Extending the NetServ Autonomic Management Capabilities using OpenFlow

E. Maccherani[1], M. Femminella[1], J.W. Lee[2], R. Francescangeli[1], J. Janak[2], G. Reali[1], H. Schulzrinne[2]

[1] DIEI – University of Perugia, Perugia, Italy {maccherani,femminella,francescangeli,reali}@diei.unipg.it
[2] Dept of Computer Science, Columbia University, New York, USA {jae,janakj,hgs}@cs.columbia.edu

Abstract— **Autonomic management capabilities of the Future Internet can be provided through a recently proposed service architecture called NetServ. It consists of the interconnection of programmable nodes which enable dynamic deployment and execution of network and application services. This paper shows how this architecture can be further improved by introducing the OpenFlow architecture and implementing the OpenFlow controller as a NetServ service, thus improving both the NetServ management performance and its flexibility. These achievements are demonstrated experimentally on the GENI environment, showing the platform self-protecting capabilities in case of a SIP DoS attack.**

Keywords: autonomic management; programmable nodes; dynamic deployment; OpenFlow; NetServ; GENI

I. INTRODUCTION

A progressively more complex and interconnected networking infrastructure is leading to an increasing difficulty in managing multi-vendor environment and services. Researchers and companies are addressing this problem investigating the application of autonomic principles, trying to simplify the network management by automating and distributing the decision making processes. In [4] the authors showed how the NetServ platform can be used for implementing an autonomic management architecture. NetServ is a programmable node architecture, intended for dynamically deploying in-network services. It is an enabling platform for every kind of network management architecture, because its implementation relies on Linux and Java technologies, allowing its deployment in every linux-enabled device. However, NetServ could be a bottleneck, since in its current implementation incoming packets to be processed must go at user space, slowing down the system [8].

This paper shows how the architecture proposed in [4] can be further extended and improved by the introduction of the OpenFlow technology [3][10]. In more detail, we will show how the OpenFlow technology can be integrated with the NetServ platform, in order to improve the flexibility and the performance of the autonomic management platform implementation.

The paper is organized as follows. Section II describes the proposed autonomic architecture and its implementation, presenting both the NetServ platform and its integration with OpenFlow. Section III shows the experiment conducted inside the GENI environment [2] as a practical demonstration of how the platform can self-protect from a denial of service (DoS) attack and how the OpenFlow integration enables us to design a scalable architecture. Conclusions can be found in Section IV.

II. PROPOSED MANAGEMENT ARCHITECTURE

NetServ is a programmable node architecture designed for dynamically deploying in-network services on a wide variety of node types, including routers, access points, set-top boxes, and end-user hosts. It includes an in-network virtualized service container and a common execution environment for hosting both networks services and management agents [4], as well as an NSIS-based signaling protocol [1], used for dynamic NetServ node discovery and service modules deployment therein. NetServ fits perfectly as the enabling platform for an autonomic management architecture, not only for its dynamic capabilities, but also for its modularity, allowing the implementation of the main control element, the NetServ Autonomic Management Element (NAME) [4], and of its modules as NetServ services.

The NetServ prototype architecture uses Linux kernel as the packet forwarding plane. We are currently developing a version of NetServ that can use an OpenFlow (OF) switch as the forwarding plane. Our approach is described in detail in our companion technical report [7]. Figure 1 shows the component organization and the packet paths when NetServ nodes are running on top of an OF switch. The NetServ node at the top of the figure hosts the OpenFlow controller (OFC), running as a NetServ service module. At the bottom of the figure, one or more NetServ nodes run packet processing modules. We call these NetServ nodes "processing units" (PUs). All the NetServ nodes are connected to the OF switch at the center of the figure. The arrow labeled as the "first packet of a data flow" indicates the path of a packet when it arrives at the OF switch and the flow table in the switch does not contain a matching entry. The packet gets routed to the OFC, which determines if the packet should be routed to one of the PUs for processing. If so, the OFC will instruct the OF switch to add an entry to its flow table so that the subsequent packets of the same flow will get routed directly to a PU. Then, the first packet is processed by a PU as well, since the OFC sends the packet back to the OF switch after the flow table entry is inserted. The path of the subsequent packets are depicted as the arrow labeled as the "subsequent data packets".

When we compare the OpenFlow-based NetServ with the Linux kernel-based one, the benefits of the OpenFlow-based implementation are clear. The fast data path is in an OpenFlow switch, a hardware component.

978-1-4673-0269-2/12/$31.00 ©2012 IEEE

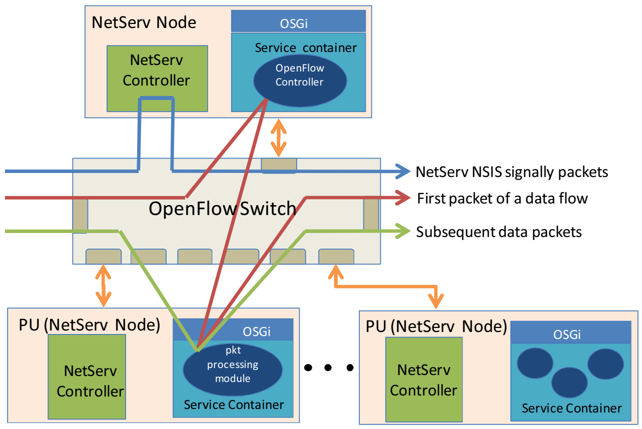

Figure 1 – OpenFlow-enabled NetServ node architecture.

The packets that do not require processing by PUs will get forwarded at the line rate. Moreover, the ability to attach multiple PUs to the OF switch effectively eliminates the scalability problem associated with a single NetServ node. This, in fact, is the key feature that enabled us to design the *scalable* autonomic management architecture that we present in this paper.

The integration of an OF switch inside the NetServ architecture enables advanced features and capabilities from an autonomic management point of view. External services, or the NAME itself, can exploit remote calls to the OFCs, and hence to OpenFlow-enabled NetServ nodes, in order to totally control the data path layer. Gathering several information about the state of the nodes or traffic statistics is useful to estimated an accurate network context. Using this information, the management engine can easily detect fault situations and alarms, and plan requested actions to restore a stable network state. The OpenFlow data path can be utilized to perform active network management operations directly mangling packet flows, such as dynamically redirect or output it to certain ports, split/join/drop flows, manipulate VLANs tags and priorities, change fields of layer 2/3/4 headers, create queues for Quality of Service support. All this capabilities are performed in hardware, so they run at wire speed and allows avoiding processing bottlenecks. The OFC exploits Link Layer Discovery Protocol (LLDP) capabilities to reconstruct its local network topology. The OpenFlow/NetServ integration also adds important features to the OpenFlow architecture, since it can exploit the capabilities of NSIS signaling to discover which part of the network is managed with OF switches and OFCs. This can be done utilizing the newly proposed NSIS extension [6].

III. Case Study

This section describes an experiment showing the NetServ platform protecting a SIP server from a DoS attack. This scenario will make use of an IP flow-based intrusion detection technique [5]. The final goal is to show how OpenFlow-enabled NetServ nodes can enhance the autonomic management capabilities of such a solution, in particular in terms of performance and reliability. In order to detect network attacks, the usual approach for intrusion detection is to inspect the content of every packet, a technique also known as deep packet inspection (DPI). However DPI cannot easily be performed at high speeds, so researchers are investigating alternative approaches, such as flow-based intrusion detection [5]. In that kind of approach the analysis is carried out on the "flow" data, instead of analyzing the contents of each individual packet. An IP flow is defined as a set of IP packets passing through an observation point during a certain time interval. All packets belonging to the same flow have a set of common properties, such as source and destination addresses, port numbers and IP protocol. The flow-based intrusion detection is a two-step process, the flow exporting and the flow collection. The first one is responsible for creating flow records from observed traffic. This module extracts the packet header from each packet, marks it with a timestamp and pass it to the flow collector. The flow collector stores these data for monitoring and analyzing it. Flows represent aggregated information regarding network interactions and do not carry payload. With these data, it is still possible to identify communication patterns between hosts and identify most attacks, such as denial of service (DoS), scans, worms and botnets. Flow-based detection can be seen as a complement of packet inspection. Both techniques can be combined into a two-stage detection process. First, a flow-based approach can be used to detect certain network attacks or a suspicious flow. Then, a DPI can be used to additionally protect a critical server or to deeply analyze the nature of a specific flow.

Although most contributions in this area rely on centralized data processing, our architecture easily allows using distributed PUs implemented as a NetServ service and deployable in every NetServ node. Moreover, we can also implement a distributed flow-based detection, because also flow exporting modules can be remotely deployed.

Figure 2 shows the network topology of this experiment, which we have implemented in the GENI platform [2]. The victim of the attack is a SIP application server (AS) attached to an OF switch, which is part of a OpenFlow-enabled NetServ (OF/NS) node. This node consists of an OF switch and multiple PUs. A separate NetServ node runs the OFC (we selected Beacon [8] since it can be easily integrated in OSGi, see also [7]) and the NAME as service modules. A Flow-based Intrusion Detection System (FIDS) service module runs inside PU1, implementing both the flow exporting and the flow collector modules. It records all packets directed toward the SIP AS, reporting statistics to the NAME module. In our experiment, we have chosen to directly forward the packets to the SIP AS by the OF switch, so as to reduce latency of data packets. At the same time, all SIP traffic is duplicated and sent to the PUs to monitor and analyze the traffic.

Figure 3 shows the signaling flow for the experiment. At time t_1 several SIP requests arrives to the SIP AS. Different flows goes through the NetServ-enabled routers (NS1, NS2, NS3) and converge to the OF/NS node. Flows are directed to the AS and, at the same time, they are replicated to the first processing unit (PU1), in charge to analyze all the packets with the FIDS module. At time t_2 an unknown attacker starts a SIP DoS passing for the NS1 node. The FIDS on PU1 detects it, activate autonomously the DPI module to further analyze traffic and informs the NAME.

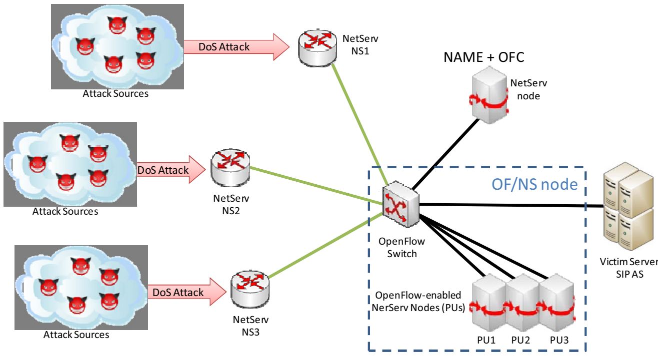

Figure 2 – Network topology for our DoS experiment on GENI.

It decides to send probe requests in all directions [4][6] to discover NetServ nodes that can potentially run an intrusion detection and prevention service (IDPS). NAME chooses the NS1 node as the best candidate for this goal. When the IDPS is deployed into NS1, the service itself takes care of the malicious flow, blocking it if necessary (at time t_3). The attack increases and it is wide spread across several paths (time t_4). In our test bed it goes through NS2 and NS3 nodes. NAME recognize the increased attack rate, and, at first, decides to instantiate additional OF/NS nodes's processing unit to avoid processing bottlenecks on PU1. It can takes advantage of OF/NS capabilities, enabling parallel packet processing through multiple modules in different PUs. So the NAME can deploys the FIDS+ IDPS service in PU2 and PU3, and instructs the OFC to split the packets flow over several output ports, in order to reach each processing node (time t_5). On the basis of reports also from PU2 and PU3, NAME deploys the IDPS service also in the other NetServ boundary nodes, in order to definitely block the attack (time t_6).

Figure 4 shows the runtime behaviour of the experiment, in particular the packet rate of the ingress (solid) and egress (dotted with marker) interfaces of NetServ boundary nodes. The ingress interface is the one receiving the aggregate malicious traffic, and the egress interface is the one forwarding the unblocked portion of such traffic towards the victim. The SIP server curve (green with marker) shows the traffic arriving to the victim server. The SIP traffic sent before the attack is 15 packets per second (pkt/s) and the sustainable rate for the SIP server is 20 pkt/s. When the attacks begins at t_2=28s, an *extra* traffic beyond the background traffic arrives at the SIP server, with an attack intensity I_r measured in pkt/s (in this case I_r = 25 pkt/s, see solid red line). The autonomic system takes few seconds to recognize it and take the appropriate decisions. In this case we have that the malicious portion of the traffic is dropped, leaving the good one untouched.

During the second phase of the attack, starting at t_4=48s, the aggregate intensity on the victim is doubled ($2\times I_r$). As in the previous case, after few seconds, the system react and returns to an acceptable state. We have repeated this experiment at different attack rates I_r, to figure out how the reaction time of our autonomic platform changes, measured at the ingress interface of the SIP AS. It shows that the reaction time is basically insensitive to increasing values of traffic intensity (see Table I). It is about 1.6 s for the first attack, and 2.4 s for the second one.

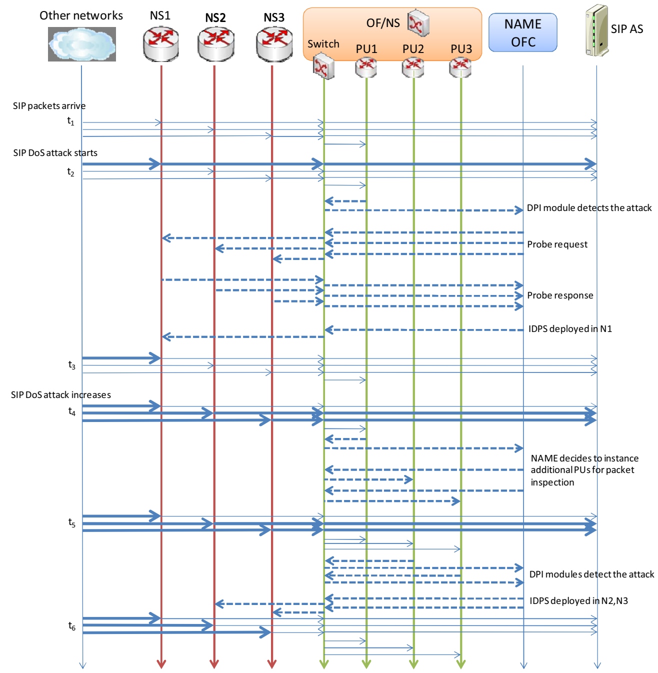

Figure 3 – Signaling Flow for the GENI experiment.

This increments is justified by the procedure for deploying additional PUs (PU2 and PU3) and to wait their processed data to deploy additional DPI modules on NS2 and NS3.

The overhead added by having the OFC exchanging messages with the OF switch is equals to 15 Kb/s. The signaling overhead between the NAME and the various NetServ modules has an average rate of 24 Kb/s, while during the modules deployment we have a peak rate of 500 Kb/s for about 20 ms.

The OF/NS architecture has been designed in order to permit packets processing inside NetServ modules. In fact, after a packet is processed by a PU, it goes out and come back to the OF switch, where it is forwarded to the destination (see also [9]). When the processing must be splitted across several PUs, the OFC informs the OF switch that a certain type of packet flow must be equally splitted to the NICs where the PUs are attached. Now, to the best of our knowledge, it is not possible to perform this type of action inside a commercial OF switch, but it will be possible with device compliant with OF specifications v.1.2. It will contain an extension of the usual flow mod command that can match every bit inside the packet header, so we can utilize, e.g., the packet identification field in the IP header (IP ID), to split the packet flow by comparing it to a bitmask. In our implementation, we replicate the whole packet flow to each involved PU. The packet separation is done inside the linux kernel, taking advantage of the netfilter u32 module that can extend a filtering rule matching a certain bit's pattern of a packet. So part of the traffic is just dropped by the kernel, whereas the portion matching the bitmask on the IP ID is forwarded to the user space to NetServ modules. We tested the scalability of this replication scheme on the PUs of the OF/NS node with a DPI service performing signature search inside the packet headers and payload.

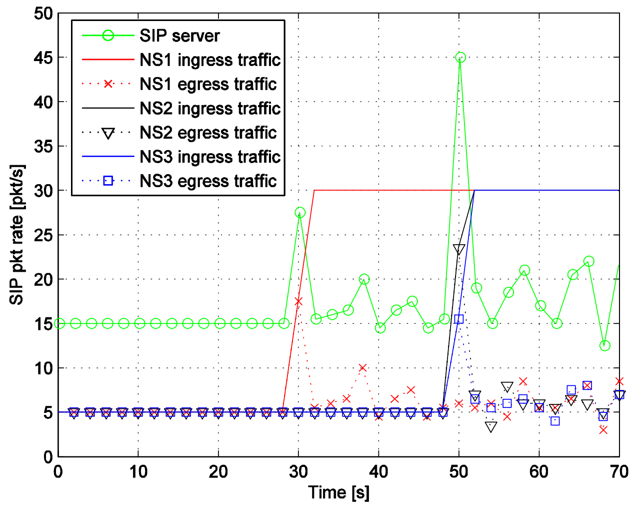

Figure 4 – Runtime behavior of network traffic.

TABLE I: REACTION TIME MEASURED AT THE VICTIM SIP SERVER

Attack intensity I_r (pkt/s)	Reaction time (s)	
	1st Attack	*2nd Attack*
15	1.4	2.4
25	1.6	2.4
35	1.6	2.4

Each PU uses 1 CPU core and 3 GB of RAM. We considered two different payload sizes for SIP packets, 600 and 1000 byte. Results are shown in Figure 5, which shows that, in both cases, using 2 PUs allows to nearly double the handled traffic. Please note that since both PUs receive the same traffic, the kernel of each PU has to drop the portion of the traffic which will be processed by the NetServ DPI module running in the other PU. However, the impact of this operation is negligible, as shown in the figure. Finally, we point out that since the signature search is performed on both header and payload, the performance in terms of handled packet rate is inferior when the payload is larger. However, the CPU load required for processing of each packet has no impact on the scalability performance, which is the same in both cases.

Analysis with 3 PUs or multiple CPU cores per PU has been not been possible, since the specific OF switch available in GENI performs MAC address rewriting in software, and this appeared to be the system bottleneck. Additional, extensive tests with a different device are planned.

IV. CONCLUSION

This paper illustrates how the OpenFlow technology can be utilized to further enhance an implementation of an autonomic management architecture based on the NetServ programmable node architecture. OpenFlow has been integrated with NetServ, utilizing an OpenFlow-enabled switch as an hardware data path for an equivalent NetServ node, enabling wire speed capabilities. Processing power constraints can be managed through the use of multiple PUs attached to the same data path.

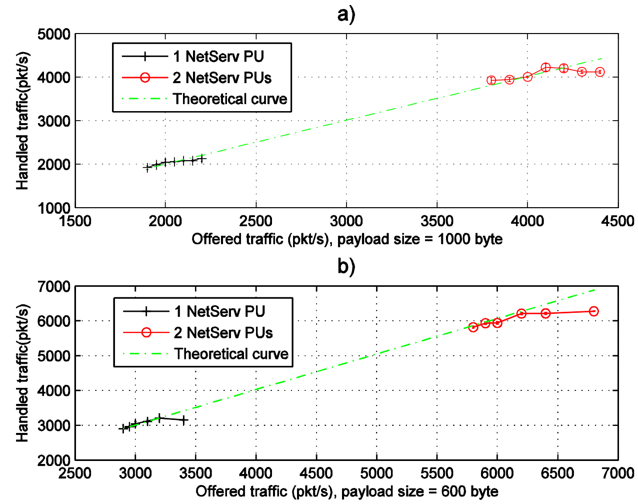

Figure 5 – Scalability analysis: handled vs. offered traffic with different PUs

We have presented a case study carried out on GENI. The experiment shows the self-protecting features against a DoS attack to a SIP application server, utilizing a flow-based intrusion detection techniques. The results are that the reaction times are nearly insensitive to increased traffic intensity, and the overall signaling overhead is acceptable. Our experiment also indicates that our solution scales well. Using 2 PUs, we were able to attain twice the packet rate of a single PU.

An additional consideration is that the deployed service is the same, both in usual NetServ nodes and in PUs of the OF/NS equivalent node. The developer writes his service logic utilizing a common API that will implement different functions depending on the different running platform, exploiting the OpenFlow data path capabilities if an OpenFlow switch is present.

V. REFERENCES

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

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

[3] N. McKeown et al, “OpenFlow: enabling innovation in campus networks”,ACM SIGCOMM Comp. Comm. Rev., 38(2), April 2008.

[4] M. Femminella, R. Francescangeli, J.W. Lee, G. Reali, H. Schulzrinne, “An enabling platform for autonomic management of the future Internet”, IEEE Network 25(6), Nov./Dec. 2011, pp. 24-32.

[5] A. Sperotto et al., “An overview of IP flow-based intrusion detection”, IEEE Communications Surveys & Tutorials, 12(3), 2010, pp. 343 – 356.

[6] M. Femminella, R. Francescangeli, G. Reali, H. Schulzrinne, "Gossip-based signaling dissemination extension for next steps in signaling", IEEE/IFIP NOMS 2012, Maui, Hawaii, USA, April 2012.

[7] E. Maccherani et al., "NetServ on OpenFlow 1.0", Technical Report CUCS-036-11, Columbia University, September 2011, available at: http://www.cs.columbia.edu/~jae/papers/netserv-openflow-tr-1.0.pdf.

[8] Beacon Controller, https://openflow.stanford.edu/display/Beacon/Home.

[9] A. Greenhalgh et al, “Flow processing and the rise of commodity network hardware”, ACM SIGCOMM Comp. Comm. Rev., 39(2), April 2009.

[10] The OpenFlow project, http://www.openflow.org/.