# Future Internet Autonomic Management Using NetServ

M. Femminella, R. Francescangeli, G. Reali

DIEI – University of Perugia
Perugia, Italy
{femminella,francescangeli,reali}@diei.unipg.it

J.W. Lee, W. Song, H. Schulzrinne

CS - Columbia University
New York, USA
{jae,wonsang,hgs}@cs.columbia.edu

***Abstract*— This demo proposal describes an autonomic management solution based on the recently defined programmable node architecture NetServ.**

## I. INTRODUCTION

This demo proposal shows how the NetServ platform can be used for implementing autonomic management architectures for the Future Internet. The NetServ-based management architecture that will be shown in the LCN demo session, fully exploits the NetServ dynamic properties. This translates to capabilities of automatically deploying, configuring, and removing at runtime both Policy Decision Point (PDP) and Policy Enforcement Point (PEP) modules on network nodes, in order to provide network management with effective autonomic capabilities. In fact, the usage of programmable nodes able to host any service, made up by combining inferential, decisional, monitoring, and actuator modules, represents a powerful instrument to implement autonomic network management functions.

In what follows, we present a novel solution for deploying autonomic network and service management architectures. We do not aim at introducing new management paradigms, but rather to increase the effectiveness of the existing ones by resorting to the potential provided by the NetServ project, which is a framework designed to deploy and execute networked services at runtime over programmable routers. The use of the NetServ capabilities in the management planes represents a step forward the state of the art, since it increases the flexibility of management solutions, their dynamic response to event requiring management actions, decreases the relevant traffic, and decreases the response time. In order to show the effectiveness of the proposed solution, we will show a case study which highlights how NetServ allows deploying self-protecting network functions. In the proposed demo, that will be carried out using the GENI testbed, we will show how the NetServ-based management architecture is able to counteract a DoS attack by selectively deploying monitoring and actuator modules at runtime.

This demo proposal is organized as follows. The next section provides an overview of the NetServ architecture. Section III describes the autonomic management scenario to be shown in the LCN demo session. Section IV lists the equipment and facilities needed along with space and time requirements. Section V gives some final remarks.

## II. NETSERV

NetServ is a programmable node architecture designed for deploying in-network services [1]. It is suited for any types of nodes, such as routers, set-top boxes, and user equipment. NetServ includes an in-network virtualized service container and a common execution environment for both network services and traditional addressable services (e.g. a Web server). NetServ is thus able to fill the gap between these two types of services that have traditionally been kept separated in the Internet architecture. In this way administrators can be provided with a suitable flexibility to optimize resource exploitation.

The NetServ prototype architecture is shown in Figure 1. It is currently based on the Linux operating system. It includes an NSIS-based signaling protocol [2], used for dynamic NetServ node discovery and service modules deployment therein. The NetServ controller coordinates NSIS signaling daemons, service containers, and the node transport layer. It receives control commands from the NSIS signaling daemons, which may trigger installation or removal of both application modules within service containers and filtering rules in the data plane. Each deployed module has a lifetime associated with it and it needs to be refreshed by a specific signaling exchange before its lifetime expiration, otherwise it is automatically removed. The NetServ controller is also in charge of setting up and tearing down service containers, authenticating users, fetching and isolating modules, and managing service policies.

Service containers are user space processes. Each container includes a Java Virtual Machine (JVM), executing the OSGi framework [3] for hosting service modules. Each container may handle different service modules, which are OSGi-compliant Java archive files, referred to as *bundles*.

The OSGi framework allows for bundles hot-deployment. Hence, the NetServ controller may install modules in service containers, or remove them, at runtime, without requiring JVM reboot.

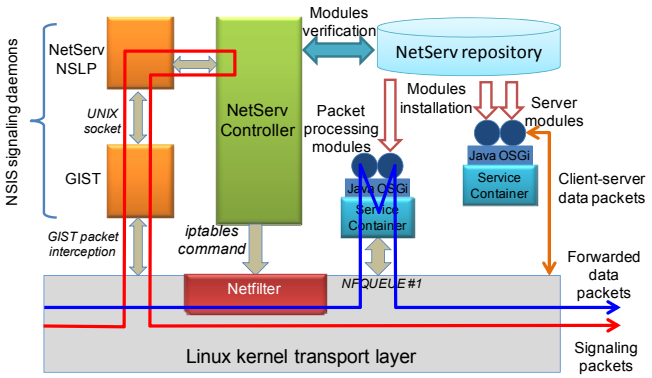


Figure 1 – NetServ node internal architecture.

Service modules, represented by circles in Figure 1, are OSGi bundles deployed in a service container. Figure 1 shows two types of modules:

- *Server modules*, circles located within the upper-right service container. They act as standard network servers, communicating with the external through a TCP or UDP port.
- *Packet processing modules*, circles located within the lower-left container. They are deployed in routers along packet path and can both inspect and modify packets in transit. The blue arrow in Figure 1 labeled "forwarded data packets" shows how an incoming packet is routed from the network interface, through the kernel, to a service container process being executed in user space. The packet is handled by two different modules before being sent back to the kernel and routed towards its final destination.

The module classification as *server module* or *packet processing module* is only logical, since each NetServ module may act in both ways. This is actually an important NetServ feature since it overcomes the traditional distinction between router and server by sharing each other's capabilities.

The *NetServ repository*, introduced in the NetServ architecture for management purposes, includes a pool of management programs deployable through NetServ signaling in the NetServ nodes present in the managed network.

Currently, the Linux kernel is used to implement the NetServ transport layer. Packet filters, used to intercept packets in the NetServ node, and rules, used to route them to the proper service container, are installed in the node forwarding plane by using the *netfilter* library through the *iptables* tool.

## III. AUTONOMIC MANAGEMENT ARCHITECTURE

The key element of our management architecture is the NetServ Autonomic Management Element (NAME). It is inspired by the FOCALE architecture shown in [4], which has been mapped into the service deployment architecture shown in Figure 1. This decision follows from the consideration that the FOCALE architecture already includes most of enabling mechanisms for autonomic network management and its modularity allows integrating the unique features of NetServ that, we believe, may introduce significant dynamics in network and service management. The NetServ additional functions are included in this architecture by implementing it as a NetServ service, and by also introducing the PEP (policy enforcement point) deployment module, which can deploy management programs over the selected NetServ managed resources at runtime. These programs are stored in the *NetServ repository*.

## IV. DEMO SCENARIO

This section describes the autonomic management scenario we would like to show at the LCN demo session. The demo will highlight the NAME effectiveness in self-protecting a network resource from a DoS attack, one of the most important Internet security threats [5]. The attack shown in the demo will just be a sample of a generic DoS attack, but it will be sufficiently structured to show the NetServ dynamic properties brought to the management architecture.

Figure 2 shows the network topology in this experiment, which we will deploy in the well-known GENI (Global Environment for Network Innovation, [6]) experimental platform. The victim, an application server, is protected by a NAME instance. The attack is a classic DoS flooding attack, performed by a number of hosts in different networks [5].

A lightweight NetServ service module, called Rate_Monitor, is executed in the NAME itself and evaluates the rate of incoming traffic and notifies the PDP module. When the attack starts (see Figure 3 at time $t_1$), the local Rate_Monitor notifies the NAME engine the value of the incoming rate above the alarm threshold. This information reveals that the network has entered an unacceptable state. The set of actions deemed necessary for leading the system to an acceptable state are:

- retrieval of a Rate_Limiter module from the NetServ repository and its deployment on the local interface, in order to protect the victim against the overwhelming service requests;
- deployment of a number of Rate_Monitor modules in the NetServ nodes all around the NAME instance, by means of epidemic signalling or directory service, so as to identify the incoming attack directions and deploy additional Rate_limiter modules on nodes where the observed value of the incoming service requests are above a given threshold.

The objective of the second action is twofold. First, any attack direction can be identified and the attack can be faced upstream. Second, in this way we relieve the network from the traffic generated by the attackers (denial of network service).

In order to execute the second action, the NAME instance starts sending NetServ PROBE messages towards all directions from itself up to three IP hops[1], so as to identify the NetServ nodes able to host and execute an instance of the Rate_Monitor module (see Figure 3). Then, by using the NetServ deployment signaling, the NAME engine deploys a Rate_Monitor module on the selected nodes, which immediately start reporting incoming rate values.

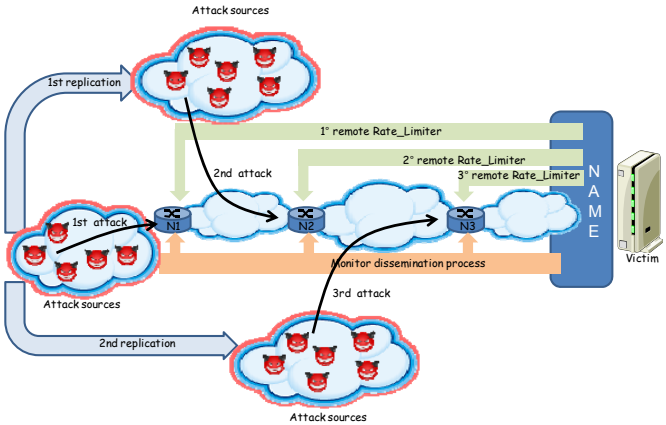


Figure 2 - Network topology for our DoS scenario.

[1] This value may be changed according to network topology and management purposes.

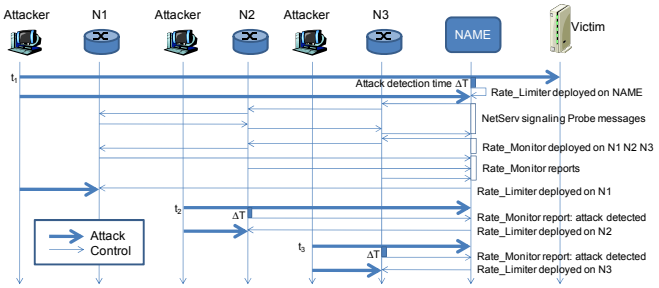


Figure 3 – Signaling flow in the GENI experiment.

Note that in this phase the application server is protected by the Rate_Limiter instance executed by the NAME itself. On the basis of reported values, which are the portion of interest of the new system state, the Action Planner of the NAME identifies the node N1 shown in Figure 3 as the best candidate to deploy a remote Rate_Limiter module, since it is the most distant node (in terms of IP hops) from the NAME with an incoming rate above the alarm threshold. Thus, by using the NetServ signaling, the NAME can instantiate the Rate_Limiter in N1. The Rate_Limiter module interacts with the NAME, which receives reports of all deployed Rate_Monitor modules, and changes the acceptable incoming rate threshold dynamically, depending on the number and frequency of detected requests. In this way, a further control loop is created so that each management action enforced by the NAME is dynamically adapted to possible context and state changes.

At time $t_2$, the attacker adds additional sources of DoS packets in other networks, thus bypassing the deployed shield. Nevertheless, since the NAME instance has been executing the monitor and rate limiter module since attack beginning, it can both protect the server and argue that the previous remote counteracting action has been bypassed. If the previously deployed Rate_Monitor modules are still active, some of them start reporting values of the observed incoming rate beyond acceptable values. This context information allows the NAME to identify the NetServ node N2 as the best candidate to deploy another remote instance of the Rate_Limiter module. If the lifetime of the previously deployed Rate_Monitor modules has expired, they are re-deployed.

Finally, the attacker starts a further attack session from another network at time $t_3$. The self-protecting procedure is repeated again, thus deploying a further instance of the Rate_Limiter on N3 that decreases the service request rate once again to a value as close as possible to the target value. When the attack ends, all the monitor and rate limiter instances are no longer refreshed. Hence, they are automatically removed, without any additional signaling.

In order to actually estimate the end of attack condition at the NAME, the remote monitor modules track both forwarded and dropped service requests, and report back the relevant statistics.

In future work, we will improve platform reliability and performance by integrating OpenFlow [8][9] as low level mechanism to balance the load towards multiple NAME modules.

## V. Demo Requirements

For the purpose of this demo, our team will need a table with enough space to host two laptops that will drive the experiment plus a projector with the respective screen or, alternatively, two big monitors to allow the visualization of the network traffic in the demo topology.

The scenario described in the previous section will actually take place in real-time in a GENI slice, using a topology of nodes deployed across the continental USA, thus decreasing the needed hardware. In order to allow the remote usage of the GENI testbed, a high speed Internet connection will be required (a wired connection is highly recommended to decrease latency related problems). Power outlets able to operate the above listed equipment are also required.

## VI. Conclusion

This demo proposal described the NetServ platform and how it can successfully be used for implementing autonomic management architectures for the Future Internet. In order to show the effectiveness of the proposed solution, we have conceived a case study which highlights how NetServ allows deploying self-protecting network functions. In the experiment, to be carried out live on the GENI testbed, we will show how the proposed architecture is able to counteract a DoS attack by selectively deploying monitoring and actuator modules at runtime.

The authors also would like to point out that another demo has been previously carried out live on the GENI testbed platform [7]. Nonetheless, the demo shown at the GENI Engineering Conference (GEC) 9, featured a previous version of NetServ with less functionalities (e.g. no management and epidemic signaling). Also, GEC9 services were more oriented towards the application level (CDN and SIP overload), whereas the demo proposed here shows NetServ as a Future Internet network service platform.

## References

[1] J.W. Lee et al, “NetServ: Active Networking 2.0”, FutureNet IV 2011, Kyoto, Japan, June 2011.

[2] X. Fu et al., “NSIS: a new extensible IP signaling protocol suite”, IEEE Communications Magazine, 43(10), 2005, pp. 133- 141.

[3] OSGi Alliance. http : // www . osgi . org.

[4] B. Jennings et al., “Towards Autonomic Management of Communications Networks”, IEEE Communications Magazine, Vol. 45, Issue10, Oct. 2007.

[5] X. Yang, X. Liu, Y. Xia, “NetFence: Preventing Internet Denial of Service from Inside Out”, ACM SIGCOMM 2010, Delhi, India.

[6] The Global Environment for Network Innovations (GENI) project, http :// www .geni .net.

[7] GEC9 NetServ demo, http : // www .cs. columbia. edu/irt/ project/netserv/.

[8] Openflow, http : // www . openflow . org

[9] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, L. Mathy, “Flow Processing and the Rise of Commodity Network Hardware”, ACM SIGCOMM Computer Communication Review, 39(2), April 2009.