An Autonomic Monitoring Framework for QoS Management in Multi-service Networks

Constantinos Marinos, Christos Argyropoulos,
Mary Grammatikou, and Vasilis Maglaris

Network Management & Optimal Design Laboratory (NETMODE)
School of Electrical & Computer Enigineering
National Technical University of Athens (NTUA)
Tel.: +30 210.772.1451, Fax: +30 210.772.1452
{cmarinos,cargious,mary,maglaris}@netmode.ntua.gr

Abstract. Autonomic monitoring procedures in multi-service networks provide not only feedback to end users, but also self-handling monitoring events to network operators. In this work, we present an autonomic monitoring framework for Quality of Service (QoS) management in multi-service networks. Our framework introduces aggregation mechanisms to deal with the excessive number of alarms, triggered in an autonomic networking environment. The proposed framework was assessed via an early prototype, deployed to IPv6 end-sites, distributed across Europe and interconnected via the Internet.

Keywords: autonomic monitoring, QoS management, multi-service networks.

1 Introduction

End-to-end Quality of Service (QoS) for multi-domain service calls remains a challenge for next generation networks, as various real-time and bandwidth-sensitive applications are all migrating to an IP based architecture. Popular applications such as Video on Demand, Voice over IP, IPTV, online gaming require strict QoS and underlying Quality of Experience.

Increasing complexity and size of computer systems and networks lead to more complicated and distributed management systems. Moreover, multiple end-node services probe network elements or the operating system, in many cases simultaneously, with no coordination. Hence unreliable measurements may be taken causing unpredictable system behaviors or service degradation due to tuned measurements. Autonomic computing and autonomic networking have been arisen as the key concept in the effort of complexity confrontation. Autonomic computing and networking exhibit several self-management properties that will characterize autonomic systems like: self-configuration, self-healing, self-optimization and self-protection [1], [2], [3]. One of the main pillars of the autonomic networking is the self-monitoring mechanism. Self-monitoring is a fundamental operation in the autonomous systems providing not only a feedback to the other entities of the autonomic framework for the changing conditions, but also providing the capability of self-handling health measurements, network measurements and alarms. When an incident occurs in an

K. Pentikousis et al. (Eds.): MONAMI 2010, LNICST 68, pp. 79–86, 2011.

autonomic environment self-management entities accomplish their functions by taking the appropriate actions, without human intervention. Following the design principles of an emerging Generic Autonomic Network Architecture (GANA) [4] for autonomic networking we present the architecture of a framework that orchestrates several monitoring entities, while handling and controlling the monitoring processes.

Advanced monitoring tools that have been developed, aiming to achieve more accurate monitoring alarms and notifications, take into account the network topology and services dependencies. Mechanisms that have as an input host and network dependencies use this information to diagnose fast and accurately, where and what problem occurred within the network. When an incident is classified as a host or service event, according to specific thresholds, the monitoring mechanism activates multiple checks. The self-monitoring mechanism defines if the specific service is actually the root cause of the emerged event or another one exists that produce collateral effects.

This paper focuses on the design and development of an autonomic monitoring framework[1] to maximize the experienced QoS of the end user during an inter-domain application. In this work we propose a dynamic self-monitoring mechanism introducing the Orchestration Engine entity with self-adapting characteristics. The main goal is to design a mechanism that it could have the ability to be self-adjusted in a varying network topology and different services. Furthermore the proposed entity will have the ability to monitor the overall node health status by collecting information, keeping monitoring records and aggregating information in different time instances, collected by different sensor interfaces.

The modular architecture of the Autonomic Monitoring Framework is based on the separation of monitoring tools and business logic mechanism. This permits the harvest of different monitoring data from different protocols and mechanisms. The different interfaces act as middleware between monitoring architectures of heterogeneous networks, regardless of the protocols and data models used by each network. In addition, the monitoring framework should have the ability to set a hierarchical map of dependencies for the hosts and the services that are under surveillance according to different policies.

The rest of the paper is organized as follows: Section 2 presents related work; Section 3 describes the autonomic framework along with its core components and its architecture for monitoring procedures. In section 4 the evaluation and testing of the framework is performed through a multi-domain application scenario. Finally in section 5 we provide some concluding remarks, including challenging issues that are part of our current and future work.

2 Related Work

To date several frameworks have been proposed to support the monitoring for specific end-to-end (e2e) network metrics like packet loss, RTT, delay, bandwidth, jitter. Each framework has its own methodology based on installing measurement points either at

[1] According to [14] "Frameworks are a special case of software libraries in that they are reusable abstractions of code wrapped in a well-defined Application programming interface (API)".

the end nodes (sites), i.e. the receiver or the sender, or by setting a measuring point at an intermediate node along the path. Notice that some of the methods make assumptions for a part or the whole e2e path, based on intermediate measurement points.

Monitoring specific QoS metrics between client and server is a well known problem. The authors of [11] outline a method that measures round-trip-time in the three-way handshake at the beginning of every TCP connection. In [12] two methods for measuring RTT from an intermediate node are examined.

With respect to the above proposals, our framework differentiates from the others in the sense that it introduces an autonomic mechanism for the detection and reconfiguration of the measurement mechanisms. Additionally, our monitoring framework has been deployed in IPv6 sites, hosted by four European partners of the FP7 European Commission Project EFIPSANS [1].

3 Framework Architecture

In this section, we describe the main principles that must be followed by a multi-domain performance monitoring tool for accurate measurements. We then describe the logical view of our architecture schema and present the development of a framework prototype.

Our monitoring framework should be supported in each end-site of a service path. In the multi-domain environment of the Internet, support of the framework in intermediate points of the path would enhance the accuracy of e2e measurements. Well defined APIs carry out the communication between frameworks of the involved sites. When an alarm or notification occurs the monitoring framework collects and handles the event without administrators interfere. In this way, an autonomic approach is achieved with the integration of the corresponding entities.

3.1 Design Objectives

Three different methods of monitoring are being used to aggregate network level performance assessment: active, passive and piggybacking. Modern network infrastructures use these main techniques in order to collect and analyze measurements.

Active measurements require test-packet generation into the network. Traditionally, active measurements include *ping* and *traceroute* utilities. More sophisticated tools like Multicast Beacons [5] have been developed, which emulate application specific traffic and use the obtained results of performance to estimate the end-user Quality of Service. More popular tools for active measurements are iperf [6], bwctl [7], owamp [8] that use sophisticated packet probing techniques.

In comparison to the active measurements, the passive measurements do not produce test packets. They require capturing of packets and their corresponding timestamps transmitted by applications running on network attached devices over various network paths. Some of the popular passive measurement techniques include collecting Simple Network Management Protocol (SNMP) and NetFlow data from network switches and routers. Piggybacking is at the moment mostly a research-oriented approach with time stamping and sequence numbering information being used.

Our Monitoring Framework uses specific monitoring tools that should follow some fundamentals architectural guidelines in order to be in conformance with monitoring

techniques. A monitoring tool that will be installed in an end-to-end path must not interfere with the application. If the monitoring tool generates traffic in order to measure the end-to-end path, that traffic must be low enough so that we will not have any interjections with the running applications. In addition, the tool should support the on-demand measurements at any time. It should be simple and easy to be installed in any node across the network path without consuming too much hardware or network resources. Finally, it should be IPv6 enabled in order to call IPv6 addresses.

Scalability is another critical design goal. At any given time, a varying number of users can make a varying number of measurements. The monitoring framework should be able to support multi-service optimization. This means that the framework should not maximize one service, if this leads to discrimination of another.

3.2 Architecture Overview

In order to build our monitoring framework and manage the Quality of Service along a requested service path, we have applied the three-tier architecture [9]. The core elements of the Monitoring Framework are the Orchestration Engine, the Policy Handling Module and the Event Correlation Module.

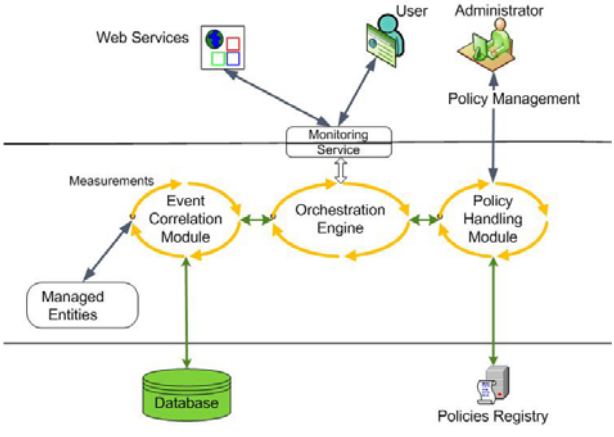

Fig. 1. Framework Architecture

Today's practice of segregated monitoring mechanisms leads to multiple service requests for QoS-related measurements generating redundant measurement data. For this reason the lowest tier, referred to as the Persistence Layer [13], of our Monitoring Framework consists of a database that stores all measurements collected from the different monitoring entities. This database also contains the different policies derived from the Policy Handler and related statistical analysis results. The Policy Repository contains data for the network and system inventory that every service can access. By continuously collecting this information we build up knowledge about the performance of a service like *ftp* or *video on demand* between two specific end-sites. The monitoring entities belong to the Middleware Layer feed the Event Correlator with periodic values for all the network metrics.

The Policy Handler implements the manipulation mechanism of the monitoring architecture. Policies originate either from the Administrator or the entities of the autonomic system with the authority to edit policy profiles. These entities are referred to as *decision elements* in [4].

The Policy Handler is responsible for informing the Orchestration Engine for the policies that govern the different services: It retrieves the different parameters and thresholds that must be fulfilled depending on the requested service, from the database and informs the Orchestration Engine. If, for example, there is a request from the Orchestration Engine for an *IPTV* stream servicing an H.264 video movie file with 24 frames/sec in standard definition, the Policy Handler will respond with a minimum requirement of 15 Mbps bandwidth. The administrator can have access to the policies – add, edit, delete - that are stored in the database through an administrative interface, if he needs to change a specific policy. An additional interface of the Policy Handler permits policy manipulation from an upper layer of self-manageability (autonomicity), such as the *node level* or *network level* in [4]. The existence of this second interface permits the adaptation of the Monitoring Framework to a more generic Reference Model for Self-Managed Networks.

The Orchestration Engine is the core entity that is responsible for the alarm propagation outside the monitoring framework and to respond to any quality control request coming from a user service request. It handles the role of the logic entity which is responsible for the smooth monitoring operation inside the Autonomic Architecture. It is responsible to satisfy any monitoring request providing system and network measurements with an efficient non-redundant way.

Moreover the Orchestration Engine takes care of active measurements scheduling and handling. QoS-related measurements in many cases are CPU and bandwidth demanding, in order to reveal bandwidth bottlenecks and packet forwarding prioritization mechanisms [10]. It schedules active measurements based on information from already stored data; new measurements are executed if data are out-to-date according to thresholds defined by the Policy Handler.

The Policy Handler defines thresholds of measurements concerning specific service policies. It accordingly initiates notifications and alarms. The Orchestration Engine is in charge of deciding for: (i) the scheduling of measurements, (ii) the severity of an alarm, (iii) the impact of the event that triggered an alarm, (iv) notifications that need to be propagated outside the monitoring entity and their verbosity, (v) the abortion of an alarm in case of flapping event state, (vi) the handling of an event without further interactions with other entities.

The Orchestration Engine's Logic combines input from the Event Correlator and Policy Handler to make decisions related to its aforementioned six tasks. The required logic rules manage the measured data from the Event Correlator according to policies taken from the Policy Handler.

The Event Correlator is responsible for: (i) generating messages, (ii) sending them to the Orchestration Engine, (iii) managing the suppression and aggregation of the measurements, (iv) correlating multiple measurements and statistics from different sources (monitoring entities), that form a logical set, escalate the severity of a notification and create a new notifications, (v) binding repeated measurements, (vi) correlating measurements based on service dependencies, (vii) suppressing transient measurements. It may combine inputs directly from the different Monitoring Entities, but also from monitoring data stored to a local repository.

Requests that could trigger the Orchestration Engine could be a query from a user that requests a custom measurement between two nodes during a streaming service; or it could be a notification from the Event Correlator about a service threshold violation. In the first case, the Orchestration Engine would send a message to the Event Correlator in order to notify the responsible monitoring entities for the custom measurement. In the second case, if a violation of a threshold service would take place, a notification would be sent to the Orchestration Engine from the Event Correlator. Then the Orchestration Engine could re-initiate the service and inform the administrator for the violation, in order to change the service settings, reconfigure the service, or start troubleshooting process, via the notification interface of the Monitoring Framework.

4 Framework Evaluation

To evaluate our framework, we used a test-bed consisting of four IPv6 enabled nodes, distributed across Europe in partner sites of the EFIPSANS project [1]. A large file was transferred over the Internet; the end-nodes were acting as measurement points with Monitoring Frameworks installed on them. Measurements were collected in a main node, located at the NETMODE Laboratory, National Technical University of Athens (NTUA). The other nodes were hosted by Telefónica in Madrid, the Telecommunications Software & Systems Group (TSSG), Waterford Institute of Technology, Dublin and the Greek Research & Technology Network (GRNET) in Athens.

An Apache HTTP server running at NTUA was playing the role of the framework User Interface (UI). Through this UI, a user could take on-demand measurements and see graphical representations of the various network paths through the four nodes. Different scenarios with respect to packet loss, one-way delay and jitter were tested between different node pairs. For each network pair we ran periodically measurements continuously for 30 days. In the following figures we present some sample measurements obtained by using our Monitoring Framework.

In Figure 2, we show the Bandwidth (Mbps) measurements between two IPv6 nodes. As we can see, it exhibits significant variations ranging from 10Mbps to 70Mbps.

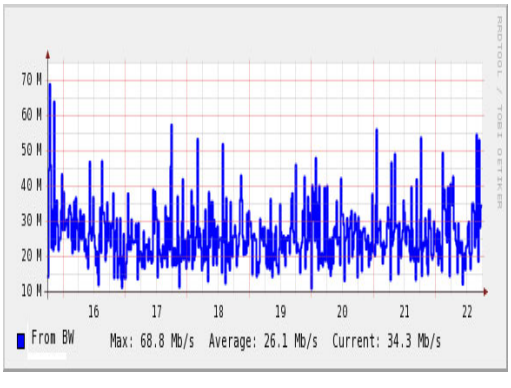

Fig. 2. Bandwidth (Mbps) representation

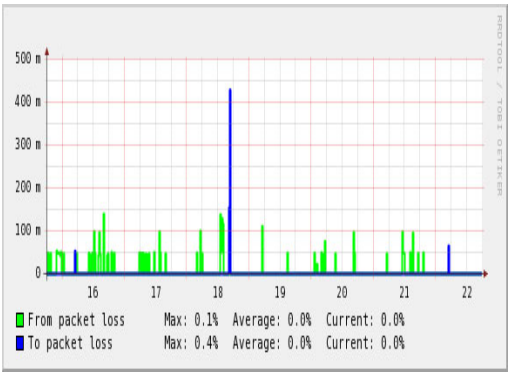

Fig. 3. Packet loss (%) representation

In Figure 3 we show Packet Loss during a week in both directions. It is mostly 0%, but frequently exhibit peaks in both directions. The difference in the two curves may be due to the asymmetric nature of the two directions between the two nodes.

Finally, the jitter (ms) representation is shown in Figure 4. In one direction we can notice that we have almost no jitter with some peaks in some cases, while in the opposite direction we have an average jitter of 1ms.

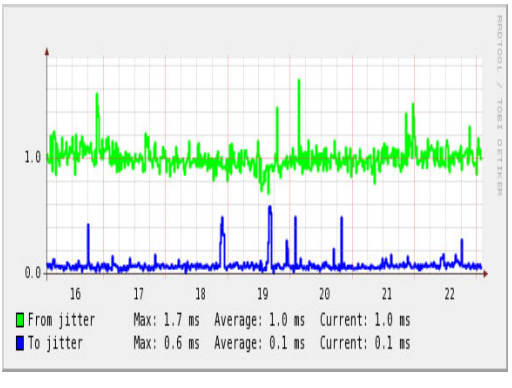

Fig. 4. Jitter (ms) representation

5 Conclusions

In this paper we propose an autonomic Monitoring Framework to assess end-to-end QoS in multi-service networks. As a proof of concept, we designed and developed an autonomic monitoring framework introducing the Orchestration Engine entity with self-adapting and self-monitoring capabilities.

We deployed our prototype in four IPv6 enabled sites across Europe. These sites were interconnected by multi-domain networks, i.e. the commercial Internet and over-provisioned high speed communication networks (NRENs and GÉANT). We

subsequently ran measurements on e2e QoS (packet loss, bandwidth, one-way delay and jitter). Our results are reported in this paper but require further analysis. We are currently considering several extensions, e.g. assessing Quality of Experience (QoE) from QoS metrics for demanding services (streaming of HD video, IPTV, VoIP and Online Games).

Acknowledgments. This paper was partially supported by EFIPSANS project (INFSO-ICT-215549) of the European Union's 7th Framework Programme for Research & Technological Development.

The authors wish to express their thanks to their EFIPSANS collaborators in Madrid, Dublin and Athens that contributed in the test-bed deployment.

References

1. EFIPSANS project, http://www.efipsans.org
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
3. Dobson, S., Denazis, S., Fernandez, A., Gaiti, D., Gelenbe, E.: A Survey of Autonomic Communications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 1(2) (2006)
4. Chaparadza, R.: Requirements for a Generic Autonomic Network Architecture (GANA). Suitable for Standardizable Autonomic Behavior Specifications for Diverse Networking Environments. IEC Annual Review of Communications 61 (2008)
5. Type of Service field in IP packets, http://en.wikipedia.org/wiki/Type_of_Service
6. IPerf, http://www.noc.ucf.edu/Tools/Iperf/
7. Bandwidth Controller, http://www.internet2.edu/performance/bwctl/
8. One-Way Active Measurement Protocol, http://www.internet2.edu/performance/owamp/
9. Three-tier Architecture, http://en.wikipedia.org/wiki/Multitier_architecture
10. Lu, G., Chen, Y., Birrer, S., Bustamante, F.E., Li, X.: POPI: A User-Level Tool for Inferring Router Packet Forwarding Priority. IEEE/ACM Transactions on Networking 18(1), 1–14 (2010)
11. Jiang, H., Dovrolis, C.: Passive estimation of tcp round-trip times (2002)
12. Veal, B., Li, K., Lowenthal, D.: New methods for passive estimation of TCP round-trip times. In: Dovrolis, C. (ed.) PAM 2005. LNCS, vol. 3431, pp. 121–134. Springer, Heidelberg (2005)
13. Persistent Data Structure, http://en.wikipedia.org/wiki/Persistent_data_structure
14. Software Framework, http://en.wikipedia.org/wiki/Software_framework