An Architecture for Multimedia Delivery Over Service Specific Overlay Networks

Ibrahim Al-Oqily[1], Ahmed Karmouch[1], and Roch Glitho[2]

[1]School of Information Technology & Engineering (SITE), University Of Ottawa, PO Box 450, Ottawa, ON, K1N 6N5, Canada. {ialoqily, karmouch}@site.uottawa.ca.

[2]Ericsson Canada, 8400 Decarie, Montreal Quebec, Canada, H4P2 N2, roch.glitho@ericsson.com

Abstract. Overlay networks are becoming widely used for delivering content, since they provide effective and reliable services that are not otherwise available. However, overlay management systems face the challenges of increased complexity and heterogeneity due to the numerous entities that are involved in realizing overlay services. We believe that autonomic management is a key solution for dealing with the complexity of overlay management. In this paper, a management architecture for service specific overlay networks is proposed. Overlays are viewed as a dynamic organization for self-management in which self-interested nodes can join or leave according to their goals. The objective of this architecture is to create autonomic overlays that are driven by different levels of policies. Policies are generated at different levels of the autonomic management hierarchy and enforced on the fly. The proposed autonomic management dynamically adapts the behavior of the overlay network to the preferences of the user, network, and service providers. A description of our novel architecture that addresses these challenges is presented.

Keywords— Autonomic Computing, Adaptive SSONs, Network Management, Overlay Networks, Policies.

1 INTRODUCTION

Overlay networks are a virtual topology on top of a physical topology, and they are becoming more popular due to their flexibility and their ability to offer new services. Recent research in this area has focused on designing specific overlay networks to deliver media in heterogeneous environment. For example, SMART [1] was developed in the context of the Ambient Network project [2] in order to optimize media delivery services by moving control and resource allocation to the network itself, since it has more knowledge about topology and network proper-

ties. SMART creates a Service Specific Overlay Network (SSON) for each media delivery service or group of services. SSON construction uses network side functions, called MediaPorts (MPs). MPs, thereby, provide the flexibility to modify the content and the services, such as caching, adaptation and synchronization [3].

A service delivered to a customer by a SP is usually formed from a composition of different services. Some services are basic in the sense that they cannot be broken down further into component services, and usually act on the underlying resources. Other services are composed of several basic services. Every service consists of an allocation of resource amounts to perform a function. However, with the increasing demands for QoS, service delivery should be efficient, dynamic, and robust. Current manual approaches to service management are costly and consume resources and IT professionals' time, which leads to increased customer dissatisfaction. With the advent of new devices and services the complexity is further increased.

Given the multiple sources of the heterogeneity of networks, users, and applications, constructing SSONs in large distributed networks is challenging. Media content usually requires adaptation before it is consumed by media clients. For example, video frames must be dropped to meet QoS constraints. Other examples are when a client with a PDA requires a scale down for a video, or when content must be cached to be viewed by a mobile user. In addition to adaptation, new users may request to join or leave the overlay, a network node may fail, or a bottleneck may degrade the SSON's performance. Consequently, the overlay must be adapted to overcome these limitations and to satisfy the new requests. It is obvious that with a large number of overlays, the management task becomes harder to achieve using traditional methods. Therefore new solutions are needed to allow SPs to support the required services and to focus on enhancing these services rather than their management.

The concept of autonomic computing (AC) [4] is proposed by IBM to enable systems to manage themselves through the use of self-configuring, self-healing, self-optimizing, and self-protecting solutions. It is a holistic approach to computer systems design and management with the goal to shift the burden of support tasks, such as configuration and maintenance, from IT professionals to technology. Therefore, AC is a key solution for SSON management in heterogeneous and dynamic environments.

Establishing a SSON involves 1) Resource discovery to discover network-side nodes that support the required media processing capabilities. 2) An optimization criterion to decide which nodes should be included in the overlay network. 3) Configuring the selected overlay nodes, and 4) Adapting the overlay to the changing network context, user, or service requirements, and joining and leaving nodes. In AC, each step must be redesigned to support autonomic functions. In other words, in Autonomic Overlays (AO), each step imposes a set of minimum requirements. For example, the resource discovery scheme should be distributed and not rely on a central entity, dynamic to cope with changing network conditions, efficient in terms of response time and message overhead, and accurate in terms of its success rate. The optimization step is mapped into a self-optimization that se-

lects resources based on an optimization criterion (such as delay, bandwidth, etc.) and should yield the cheapest overlay, and/or an overlay with the least number of hops, and/or an overlay that is load-balanced, and/or a low latency overlay network, and/or a high bandwidth overlay network. The configuration of the selected overlay nodes in a given SSON is mapped into a self-configuration and self-adaptation. Self-configuring SSONs dynamically configure themselves on the fly. Thus they can adapt their overlay nodes immediately to the joining and leaving nodes and to the changes in the network environment. Self-adapting SSONs self-tune their constituent resources dynamically to provide uninterrupted service. Our goals are to automate overlay management in a dynamic manner that preserves the flexibility and benefits that overlays provide, to extend overlay nodes to become autonomic, to define the inter-node autonomic behavior between overlay nodes, and to define the global autonomic behavior between SSONs.

This paper proposes a novel management architecture for overlay networks. Our contributions are twofold. First we introduce the concept of Autonomic Overlays (AO), in which SSONs and their constituent overlay nodes are made autonomic and thus become able to self-manage. Second, autonomic entities are driven by policies that are generated dynamically from the context information of the user, network, and service providers. This ensures that the creation, optimization, adaptation, and termination of overlays are controlled by policies, and thus the behaviors of the overlays are tailored to their specific needs.

The paper is organized as follows. Section 2 discusses the related work. Section 3 introduces the proposed autonomic overlay architecture. Section 4 discusses the experimental evaluation. Finally, in Section 5, we draw our conclusion and suggest future work.

2 RELATED WORK

The proposed autonomic overlays draw upon IBM's vision and blueprint [7]. The work presented in this paper is concerned with all possible phases of the service delivery in SSONs – from the instant of requesting a service to the instant of terminating it. Thus we present an integral approach to SPs wishing to deliver services over their infrastructure.

IBM identified the complexity of current computing systems as a major barrier to its growth [5]. The solution to this problem lies in more intelligent systems called autonomic computing (AC). AC simplifies and automates many system management tasks traditionally carried out manually. Systems that manage themselves are able to adapt to changes in their environment in accordance with business objectives. The result is a great savings in management costs and IT professionals' time. This will free the latter to focus on improving their offered service rather than managing them manually. Some of the main scientific and engineering challenges that collectively make up the grand challenge of autonomic computing

were outlined in [6]. Also, a set of characteristics required by AC were identified and explained in [7].

According to the IBM vision [4], an AC system is a system that knows itself and its environment, configures and reconfigures itself under varying and unpredictable conditions, heals itself, provides self-protection, and keeps its complexity hidden. Although the IBM vision is a holistic approach to designing computer systems, much of the research in this field focuses on a few specific aspects of this vision.

Autonomic communications were proposed in [8]. It has a similar concept to IBM's autonomic computing. The difference is that, in the former, the focus is on individual elements of the network, how their behavior is learned and altered, and how they interact with other elements. Our work focuses on service specific overlay networks; thus, the interaction between the network and computing entities is based on a service request/offer concept in which each entity is responsible for its internal state and resources. An entity may offer a service to other entities. The offering entity responds to a request based on its willingness to provide a service in its current state. A generic architecture for autonomic service delivery was proposed in [9]. It defines a resource management model based on virtualization. However, it is service-independent, and is unlikely to achieve the specific QoS requirements for each service dynamically without human intervention. A model for dynamic fault tolerance technique selection for grid work flow, that allows the system to configure its fault tolerance mechanism, was developed in [10].

Policy-based management for computer systems has also been studied. Pattern classification and clustering techniques that support online decision making and incremental learning in autonomic systems were proposed in [11]. The use of policies to configure autonomic elements to enforce the required behavior in an Apache web server was presented in [12]. A set of UML-based models were developed and used in [13] to specify autonomic properties and to deploy policies as an executing system based on composition and model modification. A policy-driven model based on multi-agent systems was also proposed in [14]. In their model, Web services are represented as agents and agent behavior is controlled using high level policies. A mapping of biological systems to PBMS was introduced in [15]. Their system is hierarchical and relies on mechanisms for organism regulation, which supports self-management at different levels of the hierarchy. Humans in an organization thus specify policy at a level of abstraction that reflects their specific needs. The difference between our work and all these approaches is that the above approaches consider a particular service to which their design is appropriate. In addition, policy generation is not a fully automatic process and human intervention is still needed.

Projects such as Service Clouds [16], Autonomia [17], GridKit [18], Auto-Mate [19], and Unity [20] are using the autonomic concept in different ways. Service Clouds provides an infrastructure for composing autonomic communication services. It combines adaptive middleware functionality with an overlay network to support dynamic service reconfiguration. Autonomia provides dynamically programmable control and management to support the development and deployment

of smart applications; primarily, it achieves the self-healing property for failed entities. GridKit proposes a middleware that offers a consistent programming model across different communication types. AutoMate enables the development of autonomic Grid applications by investigating programming models, frameworks, and middleware services that support autonomic elements. Finally, Unity designs both the behavior of individual autonomic elements and the relationships that are formed among them in order to create computing systems that manage themselves. A detailed survey on autonomic computing is available in [21].

3 AUTONOMIC OVERLAYS

To tackle the complexity of overlay management, each SSON is managed by an SSON Autonomic Manager (SSON-AM) that dictates the service performance parameters. This ensures the self.* functions of the service. In addition to this, overlay nodes are made autonomic to self-manage their internal behavior and their interactions with other overlay nodes. In order to ensure system wide performance, System Autonomic Managers (SAM) manages the different SSON managers by providing them with high level directives and goals. The following sections detail the different aspects of our architecture.

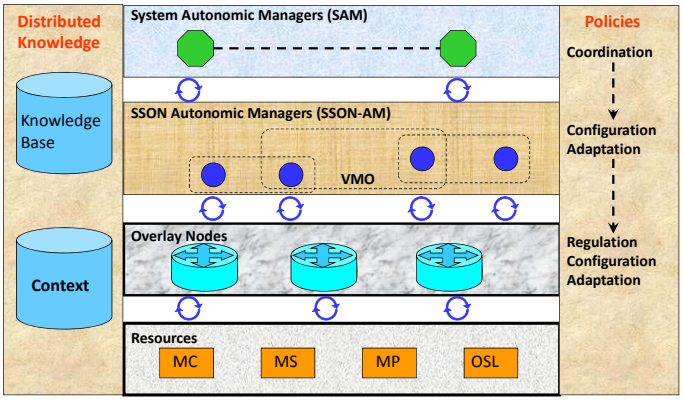

Fig. 1. Autonomic overlays architecture

3.1 Architecture Overview

The set of components that makes up our architecture is shown in Fig. 1. The lowest layer contains the system resources that are needed for multimedia delivery sessions. In particular, the Overlay Support Layer (OSL) receives packets from the network, sends them to the network, and forwards them on to the overlay. Overlay

nodes implement a sink (MediaClient, or MC), a source (MediaServer, or MS), or a MediaPort (MP) in any combination. MPs are special network side components that provide valuable functions to media sessions; these functions include, but are not limited to, special routing capabilities, caching, and adaptation. These managed resources can be hardware or software and may have their own self-managing attributes.

The next layer contains the overlay nodes. Overlay nodes are physical Ambient Network nodes that have the necessary capabilities to become part of the SSON. They consist of a control plan and a user plan. The control plan is responsible for the creation, routing, adaptation, and termination of SSONs, while the user plan contains a set of managed resources. The self-management functions of overlay nodes are located in the control plan. The Ambient manageability interfaces are used by the self-managing functions to access and control the managed resources. The rest of the layers automate the overlays' management in the system using their autonomic managers. SSON-AMs and SAMs may have one or more autonomic managers, e.g. for self-configuring and self-optimizing. Each SSON is managed by an SSON-AM that is responsible for delivering the self-management functions to the SSON. The SAMs are responsible for delivering system wide management functions; thus, they directly manage the SSON-AMs. The management interactions are expressed through policies at different levels. All of these components are backed up with a distributed knowledge. The following sections describe each component in detail.

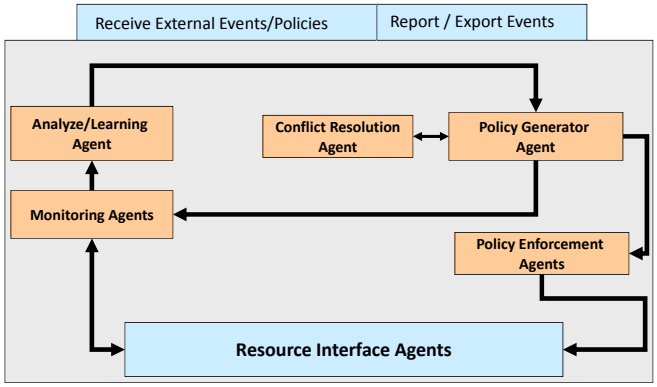

Fig. 2. The set of components that makes up the intelligent control loop

3.2 Autonomic Elements

1) Overlay Nodes Autonomic Manager(ONAM): Each overlay node contains a control loop similar to the IBM control loop [5] as shown in Fig. 2. The

Autonomic Manager (AM) collects the details it needs from its managed resources, analyzes those details to decide what actions need to change, generates the policies that reflects the required change, and enforces these policies at the correct resources. As shown in the figure, the ONAM consist of the following:

- Monitoring Agents (MAs): collects information from the overlay node resources, such as packet loss, delay jitter, and throughput. It also correlates the collected data according to the installed policies and reports any violation to the Analyze/Learning Agent (ALA). For example, an MA for a caching MP collects information about the MP's available capacity, and whenever the available capacity reaches 10% it reports to the ALA. Another example is the MA for a routing MP that relays data packets between overlay nodes: its MA collects information about the throughput and reports to the ALA whenever the throughput reaches a high value. These collected data will be used to decide the correct actions that must be taken to keep the overlay node performance within its defined goals. The MAs interact with the Resource Interface Agents (RIAs) to monitor the overlay node resources availability and to collect data about the desired metrics. They also receive policies regarding the metrics that they should monitor as well as the frequency in which they report to the ALA.
- Analyze/Learning Agent (ALA): observes the data received from the MAs and checks to see whether a certain policy with which its overlay node is abided is not being met. It correlates the observed metrics with respect to the contexts, and performs analysis based on the statistical information. In the case that one of policies is violated, it sends a change request to the Policy Generator (PG). This component is an objective of future work.
- Policy Generator (PG): The difference between this control loop and the IBMs' control loop lies in the use of a PG instead of a Plan component. The Plan function –according to IBM [5] – is to select or create a procedure that reflects the desired change based on the received change request from the Analyze Agent. This is not sufficient in our case, where each overlay node receives high level policies and it is up to the overlay node to decide how to enforce these policies based on its available resources. Therefore, we envisioned a PG instead. The PG reacts to the change request in the same way as in the Plan component, although it also generates different types of policies in response to the received high level policies. For example, based on the goal policies received by the overlay node, the policy generator generates the tuning polices and passes them to the MAs (more about this in Section 3.4). Upon generating new policies, the policy generator consults a Conflict Resolution Agent (CRA) that ensures the consistency of the new generated policies with those that already exist. Generally, we divide conflicts into two types: static conflicts and dynamic conflicts. In our model, a static conflict is a conflict that can be detected at the time of generating a new policy, while a dynamic conflict is one that occurs at run time.
- Policy Enforcement Agent (PEA): The PG generates suitable policies to correct the situation in response to a change request, and passes these policies to the

PEA. The PEA then uses the suitable RIA to enforce them. This includes mapping the actions into executable elements by forwarding them to the suitable RIA responsible for performing the actual adjustments of resources and parameters. The enforced policies are then stored in the Knowledge Base (KB).

- Resource Interface Agents (RIAs): These implement the desired interfaces to the overlay node resources. The MAs interacts with them to monitor the availability of overlay node resources and the desired metrics in its surrounding environment. Each resource type has its own RIA that translates the policy actions into an adjustment of configuration parameters that implements the policy action.
- Each overlay node has a set of interfaces to receive and export events and policies to other overlay nodes. These interfaces are essential to enable multiple overlay nodes to cooperate to achieve their goals. In particular, these interfaces are used by the SSON-AM to interact with the overlay nodes that agreed to participate in the SSON. The SSON-AM sends the system policies to the overlay nodes through these interfaces, through which it also receives reports on their current status.

2) SSON Autonomic Managers (SSON-AM): SSON-AMs implement the intelligent control loop in much the same way as ONAMs. They automate the task of creating, adapting, configuring, and terminating SSONs. They work directly with the ONAM through their management interfaces. They perform different self-management functions, such as self-configuring, self-optimizing, and self-adapting. Therefore, they have different control loops. Typically, they perform the following tasks:

- Self-configuration: SSON-AMs generate configuration policies in response to the received system policies. They use these policies to configure overlay nodes that are participating in a given SSON.
- Self-optimization: during SSON construction, SSON-AMs discover the overlay nodes required to set up a routing path for the multimedia session. Therefore, they are responsible for optimizing the service path to meet the required QoS metrics induced from high level policies as well as the context of the service.
- Self-Adaptation: SSON-AMs monitor the QoS metrics for the multimedia session and keep adapting the service path to the changing conditions of the network, service, and user preferences. They also monitor the participating overlay nodes and find alternatives in case one of the overlay nodes is not abiding to the required performance metrics.

SSON-AMs receive goal policies from the SAMs to decide the types of actions that should be taken for their managed resources. A SSON-AM can manage one or more overlay nodes directly to achieve its goals. Therefore, the overlay nodes of a given SSON are viewed as its managed resources. In addition, they expose mana-

geability interfaces to other autonomic managers, thus allowing SAMs to interact with them in much the same way that they interact with the ONAMs. See Fig. 3.

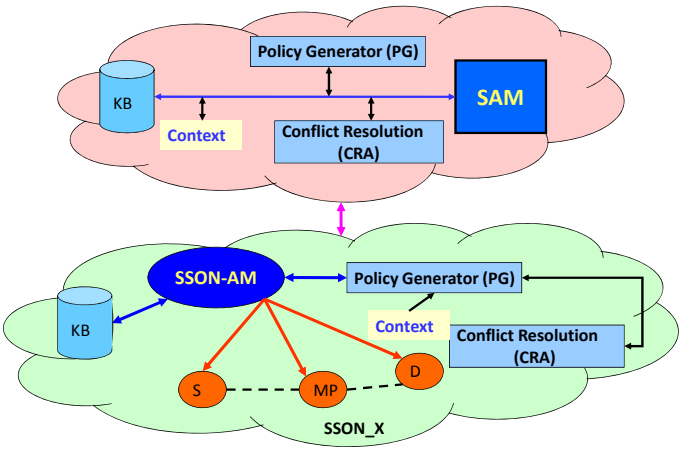

Fig. 3. The lower part represents an SSON that consists of a Source (S), a Destination (D), and a MediaPort (MP). The SSON is managed by a SSON-AM. It has its own Knowledge Base (KB). The upper part represents a SAM and its components.

3) System Autonomic Managers (SAM): A single SSON-AM alone is only able to achieve self-management functions for the SSON that it manages. If a large number of SSONs in a given network with their autonomic managers is considered, it is observable that these SSONs 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, consider two SSONs sharing the same routing MP with the same goal to maximize throughput. This will lead to a competition between autonomic managers that are expected to provide the best achievable performance. Therefore, and in order to achieve a system wide autonomic behavior, the SSON-AMs need to coordinate their self-managing functions. Typically this is achieved using SAMs.

SAMs can manage one or more SSON-AMs. They pass the system high level policies, such as load balancing policies, to the SSON-AMs. Moreover, whenever they find shared goals between two different SSON-AMs, they inform them to avoid conflicting actions. The involved autonomic managers then contact each other and create a Virtual Management Overlay (VMO) among themselves as illustrated in Fig. 4. They use this VMO to coordinate their management actions before they are passed to their overlay nodes.

Sharing goals is not the only reason to create VMOs; SSONs sharing common links as well as SSONs that belong to the same policy domain (same service class, ISP, etc.) may also create VMOs among themselves to coordinate their actions. Moreover, SSONs that share common nodes/links affect each other's performance, as they compete for the shared resources. This might result in a degraded performance as the competition will cause the control loop to be invoked frequently in an attempt to reach the desired performance goals. Also, all the SSONs in a given domain (ISP) are expected to achieve the domain wide policies together. VMOs allow these policies to be dispatched and adapted to each SSON in a way that achieves the desired goals. Moreover, VMOs also allow the sharing of control and information between different SSONs. A set of SSONs that are co-located in given vicinity (such as an area, domain, AS, etc.) usually has independent rout decisions based on its observations of its environment. Sharing this information will result in a reduced overhead for each overlay to compute this information, and allows for adapting and generating policies to achieve better performance.

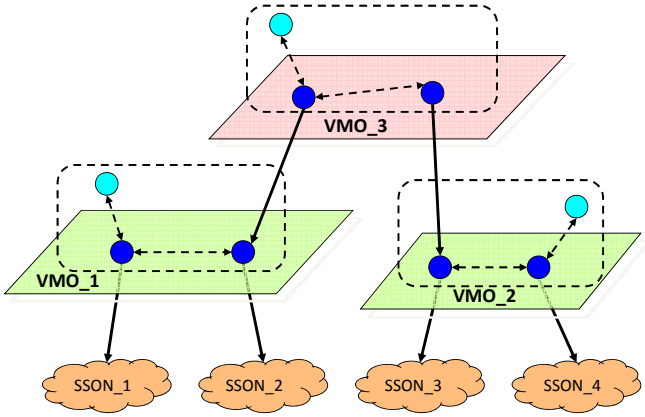

Fig. 4. Virtual Management Overlay (VMO) hierarchy

3.3 Distributed Knowledge

Each autonomic manager obtains and generates information. This information is stored in a shared Knowledge Base (KB) (see Fig. 3). The shared knowledge contains data such as SSON topology, media type descriptions, the set of policies that are active, and the goal policies received from higher level autonomic managers. The shared knowledge also contains the monitored metrics and their respective values. When VMOs are created, each autonomic manager can obtain two types of information from its VMO peers. The first is related to the coordination actions and the second is related to the common metrics in which each autonomic manager is interested. Therefore, knowledge evolves over time; the autonomic manager's

functions add new knowledge as a result of executing their actions, obsolete knowledge is deleted or stored in log files, and autonomic managers in VMOs exchange and share knowledge. Also, goal policies are passed from high level autonomic managers to their managed autonomic managers. The context information of the network, users, and services is also used primarily to aid in generating suitable policies at each level of autonomic managers.

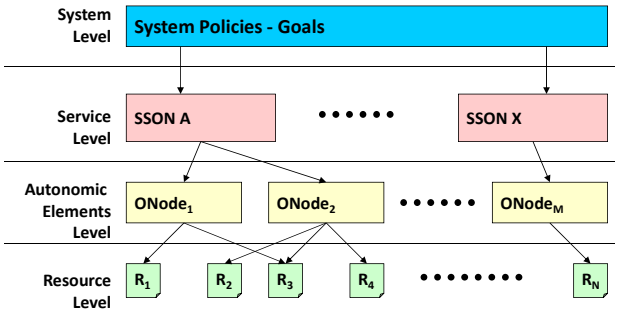

Fig. 5. Policy Levels

3.4 Policies

The use of policies offers an appropriately flexible, portable and customizable management solution that allows network entities to be configured on the fly. Usually, network administrators define a set of rules to control the behavior of network entities. These rules can be translated into component-specific policies that are stored in a policy repository and can be retrieved and enforced as needed. Policies represent a suitable and efficient means of managing overlays. However, the proposed architecture leverages the management task to the overlays and their logical elements, thus providing the directives on which an autonomic system can rely to meet its requirements. Policies in our autonomic architecture are generated dynamically, thereby achieving an automation level that requires no human interaction. In the following we will highlight the different types of policies specific to autonomic overlays. These policy types are being generated at different levels of the system.

Configuration policies: are policies which can be used to specify the configuration of a component or a set of components. The SSON-AMs generate the configuration polices for the service path that meets the SSON's QoS requirements. The ONAMs generate the specific resource configuration policies that, when enforced all together, achieve the SSON QoS metrics. The user, service, and network context are used by these autonomic managers to generate configuration policies.

Adaptation policies: are policies that can be used to adapt the SSON to changing conditions. They are generated in response to a trigger fired by a change in the user, service, or network context. SSON-AMs receive these triggers either from the

SAMs or from the ONAMs, while the ONAMs receive these triggers either from the SSON-AMs or from their internal resources. Whenever a change that violates the installed policies occurs, an adaptation trigger is fired. The autonomic manager that first detects this change tries to solve the problem by generating the suitable adaptation policies; if it does not succeed, it informs the higher level autonomic manager.

Coordination policies: are policies which can be used to coordinate the actions of two or more SSON-AMs. They are generated by the SAMs to govern the behavior of SSON managers that have conflicting goals to avoid race conditions.

Regulation policies: are generated by the overlay nodes themselves to control the MAs' behavior with respect to their goals. For example, a MA that measures throughput has a policy to report throughput < 70%. Another regulation policy can be installed to replace this policy and report throughput < 90%. The second regulation policy can be generated in response to an adaptation policy that requires throughput to be at least 90%. The MAs therefore are made more active to contribute to achieving the required tasks.

Figure 5 shows how these policies are related to our autonomic architecture. At the highest level the SAMs define the set of system polices. These policies represent the system wide goals and do not describe either the particular devices that will be used to achieve the system goals or the specific configurations for these devices. SAMs pass these policies to the SSON-AMs. SSON-AMs refine the system policies and generate service specific policies. They do so by adding further details to the system policies. These details are induced from the system policies as well as from the context information of the users, the network, and the service. At this level, the goals of the SSON under discussion, such as the permitted QoS metrics, are defined. These goals are still device independent policies. The set of service polices is then passed to the ONAMs. These autonomic managers further refine the received policies and generate the overlay node polices and their respective resource specific policies. Overlay node policies represent the goals that this overlay node is expected to achieve, while resource specific policies represents the actual actions that the resources of the overlay node has to do to achieve the overlay node goals. This separation of policies allows each autonomic element to focus on its goals and how to achieve them using its current resources while contributing at the same time to the overall system performance. By decoupling the functionality of adapting overlay node resources policies from the task of mapping system objectives and abstract users' requirements, the policy separation offers users and IT professionals the freedom to specify and dynamically change their requirements. The hierarchical policy model is used to facilitate the mapping of higher level system policies into overlay node objectives. Given sets of user, service and network context and constraints, as well as sets of possible actions to be taken, decisions for policy customizations are taken at run time based on values obtained from MAs to best utilize the available overlay node resources.

In addition to generating policies from high level goals, the policy generator located in each autonomic manager serves as a Policy Decision Point (PDP) for the

low level autonomic manager. For example, the SSON-AM serves as a PDP for the ONAM. Whenever an ONAM detects that one of the configuration policies has been violated, it tries to solve the problem locally. If it is unable to do so, it consults the SSON-AM to which the overly node is providing a service. The SSON-AM then tries to solve the problem by either relaxing the goals of the services or by finding an alternative overlay node that is able to achieve the SSON's goals. The SSON-AM then informs the ONAM of its decision, and may also consult its designated SAM to acquire decisions on situations that it cannot handle locally. The autonomic manager acting as a PDP decides which policies, if any configuration or adaptation policies have been violated, were most important and what actions to take. It uses information about the installed policies and the current context of the user, network, and service.

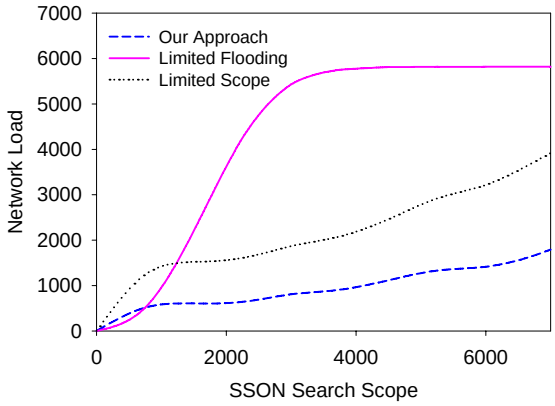

Fig. 6. Overhead due to search messages.

4 EXPERIMENTAL EVALUATION

We used a discrete event simulator to evaluate the performance of the architecture. The measurements of search overhead, and success rate were tested in a large-scale network.

Our first concern was to compare the performance of the architecture in building SSONs with limited-flooding and limited-scope approaches. Limited-flooding has been predominantly used to discover services in environments such as ad hoc and pervasive networks. In a limited-flooding protocol, a service request is broadcast to all direct neighbors of the requesting node. Close neighbors send it on to their neighbors; the propagation is controlled by a TTL value that indicates how far the query should be sent from the requesting node. In a limited-scope approach, the service request is sent to nodes that bring it progressively closer to the destination and located within a scope angle.

The topology used had 2000 nodes in a 1000 X 1000 node two-dimensional overlay space. The bandwidth assigned to each node was randomly selected between 128 and 512 kbits/s. The links propagation delay was fixed at 1 ms. To follow a flash crowd characteristic, all nodes issued their queries at a random point during the first 30 seconds, with the simulation lasting for another 1000 seconds. We ran the simulation a number of times with different search scope values (relative value, similar to TTL except it measures how far the query travels in the network in terms of network distance which is a relatively stable characteristic).

4.1 Network Load

This quantifies the cost of each approach. That is the total number of messages used to construct an SSON. Fig. 6 shows that limited-flooding has the worst performance: it produces a greater number of search messages, except in searches with small scope values. But as the search scope increases, the number of messages in limited-flooding and limited-scope is at least two times higher than the number of messages in our approach.

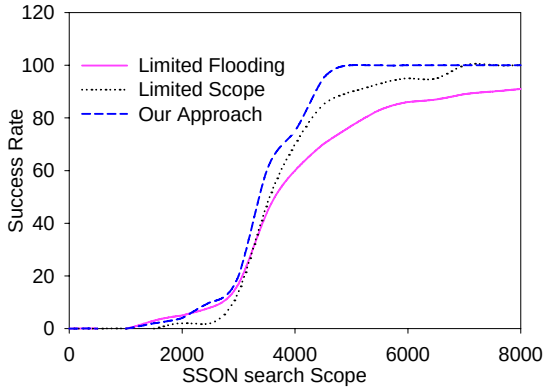

Fig. 7. Success rate.

4.2 Success Rate

Success rate measures the accuracy of each approach. Success rate is defined as the number of requests that receive positive responses, divided by the total number of requests. Fig. 7 shows that our approach results in a higher success rate. Limited-flooding reach the 100% success rate after a 4000 search scope value. While limited-scope approach attains it after a 6500 search scope value. However,

our approach reaches the 100% success rate earlier. We believe that this is due to the huge network load generated by limited-flooding. For large search scope values, limited-flooding generates a large number of messages and receives a large number of reply messages. As a consequence, messages are dropped or lost due to collisions.

5 CONCLUSION AND FUTURE WORK

This work-in-progress provides a complete integrated architecture for autonomic SSONs management. It shows how it can be useful to avoid the complexity of existing service management systems. The road towards fully autonomic system architecture is still long; however, this paper presents an autonomic overlay architecture that represents the basic building blocks needed by autonomic systems. The creation, provision, management and termination of SSONs are automated dynamically based on the context information available from the user, service, and the network provider. The required knowledge capability, reasoning capability, and the different autonomic manager components capabilities are being studied. The PG and the CRA components are being investigated in terms of their requirements and implementation to generate different types of policies. The dual goals of these policies are to drive the autonomic system and dynamically manage SSONs.

Acknowledgments

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

REFERENCES

1. S. Schmid, F. Hartung, M. Kampmann, S. Herborn, and J. Rey, “SMART: Intelligent Multimedia Routing and Adaptation based on Service Specific Overlay Networks,” In Proc. of Eurescom Summit 2005, Heidelberg, Germany. pp. 69-77, 2005.
2. N. Niebert, A. Schieder, H. Abramowicz, G. Malmgren, J. Sachs, U. Horn, C. Prehofer, and H. Karl, "Ambient Networks -- An Architecture for Communication Networks Beyond 3G," IEEE Wireless Communications (Special Issue on 4G Mobile Communications -- Towards Open Wireless Architecture), 2004.
3. W. T. Ooi, and R. van Renesse, “The design and implementation of programmable media gateways”, In Proc.NOSSDAV’00, Chapel Hill, NC, Jun. 2000.
4. J.Kephart, and D.Chess, “The vision of autonomic computing,” IEEE Computer Mag.. Vol. 36, No.1, PP.41–50. Jan. 2003.

5. IBM Corporation, “An architectural blueprint for autonomic computing,” White Paper, Jun. 2006.
6. IBM Corporation,”Autonomic computing - a manifesto,” http://www.research.ibm.com/autonomic /manifesto/, Oct. 2001.
7. J.Kephart , “Research Challenges of Autonomic Computing,” Proc. of the 27th int. conf. on Software Engineering (ICSE’05), St. Louis, Missouri, P. 15 – 22, May 15–21, 2005.
8. A. C. Forum, “Autonomic communication.” http:// www.autonomic-communication.org.
9. R.Farha and A.Leon-Garcia, “Blueprint for an Autonomic Service Architecture” 2006
10. J.Nichols, H.Demirkan, and M.Goul, “Autonomic Workflow Execution in the Grid,” IEEE Tran. on Systems, Man, and Cybernetics—Part C: Applications And Review, VOL. 36, NO. 3, MAY 2006.
11. E.Kasten and P.McKinley, “MESO: Supporting Online Decision Making in Autonomic Computing Systems,” IEEE Trans. on Knowledge And Data Engineering, VOL. 19, NO. 4, April 2007.
12. R.Bahati, M.Bauer, E.Vieira , O. Baek, and C.Ahn, “Using Policies to Drive Autonomic Management,” Proc. of the Int. Sym.. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06), 2006.
13. J. Pena, M.Hinchey, R. Sterritt A.Ruiz-Cortes and M Resinas, “A Model-Driven Architecture Approach for Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems,” Proc.of the 2nd IEEE Int. Symposium on Dependable, Autonomic and Secure Computing (DASC'06), 2006
14. F.Zhang, J.Gao, B.Liao, “Policy-Driven Model for Autonomic Management of Web Services Using MAS,” Proc. of the 5th Int. Conf. on Machine Learning and Cybernetics, Dalian, 13-16 Aug. 2006.
15. S.Balasubramaniam, K.Barrett, W.Donnelly, S.Meer and J.Strassner, “ Bio-inspired Policy Based Management (bioPBM) for Autonomic Communications Systems,” Proc. of the 7th IEEE Int. Work. on Policies for Distributed Systems and Networks (POLICY'06), 2006
16. P.McKinley, F.Samimi, J.Shapiro, and C.Tang, “Service Clouds: A Distributed Infrastructure for Constructing Autonomic Communication Services,” Proc. of the 2nd IEEE Int. Symposium on Dependable, Autonomic and Secure Computing (DASC'06), 2006.
17. D. Xiangdong, S.Hariri, L.Xue, H.Chen, M.Zhang, S.Pavuluri, and S.Rao, “Autonomia: an autonomic computing environment,” in Proc. of the IEEE Int. Performance, Computing, and Communications Conf., pp. 61–68, Apr. 2003.
18. P.Grace, G.Coulson, G.Blair, L.Mathy, W.Yeung, W.Cai, D. Duce, and C. Cooper, “GRIDKIT: pluggable overlay networks for grid computing,” in Proc. of the distributed objects and applications conf. (DOA'04), Cyprus, p1463-81, October 2004.
19. M.Parashar, H.Liu, Z.Li, V.Matossian, C.Schmidt, G.Zhang, and S.Hariri, “AutoMate: enabling autonomic applications on the grid,” Cluster Computing, Vol. 9, No. 6, PP.161–174, 2006.
20. D.Chess, A.Segal, I.Whalley, and S.White, “Unity: Experiences with a Prototype Autonomic Computing System,” Proc. of the Int. Conf. on Autonomic Computing (ICAC’04), 2004.
21. S.Dobson, S.Denazis, A.Fernández, D.Gaïti, E.Gelenbe, F.Massacci, P.Nixon, F.Saffre, N.Schmidt, F.Zambonelli and A.Fernández, “A survey of autonomic communications,” ACM Transactions on. Autonomous and Adaptive Systems, Vol. 1, No. 2, P. 223-259, Dec.2006.