrics and automatic configuration* result.

Computations are distributed in a logical hierarchy described in Reference 16. Layers in the hierarchy are built by recursively grouping logical or physical

Figure 4 XML representation of the capabilities of a network processor

```
<Network_Processor>
  <base_capabilities>
    <API_supported> ForCES, NPF </API_supported>
    <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>
        <Advanced Queue Management> RED </AQM>>
      </buffers_management>
      <forwarding>
        <type> layer-4 </type>
        <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 diffserv
    <API_supported> ForCES, NPF </API_supported>
      <general>
        <classifier>
          <fields> 6 </fields>
          <options> ranges_support </options>
          // etc
        </classifier>
         // etc
      </general>
      <resource_usage>
        // current usage for the defined capabilities
      </resource_usage>
  </diff_serv>
  // etc
</Network_Processor>
```

nodes and representing them by a single logical node at the next layer of the hierarchy.

During the solicitation step, service requirements are distributed to nodes, following the service-deployment hierarchy downwards (see Figure 5). The summarization step collects the metrics created at the lowest level of the hierarchy, namely the physical nodes, and successively hands them over to the next higher layer of the hierarchy, possibly repeatedly aggregating the results to prevent information overflows. Once the result reaches the top level of the hierarchy, the dissemination step can start, in which the metrics collected at each level are evaluated and

Figure 5 A sample three-layer hierarchy

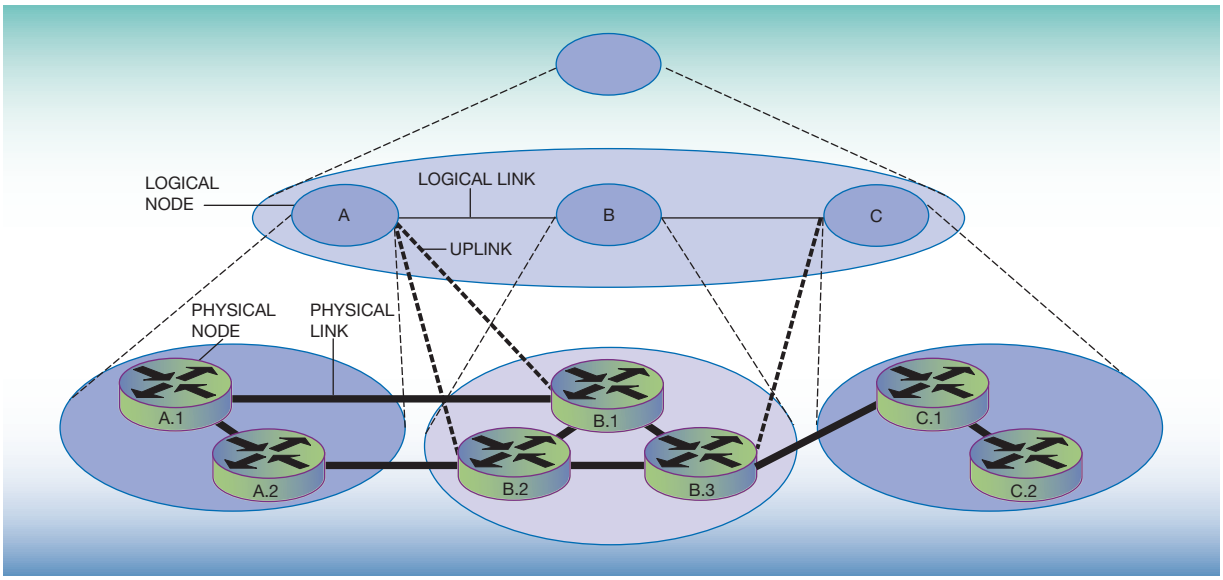
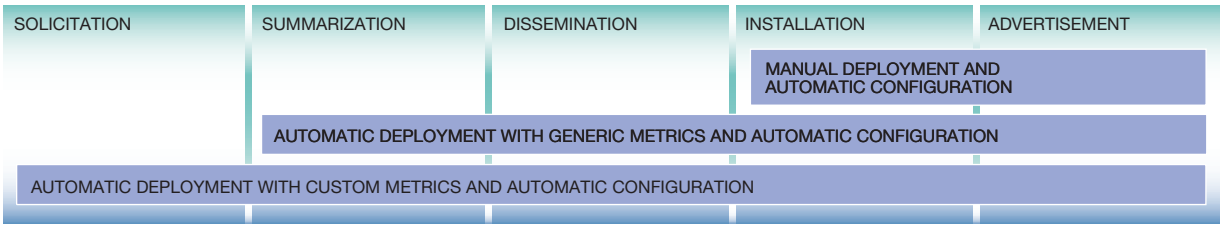


Figure 6 The five steps of the service-deployment mechanism



the most appropriate nodes are chosen until the physical nodes are reached. In these nodes the installation step loads the service. The final advertisement step mimics the summarization step, albeit only involving nodes that have been chosen to run the service.

This procedure can also be viewed as two successive query-reply processes taking place over the entire network, the queries going down the hierarchy and the replies going upwards, in which the nodes at the intermediate layers actively process the queries and replies. The first query-reply pair is used to extract the information from the network on how it can support a certain service (solicitation and summarization steps), whereas the next query-reply pair is used to actually install the service in the manner most appropriate (dissemination, installation, and advertisement steps).

Hierarchical iterative gather-compute-scatter algorithm. The procedure described in the previous subsection with its five steps can be divided into a sequence of iterations, each consisting of gather, compute, and scatter phases, distributed over the service-deployment hierarchy, hence the name HIGCS (hierarchical iterative, gather-compute-scatter). HIGCS uses an approach similar to the one in Reference 17, albeit with specific enhancements. A possible implementation of such a mechanism could involve mobile agents whose navigation model is extracted from the structure of the hierarchy. We

first describe the formal model and then how it is used to express the service-deployment procedure.

Service-deployment HIGCS messages are exchanged at each iteration, following the tree-like topology of the underlying hierarchy. The sets of destinations and origins relevant for the messages exchanged in the scatter and gather phases, respectively, are obtained from the compute phase. The logical node of a group is merely responsible for communicating with its underlying peer-group members of the scatter set. The logical node has neither to monitor all members of its group nor perform all computations centrally. For ease of description, we assume in the following discussion that the scatter set is always determined by a central computation performed by the logical node rather than distributing this computation among group members.

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, as illustrated in Figure 7. 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 that in Reference 17, 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 network-level service-deployment mechanism is defined in Table 2.

To perform service deployment, iterations are associated with the steps as described in Figure 6. 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, the service is installed).
- C_3 summarizes the results from the deployment on the set of nodes G_3 (on a physical node, this information is obtained locally) and places it in *iMetric*.

Figure 7 The HIGCS agent operation model

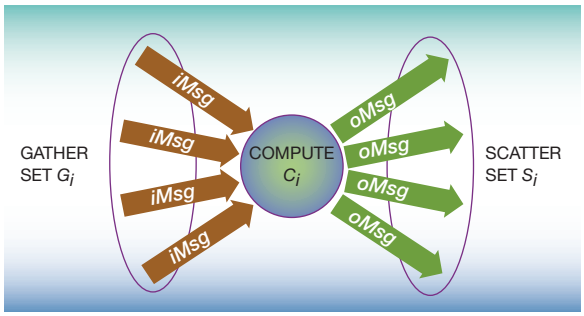


Table 2 The HIGCS service-deployment message format

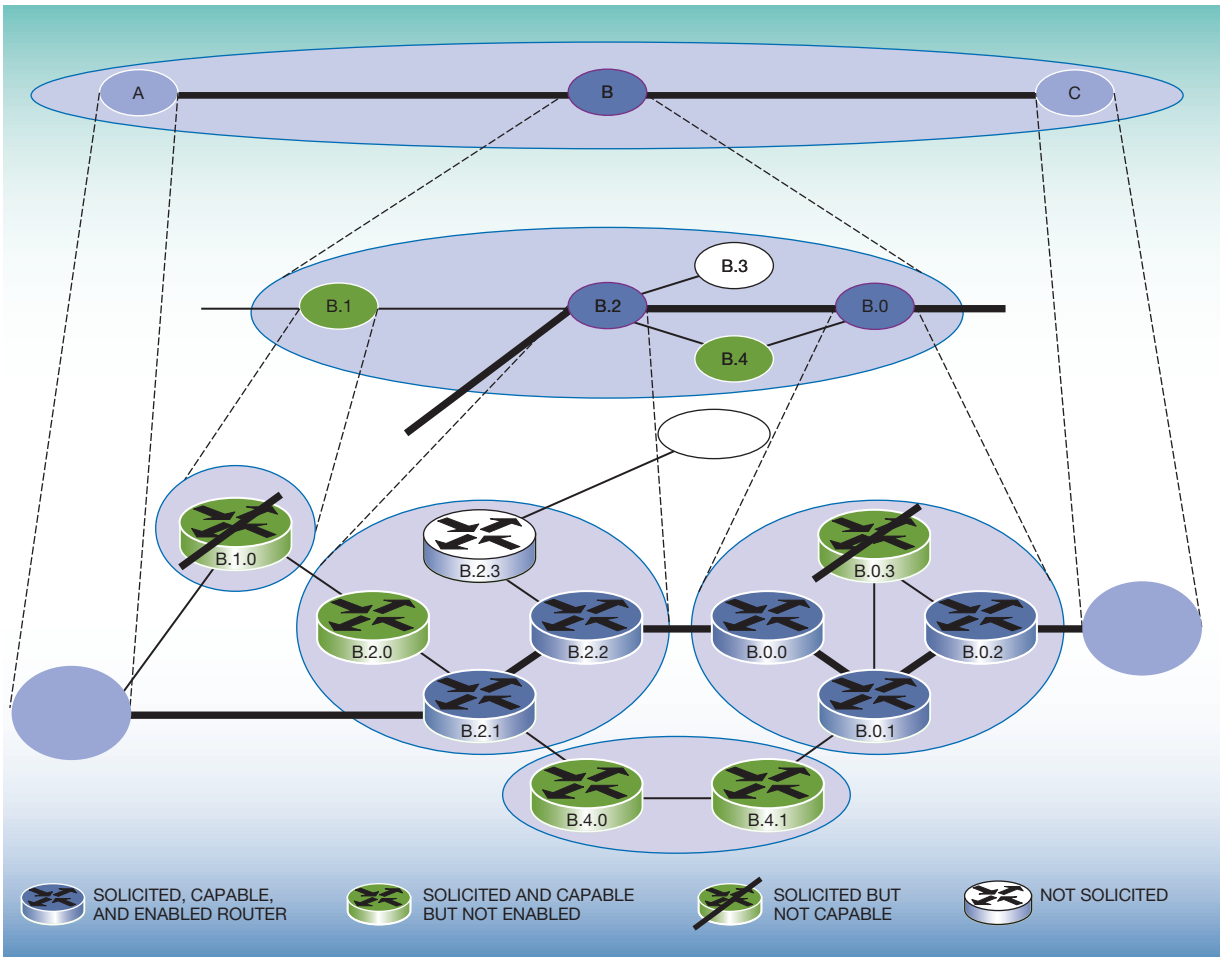
Parameter	Generic Value
serviceId	Instance of service deployed
hierarchyId	Identifier of service hierarchy
srcId	Source of message
destId	Destination
iterationId	Current iteration
G_i	Gather set for iteration i
C_i	Compute function for iteration i
S_i	Scatter set for iteration i
servSpec	Service specification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

The actions executed by nodes in the hierarchy during the first query-reply pair, namely, the solicitation and summarization steps, are expressed in the C_0 and C_1 compute phases, whereas C_2 and C_3 correspond to the second query-reply pair, namely, the dissemination, installation, and advertisement steps.

Clearly, these C_i functions will be implemented differently for each service to be deployed, as we describe next. More detailed examples of the HIGCS iterations and C_i functions for each category of service deployment are given in Reference 18.

Path-based deployment. Table 3 shows the computations executed for the deployment of a hypothetical path-based service on all nodes of a path between two customer sites (represented by the A and C top-level nodes in Figure 8). For the sake of simplicity, the straightforward update of other fields in the

Figure 8 Result of the HIGCS computations



oMsg (such as *srcId*) and fields that do not change from the *iMsg* message are not shown.

Figure 8 illustrates the result after all four HIGCS iterations have completed across the entire hierarchy. The path selected is shown as a thicker line. The first iteration was initiated by node B, which received the initial HIGCS message with *iMsg.ends* set to {A, C}. Logical node B.3 and router B.2.3 did not participate in the service deployment, since they are not on a path between A and C. All other nodes had been solicited, and nodes B.0.3 and B.1.0 were excluded, since their capabilities did not match the requirements of that service. Of the remaining routers, B.0.0, B.0.1, B.0.2, B.2.1, and B.2.2 are finally enabled with the service, since they are on the shortest path between A and C.

For path-based deployment, routing decisions have to be taken successively at each layer of the hierarchy (executed by SelectNodesOnShortestPath in Table 3). Among the various suitable data representations, transition matrices offer an accurate view of the cost of traversing a logical node.

Transition matrices used in path-based service deployment contain topology-, capability-, performance-, or cost-related information, or all of these types of information. For instance, the sMetric generated for node B in Figure 8 is the following transition matrix:

$$\mathcal{T}_B = \begin{pmatrix} 1 & \cdots & \\ 0 & 0 & \cdots \\ 5 & 0 & 1 \end{pmatrix}$$

Elements $t_{i,j}$ indicate that there is connectivity between border nodes B.i and B.j, with a path composed of nodes capable of running that particular service. In addition, the value of $t_{i,j}$ corresponds to the performance of that path in terms of number of hops. For instance, element $t_{0,2}$ in $\mathcal{T}_B$ with a value of 5 indicates that there is a path composed of capable nodes when crossing B from B.0 to B.2 (and vice versa). This path is shown with a thicker line at each hierarchy level in Figure 8, which consists of five nodes (or hops) at the bottom hierarchy layer.

Straight-path search. In path-based deployment, it is only necessary to solicit those nodes that form a path between the endpoints specified. Nodes that lie in stub networks, as, for instance, B.3 in the example of Figure 8, do not have to be solicited. In addition, assuming that node capabilities are available in the same way irrespective of the input and output port on a node (this normally applies to physical nodes), it is possible to further constrain the set of possible paths as follows: Assume S is the set of all possible paths without loops between two endpoints. S_s contains all paths from S for which no subpath exists in S. (A path P_s is defined as a subpath of another path P if by removing one or more nodes from the list of nodes that form P we obtain the list of nodes that form P_s.) We define such paths in S_s as *straight paths*. Clearly, the shortest path is a straight path. Note also that straight paths are not transitive. Appending a straight path between nodes b and c, such as {b, c}, to a straight path between a and b, such as {a, e, b}, does not necessarily lead to a straight path between a and c, as {a, e, c} is a subpath of {a, e, b, c}, as shown in Figure 9. In this example, the only straight paths between a and c are {a, d, b, c} and {a, e, c}.

Here, we present a novel algorithm that searches for straight paths (SPS) and performs similarly to the Depth-First Search (complexity of $O(n^2)$, where n is the total number of nodes). The algorithm discovers all nodes that lie on a straight path between two endpoints *source* and *dest* in a graph G composed of nodes $V[G]$, where $Adj[u]$ is the set of nodes adjacent to node u.

```
SPS(source,dest)
 1  for each u ∈ V[G] do
 2     color[u] ← WHITE
 3     marked[u] ← FALSE
 4  SPS-Visit(source,dest)
```

Figure 9 Example network for SPS

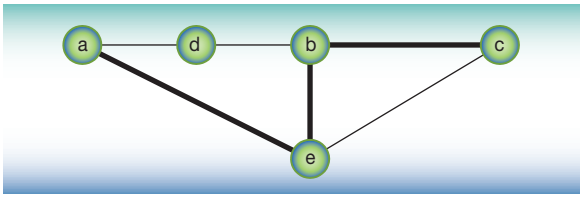


Table 3 C_0, C_1, C_2, and C_3 functions

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

```
SPS-Visit(u,dest)
 1  localresult ← 0
 2  color[u] ← GRAY
 3  for each v ∈ Adj[u] do
 4     if v = dest then
 5        localresult ← 2
 6  if localresult < 2 then
 7     for each v ∈ Adj[u] do
 8        if color[v] = WHITE then
 9           if NumGrayNeighbors(v) < 2 then
10              localresult ← max(SPS-Visit(v,dest),
                   localresult)
11           else localresult ← max(localresult,1)
12  if localresult = 2 then
13     marked[u] ← TRUE
```

```
14  color[u] ← WHITE
15  return(localresult)

NumGrayNeighbors(u)
 1  result ← 0
 2  for each v ∈ Adj[u] do
 3     if color[v] = GRAY then
 4        result ← result + 1
 5  return (result)
```

In the initialization phase of the algorithm (lines 1–3 of SPS( )), all nodes in the graph are marked as FALSE and colored WHITE. Nodes that lie on a straight path between *source* and *dest* will be marked by the SPS algorithm. The color of each node can alternate between WHITE and GRAY as nodes are being visited. Note that nodes may be visited multiple times on different paths.

The principle of the algorithm is to grow all possible paths in the graph recursively and temporarily color nodes on such paths gray, until one of the two following conditions applies: the path either reaches the destination, or the path reaches a node that is adjacent to a gray node (therefore already in the same path). This is the case if NumGrayNeighbors(v) is equal to 2, on line 9 of SPS-*Visit*( ). If the first condition applies, then all nodes on this path are marked as belonging to a straight path (line 13 of SPS-*Visit*( )). In both cases, the color reverts from gray to white as the algorithm continues and recursively searches for other paths. It stops once all possible paths have been visited.

As a brief proof that all nodes on straight paths and only those nodes are indeed marked by this algorithm, we use the following property: If any subpath of some path is not a straight path, then the path itself is not a straight path either. Note that a node can belong to many of the paths in S simultaneously, including paths in the subset S_s, but only those nodes that belong to at least one path of S_s will be marked. By construction, the algorithm extends all paths from *source* to *dest*, except those paths it abandons at some intermediate node because the subpath from *source* to that intermediate node is not straight (and hence, according to the property above, any extension of that subpath to *dest* would not lead to a straight path). Therefore, all paths that have been successfully extended to *dest* comprise the set S_s of all possible straight paths between *source* and *dest*. By construction, the algorithm marks only those nodes—but all of them—that belong to each such path in S_s, thereby marking certain nodes possibly more than once. As a result, all nodes on straight paths are marked at least once by the algorithm.

A performance improvement[19] to the algorithm consists in reusing the marking obtained at each node. Because straight paths are not transitive, this might lead to false markings; therefore, more nodes than necessary might be solicited.

Path- and node-based deployment. 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 (IETF Resource Reservation Protocol, or RSVP) nodes between those endpoints.

To accommodate both path- and node-based characteristics, we choose to extend the transition matrix presented in the previous subsection with the necessary node information. The extended transition matrix is defined as follows:

$$\mathcal{T}_{\mathrm{N}} = (\mathcal{M}; \mathcal{P})$$

Elements $m_{i,j}$ indicate the shortest number of RSVP-capable hops between border nodes N.i and N.j in node N. 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, which are represented by logical nodes.

Figure 10 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:

$$\mathcal{T}_{\mathrm{A.1}} = \left(\begin{array}{cccc|c} 1 & \ldots & & & 1 \\ 0 & 0 & \ldots & & 0 \\ 4 & 0 & 1 & \ldots & 3 \\ 2 & 0 & 3 & 1 & 1 \end{array}\right)$$

In $\mathcal{T}_{\mathrm{A.1}}$, element $m_{2,0}$ indicates that the number of hops to cross A.1 from A.1.2 to A.1.0 is four. Element $p_{2,0}$ indicates that the number of hops to reach a VPN-capable endpoint in A.1 entering from A.1.2 with a path composed of RSVP-capable nodes only is three. There are actually two such paths, namely, (A.1.2, A.1.4, A.1.3) and (A.1.2, A.1.6, A.1.3).

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

Hierarchical Steiner-tree construction. When constructing a VPN, it might be relevant to minimize the total cost of interconnecting the VPN endpoints. The Minimum Steiner Tree is the minimum-cost tree that interconnects such endpoints, and is an NP-complete problem in the case of additive edge weights. Several algorithms approximate the optimum solution, such as Selective Closest Terminal First (SCTF[20]), which finds a solution guaranteed to be at most twice as expensive as the optimum tree. Here, we address how Steiner trees can be built over hierarchical networks.

During the summarization step, as the locations of the branching points of the Steiner tree are not yet known, only the shortest paths between all pairs of nodes at that hierarchy level are computed and stored as transition matrices in the sMetric. It can be shown that extending each transition matrix with the costs of all Steiner trees that can be constructed between all combinations of three or more border nodes would incur a large amount of additional transported information and computation effort in the summarization step. In addition, the SCTF algorithm cannot operate with the resulting interdependent edge costs.

Based on this information, a Steiner tree is computed successively at each hierarchy level during the subsequent dissemination step, for instance, using the SCTF algorithm. The cost used in a computation at a given level is an upper bound of the actual cost of the Steiner tree that can be built at the level below. This can lead to suboptimum decisions being made, as shown in Figure 11. Here, nodes A.1 and A.2 have identical transition matrices, and therefore the algorithm will select one of these two nodes indistinctively to become the branching point for the Steiner tree in node A. The cost of this tree, indicated in Figure 11 by a thicker line in node A, is forecast to be seven, which is the upper bound given the cost of four for traversing node A.2 according to its transition matrix. While the algorithm repeats in node A.2, it computes a local Steiner tree of cost three, leading to a final cost of six for the overall Steiner tree, instead of the upper bound of seven that would have been reached if node A.1 had been selected instead of A.2.

Figure 10 A group of nodes solicited for the VPN service

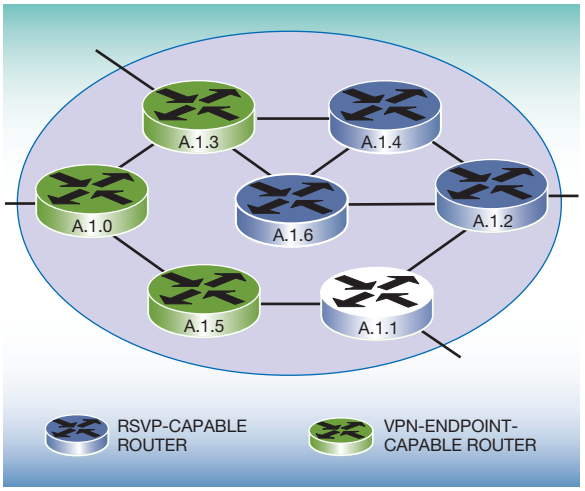
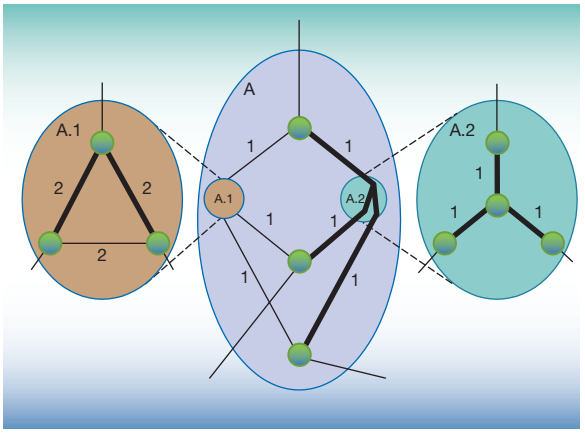
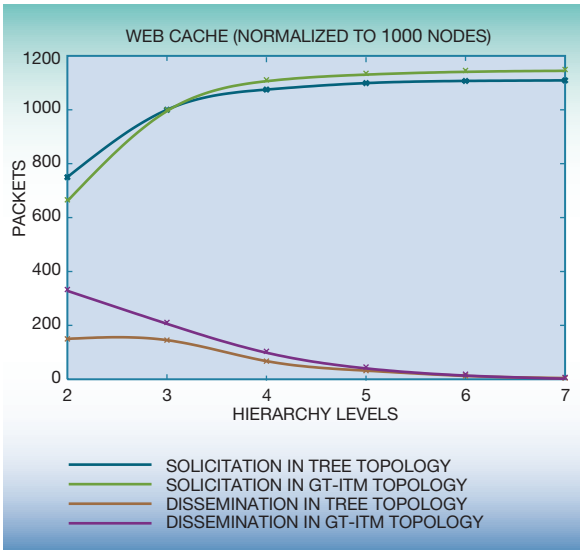


Figure 11 Hierarchical construction of a Steiner tree



The information-aggregation method used by transition matrices is designed so that the shortest path can be accurately computed over a hierarchical network as presented earlier, but it lacks information that would allow the construction of better Steiner trees as described above. This method illustrates the necessary trade-off between acceptable quality and amount of information that has to be found for each

Figure 12 Total HIGCS messages for Web-cache deployment



service to be deployed, hence the complexity of the processing it entails.

Simulation results

The elements and algorithms of network-level service deployment presented in the previous section have been implemented in a simulation environment to evaluate their scalability over large networks. The HIGCS agent-based deployment protocol has been implemented as an extension to the discrete-event network simulator ns-2. We present here some simulation results obtained with various fixed network topologies and for two types of services:

- A Web-cache node-based service, for which the Web cache is deployed in the most appropriate physical node in each lowest-layer peer group (B.0, B.1, B.2, B.3, and B.4, in Figure 8)
- A three-endpoint diffserv path-based service, using the directed broadcast mechanism shown earlier for the solicitation step, the Floyd-Warshall all-pairs shortest-path algorithm for the summarization step, and the Steiner-tree computation shown previously for the dissemination step.

Whereas typical Internet topologies consist of three routing-hierarchy levels, we have simulated the service-deployment protocol over topologies ranging from a flat two-level up to a highly hierarchical seven-level topology, as produced by the GT-ITM (Georgia Tech Internetwork Topology Models) topology modeler. The average size of peer groups is fixed to three nodes at every level so as to obtain comparable results for the various topologies, and also to keep the maximum number of nodes ($\sum_{n=0}^{6} 3^n = 1093$) within the simulator capabilities. For each discrete number of hierarchy levels, ten different topologies are generated using a random placement of nodes on a grid, and connectivity is determined with the Waxman probabilistic method.[21] Random numbers are generated using the Stanford GraphBase pseudo-random-number generator.[22]

As a benchmarking topology, a tree-network topology that ideally fits the hierarchical structure of HIGCS is also simulated. In this topology, HIGCS messages that travel between logical nodes only go over a single physical hop.

Interesting measures of the overhead incurred by the service-deployment protocol are the total number of messages exchanged during the five deployment steps and the total compute time spent in all nodes participating in the procedure.

Figure 12 shows the total number of HIGCS messages sent over the topology (normalized to a 1000-node topology) for the deployment of the Web-cache service. The standard deviation is zero for the tree topology (number of packets is deterministic) and very small for the randomly generated GT-ITM topologies. Note that despite the arbitrary logical-node selection policy applied in the simulations, the GT-ITM and the tree topologies show close to identical results. The more hierarchy levels, the more solicitation packets are transmitted, since every logical or physical node receives such a packet, but this increase becomes less and less important. During the dissemination step, the overhead is negligible because only one node in each lowest-level peer group has to be reached.

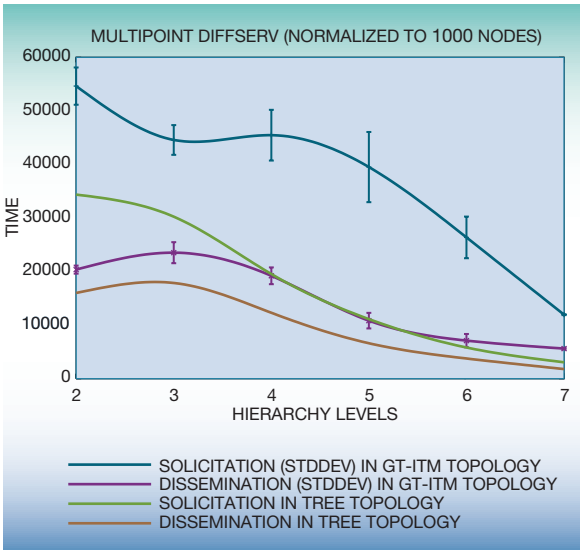
Figure 13 shows the total compute time (defined as a relative measure on normalized topologies) for the deployment of a three-endpoint diffserv service. Standard deviation is shown only for the ten randomly generated GT-ITM topologies for each number of hierarchy levels, since the measurement variations in the fixed-tree topologies are negligible.

The *divide and conquer* method takes effect as it can be seen that compute time decreases noticeably when

more hierarchical levels are introduced, because in turn the set of nodes that each logical node is responsible for decreases.

These results confirm the influence of the number of levels in the deployment hierarchy. The total compute time, shown here for the solicitation and dissemination steps only, decreases when more hierarchy levels are introduced in a given topology, whereas only a minor increase in message overhead is incurred. Hierarchical network-level deployment based on the suitable algorithms for the service-deployment categories presented therefore allows an efficient and scalable allocation of services in a network.

Figure 13 Total HIGCS compute time for diffserv deployment



Conclusion

Organizing the autonomic deployment of services is a key step toward an intelligent network infrastructure. Openness and programmability are starting to appear in network equipment, and it is therefore necessary to provide generic mechanisms that can enable the deployment of any type of service.

In this paper, we have introduced a two-phase mechanism that achieves an efficient and flexible deployment of services in networks: a macro-level deployment phase that operates in a hierarchically distributed manner to query and collect capabilities of nodes in the network, and then execute the deployment itself; and a micro-level deployment phase that refines the actual installation of a service according to the specific capabilities of each element comprising the network node.

In addition, we have introduced various categories of services according to topological service-deployment needs and proposed novel information aggregation methods, as well as a straight-paths graph algorithm used during the macro-level deployment phase that allows the number of nodes queried to be limited. The trade-off between accuracy, amount of information, and availability of efficient algorithms required for near-optimum deployment was illustrated using the hierarchical Steiner-tree computation.

By making the network itself responsible for executing the deployment of a service, an autonomic deployment is achieved, avoiding time-consuming and error-prone manual operations. As networks grow in size and heterogeneity, the deployment mechanism 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, as confirmed by our simulations.

Cited references

1. *Power Network Processor Software Overview*, Technical Report, IBM Corporation (April 2002); see http://www.chips.ibm.com/techlib.
2. D. Verma, M. Beigi, and R. Jennings, "Policy-Based SLA Management in Enterprise Networks," *Proceedings of the Workshop on Policies for Distributed Systems and Networks (POLICY 2001)*, Bristol, UK (2001), *Lecture Notes on Computer Science*, Vol. 1995, Springer, Berlin (2001), pp. 137–152.
3. K. Nahrstedt, D. Wichadakul, and D. Xu, "Distributed QoS Compilation and Runtime Instantiation," *Proceedings of IEEE/IFIP 8th International Workshop on Quality of Service (IWQoS'2000)*, Pittsburgh, PA (June 5–7, 2000).
4. A. Nakao, L. Peterson, and A. Bavier, "Constructing End-to-End Paths for Playing Media Objects," *Proceedings of OpenArch 2001—The 4th IEEE Conference on Open Architectures and Network Programming*, Anchorage, AK (March 2001).
5. S. Choi, J. Turner, and T. Wolf, "Configuring Sessions in Programmable Networks," *Proceedings of INFOCOM 2001*, Anchorage, AK (April 2001), pp. 60–66.
6. D. Wetherall, U. Legedza, and J. Gutta, "Introducing New Internet Services: Why and How," *IEEE Network: The Magazine of Global Information Exchange* **12**, No. 3, 12–19 (1998).
7. A. Doria, F. Hellstrand, K. Sundell, and T. Worster, *General Switch Management Protocol V3*, IETF RFC 3292 (June 2002); see http://www.ietf.org.
8. ForCES, *Requirements for Separation of IP Control and For-*

warding, IETF draft <draft-ietf-forces-requirements-05.txt> (June 2002).
9. N. Greene, M. Ramalho, and B. Rosen, *Media Gateway Control Protocol Architecture and Requirements*, IETF RFC 2805 (April 2000); see http://www.ietf.org.
10. J. Biswas et al., *Application Programming Interfaces for Networks*, Technical Report, IEEE PIN1520 Working Group (2000); see http://www.ieee-pin.org.
11. Software Working Group, Network Processing Forum, Fremont, CA (2002), at www.npforum.org.
12. R. Haas, P. Droz, and B. Stiller, "Cost- and Quality-of-Service-Aware Network-Service Deployment," *Proceedings of Advanced Internet Charging and QoS Technology (ICQT 2001)*, Vienna, Austria (September 2001), pp. 166–171.
13. P. Grillo and S. Waldbusser, *Host Resources MIB*, IETF RFC 1514 (September 1993); see http://www.ietf.org.
14. F. Baker, K. Chan, and A. Smith, *Management Information Base for the Differentiated Services Architecture*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 3289 (May 2002); see http://www.ietf.org.
15. *Private Network-Network Interface Specification Version 1.0 (PNNI V1.0)*, af-pnni-0055.000, the ATM Forum (March 1996); see http://www.atmforum.com/pages/aboutatmtech/approved.html.
16. R. Haas, P. Droz, and B. Stiller, "A Hierarchical Mechanism for the Scalable Deployment of Services over Large Programmable and Heterogeneous Networks," *Proceedings of the IEEE International Conference on Communications (ICC 2001)*, Helsinki, Finland (June 11–15, 2001), pp. 2074–2078.
17. Y. Chae, S. Merugu, E. Zegura, and S. Bhattacharjee, "Exposing the Network: Support for Topology-Sensitive Applications," *Proceedings of OpenArch 2000—The 3rd IEEE Conference on Open Architectures and Network Programming*, Tel Aviv, Israel (March 26–27, 2000).
18. R. Haas, P. Droz, and B. Stiller, "Distributed Service Deployment over Programmable Networks," *Proceedings of the 12th International Workshop on Distributed Systems: Operations & Management (DSOM'01)*, Nancy, France (October 2001), pp. 113–128.
19. A. Kumar and R. Haas, *Design and Implementation of a Distributed-Agent-Based Simulation for Hierarchical Service-Deployment*, Technical Report RZ 3378, IBM Corporation, Zurich Research Laboratory, Rüschlikon, Switzerland (2001).
20. S. Ramanathan, "Multicast Tree Generation in Networks with Asymmetric Links," *IEEE/ACM Transactions on Networking* **4**, No. 4, 558–568 (1996).
21. E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proceedings of INFOCOM 1996*, Vol. 2 (March 1996), pp. 594–602.
22. D. E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison-Wesley Publishing Co., Reading, MA (1994).

Accepted for publication August 9, 2002.

Robert Haas *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (electronic mail: rha@zurich.ibm.com).* Mr. Haas received the M.S. degree in communication systems from the Swiss Federal Institute of Technology (EPFL), Lausanne, and the Eurecom Institute, Sophia-Antipolis, France, in 1996. He also received the D.E.A. degree in distributed systems from the University of Nice Sophia-Antipolis. He joined the IBM Thomas J. Watson Research Center in 1996 as a research staff member, designing and prototyping a layer-3 switch. In 1998 he joined the Zurich Research Laboratory, and he is currently studying for a Ph.D. degree with the Swiss Federal Institute of Technology (ETHZ), Zurich. His research interests include network protocols and architecture, specifically auto-configurable and programmable networks.

Patrick Droz *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (electronic mail: dro@zurich.ibm.com).* Dr. Droz received an M.S. degree in computer science from the Swiss Federal Institute of Technology (ETHZ), Zurich, in 1992. He then joined the Zurich Research Laboratory and worked in the ATM Networking Group on the design and implementation of the ATM control point for the 8260 campus backbone hub, and completed his Ph.D. degree on "Traffic Estimation and Resource Allocation in ATM Networks" in 1996. He is now manager of the Network Processor Software group. He is cochair of the IETF ForCES working group. His current research activities focus on software enablement for network processors.

Burkhard Stiller *Information Systems Laboratory IIS, University of Federal Armed Forces Munich, D-85579 Neubiberg, Germany, and also Computer Engineering and Networks Laboratory TIK, ETH Zurich, CH-8092 Zurich, Switzerland (electronic mail: stiller@informatik.unibw-muenchen.de).* Prof. Dr. Stiller received an M.S. degree in computer science and a Ph.D. degree from the University of Karlsruhe, Germany, in 1990 and 1994, respectively. He worked there as a research assistant at the Institute of Telematics until 1995, and was a visiting scientist at the University of California, Irvine, California, in 1992, and at the University of Cambridge, Computer Laboratory, in the United Kingdom in 1994/1995 under a European Community Research Fellowship. From 1995 until 1999 he was with the Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETHZ), Zurich, as a research associate and lecturer for multimedia communications. Since August 1999 he has been an assistant professor for communication systems at ETHZ. In addition, he was appointed full professor for computer science at the University of Federal Armed Forces, Munich, Germany, in April 2002.