Journal of Network and Computer Applications 32 (2009) 461–473

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Towards automating overlay network management

I. Al-Oqily *, A. Karmouch

School of Information Technology & Engineering (SITE), University of Ottawa, PO Box 450, Ottawa, ON, Canada K1N 6N5

ARTICLE INFO

Article history:
Accepted 15 February 2008

Keywords:
Adaptive SSONs
Network management
Overlay networks
Policies

ABSTRACT

Overlay networks are becoming widely used for content delivery because they provide effective and reliable services that are not otherwise available. However, they can negatively affect each other as well as the underlying network. A management system that controls and adapts their behavior is therefore needed. This will meet not only the specific demands of the users but also those of the network and service providers. This paper presents a novel approach to the issue of automating overlay network management. In contrast to existing management approaches which require static a priori policy configurations, policies are created dynamically. A policy layer consists of a set of policy enforcement points and policy decision points. This is used to capture the goals of users, services, and networks into network-level objectives. The behavior of the overlay network is adapted to the changing conditions in its environment. The creation, adaptation, and termination of overlays are achieved through policies. Policies are generated and enforced on the fly from the context information of the user, the network and the service provider. The new approach provides users and applications with more flexibility to dynamically change their quality-of-service requirements while maintaining smooth quality-of-service delivery. We show the advantages of our architecture and provide simulation results to verify its effectiveness.

1. Introduction

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network in order to implement a service not originally available. Overlays can be used to increase routing robustness, to increase security, to reduce duplicate messages and to provide new services for mobile users. They can also be incrementally deployed on end hosts without the involvement of ISPs, and they do not need new equipment or modifications to existing software or protocols (Andersen et al., 2001; Jannotti et al., 2000; Subramanian et al., 2004). These attractive benefits come at the cost of increased overhead and complexity. Overhead is increased because of the additional packet header and the redundant work at the overlay and IP layers. The constantly increasing traffic carried by the overlays also tends to overload the network and consume its resources (Sripanidkulchai et al., 2004). In addition, overlays are usually designed independently. This increases the chances that they will negatively affect each other; bottlenecks are created that reduce the performance both of the overlays and of the underlying network. Overlays therefore need to incorporate a management mechanism that reduces this complexity and keeps them operating correctly.

Overlay management is challenging for several reasons. First, the dynamic changes in network conditions and topology quickly renders management information obsolete. For example, network nodes may fail, links may get congested and routing information may change. In addition, any changes in the routing path are affected by the required QoS (Shin et al., 2001), bandwidth, latency, and the existence of other overlays. Second, overlay members are dynamic as new users may join or leave. Finally, each overlay node has limited knowledge of the network, and the knowledge varies among overlay members. With a large number of overlays, management by traditional methods becomes harder to achieve and a new management scheme must be supplied. This new scheme must account for the different phases that overlays go through during their lifetime: creation, optimization, adaptation and termination.

Creation requires the setup of a routing table in each overlay node along the end-to-end path, a path that must optimize the QoS metrics. Adaptation produces a new behavior that reflects a change in the overlay environment. Adaptation may be necessary to assist

* Corresponding author. Tel.: +1 613 562 5800x6203; fax: +1 613 562 5175.
E-mail addresses: ialoqily@site.uottawa.ca (I. Al-Oqily), karmouch@site.uottawa.ca (A. Karmouch).

1084-8045/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2008.02.013

mobility, to deal with the failure of an overlay node, or to control congestion. Termination means claiming the reserved resources and updating routing tables.

The use of policies offers an appropriately flexible and customizable management solution that allows network entities to be configured on the fly (Yang et al., 2002; Damianou et al., 2001). Usually, administrators define a set of rules to control the behavior of network entities. These rules are translated into component-specific policies that are stored in a policy repository, to be retrieved and enforced as needed. However, existing management systems usually direct the management task to physical entities such as routers, switches, and gateways. In our architecture, the task is assigned to the overlays and their logical elements. Policies are generated dynamically, and no human interaction is required.

This paper proposes a new approach to the autonomous, context-aware, policy-based management of overlay networks. The approach novelty lies in that sets of policies, specifically adapted to the current availability of resources and users' demands, are dynamically generated from the available context information and enforced on the fly. Policies also control the various construction phases harmoniously. Our goal is to automate overlay management in a dynamic manner that preserves the flexibility and benefits that overlays provide.

The remainder of this paper is organized as follows. Related work and existing management approaches are briefly discussed in Section 2. Section 3 presents an overview of the proposed architecture and Section 4 describes it in detail. Section 5 discusses simulation results and evaluates performance. Section 6 concludes the paper with indications for future work.

2. Related work

To the best of our knowledge, no similar work is published in the literature at the time of writing. But overlay networks have been proposed as a way to increase the scalability of distributed systems. Frameworks that have been developed for this purpose fall mainly into one of two configurations: static and automatic. They can be further classified into specific and generic application overlay networks. Although an automatic configuration seems to provide a more promising solution, existing techniques have limitations. They usually lack the flexibility that is essential to accommodate the specific demands of individual overlay services.

Policy-based management has also been introduced as a solution. However, static policy configurations built a priori into network elements may not be sufficient to handle changes. Research trends use policies in different ways. In Ferdinando et al. (2003), policies are used to control the topology growth of peer-to-peer systems. Policies are distributed to all hosts in the system with each host able to adopt only one policy at a time. But human interaction is still required to define the policies and to inject them into the system. In Massimi and Wolz (2003), peer-to-peer concepts are used for wearable mobile devices to protect users from one another. A policy manager stores and forwards policies. Unfortunately, the policies are static and built a priori.

Specific application overlay networks have been designed and tailored to a specific application. In Agarwalt et al. (2003), for example, overlays are used to achieve fast fail-over and traffic load balancing in the border gateway protocol (BGP). A set of policy agents installed in each participating autonomous system enforces necessary changes in the local BGP. The policy agents communicate through the overlay. Our work differs from this approach in that it allows (a) a number of overlays to be managed at the same time and (b) policies to be generated dynamically from the context information.

In generic overlay networks, knowledge is shared through an intermediate layer that measures a number of network properties. In Nakao et al. (2003), an underlay with a multi-tier overlay routing scheme is proposed. AS-level Internet topology and routing information is acquired by a topology-probing kernel from nearby BGP routers. A more generic approach is described in Shen (2004). A number of quality metrics (such as low latency, low hop count and high bandwidth) are acquired from end-to-end network measurements and used to construct overlays. Although generic overlay networks are efficient in reducing the cost of acquiring the shared knowledge, they lack the flexibility to support specific application overlay networks. Moreover, they do not take into account specific demands for individual services such as user or terminal mobility. More importantly, they do not explicitly address the use of policies to configure overlays dynamically. Our work does so and, in addition, addresses the use of intelligent network side functions in the overlay path that permit additional services to be deployed.

3. Proposed architecture

Our work is in part inspired by SMART (Schmid et al., 2005). SMART was developed to optimize media delivery services. It does so by moving the control and resource allocation to the network that has more knowledge about the topology and network properties. We begin with a brief description of SMART architecture in order to present our architecture as an extension.

3.1. SMART modeling for overlay networks

As part of the Ambient Networks Integrated Project (Niebert et al., 2004), SMART supports a variety of content distribution models: peer-to-peer, content-distributed networks, unicast and multicast. A service-specific overlay network (SSON) is constructed for each media delivery service or group of services. An SSON is a virtual network composed of a set of overlay nodes and links. This customizes the network to the particular requirements of the service (such as QoS, media formats, responsiveness, cost). SSONs have the ability to transparently include network side entities called MediaPorts (MPs) in the communication path, thereby providing the flexibility to modify the content and services such as caching, adaptation and synchronization (Ooi et al., 2000).

Overlay nodes are physical ambient network nodes that have the capabilities needed for them to become part of the SSONs. These are a control plan and a user plan. The control plan is responsible for the creation, routing, adaptation, and termination of SSONs. The user plan contains the overlay support layer that receives packets from the network, sends them to the network, and forwards them on the overlay. Overlay nodes implement a sink (MediaClient or MC), a source (MediaServer or MS) or a MP in any combination. MPs are special side components that provide valuable functions to media sessions such as special routing capabilities, smart caching and adaptation.

MPs, MCs and MSs are managed by the control plan. The control plan also contains a MP directory service (MPDS) to maintain information about the available MPs, such as location, load and cost.

SMART architecture is described in detail in Hartung et al. (2004). But SMART does not specify the means by which SSONs are constructed and managed, nor does it specify how SSONs can be adapted dynamically according to the users' context. Our architecture addresses these drawbacks. First, the control plan is equipped with a new entity called the overlay policy enforcement point (OPEP). The OPEP is designed to control node resources and functionalities by enforcing configuration changes based on context information. This in turn is used to dynamically generate policies. Second, an SSON overlay policy decision point (OPDP) is used for each SSON or group of SSONs to make the appropriate decisions about the creation, adaptation and termination of SSONs. In addition, a set of system PDPs (SPDPs) is used to coordinate the actions of OPDPs. COPS protocol (Durha et al., 2000) is used to exchange policy objects between the OPEP, OPDP, and SPDP.

3.2. Architecture overview

A schematic description of the main components of our proposed architecture is shown in Fig. 1. The central components are the OPDP, SPDP, OPEP and the policy generator (PG). The OPEP is a component in the overlay nodes while the OPDP and the SPDP are remote entities that may reside at a policy server. The PG generates and adapts policies using the available context information. The OPEP is the point at which policy decisions are actually enforced. Policy decisions are made primarily at the PDP.[1] The PDP receives policies from the PG, evaluates them and distributes them appropriately. The OPEP requests decisions and enforces them. With any change in the context information, an adaptation process is triggered by first generating the policies that reflect the new context and then by proactively sending them to be dynamically enforced.

We distinguish between the sources of context information, such as user context, service provider context and network provider context. All these types of context must be considered when building a comprehensive management system. We assume that the context information has been gathered in a context memory (Bałos et al., 2006; Salo et al., 2007) that feeds it to the PG. The PG generates different types of policies: user policies, application policies, service provider policies, network provider policies and service-specific policies. Any change in the SSON environment triggers an adaptation process in which new policies are generated dynamically and sent to the appropriate PDP.

The policy repository saves all the policies generated for each SSON. It also contains other information relevant to the management task. This may include the SSONs constructed so far and the media port directory service (MPDS) that lists the available MPs and their capabilities, user registration, and accounting information.

There are two different types of PDP: the SSON PDP (OPDP) and the system PDP (SPDP). Since the number of overlay nodes expected in each SSON is small, each OPDP is assigned one or more SSONs. The OPDP is responsible for automating the task of creating, adapting, configuring, and terminating its designated SSONs. It communicates directly with the participating overlay nodes to achieve its tasks. Typically, its tasks are the following: (1) it makes configuration decisions in response to the system policies received and uses these decisions to configure the overlay nodes participating in a given SSON. (2) During construction of an SSON, it is responsible for optimizing the service path to meet the required QoS metrics of the high-level system policies as well as the context of the service. (3) It monitors the QoS metrics for the multimedia session and continuously adapts the service path to the changing conditions of the network, the service, and user preferences. (4) It also monitors the participating overlay nodes and finds alternatives in case any of the nodes do not conform to the required performance level. OPDPs receive goal policies from SPDPs in order to decide the types of actions required.

A single OPDP is able to automate the management functions only for the SSONs that it manages. If a network contains a large number of SSONs, it may be that they are not really isolated. On the one hand, each overlay node can be part of many SSONs if it offers more than one service or if it has enough resources to serve more than one session. On the other hand, the SSONs' service paths may overlap, resulting in two or more SSONs sharing the same physical or logical link. For example, if two SSONs share the same routing MP with the same goal to maximize throughput, the result will be race conditions on the shared resources. Therefore, in order to achieve a system-wide balance, the OPDPs need to coordinate their actions. This is achieved using SPDPs.

SPDPs interact with one or more OPDPs. They pass the high-level system policies, such as for load balancing, to the OPDPs. Whenever they find shared goals between two different SSONs, they send information that avoids conflicting actions. The OPDPs then contact each other and create a virtual management overlay (VMO) as illustrated in Fig. 2. This VMO coordinates their actions before they are passed to their overlay nodes.

Sharing goals is not the only reason to create VMOs. SSONs that share common links and SSONs that belong to the same policy domain (same service class, ISP, etc.) may also create VMOs among themselves. Additionally, SSONs that share common nodes or links affect each other's performance as they compete for the shared resources. This can result in degraded performance as the competition causes them to frequently evaluate their decisions in an attempt to reach the desired performance goals. All SSONs in a given domain (ISP) are also expected to achieve the domain-wide policies together. VMOs allow these policies to be sent and adapted to each SSON in a way that achieves the desired goals. VMOs also allow the sharing of control and information between different SSONs. A set of SSONs co-located in a given vicinity (such as an area, domain, AS) usually has independent routing decisions based on its observations for its environment. Sharing this information results in reduced overhead for each overlay and allows policies to be adapted and generated in order to achieve better performance.

When VMOs are created, each OPDP can obtain information of two types. The first is related to the coordination actions and the second is related to the common metrics in which each OPDP is interested. Goal policies are passed from SPDPs to the OPDPs they manage. The context information of the network, users, and services is used primarily to aid in generating suitable policies at each level.

[1] We use PDP to refer to both OPDP and SPDP unless it is necessary to make a distinction.

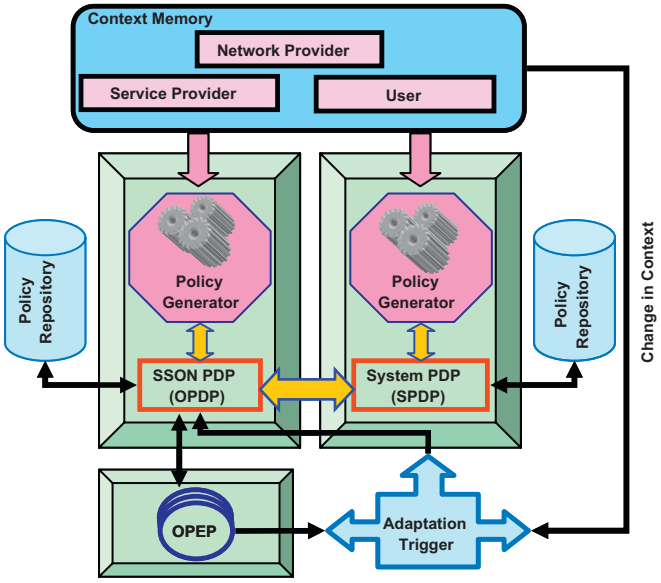

Fig. 1. Context-aware overlay policy architecture.

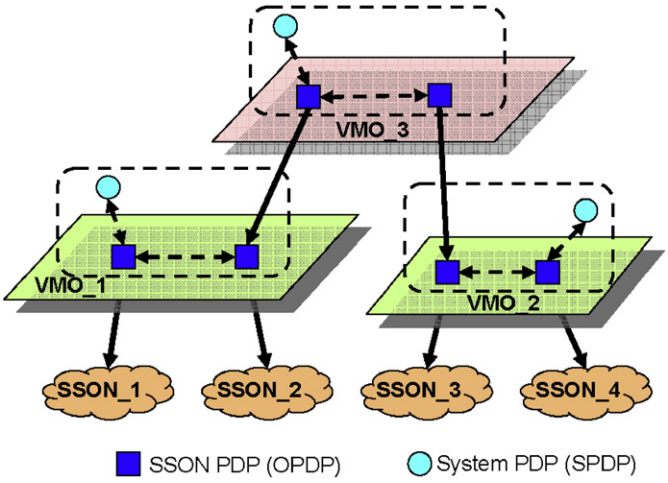

Fig. 2. Virtual management overlay (VMO) hierarchy. A VMO is created for SSONs sharing the same goals, links, nodes, and/or domain. Basically they are used to coordinate the management actions between OPDPs to prevent conflicting goals and racing conditions.

4. Proposed architecture details

This section describes in detail the central components of the proposed architecture and the various steps in the construction, adaptation and termination of SSONs.

4.1. Policy generator

The central component in the PG (Samaan and Karmouch, 2005) is the automated policy adaptor (APA). The key feature in APA design is to separate the mapping of abstract higher-level goals to network-level objectives from the functionality that adapts the behavior of network components. Although we used the same concepts as Samaan and Karmouch (2005) to generate policies, our PG goes beyond those concepts as follows. First, policies for system and business objectives are derived from the relevant context information, as are policies for users and applications. While in Samaan and Karmouch (2005), network administrators and users/applications specify these policies using a graphical user interface and register them with the APA, we derive these policies from the context, thereby further automating the process. Second, instead of sending the policies directly to the managed resources, we send them to the PDPs. This separates the process of generating policies from the process of making decisions. This is done because adapting an existing policy may not be sufficient to adapt an entire SSON. In addition, adapting one policy may require adaptations in other policies in order to achieve an SSON-wide adaptation. The separation of policy generation from decision making allows for more flexibility to adapt SSONs dynamically based on their specific requirements as units.

As a result, overlay management is seen as a process of learning from current system behavior by creating new policies at runtime in response to changing requirements. The PG generates and adapts five kinds of policies and sends them to PDPs. These are user policies,

network policies, application policies, service provider policies, and service-specific policies. The adaptation process is either triggered at pre-set intervals or by events received from the OPDP and network monitors in the OPEP. These events are in response to user-related or application-related events such as low battery level, or changes in a user's location or an application's QoS requirements.

The PG considers policy adaptation to be a process of learning from current system behavior. This learning process assembles new policies at runtime. The policy-making passes through three main phases: *stage setting*, *consideration of alternative decisions* and *reassessment of the applied decision*. As a final step, a feedback mechanism ensures that the new policies are correct.

In the first phase, all necessary information is obtained from the context information. In the second phase, the PG selects one or more actions from the action space that best attains the specified change. The selection is made by calculating an expected loss value for each action. Choosing the optimum policy is simply choosing the action that minimizes the expected loss values. The third phase involves the assembly of a new policy as a result of the actions selected in the previous step. The newly assembled policy consists of a triggering event translated by the PG from higher-level policies such as user location. Conditions are specified by the characteristics of the satisfied objective and the selected action. The new policy can also be associated with a lifetime, a duration after which it expires and is deleted. Once a policy is assembled, it is sent to the appropriate PDP.

The final step is performed by the reassessment module that evaluates the success of the new policy. Network monitors in OPEPs measure the average values for the actual QoS parameters. For example, an SSON's actual throughput of traffic is calculated and compared with the objective. The difference between the values measured by the monitors and the required objectives is fed back to the first stage as a new objective change. If the difference is substantial, the adaptation process is repeated.

4.2. Policy decision point

As shown in Fig. 3, the OPDP is composed of a management agent (OPDPMA), an overlay conflict resolution agent (OCRA), a plans generator, and a communication agent (CA).

The management agent receives the policies from the PG, analyses them, and makes appropriate decisions. It assigns a unique ID to each SSON and to each flow. Flows can then be routed independently when, for example, this is necessary to meet the required QoS. To create or adapt an SSON, the management agent sends the IDs, the SSON's performance requirements and the requested MP capabilities to the Plans Generator.

The Plans Generator is responsible for constructing the topology of the SSON that meets the requested performance properties (such as low overlay path latency or a specific overlay path bandwidth) and the requested MP capabilities (such as a caching MP with at least 300 Mb disk space). It searches for the path that best meets the QoS constraints. For this to be done, a set of QoS metrics has to be available. These metrics (such as link costs, delay, jitter and bandwidth) are either part of the context information available in the plan history or are obtainable by reusing measurement techniques similar to those presented in Li and Mohapatra (2004) and Vleeschauwer et al. (2006). These costs are updated using a link state update protocol that is outside the scope of this paper. To facilitate the process of finding the optimal path, and to allow the network to interactively contribute to successful media delivery, the Plans Generator includes the suitable MPs whenever necessary.

The MP location is chosen to be as close as possible to the shortest path between the source and the sink. The Plans Generator either chooses either the most suitable MP or any suitable MP and then searches for the optimal path from the source to the MP and from the MP to the sink. The former choice ensures that the MP is as close as possible to the shortest path but it does not guarantee that an MP will be found. To avoid this problem, we expand the search parameters and run the search again, accepting that this increases the time needed to find an optimal path and consumes more resources. The latter choice allows a parallel search for the optimal path and reduces the management overhead, but it does not guarantee that the MP closest to the shortest path will be found. Any MP can be chosen at random from the MPDS, in which SMART assumes that MPs register their locations and capabilities. But since geographically closer nodes are expected to have fewer hops between them (Lakhina et al., 2003; Melodia et al., 2005), it is more efficient to choose an MP that is geographically close to the sink. This is the approach taken in this paper.

Once the Plans Generator finds the optimal path, it constructs a connection matrix that represents the SSON topology and sends it to the management agent, and to the PG. The PG generates the policies that construct or adapt the SSON and sends them to the Management Agent.

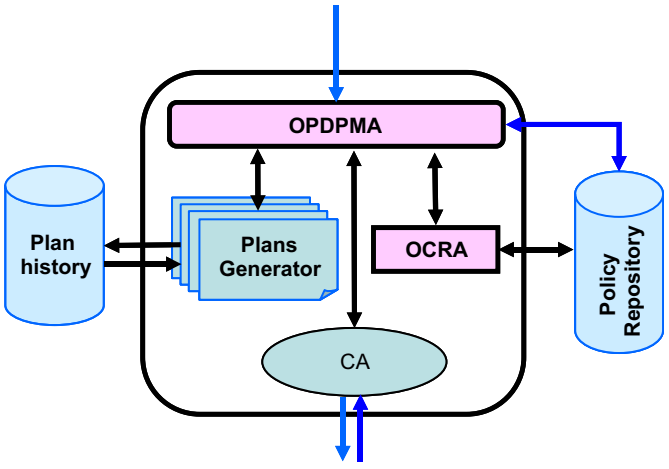

Fig. 3. OPDP architecture.

The OCRA (an object of future work) receives the policies from the management agent and checks them for any conflict with previous policies. This ensures that new or adapted SSONs do not negatively affect the operation of those already deployed. If a conflict is found, the policies are rejected and the SSON has to enter an adaptation process. Conflicts are generally divided into two types: static and dynamic. In our model, a static conflict is one that is detected at the time a new policy is generated. A dynamic conflict is one that occurs at runtime. If no conflicts are found, the management agent either sends the policies to the appropriate OPEP through the CA for immediate enforcement or retains them in the policy repository to be activated at a future time. The CA is responsible for sending policy objects to the appropriate OPEP as well as for receiving policy objects from OPEPs.

The SPDP's main tasks are to coordinate the actions of two or more OPDPs and to distribute system-level policies that guarantee system-wide performance. These policies are derived from the network and service provider context information. SPDPs consist of the same components as the OPDP except that they do not contain the Plans Generator module. They therefore receive the system policies such as load balancing from the PG, analyze them and send them to a conflict resolution module. This module checks the consistency of the new policies against those already installed. In case of conflicts, the new policies are fed back to the PG for re-adaptation. If no conflicts are found, the policies are sent to their OPDPs through the CA.

4.3. Overlay policy enforcement point

The OPEP is the point at which policies are actually enforced. As shown in Fig. 4, the OPEP is composed of four cooperating agents. It receives notifications or messages from the monitoring agents (MAs) that require a policy decision. It then constructs a request for a policy decision and sends it to the OPDP. Once the policy decision is received, the OPEP enforces the decision by accepting or denying as appropriate.

- *Overlay policy management agent* (*OPMA*): The OPMA manages the various aspects of the OPEP through two-way transmission of policy objects with the OPDP. It also receives notifications and adaptation events from the MAs. Once a new policy object is received, the OPMA analyzes it to determine the appropriate action. If it is a decision, it is sent to the policy enforcement agent (PEA) that decides how the policy will be enforced. If it is an adaptation, it constructs an appropriate policy object and communicates with the OPDP to acquire a decision.
- *Policy enforcement agent*: The PEA is responsible for enforcing or removing the overlay policies at the overlay node. It receives and analyzes policy decisions from the OPMA. With the assistance of the resource interface agent (RIA), it enforces them at the appropriate overlay node component. It also sends a report to the OPMA describing the success or failure of the enforcement.
- *Resource interface agent*: The RIA is an interface between the OPEP and the components of the overlay node. As such, it communicates with the appropriate component to enforce a policy. For example, it communicates with an overlay service layer (OSL) component to update a routing table entry and with the MP to reserve or free its resources.
- *Monitoring agents*: MAs are placed at various layers of the system as required. Each MA is responsible for monitoring its layer and reporting to the main monitoring agent in the OPEP. MAs are therefore able to monitor the available resources and capabilities, as well as the connectivity of the overlay node to neighboring nodes. MAs also monitor the performance of overlay nodes and MPs. Whenever reduced performance is detected, MAs send an adaptation event to the management agent so that additional resources can be freed.

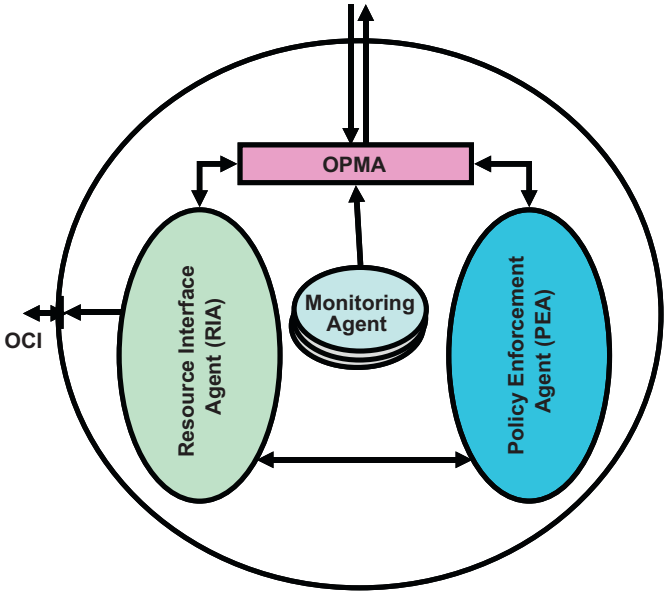

Fig. 4. OPEP architecture.

4.4. Use case scenario

This section provides a simple illustration of how our architecture uses policies to create, adapt, and terminate an SSON.

(1) *SSON construction*: The process of constructing an SSON starts with the service provider (or user) defining the properties of the service to be offered. These properties include the required QoS, the required network side functions (such as caching) and any other requirements specific to the service. The OPEP sends the properties to an SPDP that assigns the task to an OPDP and forwards the properties to the PG. The PG then converts them into policies and sends them to the OPDP. For example, the PG would generate policy (1) for a user requesting a video from a streaming video server:

```
If (User = "x") and (Application = "video") Then
   Max_Bandwidth < 128 kbps, Aggregate_Bandwidth < 1024 kbps
   OneWay  Delay < 200ms, Special  Functions = "caching", Priority = 3
```
(1)

Once the OPDPMA decides, with the help of the policy repository, that this is a new service, it assigns a unique SSON_ID to the service. If the service has multiple flows, it assigns a distinct FLOW_ID to each of them. The video session in our example has two flows (video and audio). Each flow can be routed on a different path if necessary, but our initial assumption is that both are routed on the same path. The OPDPMA therefore constructs the following policy object and passes it to the Plans Generator.

Action	SSON_ID	FLOW_Ids		Application	User	Policies
Create	3432	Audio = 1	Video = 2	video	x	Policy (1)

The Plans Generator searches for the optimal path, including the suitable MPs. It then sends back a policy object (2) containing the proposed topology in the form of a connection matrix.

```
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) and (User = "x")    Then
   Connection_MAT = {ONodeA (client), ONodeB (caching MP), ONodeC (server)}
   Client = ONodeA, Next_HOP = ONodeB, Server = ONodeC
```
(2)

This plan is sent to the PG that uses the relevant context to generate the policies (3) that will make up the SSON.

```
Server = ONodeC
   If (User = "x") and (Application = "video") Then
      SSON_ID = 3432, Audio_FlowID = 1, Video_FlowID = 2
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) Then
      Next_HOP_ONodeID = ONodeB, Next_HOP_ONodeIP = xxx.xxx.xxx.xxx
      Media_Ports = none
Target = ONodeB
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) Then
      Media_Ports = "Caching", Caching = "ok"
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) and (Caching = "ok") Then
       Disk_Space = 300 MB,   Freshness_Factor = 50.0,
      Expiration_Time = now + 10h,
      On cache miss refer to: ONodeC_IP = "xxx.xxx.xxx.xxx",
      On overload refer to: ONodeC_IP = "xxx.xxx.xxx.xxx"
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) and (User = "x") Then
      Next_HOP_ONodeID = ONodeA, Next_HOP_ONodeIP = xxx.xxx.xxx.xxx
      Sending_Rate = default
Target =ONodeA
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) Then
      Application = "video"
```
(3)

The first policy instructs the server's OPEP to mark each packet with the session information. The second policy updates its routing table to route the packets to the next overlay node in the SSON topology. The rest of the policies update the routing tables of overlay nodes. In OnodeB, a caching media port is configured to cache the data and to deliver it to ONodeA (the client). At that point, the OSL component is instructed to deliver the packets to the requesting application.

Once the OPDPMA receives these policies, it sends them to the conflict resolution agent to check for conflicts with policies and SSONs that are already installed. If there are no conflicts, the OPDPMA constructs a policy object for each participating overlay node. This policy object contains information about its neighboring overlay nodes in order to facilitate routing and the enforcement of applicable policies. If a conflict is detected, the policies are rejected and an adaptation process is triggered so that the conflict can be overcome.

Once the policy object is received by the OPEP, it is analyzed and enforced. In our example, the caching MP at ONodeB starts caching the video content and sending the video to the client from the cached version. The OPEP reports to the OPDP informing it about the success or failure of enforcing the decision. Assuming success at all overlay nodes, the SSON is now constructed.

(2) *SSON adaptation*: If another user requests the same service, the overlay must be adapted to include the new user. In our example, the new user's OPEP constructs a policy object containing the request and user information. After authenticating the user for security and accounting purposes (operations outside the scope of this paper), the OPDPMA checks to see if there is an SSON for the requested service. When one is found, it triggers an adaptation process by sending a message to the PG requesting any policies specific to the user. Assuming that the new user's context information has already been fed to the context memory, the PG generates policy (4).

```
If (User = "y") and (Application = "video") Then
   Available_Bandwidth = 16 kbps, One_Way_Delay < 300ms
   Special_Functions = "routing, scaling", Priority = 2
```
(4)

Based on policy (4), the OPDPMA decides to scale down the video frames so that they may be routed to the new user. It therefore includes another MP with routing and adaptation capabilities. Along with the SSON information, this is sent to the Plans Generator that invokes the plan and decides which Media Port to include. It sends back a policy object (5) containing the proposed topology in the form of a connection matrix.

```
IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) and (User = "y") Then
   Connection_MAT = {ONodeB (caching), ONodeD (media adaptation MP),  ONodeE (client)}
   Client (y) = ONodeE, Next_HOP = ONodeD; scaling, Server = ONodeB; caching MP
```
(5)

In our proposal, the caching Media Port is used to route the data to the new user rather than the original streaming server. This adaptation saves network bandwidth and resources because the same content is distributed only once for each SSON rather than once for each user.

The PG generates the policies (6) that follow. The sending rate of the caching MP is chosen to match the user preferences, the adaptation MP is configured to scale down the video frames and the video data are buffered at the client side before being forwarded to the requesting application. The rest of the adaptation steps are the same as those in the section on creation.

```
Target =ONodeB
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) and (User = "y") Then
      Next_HOP_ONodeID = ONodeD, Next_HOP_ONodeIP =
      xxx.xxx.xxx.xxx ,     Media_Ports = "caching" ,
      Sending_Rate = 10p/100mls
Target = ONodeD
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) Then
      Media_Ports = "Scaling", Scaling = "ok"
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) and (Scaling ="ok") Then
      Frame_rate = 10fps, Frame_Size = 320x240
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) Then
      Next_HOP_ONodeID = ONodeE, Next_HOP_ONodeIP =
      xxx.xxx.xxx.xxx
Target =ONodeE
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) Then
      Buffer_Size > 2 MB, Application = "video"
```
(6)

(3) *SSON termination*: If a user leaves a session normally (or even unexpectedly because of a crash), the SSON must be adapted accordingly. If the session is ended normally, the OPDP receives a notification of leave from the user; it then sends a policy object to the overlay nodes to uninstall the existing policies. If the notification is received from the last user of the SSON, the session is terminated by sending policy objects to the overlay nodes that are part of the SSON. In our example, a leave request from user *y* causes the following policies (7) to be sent by the OPDP to ONodeB, ONodeD, and OnodeE.

```
Target =ONodeB
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) and (User = "y") Then
      Delete_Policy
Target = ONodeD
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) Then
      Media_Ports = "None", Scaling = "No"
       Delete_Policy
   IF (SSON_ID = 3432) and (Video_FlowID = 2) and (Audio_FlowID = 1) Then
      Delete_Policy
Target =ONodeE
   IF (SSON_ID = 3432) and ((Video_FlowID = 2) or (Audio_FlowID = 1)) Then
       Delete_Policy
```
(7)

Once these policies are received by the respective OPEPs, they are deleted immediately and all ongoing packets are dropped.

To deal with the failures that may occur in a dynamic network, we adopt a fault recovery mechanism similar to the one described in Chakraborty et al. (2002). This guards against failures with a checkpoint technique. Each overlay node that is part of an SSON sends the checkpoint back to the OPDP. The OPDP caches any checkpoints obtained. If an overlay node fails, the OPDP can receive no more checkpoints. It then decides whether to readapt the SSON or to terminate the SSON if it has no users.

5. Simulation details and results

In our simulation, the topology was constructed using the BRITE (Medina et al., 2001) topology generator and the network was simulated using the J-Sim network simulator (Tyan and Hou, JavaSim On-Line Manuals and Tutorials), a simulator with a Java(tm)-based engine. We conducted two experiments to test our architecture. The first simulated a moderate-sized network to test a mobility scenario. The second simulated a large network to test the response to heavy demand.

5.1. Experiment 1: Mobility scenario

In the first experiment, three interconnected networks were simulated as shown in Fig. 5. One was a LAN with randomly generated topology and the other two were WLANs with different bandwidth capacities. WLAN A had a higher bandwidth capacity (15 Mb/s) than WLAN B (5 Mb/s). The LAN contains different MPs such as high bandwidth (HB) routing MP, low bandwidth (LB) routing MP, HB caching MP, and a splitter MP. The mobile user moves average speed of 5 km/h (Tabbane, 1995).

The scenario consisted of three scenes, the first showing the creation of the overlay and the second and third showing the dynamic adaptation and routing. In the first scene, a user tunes her office PC into a video server and starts to view a video. An SSON that requires a HB MP is created. In the second scene, the user moves to a cafeteria during her lunch break. When she enters the cafeteria's coverage area, the network detects her PDA and its wireless headset. The SSON adapts by choosing a splitting MP that splits the audio and video of the session into different flows. The audio flow is sent through a routing MP to the headset. Assuming that the user previously established her context by stating that she has a meeting in a conference room after lunch, a caching MP close to the conference room's wireless LAN is selected and automatically configured to catch the video flow. In the third scene, the user moves to the conference room and the SSON is automatically reconfigured. The video flow from the caching MP is resumed, thereby reducing transmission delays. The throughput for each scene is shown in Fig. 6.

We observe that as the user moves to the cafeteria, the throughput is decreased until it reaches its minimum. As the throughput starts decreasing, the OPEP sends an adaptation request to its SSON OPDP. The OPDP then decides, based on user and network context, to split the session into video and audio flows. It then adapts the SSON to route the audio flow to the new location. This is shown as Scene 2 in Fig. 6 where the audio throughput is less than the original video and audio throughput. When the user moves to the conference room, the

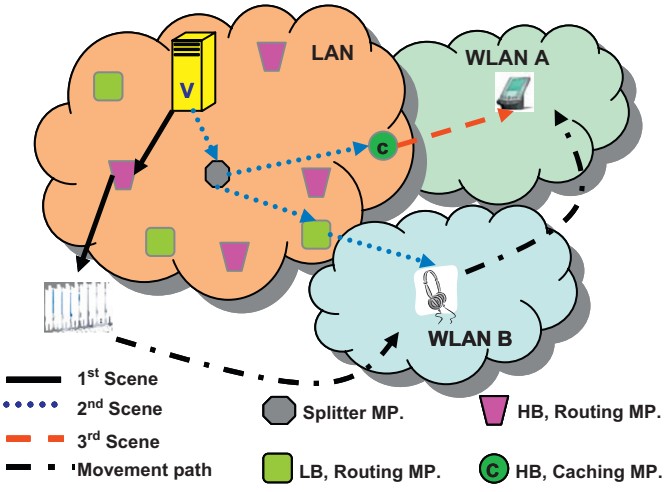

Fig. 5. Mobility scenario.

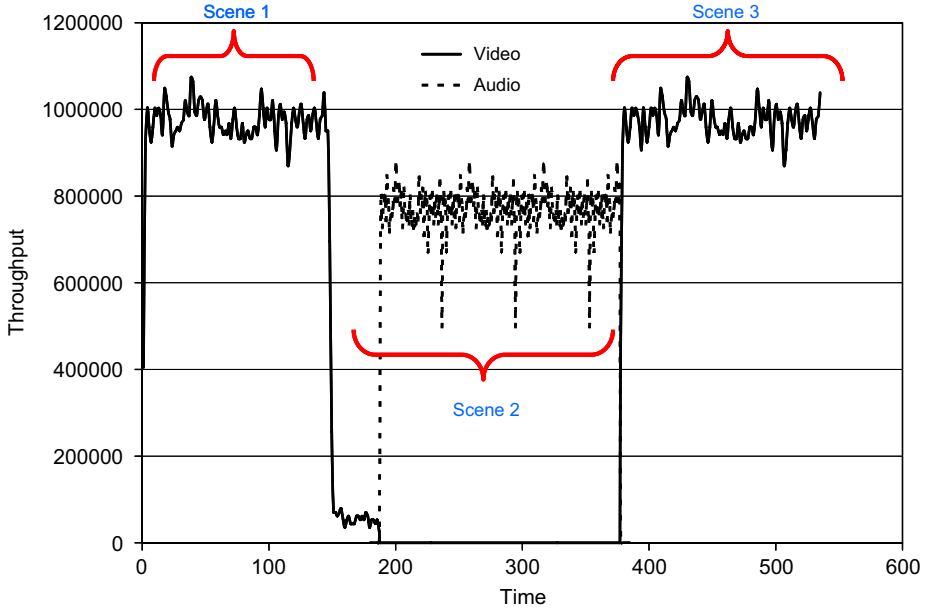

Fig. 6. Mobility scenario throughput.

MAs in the user OPEP detect the move and report to the SSON OPDP. The OPDP then readapts the SSON to the new context and resumes the transmission of the session from the cached version. Our architecture dynamically handles the adaptations of SSONs to the available context information since all adaptations are done transparently and with no explicit interaction from the user.

5.2. Experiment 2: Large-scale network

The topology used in the second experiment has 1000 nodes. The bandwidth assigned to each node is randomly selected between 128 and 512 kbit/s, and the links propagation delay is fixed at 1 ms. Each source generates packets according to a Poisson process with a bitrate of 3400 kbit/s and with a uniform random selection of destination nodes. Following a flash crowd characteristic, all nodes request their sessions at a random point during the first 2 s while the simulation lasts for another 1000 s. We ran the simulation 10 times and collected the results after each run. The first run simulated 100 SSONs with each subsequent run adding 100 SSONs. To reach steady-state behavior, each SSON issued one adaptation request randomly 30 s after the start of the simulation. As previously described, each SSON has one or more MPs when created and different MPs when adapted. In our selection of MPs, we compared two approaches. In the *Geographical* approach, we selected MPs that were geographically close to the shortest path between the source and the destination. In the *Random* approach, MPs were selected at random. We measured the overlay latency, packet stretch and management overhead.

(1) *Overlay latency*: Fig. 7 illustrates the average latency incurred by 1000 overlays with varying numbers of MPs. MPs join the overlays until the desired number is reached; the measurements are taken after the system stabilizes. The *Random* approach has the worst performance, especially for large sessions. Three factors contribute to the latency overhead. First, the encapsulation and decapsulation time depends on overlay node capabilities such as CPU speed and memory. Second, as overlay packets add more information to the header of the normal IP packets, the packet size increases, thereby increasing the time needed for delivery. The solution is to equip devices, especially small ones such as PDAs, with faster CPUs and more memory in order to maintain, or if possible to reduce, latency level. Third, the processing time needed at MPs, for example to record the data into caches. The average overlay path latency increases linearly with the number of MPs in the path. Although each MP adds its own latency depending on how fast it can provide its service, the average latency incurred by each MP is 0.01 s. Therefore, in order to compensate for the delay in multimedia transmission at the source and its presentation at the destination, we need to set a buffer size at the destination relative to the number of MPs used in the SSON.

(2) *Packet stretch*: The stretch of the SSONs' topologies is defined as the number of physical hops taken by an overlay packet divided by the number of hops a packet takes when using an IP-layer path between the same source and destination. A high stretch value indicates an inefficient SSON topology as their packets have longer routes and delays. Fig. 8 shows the simulation results for the average overlay path stretch and Fig. 9 shows the distribution of the actual stretch values for the geographical approach. The results show that the stretch for the geographical approach ranges from 1.73 to 1.79. This low stretch value is not significant considering the gains obtained by using policies. The distribution of the actual stretch values shows that 3.6% of sessions have a stretch greater than 3 and 76.8% have a stretch less than 2. Although the geographical approach improves the overlay path stretch, the improvement is not significant compared to the random approach. This is due mainly to the limitation that results from using an MPDS. The MPDS is a centralized entity that, in addition to its disadvantage of being a single point of failure, frequently registers MPs services and capabilities. In a dynamic network, this is not sufficient as the services change over time. The capacities of MPs are also dynamic as they change when sessions are added and removed. In order to decrease the stretch, therefore, there is a need to design a resource location mechanism. This would integrate the search for an optimal overlay path that satisfies a certain QoS metric with the search for the MPs needed to construct the SSON. The resource discovery approach should be decentralized and should exploit the semantics of the services offered by MPs.

(3) *Management overhead*: The management overhead of the architecture consists of (a) the time needed to generate and enforce policies, (b) the time needed to exchange messages between the SSON OPDP and OPEPs, as well as between the SSON OPDP and the System OPDP, and (c) the time needed to access information in the policy repository and plan history. Fig. 10 shows the average management overhead. Results show that, while the number of sessions increases, the management overhead increases only slightly. For a small number of sessions, stretch is a significant factor as it results in larger delays and thereby increases the management time. For example, the second simulation run shows that the average management overhead is nearly 0.275 s and the stretch is 1.75 (see Fig. 8). As more sessions are added, therefore, the average stretch decreases and the time needed to generate policies and to access the repository and plan history outweighs the effect of the stretch. The management overhead time slightly increases but the average increase is insignificant compared to the gain achieved by the architecture itself.

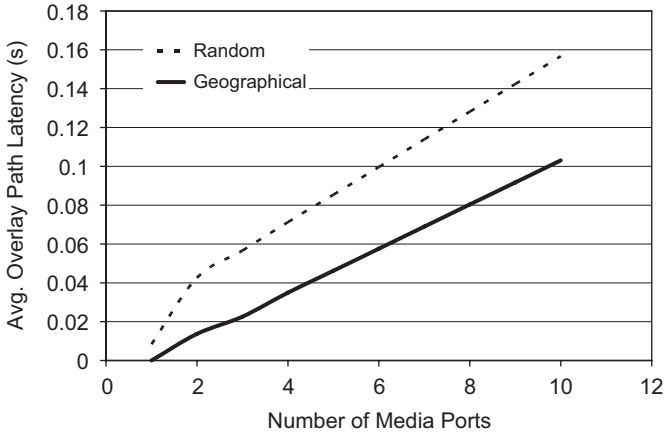

Fig. 7. Average overlay path latency.

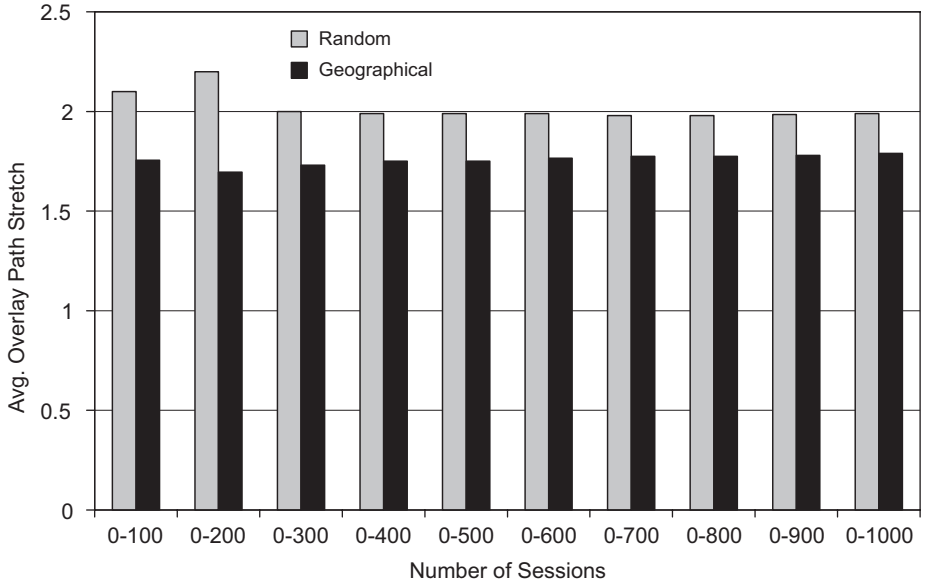

Fig. 8. The average overlay path stretch.

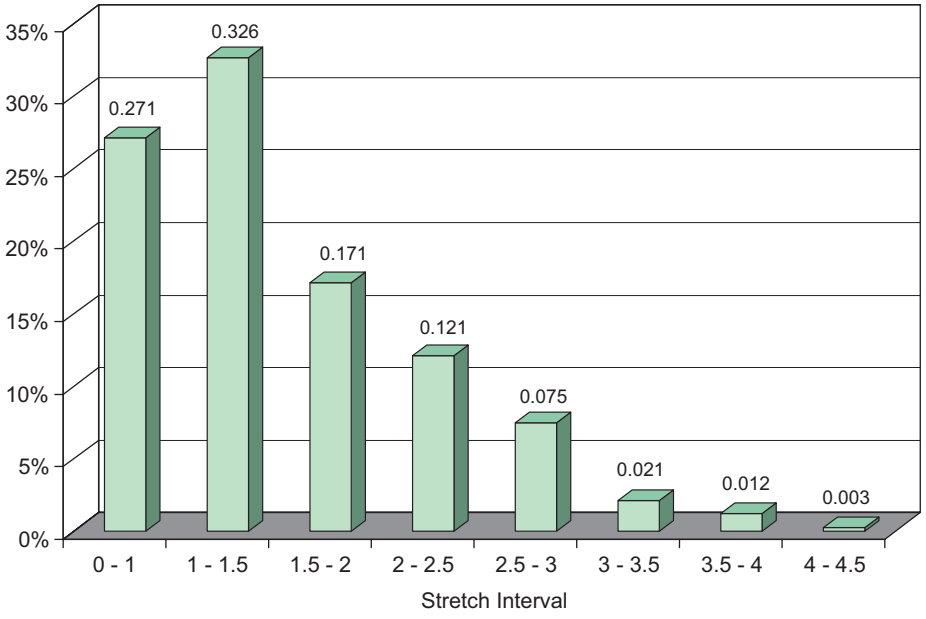

Fig. 9. The distribution of the actual stretch.

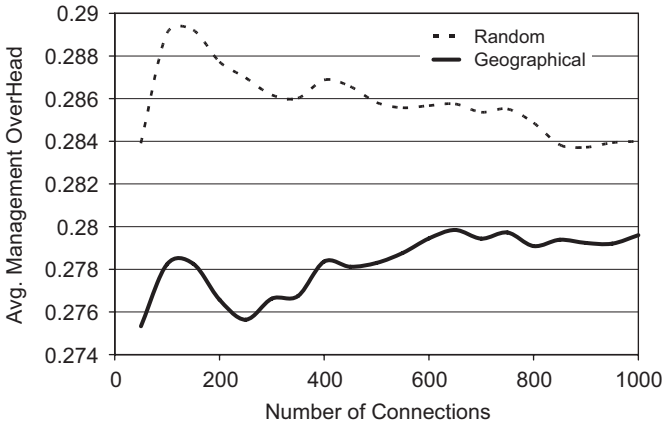

Fig. 10. The average management overhead.

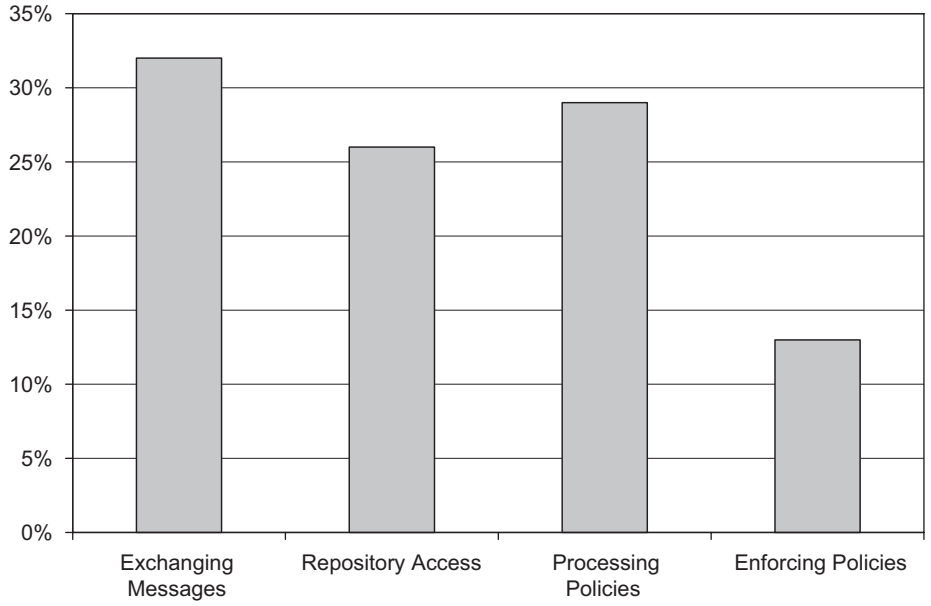

Fig. 11. The time needed to create or adapt an SSON.

Fig. 11 shows that the time needed to process and enforce policies is nearly 42% of the total time consumed in creating or adapting an SSON. Exchanging messages between the OPDP and OPEPs takes 32% of the time, and the remaining 26% is used to access information in the policy repository and plan history. This indicates that the overhead caused by introducing policies is compensated for by the gain achieved by the context-aware architecture and the dynamic deployment of SSONs. Since the extra time is needed only once to create an SSON and once for each adaptation, it is insignificant for overlays that do not require adaptation, or that have a long operation time between creation and adaptation.

6. Conclusion and future work

This paper presents a novel and flexible policy-based, context-aware management architecture that automates the task of managing overlay networks. The contribution of the architecture is that the creation, optimization, adaptation, and termination of SSONs are controlled using policies. Policies are generated dynamically from user, network, and service provider contexts and enforced on the fly. As a result, the behavior of overlays is tailored to their specific needs. Simulation results demonstrate flexibility in managing overlays that is not possible without introducing policies at the overlay level. In future work, we plan to investigate the functionality of the OCRA in order to detect and prevent the negative effects of coexisting overlays. We plan to study the use of a more efficient MP resource discovery mechanism than the present MPDS. We also plan to further evaluate performance by designing a prototype.

Acknowledgements

The authors would like to thank the anonymous referees for their suggestions and comments that have improved the quality of the paper.

This work was partly supported by a Strategic Research Grant from the Natural Sciences and Engineering Research Council of Canada.

References

Agarwalt S, Chuaht C, Katz R. OPCA: robust interdomain policy routing and traffic control. In: Proceedings of the IEEE OPENARCH, New York, April 2003.

Andersen D, Balakrishnan H, Kaashoek F, Morris R. Resilient overlay networks. In: Proceedings of the 18th ACM symposium on operating systems principles (SOSP), Banff, Canada, October 2001.

Bałos K, Szydło K, Szymacha R, Zieliński K. Context dissemination and aggregation for ambient networks. In: First European conference on smart sensing and context, Netherlands, 26–27 October 2006.

Chakraborty D, Perich F, Joshi A, Finin T, Yesha Y. A reactive service composition architecture for pervasive computing environment. 7th Personal Wireless Communications Conference, 2002.

Damianou N, Dulay N, Lupu E, Sloman M. The ponder specification language. Workshop on policies for distributed systems and networks. HP Labs, Bristol, 29–31 January 2001.

Durha D, Boyle J, Cohen R, Herzog S, Rajan R, Sastry A. The COPS (common open policy service) protocol. IETF RFC 2748, 2000.

Ferdinando A, McKee P, Amoroso A. A policy based approach for automated topology management of peer to peer networks and a prototype implementation. Proceedings of the fourth international workshop on policies for distributed systems and networks (POLICY 2003), Italy, 2003. p. 235–8.

Hartung F, Herborn S, Kampmann M, Schmid S. Smart multimedia routing and adaptation using service specific overlay networks in the ambient networks framework. WWRF #12, Toronto, 4–5 November 2004.

Jannotti J, Gifford D, Johnson K, Kaashoek M, O'Toole J. Overcast: reliable multicasting with an overlay network. In: Proceedings of the USENIX OSDI, October 2000.

Lakhina A, Byers JW, Crovella M, Matta I. On the geographic location of Internet resources. IEEE J Sel Areas Commun 2003;21(6):934–48.

Li Z, Mohapatra P. QRON: QoS-aware routing in overlay networks. IEEE J Sel Areas Commun 2004;22(1).

Massimi M, Wolz U. Peer-to-peer policy management system for wearable mobile devices. In: Proceedings of the 17th IEEE international symposium on wearable computers (ISWC'03), 2003. p. 246–7.

Medina A, Lakhina A, Matta I, Byers J. BRITE: universal topology generation from a user's perspective. Technical Report. BUCS-TR-2001-003, Boston University, 12 April 2001.
Melodia T, Pompili D, Akyildiz IF. On the interdependence of distributed topology control and geographical routing in ad hoc and sensor networks. IEEE J Sel Areas Commun 2005;23(3):520–32.
Nakao A, Peterson L, Bavier A. A routing underlay for overlay networks. In: Proceedings of the ACM SIGCOMM, August 2003.
Niebert N, Schieder A, Abramowicz H, Malmgren G, Sachs J, Horn U, et al. Ambient networks—an architecture for communication networks beyond 3G. IEEE Wireless Communications [special issue on 4G mobile communications—towards open wireless architecture], 2004.
Ooi WT, Renesse RV, Smith B. The design and implementation of programmable media gateways. In: Proceedings of the NOSSDAV'00, Chapel Hill, NC, June 2000.
Salo J, Tarlano A, Galis A. Context sources and their presentation in the WWI system architecture. Wireless World Research Forum-WWRF18, Helsinki, Finland, 13–15 June 2007.
Samaan N, Karmouch A. An automated policy-based management framework for differentiated communication systems. IEEE J Sel Areas Commun 2005;23(12):2236–47.
Schmid S, Hartung F, Kampmann M, Herborn S, Rey J. SMART: intelligent multimedia routing and adaptation based on service specific overlay networks. In: Proceedings of the Eurescom Summit 2005, Heidelberg, Germany, 2005. p. 69–77.
Shen K. Saxons: structure management for scalable overlay service construction. In: USENIX NSDI '04, 2004. p. 281–94.
Shin J, Kim JW, Kuo CJ. Quality-of-service mapping mechanism for packet video in differentiated services network. IEEE Trans Multimedia 2001;3(2):217–30.
Sripanidkulchai K, Maggs B, Zhang H. An analysis of live streaming workloads on the internet. In: ACM IMC, October 2004.
Subramanian L, Stoica I, Balakrishnan H, Katz R. OverQoS: an overlay based architecture for enhancing Internet QoS. In: Proceedings of the NSDI, 2004.
Tabbane S. An alternative strategy for location tracking. IEEE J Sel Areas Commun June 1995;13(5):880–92.
Tyan HY, Hou CJ. JavaSim on-line manuals and tutorials. Available on-line at 〈http://j-sim.cs.uiuc.edu/〉.
Vleeschauwer B, Turck F, Dhoedt B, Demeester P. Dynamic algorithms to provide a robust and scalable overlay routing service. In: Proceedings of the international conference on information networking (ICOIN 2006), Sendai, Japan, 16–19 January 2006.
Yang K, Galis A, Mota T, Gouveris S. Automated management of IP networks through policy and mobile agents. In: Proceedings of the fourth international workshop on mobile agents for telecommunication applications. Lecture notes in computer science, vol. 2521. Spain: Springer; October 2002.