Maintenance of Monitoring Systems Throughout Self-healing Mechanisms

Clarissa Cassales Marquezan[1,2], André Panisson[1], Lisandro Zambenedetti Granville[1], Giorgio Nunzi[2], and Marcus Brunner[2]

[1] Federal University of Rio Grande do Sul - Porto Alegre, Brazil
{clarissa, panisson, granville}@inf.ufrgs.br
[2] NEC Europe Network Laboratories - Heidelberg, Germany
{marquezan, nunzi, brunner}@nw.neclab.eu

Abstract. Monitoring is essential in modern network management. However, current monitoring systems are unable to recover their internal faulty entities forcing the network administrator to manually fix the occasionally broken monitoring solution. In this paper we address this issue by introducing a self-healing monitoring solution. This solution is described considering a scenario of a monitoring system for a Network Access Control (NAC) installation. The proposed solution combines the availability provided by P2P-based overlays with self-healing abilities. This paper also describes a set of experimental evaluations whose results present the tradeoff between the time required to recover the monitoring infrastructure when failures occur, and the associated bandwidth consumed in this process. Based on the experiments we show that it is possible to improve availability and robustness with minimum human intervention.

1 Introduction

Network and service monitoring is an activity essential to identif problems in underlying IT communication infrastructures of modern organizations. Monitoring is typically materialized by systems that periodically contact elements (*e.g.*, network devices and services) to check their availability and internal status. A monitoring system may be simple (like the Multi Router Traffic Grapher (MRTG) [1]) or complex, being composed of as diverse entities as monitors, agents, and event notifiers. The information collected and processed by monitoring systems enables human administrators, responsible for managing the IT infrastructure, to identify (and possibly predict) problems, and thus react in order to keep the managed infrastructure operating in a proper way.

Monitoring systems must run uninterruptedly to ensure that failures in the managed elements are detected. Problems in the monitoring systems break the monitoring process and can lead the human administrator to believe that the managed elements are working properly even when they are not. Robust monitoring systems should thus employ mechanisms not only to identify failures on the managed infrastructure, but also to recover the faulty monitoring solution itself. Currently, however, most monitoring systems force the administrator to manually recover the occasionally broken solution. Such a manual approach may not drastically affect the monitoring of small networks, but in larger infrastructures the approach will not scale and should be replaced efficient

F. De Turck, W. Kellerer, and G. Kormentzas (Eds.): DSOM 2008, LNCS 5273, pp. 176–188, 2008.

alternatives. The self-managed approach is one alternative emerging as a solution for the manual approach. Typically, a self-managed system is built on top of self-* features capable to reduce the human intervention and provide more efficient results.

In this paper we address the problem of monitoring systems that lack self-healingness feature by considering the example of a Network Access Control (NAC) [2] installation. A NAC secured network is composed of devices and services (*e.g.*, routers, firewalls, RADIUS servers) that control how users and devices join the network. Traditional monitoring systems (*i.e.*, without self-healing support) fail to protect NAC, for example, in two situations. First, consider a crashed RADIUS server whose associated RADIUS monitor crashed too. In this case, the administrator reacts to the RADIUS problem only when users complain about unsuccessful login attempts. Worse than that, however, is the second situation. Suppose a failure in the *rogue user* service responsible for detecting unregistered devices, and another failure in the monitor associated to it. In this case, unregistered devices will silently join the network without generating user complains. In contrast to the first situation, the "silent failure" remains because no signal is issued either by network users or, and most seriously, by the now broken monitoring system.

Recent researches on autonomic management certainly present self-* concepts that could be used to address the aforementioned problems. Such researches, however, take a mostly abstract approach and rarely touch concrete implementation issues. In this paper, in turn, we investigate the employment of self-* features to actually implement, deploy, and evaluate a self-healing monitoring system able to recover from internal failures without requiring, at some extend, human intervention. The goal of our research is to understand the advantages and drawbacks of using self-* features in a real scenario of a service monitoring system.

The monitoring elements of our solution implement two main processes: regular monitoring (to monitor final devices and services) and recovery (to heal the monitoring system). We evaluate our solution in terms of recovery time when fail-stop crashes occur in the monitoring system. In addition, we also measure the traffic generated by the communication between the elements of our solution. This leads us to the contribution of showing the tradeoff between the recovery time and associated network traffic. The determination of such tradeoff is important because it shows when a faster recovery process consumes too much network bandwidth. On the other side, it also shows when excessively saved bandwidth leads to services that remain unavailable longer.

The remainder of the paper is organized as follows. In Section 2 we review network monitoring systems in terms of self-* support, distribution, and availability. In Section 3 we introduce our self-healing architecture for NAC monitoring, while in section 4 we evaluate our proposal in an actual testing environment, presenting associated results and their analyses. Finally, the paper is concluded in Section 5.

2 Related Work

Although network and service monitoring is widely addressed by current investigations [3] [4] and products on the market [5], we review in this section solutions that are mainly related to the aspects of self-monitoring and self-healing on distributed monitoring.

Distributed monitoring are specially required in large-scale networks, like country or continental-wide backbones [6] [7]. Most of the research in this area propose complex monitoring system that generally identify internal failures and employ algorithms to reorganize itself without the failed components. In this sense, the solutions present a certain level of self-awareness and adaption, although self-healing is in fact not present, *i.e.*, failing entities are not recovered or replaced. It means that in scenarios where most of the monitoring entities crash, the monitoring systems stop working because no mechanism is employed to maintain the execution of the monitoring entities.

Yalagandula *et al.* [8] propose an architecture for monitoring large networks based on sensors, sensing information backplane, and scalable inference engine. The communication among the entities relies on a P2P management overlay using Distributed Hash Tables (DHTs). Prieto and Stadler [9] introduce a monitoring protocol that uses spanning trees to rebuild the P2P overlay used for the communications among the nodes of the monitoring system. Both Yalagandula *et al.* and Prieto and Stadler work can reorganize the monitoring infrastructure if failures are detected in monitoring nodes. After such reorganization the failing nodes are excluded from the core of the rebuilt monitoring infrastructure. Although reorganized, with a few number of nodes, the monitoring capacity of the system is reduced as a whole. Again, adaptation in the form of infrastructure reorganization is present, but proper self-healing it is not.

Few investigations in fact explicitly employ autonomic computing concepts in system monitoring. Chaparadza *et al.* [10], for example, combine self-* aspects and monitoring techniques to build a traffic self-monitoring system. The authors define that self-monitoring networks are those that autonomously decide which information should be monitored, as well as the moment and local where the monitoring task should take place. Nevertheless, such work does not define how the monitoring system should react in case of failures in its components. The meaning of self-monitoring in this case is different than the one of our work. While self-monitoring in Chaparadza's work means autonomous decision about the monitoring process, in our view self-monitoring is about detecting problems, through monitoring techniques, in the monitoring system itself.

Yangfan Zhou and Michael Lyu [11] present a sensor network monitoring system closer to our view of self-monitoring. The authors' contribution resides on the use of sensors themselves to monitor one another in addition to performing their original task of sensing their surrounding environment. Although self-monitoring is achieved, given the restrictions of the sensor nodes (*e.g.* limited lifetime due to low-capacity batteries) the system cannot heal itself by reactivating dead nodes.

Considering the current state of the art, there is a necessity for new approaches for self-monitoring systems. New proposals should explicitly include, in addition to self-awareness already available in the current investigations, self-healing support on the monitoring entities in order to autonomously keep the monitoring service up. In the next section we thus present our self-healing approach for monitoring infrastructures.

3 Self-healing Architecture for Monitoring Infrastructures

The self-healing architecture built in our investigation forms a P2P management overlay on top of the monitored devices and services. The usage of P2P functionalities in

our architecture provides a transparent mechanism to enable communications target to publish, discover, and access management tasks inside the overlay. In this way, the control of such basic communications is delegated to the P2P framework used to implement our architecture. Furthermore, in a previous work we have presented that network management based on P2P can aggregate benefits, like reliability and scalability, on the execution of management tasks [12]. Now, we use P2P overlays to have the transparency on basic overlay operations, to distribute the identification of failures and also to provide scalability on the recovery process.

The overlay proposed in previous work is called ManP2P and its architecture has been described in the work of Panisson *et al.* [13]. In this current paper, we extend the ManP2P functionalities in order to explicitly support self-healing processes. We believe that combined, self-healing and P2P overlays can bring together a self-monitoring infrastructure able to address current problems on monitoring systems. In this section we review the ManP2P architecture, present self-healing extensions, and exemplifies their employment in the concrete scenario of a NAC installation.

3.1 P2P Management Overlay and Services

The collection of management peers forms the ManP2P management overlay. Each peer runs basic functions (*e.g.*, granting access to other peers to join the overlay or detecting peers that left the P2P network) to maintain the overlay structure. In addition, each peer hosts a set o *management services* instances that execute management task over the managed network. In our specific case, such tasks are monitoring remote equipments. A management service is available if at least one single instance of it is running on the overlay. More instances of the same service, however, must be instantiated in order to implement fault tolerance. Figure 1 exemplifies a scenario where management services (LDAP monitors, Web servers monitors, rogue user monitors) for a NAC installation are deployed on the ManP2P management overlay.

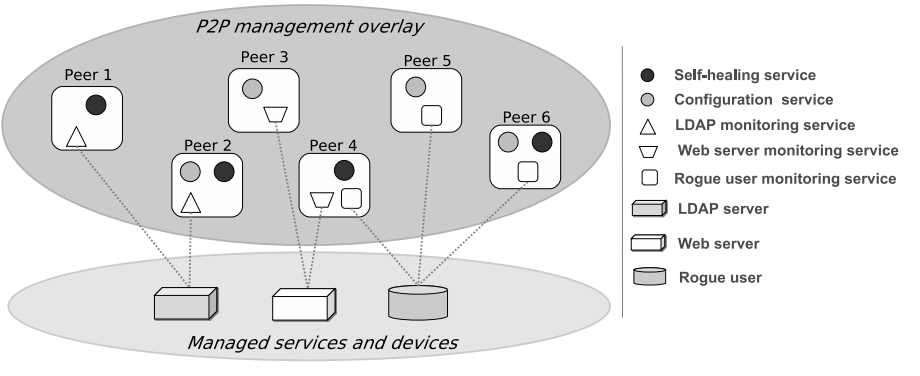

Fig. 1. NAC meta-monitoring infrastructure

In Figure 1, peers #1 and #2 host service instances, pictured as a triangle, that monitor an LDAP server. Peer #4, on its turn, contacts both the Web server and the rogue user service because it hosts management services to monitor these elements. The special services of self-healing and configuration, depicted as black and gray circles respectively, will be explained farther in this paper. Each peer, in summary, may host different services at one. In the extreme cases, there could exist peers with no management services (thus useless peers) or peers hosting one instance of each available management service (this possibly becoming an overloaded peer).

We consider that the a management service is able to heal itself if, after the crashing of some of its instances (possibly due to peers crash), new instances become available, thus recovering the service and guaranteeing its availability. In order to cope with that, two functions must be supported: failure detection and service instance activation.

3.2 Failure Detection

Failures in a management service are detected by a self-monitoring procedure where each service instance, in intervals of t seconds, sends a signal (heartbeat) to all other instances of the same service to inform that the former is running. Self-monitoring, in this sense, means that there is no external entity monitoring the instances of a management service deployed inside the overlay. Indeed, the instances of the management service themselves can monitor their liveness throughout the heartbeat messages. So, if one instance crashes, the other instances will miss the former's heartbeats and then will initiate the process to recover this instance, as it will be explained later on this paper.

Heartbeats that get lost in the network may wrongly suggest the unavailability of a service instance. Instead of immediately assuming an instance as down given the lack of a heartbeat, it first becomes suspect by the other instances. In order to double check the availability of the suspicious instance, one of the other alive instance tries to contact the suspicious instance back. If no contact is possible, the suspicious instance is finally declared unavailable. Assuming s as the time spent to double check the availability of a suspicious instance, the maximum detection time is $td = t + s$.

The distribution of heartbeats from one service instance to all others is accomplished using group communications. At the network level, in the best case, group communication is supported by multicast communications. In this case, the number of heartbeat messages h issued by i service instances in t seconds will be $h = i$. However, if multicasting is not available, the notifying service instance is forced to send, via unicast, copies of the same heartbeat to all other instances. In this case, the number of messages will be $h = i^2 - i$. In this way, the presence of multicasting directly influences the network traffic generated by the failure detection function.

Failure detection is essentially a consensus problem. Solutions on this topic, coming from the dependability field, could be employed and formalisms used to model and validade our detection approach. Instead of that, however, we preferred to use the practical approach of actually implementing the aforementioned function. Although no formal proof is provided, our experiments have shown that this approach is effective in detecting failures in the management service instances.

Table 1. Service policy repository

Management service	*Minimum instances*	*Activate instances*
LDAP monitor	2	1
Web server monitor	2	2
Rogue user monitor	2	1

3.3 Service Instance Activation and Policies

Instance activation is crucial to recover the management service that just lost some of its instances. It is on instance activation that the self-healing and configuration services, presented in Figure 1, play a key role.

Once an instance detects a remote crashed one, it notifies the *self-healing service* that determines how many, if any, new instances of the faulty service must be activated. To do so, the self-healing service internally checks a repository of service policies that describes, for each management service, the minimum number of instances that must be running, as well as the number of new instances that must be activated once the minimum boundary is crossed.

Table 1 shows the service policy repository for the NAC installation of Figure 1. As can be observed, the LDAP monitoring service must have at least 2 instances running. In cause of failure, another new one instance must be activated. In the case of the Web server monitor, on the other hand, although 2 instances are running, whenever activation is required 2 other new instances will be initiated. If the number of remaining running instances of a services is still above the minimum boundary, the self-healing service ignores the faulty service notifications. For example, in the case of the rogue user monitor from Figure 1, if a single instance crashes no action will be executed because the remaining 2 instances do not cross the minimum boundary. Although it is outside the scope of this paper stressing the administration and usage of management service policies (refer to the work of Marquezan *et al.* [14] for that), we assume that policies are defined by the system administrator and transferred to the self-healing service instances long before any failure occurred in the P2P management overlay.

Once required, the self-healing service tries to activate the number of new instances defined in the service policy by contacting the *configuration service*. Such configuration service is then responsible for creating new instances of the faulty service on peers that do not have those instances yet. A peer hosting solely a configuration service can be seen as an spare peer ready to active new instances of any service in failure.

Different than the failure detection function, instance activation is performed outside the group of instances that implement the failing management service. That is so because decoupling the instance activation function from the services that require them allow us to more flexibly deal with the number of components for each function. That directly impact on the number of message exchanged in the overlay.

So far, we have defined a self-healing architecture that extends the ManP2P functionalities. However, to ensure that the failure detection and instance activation functions work properly, two requirements must be filled on the P2P management overlay. First, each management service (including the self-healing and configuration services) must run at least 2 instances in order to detect and recover problems on the management

service. That is so because a single faulty instance cannot react itself if it is crashed, then at least another instance is required. Second, each peer must not host more than one instance of the same management service in order to avoid several instances of that service crashing if the hosting peer crashes too. We assure that the maintenance of the monitoring infrastructure can be accomplished while these requirements are fulfilled.

3.4 System Implementation

As mentioned before, our architecture extends the ManP2P system. The implementation of our architecture in an actual monitoring system is than based on the previous code of ManP2P. Figure 2 depicts the internal componentes of a peer of our solution.

Components are divided by the *core peer plane* and *management service plane*. The core peer plane's components are responsible for controlling the communication mechanisms between peers. At the bottom the *JXTA* and *network multicast* components implement group communication using unicast (via JXTA) or network multicast. On top of them, the *group manager* and *token manager* components control, respectively, group membership and load balancing (via a virtual token ring). Messages are handled by the *message handler* component that interfaces with Axis2 to communicate with the management service plane's components. A ManP2P component on the top of the core peer plane is used to implement complementary functionalities that are not inside the scope of this paper.

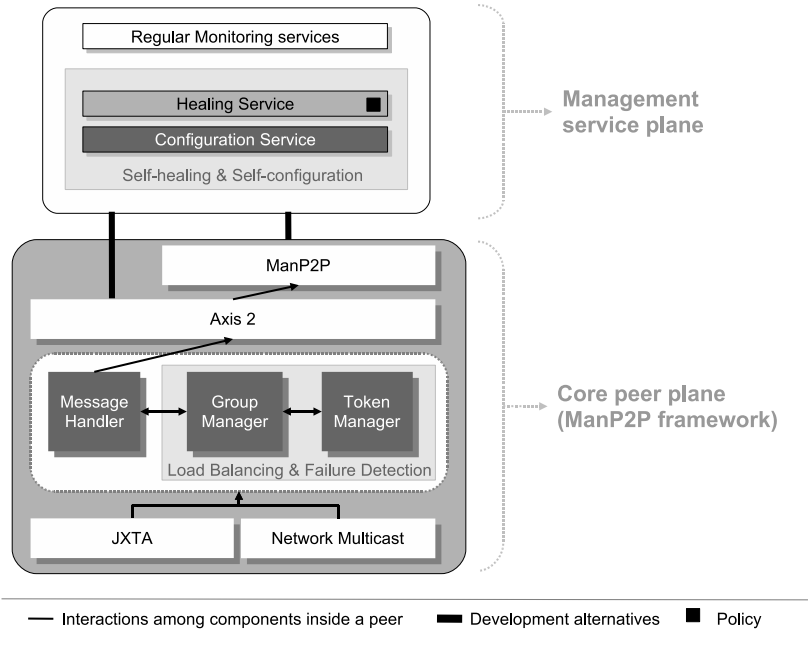

Fig. 2. Meta-monitoring architecture

At the management service plane the regular monitoring services are found. Although located in this plane, monitoring services themselves do not monitor remote instances for fault detection; this verification is in fact performed by the group manager component. That is so because we wanted the self-monitoring function to be native in any peer, freeing the developer of new management services to concentrate their efforts on the management functionalities he/she is coding without worrying about self-monitoring. At the management service plane the self-healing and configuration services are also found. As mentioned before, they are responsible for activating new instances of monitoring services when required. The black little square inside the self-healing service represents the policies that define the minimum number of instances of each management service, as well as the number of new instances that must be activated. Peers and internal monitoring services have been coded in Java using Axis2, JXTA, and ManP2P previously developed libraries. Monitoring services have been specifically developed as dynamic libraries that can be instantiated when required be a hosting peer.

4 Experimental Evaluation

In our experimental evaluation we measured the recovery time and the generated network traffic when fail-stop crashes occur in peers of the proposed self-healing monitoring infrastructure. We evaluate the effects of such failures considering variations on: (a) the number of simultaneously crashing peers, (b) the number of peers in the management overlay, and (c) the number of management services running on the overlay.

We have run our experiments in a high performance cluster, called LabTec from the GPPD research group at UFRGS [15], from which we used 16 nodes to host the management peers of our architecture. The recovery time and the generated traffic have been measured capturing the P2P traffic and timestamping it using a packet capture `tcpdump` software. Traffic volume is calculated considering the headers and payload of all packets generated by the system operations. Recovery time has been measured 30 times for each experimental case and computed with a confidence interval of 95%.

Although the size of P2P systems is typically of scales much higher than 16 nodes, we emphasize here that we do not believe that, in an actual management scenario of a single corporation, administrators would use a large number of managing nodes. We thus assume that 16 peers are sufficient for most actual management environments. Over the P2P management overlay we deployed up to 12 different NAC management services (namely, monitors for LDAP, DNS, DHCP, Radius, data base, Web servers, rogue user, firewall, proxy, access point, switches, and routers), in addition to the self-healing and configuration special services required in the recovery process. The single service policy enforced in all management services of our experiments defines that at least 2 instances per service must be running and, in case of failures, just 1 another instance must be activated per crashed instance.

Considering the above, two main sets of experiments have been carried out: multiple crashing peers, and variable number of peers and services. These experiments and associated results are presented in the next subsections.

4.1 Multiple Crashing Peers

The first experiment was designed to check the performance of the self-healing monitoring architecture when the number of simultaneously crashing peers hosting management services increases until the limit where half of them are broken. In addition, we want to check whether the number of instances of the self-healing and configuration services influences the recovery time and generated traffic.

For this set of experiments, we used to following setup: 12 management services are always deployed, each one with 2 instances running on the overlay. The total 24 service instances (*i.e.*, 12×2) are placed along 8 peers, each one thus hosting 3 (*i.e.*, $24 \div 8$) service instances. The number of crashing peers varies from 1 to 4. Since each peer hosts 3 instances, the number of crashing instances varies from 3 (12.5%) to 12 (50%), out of the total of 24 instances. Additional 4 peers have been used to host the self-healing and configuration services. Their varying number of instances has been organized, in pairs of self-healing/configuration, as follows: 2 and 4 instances, and 4 and 4 instances. Finally, we consider that group communication support is implemented interchangeably using multicast and unicast.

Figure 3 shows in seconds the time taken by the monitoring system to detect and activate new instances of the crashing services using the "spare" cluster nodes that host the configuration service. The first occurrence of 3 crashing services correspond to the situation where 1 peer fails; 6 crashing services correspond to 2 failing peers, and so on. No value is provide in 0 (zero) because with no failing peers there will not be any crashing service. Figure 4, in its turn, presents the network traffic generated by the management overlay in this recovery process. In this case, for 0 (zero) there exists an associated network traffic because, in the self-monitoring process, heartbeat messages are constantly sent regardless the presence or not of a failure.

The recovery time as a function of the number of crashing peers stayed mostly constant. With that we can conclude that the system scales well considering a management scenario of 16 nodes. There is a little variance on the recovery time as a function of the self-healing and configuration services. In fact, such difference is the result of

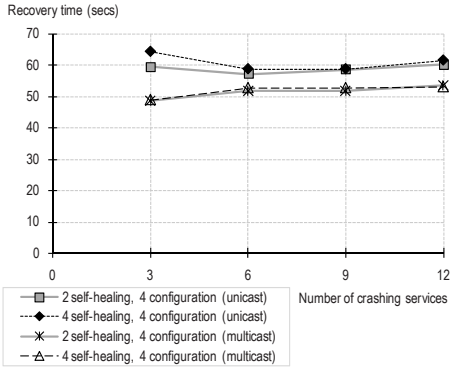

Fig. 3. Recovery time with multiple crashing

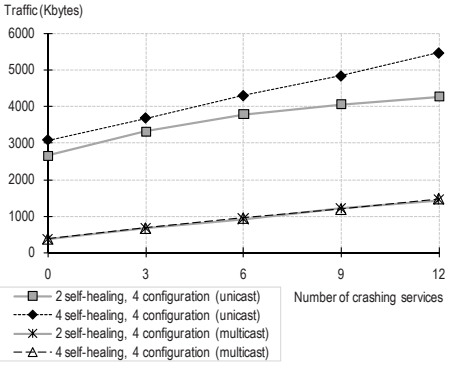

Fig. 4. Traffic to recover crashing peers

employing multicast or unicast. When peers use multicasting they quickly become aware of changes in the system, and can rather react faster. Using unicast, however, more messages are sent, delaying the communication and, as a consequence, the reactions. In summary, the recovery time is not strongly influenced either by the self-healing and configuration services or by the number of crashing services. There is, however, a little influence from the use of multicast or unicast in the group communication support.

Network traffic, in its turn, presents a stronger influence of multicast or unicast support. As can be observed in Figure 4, multicast-based communications saves more bandwidth, which is expected. The important point to be observed, however, is that with the increasing number of crashed services the traffic generated to recover them is closely linear, but with doubling the number of failures, the traffic generate does not double together. Although not so efficient as in the case of recovery time, the bandwidth consumption is still scalable in this case. Putting these two parameters together and observing the graphs, if multicasting is used the number of self-healing and configuration services and the number of crashing peers do not influence the recovery time, and slightly increase the bandwidth consumption. In the case of unicast, however, the option of employing 2 self-healing instances instead of 4 is better, because this setup reacts slightly faster yet generating less traffic.

4.2 Varying Number of Peers and Services

The second experiment shows the relationship between recovery time and generated traffic when single crashes occur (which tends to be more frequent than multiple crashes) but the number of peers and services varies. We consider the recovery process when the number of management services increases (from 1 to 12, *i.e.* from 2 to 24 instances) over three setups where 2, 6, and 12 peers are used to host the management services. In addition to single crashes, we also fixed the number of 2 self-healing and 2 configuration services instances, hosted by 2 peers. We did so because, as observed before, the number of such instances few impacts on the recovery time.

In Figure 5, where the recovery delay is presented, services communicating via multicast are depicted with dashed lines, while services using unicast are depicted with solid

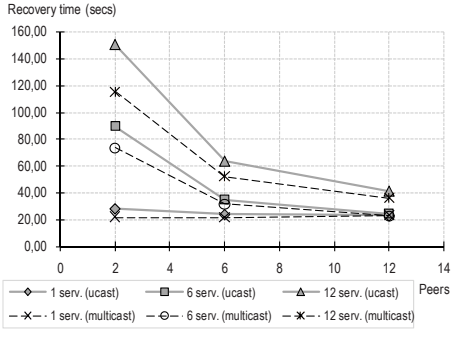

Fig. 5. Recovery time for multiple peers

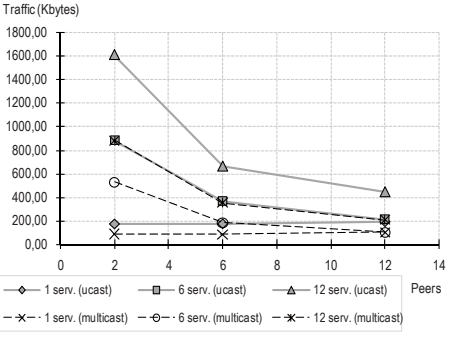

Fig. 6. Recovery traffic with multiple peers

gray lines. The recovery time when only 2 peers are employed is usually higher because each of the 2 peers hosts more service instances. When one of the peers crashes, more instances need to be activated. On the other extreme, with 12 peers, each peer hosts less services, leading to the situation where a crashing peer actually triggers the activation of less service instances.

The fact that more instances need to be activated as the result of a more load peer can be observed in Figure 6, that shows the traffic generated to recover the system. Again, multicast communications save more bandwidth than unicast, as expected. However, it is important to notice now that the number of services in each peer influences too. For example, 6 instances running on the same peer (line "6 serv. multicast", with 2 peers in the x axis) despite being multicast still takes longer and generates more traffic to recover the system than the case where, via unicast, only 1 services is deployed (line "1 serv. unicast", with 2 peers in the x axis).

This confirms that the number of peers and service instances must be similar in order to recover more promptly the system without generating too much traffic. If an administrator is restricted in terms of peers available, he/she must try to restrict the number of services employed as well. If new services are required, however, the option of also increasing the number of peers should be considered.

Now considering the whole picture, administrators should not worry about simultaneous crashes nor the number of self-healing and configuration services. Increased multiple crashes are more scare, and even if they happen the system is able to recover reasonably fast. As observed, the number of serf-healing and configuration services does not affect the overall performance of the system. However, administrator should do pay attention to the number of available peers and service instances, as mentioned before. Finally, the employment of multicast and unicast in the group communication mechanism influences in the recovery time (less) and the generated traffic (more). Choosing multicast whenever possible helps to improve the response time of the system. Unfortunately multicasting is not always available, which forces the administrator to use unicast to implement group communication.

5 Conclusions and Future Work

We have presented in this paper the design and evaluation of a P2P-based self-healing monitoring system employed in a NAC environment. The solution achieves self-healing capacity by splitting in two different processes the functions of failure detection and system recovery. Failure detection is executed inside management services that monitor final devices, while system recovery relies on special services called self-healing (that decides when new service instances must be activated) and configuration (that activates the new service instance as an reaction for the self-healing service decision).

The results of our experimental evaluations allow us to conclude that the number of instances of the self-healing and configuration service is not a major player in the performance of the system. They also permit us to state that simultaneously crashes on the management services does not influences so expressively the system performance either. A network administrator willing to employ a self-healing monitoring solution should not concentrate his/her efforts in finding an ideal number of self-heling and

configuration services. Our experiments employed 2 and 4 instances, respectively, and the system response was satisfactory. The fact that must be observed, however, is the group communication solution available on the managed network: multicast turns recovery faster while consuming less network bandwidth. Unfortunately, IP multicasting can not always be provided, and unicast ends up being chosen in such situations.

The most important aspects that must be observed in a self-healing monitoring solution is the number of peers employed in the P2P management overlay and the number of service instances deployed. With few instances, there is no need for several peers. On the other hand, with a large number of instances the number of peers should grow consistently, otherwise, on the occurrence of a failure, the recovery time will be higher and more network bandwidth is consumed by the intensive P2P traffic generated.

Currently, we are working on the optimization of the detection mechanism because the current version of it is responsible for a considerable amount of generated traffic. Another future work is the investigation about how service policies impact on the consumed network bandwidth and recovery time of our system.

Acknowledgement

Thanks to Dominique Dudkowski and Chiara Mingardi for contributing with this paper, and also to the Network and Support Division team at Data Processing Center of UFRGS for the experience exchanged on the development of the UFRGS NAC system.

This work was partly funded by the Brazilian Ministry of Education (MEC/CAPES, PDEE Program, process 4436075), and by the European Union through the 4WARD project in the 7th Framework Programme. The views expressed in this paper are solely those of the authors and do not necessarily represent the views of their employers, the 4WARD project, or the Commission of the European Union.

References

1. Oetiker, T.: MRTG - The Multi Router Traffic Grapher. In: LISA 1998: Proceedings of the 12th USENIX conference on System administration, Berkeley, CA, USA, USENIX Association, pp. 141–148 (1998)
2. López, G., Cánovas, O., Gómez, A.F., Jiménez, J.D., Marín, R.: A network access control approach based on the AAA architecture and authorization attributes. J. Netw. Comput. Appl. 30(3), 900–919 (2007)
3. Perazolo, M.: A Self-Management Method for Cross-Analysis of Network and Application Problems. In: 2nd IEEE Workshop on Autonomic Communications and Network Management (ACNM 2008) (2008)
4. Trimintzios, P., Polychronakis, M., Papadogiannakis, A., Foukarakis, M., Markatos, E., Oslebo, A.: DiMAPI: An Application Programming Interface for Distributed Network Monitoring. In: Proceedings. 10th IEEE/IFIP Network Operations and Management Symposium, 2006. NOMS 2006, pp. 382–393 (2006)
5. Packard, H.: Management Software: HP OpenView (2008), `http://www.openview.hp.com/`
6. Agarwal, M.K.: Eigen Space Based Method for Detecting Faulty Nodes in Large Scale Enterprise Systems. In: IEEE/IFIP Network Operations and Management Symposium (NOMS 2008) (2008); CDROM

7. Varga, P., Moldován, I.: Integration of Service-Level Monitoring with Fault Management for End-to-End Multi-Provider Ethernet Services. IEEE Transactions on Network and Service Management 4(1), 28–38 (2007)
8. Yalagandula, P., Sharma, P., Banerjee, S., Basu, S., Lee, S.J.: S3: a scalable sensing service for monitoring large networked systems. In: INM 2006: Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management, pp. 71–76. ACM Press, New York (2006)
9. Prieto, A.G., Stadler, R.: A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives. IEEE Transactions on Network and Service Management 4(1), 2–12 (2007)
10. Chaparadza, R., Coskun, H., Schieferdecker, I.: Addressing some challenges in autonomic monitoring in self-managing networks. In: 13th IEEE International Conference on Networks, p. 6 (2005); CDROM
11. Zhou, Y., Lyu, M.R.: An Energy-Efficient Mechanism for Self-Monitoring Sensor Web. In: 2007 IEEE Aerospace Conference, pp. 1–8 (2007)
12. Granville, L.Z., da Rosa, D.M., Panisson, A., Melchiors, C., Almeida, M.J.B., Tarouco, L.M.R.: Managing Computer Networks Using Peer-to-Peer Technologies. IEEE Communications Magazine 43(10), 62–68 (2005)
13. Panisson, A., Melchiors, C., Granville, L.Z., Almeida, M.J.B., Tarouco, L.M.R.: Designing the Architecture of P2P-Based network Management Systems. In: Proceedings. IEEE Symposium on Computers and Communications (ISCC 2006), pp. 69–75. IEEE Computer Society, Los Alamitos (2006)
14. Marquezan, C.C., dos Santos, C.R.P., Nobre, J.C., Almeida, M.J.B., Tarouco, L.M.R., Granville, L.Z.: Self-managed Services over a P2P-based Network Management Overlay. In: Proc. 2nd Latin American Autonomic Computing Symposium (LAACS 2007) (2007)
15. GPPD: Parallel and Distributed Processing Group – GPPD (2008), `http://gppd.inf.ufrgs.br/new/`