MMS: An Autonomic Network-Layer Foundation for Network Management

Hemant Gogineni[†], Albert Greenberg[‡], David A. Maltz[‡], T. S. Eugene Ng[§], Hong Yan[†], Hui Zhang[†]
[†]Carnegie Mellon University [‡]Microsoft Research [§]Rice University

***Abstract*—Networks cannot be managed without communication among geographically distributed network devices and control agents. Unfortunately, computer networks today lack an autonomic mechanism that enables such communications, and the stopgap solutions used in practice are seriously flawed. To address the problem, this paper presents the design and implementation of the Meta-Management System (MMS), a network-layer subsystem that provides robust and universal support for management plane communications. The MMS is autonomic, able to self-configure, self-heal, self-optimize, and self-protect. Furthermore, it is efficient, scalable, and evolvable. We demonstrate the practicality of the MMS via a fully functional implementation that runs on commodity hardware. The MMS software is freely available.**

***Index Terms*—Autonomic communication, network management, security, performance, system design and implementation.**

I. Introduction

Modern computer networks have many functions. Besides providing basic packet delivery services, they play a critical role in securing computing resources (e.g. blocking unauthorized traffic, detecting attacks), in ensuring application performance (e.g. balancing network and server load, differentiating service for different applications), in enhancing application reliability (e.g. transparently allowing a backup server to take over), in enabling utility computing services (e.g. virtual private networking, data center virtualization) and more.

The industry and the academic community have both recognized the importance of autonomic management for these increasingly complex functions [1][2][3]. Numerous architectures for autonomic network management have been proposed in the literature (e.g. [4], [5], [6], [7], [8], [9], [10], [11], [12]). While they generally differ in terms of system organization (e.g. centralized agent, hierarchical agents, peer-to-peer agents) and control mechanisms (e.g. policy-based and bio-inspired adaptation), they all aim at forming the autonomic control loop between network devices and control agents.

Forming the autonomic control loop fundamentally requires communications among network devices and control agents. Surprisingly, we have not yet come across any architectural proposal that studies the mechanism for this communication. Perhaps there is a mis-perception that a suitable mechanism which is itself autonomic is already widely available. Unfortunately, the reality is that computer networks today lack such an autonomic mechanism and the stopgap solutions used in practice vary widely. Many commercial networks still rely on dial-up modems to access the serial console ports of routers for control; this method has poor performance and is clearly not self-healing nor self-optimizing. Alternatively, many networks rely on an orthogonal Ethernet network to access the special management Ethernet ports of routers for control; however, Ethernet is insecure, not self-protecting, nor self-optimizing. Other networks even rely on in-band connectivity to control routers (i.e. control communication is mixed with user data communication and relies on the very same IP routing tables); this method is dangerous as it risks losing remote access with no recourse if the router is accidentally misconfigured.

From a system design point of view, we argue that a fundamental architectural element missing from autonomic network management is *a subsystem, which is itself autonomic, that provides robust and universal support for management plane communications*. Such a subsystem is a necessity for the collection and exchange of environmental observations that drive the autonomic control loops and for the conveyance and negotiation of autonomic control decisions. More broadly speaking, beyond autonomic control, such a subsystem is also a necessity for the access and storage of management data that reside in the network, and for the recovery from management system failures. For example:

- Many current management systems adopt an external control model where network switches communicate their environmental observations to an external intelligent controller(s), the controller(s) reacts to the observations and communicates control decisions to the network switches. This external control model critically depends on robust, secure, and low-latency management-plane communications. Examples of such systems include:
  1) AT&T's Intelligent Route Service Control Point [13] which can flexibly direct where and how global traffic flows in a backbone ISP;
  2) Commercial products such as HP's OpenView and IBM's Tivoli management solutions which are increasingly network-aware, able to manage network configuration changes, interact with network devices via SNMP, monitor network conditions and direct computing systems to self-optimize accordingly;
  3) Experimental systems such as Tesseract [14], Ethane [15], and OpenFlow Switch [16] that have

This research was sponsored by the NSF under grant numbers ANI-0331653, ANI-520187, CNS-0520280, CNS-0721990, CNS-033162, and by Microsoft Corporation. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Microsoft Corporation, or the U.S. government.

provided experimental platforms for sophisticated network control such as policy-driven network access control and integrated routing and firewalling.

- Another example of management-plane communications is the access to bulk data, such as software update images, located at network attached storage servers. The update of the software running on a network's routers is an intricate multi-step process that must be carefully planned and executed. An autonomic controller could orchestrate the process by systematically controlling and routing customer traffic around the switches to be updated, and by mediating the download and installation of the updates. This type of bulk data access communication especially requires high bandwidth.
- The management communication subsystem could also play a critical role if the autonomic management system fails. Because the management communication subsystem is decoupled from the rest of the management system, through it the network operator could regain control over the controller(s) and the switches. Appropriate actions such as killing and re-starting processes, patching vulnerable software, re-booting devices etc. could then be performed. In this case, it is critical for the management communication subsystem itself to be self-configuring and allow no chance for human error.

To provide this missing architectural element, this paper takes a system design and implementation approach. We present a solution called the Meta-Management System (MMS) – a self-configuring, self-healing, self-optimizing, and self-protecting network-layer module designed to provide a high performance, dependable communication service for the management plane.

Besides providing self-* capabilities, the design of the MMS addresses the real-world constraints imposed by the network environment in which it must operate. For example, in practice, the MMS may run along side other complex software in a network device. The danger of run-time resource starvation threatening the liveness of the MMS is real and must be addressed. Furthermore, it is prudent to design the MMS to have built-in support for updates so that evolving the MMS is seamless. The MMS must also streamline its memory footprint so that it may be deployed on as wide a range of network devices as possible. We show that the MMS has met these criteria via a fully functional implementation that runs on commodity hardware. The MMS software is freely available at `http://100x100.jot.com/mms`.

In the next section, we review the techniques used in other problem domains and explain how they fall short of meeting the needs of autonomic management plane communication. In Section III, we present the design and implementation of the MMS. In Section IV, we evaluate MMS's performance and robustness. In Section V, we present two case studies. First, we show how the MMS can self-optimize for variations in link quality in wireless mesh networks. Second, we show how the MMS can enable remote recovery when a network device's control plane is overloaded. Finally, we conclude in Section VI.

II. Techniques from Other Domains

There is a large body of available routing techniques for different problem domains. With the exception of static routing, essentially all routing techniques are self-healing in that they respond to link or node failures and re-route. However, not all routing techniques are self-configuring. Many widely used commercial routing techniques such as OSPF [17], IS-IS [18], and RIP [19] are not self-configuring. Take OSPF as an example: it requires a large amount of information to be configured such as OSPF area specifications, message timers, link metrics, interface types, authentication keys, etc. Any misconfiguration of these parameters could render the network inoperable.

Among commercial solutions, Ethernet [20] is the most notable self-configuring and self-healing system. Unfortunately Ethernet does not self-protect nor self-optimize. Any host on an Ethernet can launch a denial of service attack by flooding the entire network. Ethernet's spanning tree protocol is also insecure and cannot survive a compromise. A malicious host can inject fake protocol messages and manipulate the spanning tree topology. The use of a spanning tree topology also makes Ethernet highly inefficient because redundant links in the network cannot be used to forward traffic. Many research proposals that could serve as more efficient replacements for Ethernet also lack self-protection and self-optimization capabilities (e.g. SEATTLE [21], ROFL [22], UIP [23], Ethane spanning tree routing [15], Tesseract path explorer routing [14]).

There are numerous self-configuring and self-healing routing techniques proposed in the context of ad hoc mobile networks [24]. Some of the routing techniques in ad hoc mobile networks emphasize adaptation to node mobility and minimizing packet transmission energy consumption. Therefore, the techniques they employ may sacrifice routing efficiency in favor of these other concerns [25][26][27][28]. In this paper, we are interested in management in service provider networks and thus mobility and energy consumption are not likely to be the primary concerns.

Many techniques in the ad hoc mobile network environment are designed to route traffic between potentially all pairs of mobile nodes (e.g. [29], [30]). Instead, management plane communications are mainly between network switches and management entities (e.g. controllers, storage servers) rather than all possible pairs of nodes. The solution could therefore exploit this characteristic.

A large number of secure routing techniques have been proposed in the contexts of wired networks and ad hoc mobile networks. The general lesson we can learn from these techniques is that there is a large toolbox of available primitives. Which primitive is optimal for solving a problem however depends on the problem domain.

Routing techniques that are fixed on forwarding data via the shortest paths (e.g. OSPF, IS-IS, RIP) give too much power to any compromised node that happens to lie on the shortest path. A self-protecting technique will need to have more flexible control over routing. Some solutions rely on flooding redundant copies of a packet to ensure packet delivery despite a network compromise [31]; however, this technique has a rather high performance penalty.

Many techniques also turn to cryptographic primitives to provide security. One class of techniques use asymmetric public key cryptography to authenticate messages [32][33][34]. However, asymmetric cryptography is computationally very expensive. Protocols that use asymmetric cryptography heavily are vulnerable to attacks. For example, an attacker can cause a victim node to verify a large number of forged signatures to exhaust the victim's computation cycles. To avoid asymmetric cryptography, some techniques simply use a single shared secret key among all nodes [35]. Unfortunately, these techniques cannot survive even a single node compromise. Alternatively, some techniques require that nodes have pre-configured pair-wise shared secrets [36]; however, such techniques are no longer self-configuring as the number of configured keys required scales quadratically with the number of nodes. There are also hash chain techniques for authentication [37][38][39]. Hash chain techniques are most effective for broadcast traffic authentication but cannot provide secrecy. In the problem domain of this paper where communications are point-to-point and may need to be secret, hash chain techniques do not out-perform pair-wise shared secret techniques.

To meet the needs of autonomic management plane communications, the solution should strike a balance between computation overhead, complexity and security by automatically establishing shared secret keys and by using efficient symmetric cryptography for packet handling.

III. MMS Design and Implementation

In this section, we present the design and implementation of the Meta-Management System (MMS). The MMS module runs on network elements (NE), by which we mean routers, switches, firewalls and other devices that are being managed. The MMS also runs on management stations (MS), by which we mean the network-connected hosts used to control, manage and configure the network, as well as those that store management data.

A. MMS Features Overview

We begin by providing an overview of the features of the MMS and point out which feature contributes to which autonomic objectives – i.e. self-configuring (C), self-healing (H), self-optimizing (O), and self-protecting (P).

Automatic creation of management channels (Self-C) - When a MS with a valid security certificate is attached to a network, the MMS automatically establishes secure end-to-end management communication channels between the MS and the NEs in the network. Likewise, when a NE with a valid security certificate is attached to a network, the MMS automatically establishes secure end-to-end management communication channels between it and the MSs.

The MMS transports management communication over MMS network links. MMS network links can either be logical partitions (with performance guaranteed by priority queuing, for instance) of data traffic links or dedicated management traffic links. The MMS thus logically separates management communication from user data communication so that they no longer share the same fate.

There is no manual configuration beyond exchanging security certificates at device installation time.[1] The MMS integrates, and thereby enforces, best practices. Once the MMS solution is installed, the rest is automatic.

The MMS exposes a familiar datagram service to applications, so existing management applications can access the MMS management channels via standard socket API.

Integrated security assurance (Self-P) - The MMS assumes a hostile environment in which malicious end hosts attached to the network may launch a DoS attack at the MMS or try to compromise NEs. The MMS is robust to such attacks and NE compromise. First, regular end hosts have no way to address MSs in the network, thus launching a DoS attack at the MSs is not possible. The MMS management channels have priority over data traffic and thus DoS attack against NEs in the data plane cannot disrupt management traffic. If a NE is compromised, it can drop MMS traffic or generate spurious messages in a DoS attack. However, due to the MMS's use of onion-encrypted source routing, such NEs can easily be detected. The MMS can quarantine such NEs by issuing new source routes that by-pass the quarantined NE. Finally, the MMS provides a mechanism to revoke a MS certificate and replace it with a new one, which is useful, for example, when a MS laptop computer storing the certificate is lost.

Integrated liveness assurance (Self-H, Self-O, Self-P) - MMS maintains the liveness of the management channels in an integrated fashion. It can dynamically re-route when a loss of network connectivity occurs. Furthermore, the MMS can take link performance (e.g. loss rate) into account and flexibly optimize communication performance by choosing new source routes. It is designed to protect itself against CPU resource starvation. Moreover, due to the use of source routing, MMS does not require NEs to maintain forwarding tables that grow with the network size. All the memory MMS needs can be statically allocated at boot time, thereby defending against memory starvation. Furthermore, the MMS provides remote process management and packet filtering APIs to ensure the liveness of critical higher layer management software tools.

Handles large networks and a wide range of devices - The protocols used in the MMS are specifically designed so that the amount of memory and CPU computation required of network elements is small and independent of the size of the network. This means that the MMS can run on a wide range of devices, and the network can grow without forcing the upgrade of all NEs. Instead, the computation and memory requirements are placed on the MSs. The MSs can target their resources at reaching the specific NEs they wish to configure. Furthermore, since MSs are just end hosts and comparatively few in number, they are easy to upgrade.

Management stations can be connected to the network at any port, so service technicians in the field and operators in the network operation center can all access network elements using the MMS. There is no need to travel to special "network

[1] Major vendors today already install security certificates onto their network elements before shipping them to customers.

management ports" to connect. After a MS is plugged into a network, in about 30 seconds it can establish MMS secure channels to one thousand core devices. Note that many large enterprises and ISPs have roughly this size [40].

Evolvable after deployment - The MMS can be used to manage and evolve the MMS itself with zero down time. The design of MMS enables multiple parallel instances of the MMS to operate over the same network at the same time. This allows a new MMS instance to be brought up in order to manage or replace the old instance. Specifically, a new version of the MMS can be installed and brought up through the management channels provided by the old working version. The new version can be tested thoroughly before the old version is removed.

B. Partitioning Data Links for MMS Communication

The MMS can run on links dedicated to management or the same links that carry user data. If the MMS leverages the same physical links used for regular data packet transmissions, the link-layer must logically partition the link so that the logical link used by the MMS has a guaranteed minimum throughput. This prevents regular data traffic from interfering with the delivery or processing of MMS frames. This abstraction can be realized on all common links, though by different mechanisms. For example, SONET links can use the supervisory channel to carry MMS frames. In a datagram network, weighted fair queuing or priority queuing can be used.

In our implementation, the network consists of point-to-point Ethernet links, and MMS frames are sent to a reserved multicast address and tagged with a specific protocol type. When the MMS module is activated, it configures the OS to hand it any MMS-tagged frames going to this multicast address. To prevent user traffic (e.g., DoS attacks) from interfering with management communication, we use the simple priority queuing system provided by the interface driver. MMS frames are put into the highest priority queue and thus served first by the scheduler.

C. Automatic Construction of Secure Channels

One of the MMS's most important and basic features is constructing a set of secure channels for management information to flow between a MS connected to the network and the NEs that make up the network. These channels must be authenticated, must survive DoS attacks and local link or NE failures, and must be able to recover from a NE compromise. This section explains our design for establishing and maintaining these management channels.

1) Threat Model: The MMS is designed to withstand the following threats:

- **Operator error -** Mistakes made while altering the configuration of network elements.
- **Attack from an end-host -** Hosts connected to the network may attempt to DoS or inject false commands into the management channel.
- **Compromise of a NE -** Attackers may compromise any NE in the system, learn its secrets, sniff frames traversing it, and use it as a platform for launching DoS attacks against the MSs and NEs.

2) Minimizing State Held by Network Elements: The first step in constructing a secure channel is defining and authenticating the endpoints of the channel.

Estimates show that configuration errors are responsible for 60 to 70 percent of network outages today [41][42]. Since the MMS must provide an always-available management channel, configuration errors that prevent communication between the MS and the NEs are intolerable. We argue the best approach to eliminate configuration errors is to reduce the configuration state to the bare minimum needed.

In our design, each NE is configured with the following critical pieces of information prior to deployment. The first is a *network certificate* identifying the public key with ultimate authority over the network. NEs will accept commands only from MSs who have a *MS certificate* signed by the network certificate's private key. The second is a private/public key pair that uniquely identifies the NE. The NE's public key must be made available to the MSs before the MSs can communicate with the NE. The network certificate and the private/public key pair should be preserved in non-volatile storage on the NE.

This basic configuration provides the toehold from which the MS will be able to authenticate and communicate securely with each NE. In addition to the basic configuration, each NE stores the following dynamically generated soft-state for each MS with which it communicates: (a) a secret key shared only between the NE and the MS, (b) one or more onion-encrypted source routes by which the NE can communicate with the MS, (c) the version number of the MS's certificate, and (d) the time at which this per-MS state was last used. The exact definition of these fields and the means by which they are created will be explained next.

3) Secure Routing: The MMS is completely decoupled from the regular IP data plane services and therefore has its own routing subsystem. The forwarding of messages in MMS is controlled by *onion-encrypted source routes* [43]. These are strict source routes placed in the headers of the MMS frames that list the series of NEs through which the frames must pass. A source route is built like an onion, with the list of hops remaining in the route encrypted in the secret key of the NE making the next forwarding operation. A NE without a valid onion-encrypted source route can only transmit MMS frames to its immediate neighbors. Since the MS knows the secret keys of all NEs, it can construct an onion-encrypted source route between any two NEs. As a frame is forwarded, each hop re-encrypts the portion of the route over which the frame has already traveled.

We use onion-routing for two main reasons. First, it creates in each MMS frame a secure log of the frame's traversed path which only the MS can fully decrypt. As described in Section III-C5 this property will be used to detect and evict misbehaving NEs. Second, source routing ensures that the MMS on each NE does not need to maintain a dynamic routing

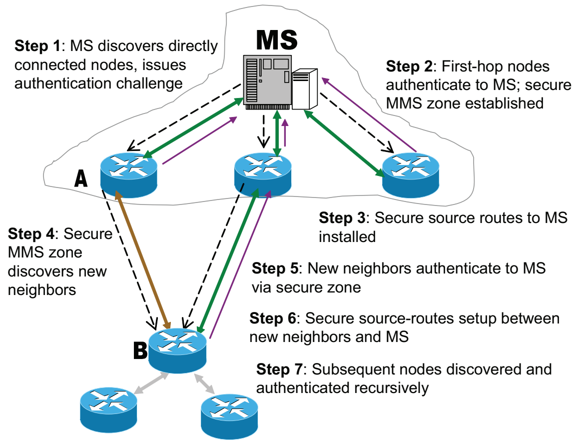

Fig. 1. Recursive MMS Authentication.

table that grows with the network size. Thus, the MMS on a NE only needs a small static amount of memory and will not run into memory allocation failures.

To establish the MMS onion-encrypted source routes, a MS first recursively authenticates and establishes secret keys with the NEs in the network. During this process, the MS computes an onion-encrypted source route for each NE to use to communicate with the MS, and the MS installs this route on the NE. Subsequently, the MS learns changes in the topology of the network by collecting encrypted link state advertisements (LSAs) from NEs. The MS reacts to topology changes by recomputing and pushing out new onion-encrypted source routes as needed. There can be multiple MSs in the network, but each MS performs these tasks independently. The details of the authentication process are explained next.

Recursive authentication - The MS is responsible for authenticating the NEs and sending them encrypted source routes that can be used to communicate with the MS. A NE proves its identity to the MS using a challenge-response protocol, and the MS proves its own identity to a NE by sending it a verifiable signed source route.

Figure 1 gives an overview of the process by which a MS establishes communication channels to the NEs in the network. The MS initiates and drives this process, enabling it to limit the set of NEs it contacts to the ones of interest. This will be important in very large networks with many edge NEs. The MS begins by initiating the authentication process with the directly connected NEs (e.g., A).

The MS authenticates a NE by sending it a challenge via an onion-encrypted source route. For a NE directly connected to the MS, this source route is trivial. This challenge contains several things. The first is a 128-bit session key that serves as a shared secret between the MS and that NE. The shared secret is encrypted by the NE's public key and signed by the private key from the MS certificate. The second is the public key from the MS certificate signed by the private key from the network certificate. This signed public key is pre-configured on a MS by the administrator. It is important to note that a MS does not know the private key of the network certificate. Thus, even if a MS is compromised, the network certificate is still safe. The third component is an onion-encrypted source route from the NE to the MS signed by the private key from the MS certificate.

By verifying the certificates and decrypting the session key, the NE proves its identity, verifies it is communicating with a valid MS, and obtains an onion-encrypted source route it can use to communicate with the MS (since it can decrypt the first layer of the route using the session key). The NE then encrypts its current LSA by the session key, and sends it to the MS using the onion-encrypted source route. If the LSA informs the MS of new NEs it should communicate with, the MS recursively authenticates those NEs (e.g. B) by sending them challenges via onion-encrypted source routes over authenticated NEs.

A MS certificate contains a version number, and NEs will only accept a MS certificate with the highest version number they have seen. This means if a MS certificate is compromised, it can be cheaply "revoked" by creating a new MS certificate with a higher version number and using it to authenticate all the NEs.

Authentication in large networks - Since the MS drives the recursive authentication process, it can target the authentication towards the NEs it wants to control. This is important in large networks, e.g., one with millions of edge NEs. As a simple example, the MS can authenticate with all the core NEs (as identified by an inventory database), obtaining LSAs that list the edge NEs and their attachment points. Even the largest networks have no more than a few thousands of core NEs, which the MMS can easily handle (see Section IV). Subsequently, the MS can initiate authentication with only the desired edge NEs.

LSA creation - The MMS implements a simple HELLO protocol by which each NE discovers the identities of its neighbors. Also, as part of this HELLO protocol, neighbors exchange lists of the MSs that they have authenticated with. These lists need not be verified by a NE, it is advisory only. From this information, a NE creates an encrypted LSA and sends it to a MS it has authenticated with. In each LSA, for each neighbor, a bit is used to indicate whether that neighbor claims to have authenticated with that MS.

When the link state changes, the NE detecting the change sends new LSAs to the MSs it has authenticated with. Each NE limits the rate at which it sends LSAs so that a compromised NE attempting to attack the MMS by flooding LSAs can only flood its immediate neighbors (which is unavoidable), but not the rest of the network.

New LSAs are retransmitted periodically until acknowledged by the MS (our implementation uses a period of 500 ms). If a new LSA is generated, it replaces the one currently being sent. To make the system as simple as possible, an LSA is acknowledged by the MS by sending a hash of the LSA back to the NE. There is no need to use sequence numbers as there can be only one outstanding LSA at a time, and the hash provides protection against bit-corruption in the LSA.

4) Resilience to Failures: If the connectivity between NEs changes, new LSAs are sent to the MS and the MS recalculates onion-encrypted source routes for affected NEs and sends the new routes to the NEs. Should a NE reboot or otherwise lose its soft-state for a MS, LSAs sent by this NE's neighbors will show that this NE is unauthenticated to the MS, and the MS can re-authenticate the NE if needed. Should a MS fail, all NEs will eventually purge their soft-state for it.

The MMS is designed to survive even simultaneous failures of multiple links. In addition to the experimental results presented in Section IV, we are able to prove this formally.

Convergence Property: If each NE knows the shortest path to a MS and the MS has the initial network topology, the above LSA propagation scheme ensures that the MS will eventually re-discover the shortest paths to all NEs in its network partition after any period of link failure events followed by a period without failures.

Proof: Let G_{MS} be the network topology, including the MS itself, perceived by the MS, G_{real} be the topology after the link failure event(s), $p(x)$ be the shortest path in G_{MS} from any NE x to the MS, S be the set of NEs who have different link state in G_{MS} and G_{real}. We define a path $p(x)$ as a *working path* if it is a path in both G_{real} and G_{MS}.

After the failure event(s), at least one NE in S has a *working path* to the MS. This follows since there is always at least one NE $a \in S$ such that $p(a)$ is the shortest. Since no other NE in S is between a and the MS, there is no failed link along $p(a)$. It follows that the LSA from at least one NE in S can reach the MS, and that NE will continue to send that LSA until is is acknowledged. After the MS receives and processes the LSA, G_{MS} and p are updated and a is removed from S. The MS repeats the above procedure until S is empty. When S is empty, G_{MS} is identical to G_{real}. Thus, it takes at most $|S|$ steps to make S empty and at which point the network has converged. ■

Therefore, as long as the MS assigns each NE the shortest onion-encrypted source route, the network is guaranteed to converge even when multiple failures occur simultaneously. In addition to the shortest route, the MS can optionally give a NE a *preferred* route which is not necessarily the shortest. A NE can use the preferred route to send management traffic to the MS and use the shortest route only to send LSAs. The flexibility of assigning preferred routes allows more advanced features to be implemented on the MS.

5) Resilience to Attacks: Under this security framework, only authenticated NEs can communicate with MSs via the MMS. When used with traffic isolation techniques (see Section III-B), data plane DoS attacks cannot disrupt management communication. Even if a NE is compromised, the attacker cannot modify the MMS frames in transit because they are all encrypted with the secret key of another NE. The compromised NE also cannot announce bogus connectivity to non-compromised NEs in order to attract traffic to it because the MS can detect the inconsistency in the LSAs.

A compromised NE can attempt to launch a DoS attack on the MMS by dropping frames in transit or by sending useless frames to the MS. The use of onion-encrypted source routes, however, offers both a mechanism to identify the origin of the DoS and a mechanism to isolate the offender once identified.

If the compromised NE is dropping frames, the MS detects it by stealthy measuring the packet loss rate along the prefixes of the lossy path using Stealth Probing [44] and then computes a new source route that avoids the compromised NE. A simple attacker that sends useless frames to the MS using its own source route would be trivially caught, since the source route identifies the sender. A sophisticated attacker could attempt to hide its identity by reusing a source route extracted from a frame it has forwarded, thereby making its attack traffic appear to come from the origin of the source route.[2]

Fortunately, using onion encrypted routes gives us strong assurance that any malicious packet received by the MS must have been sent by a NE listed in the packet's source route. The techniques of Zhang et al. [45] are then used to identify the malicious NE. Assume that a MS determines it is receiving malicious packets if they are sent at a rate above some detection threshold. Over time, the MS orders NEs to change the source routes they use. This allows identification of the attacker by forcing it to move its malicious traffic among different source routes, and the attacking node will eventually be the only node in common among the source routes along which malicious traffic arrived. The attacker's only strategy is to limit the number of malicious packets it sends to stay below the detection threshold, but this bounds the impact of its DoS attack. If an attacker is identified, the MS issues new onion routes that avoid it.

D. Assuring Liveness

To achieve liveness, beyond the ability to react to link or NE failures as explained in Section III-C4, there are additional challenges.

1) Protecting Against CPU Starvation: A common issue on NEs is CPU starvation caused by a run-away process or a data-plane DoS attack. However the MMS must maintain management communication channels during these events so that management agents or human operators can remotely diagnose and fix the problem.

The MMS relies on the NE's kernel scheduler to remain sufficiently live so that the MMS can send and receive frames.[3] To minimize the CPU cycles needed to run the MMS on NEs, the MMS is designed so that the most compute intensive work, i.e. route computation, is carried out on the MS.

However, even when the core kernel services of a NE remain live, it is possible for a process running on the NE (e.g., the OSPF or the BGP process) to consume so many CPU cycles that critical processes (e.g., the command shell) become unresponsive. For instance, this could happen when a misconfiguration causes hundreds of thousands of inter-domain routes to be mistakenly injected into an intra-domain routing process. If the command shell remains unresponsive, neither autonomic management agents nor human operators can remotely resolve the problem.

[2] Including nonces or timestamps in the source route could prevent this replay attack, but would require NEs share keys with all downstream NEs, rather than just the MS. We rejected that approach for scalability reasons.

[3] The problem of surviving arbitrary failures of the NE's kernel or operating system is intractable.

To enable recovery from this type of situation, the MMS provides a process management API and a packet filtering API. Using these APIs, a MS can command the MMS to return a list of the processes running on a NE, kill a particular process, change a process' priority, install an IP data plane packet filter, or reboot the NE. We elaborate on the features of these APIs in Section III-F1.

Together, these mechanisms allow an operator to remotely restore liveness to a NE's command shell via the MMS, investigate the cause of the problem and reconfigure the NE as needed to prevent a recurrence of the problem. In the extreme case an operator can remotely reboot a NE via the MMS.

2) Protecting Against Memory Outages: The MMS is designed to avoid "out of memory" errors by using static rather than dynamic memory allocation. In this way, as long as the MMS is successfully loaded at system startup time, it is unlikely to be impaired by memory allocation problems caused by misbehaving processes. This design requires the MMS to limit runtime state. In particular, this led to our use of source routing in the MMS, assuring that only MSs need to build the complete network topology, which requires memory proportional to the network size. The state stored by each NE scales only with the number of ports on the NE, which is known at boot time, and with the number of simultaneously active MSs communicating with the NE. In our implementation, the soft state maintained by a NE for each MS takes approximately 500 bytes of memory, so a small static array can support many simultaneously active MSs.

E. Evolving the MMS after Deployment

Networks are constantly evolving in ways difficult to anticipate. No matter how well the MMS has been designed and engineered, one cannot rule out the need for updating the MMS running in the field. Thus, the MMS must provide a robust means by which the MMS itself can be remotely managed and evolved.

Our approach to robustly evolving the MMS is to allow multiple versions of the system to operate over the same network at the same time. This allows the new version to be brought up and thoroughly tested before the old version is removed. Each version of the MMS operates independently and in parallel. Copies of all MMS frames are delivered to each version. A MMS frame contains a version number in the header, and a MMS version skips over frames marked for other versions.

In our design, management applications can specify which version of the MMS should carry its traffic through the use of a socket option. Packets sent by applications that do not specify a MMS version are handed to every version of the MMS running on that MS or that NE. Each copy is independently routed by its respective version of the MMS to the destination. Management applications therefore need not be aware of the old and new MMS versions and will continue to receive service even if the new version turns out to be faulty.

Management applications built on top of TCP will not see duplicated packets, as they will be discarded by TCP. For non-TCP applications, we leave it up to the application itself to ensure that these duplicated packets do not cause a problem. Robust UDP- and ICMP-based applications already cope with duplicated packets, and in our experiments, we did not find duplicated packets to be a problem. We choose this design so that management applications would work unmodified over the MMS without additional configuration, and we accept the performance cost of handling duplicated packets as a reasonable trade-off.

F. MMS Interfaces for Communication and Recovery

The MMS provides two key APIs: one for remote recovery to address liveness issues, and another to support existing network management applications that use TCP/IP protocols for communication.[4]

1) MMS API for Remote Recovery: We design the process management and packet filtering APIs based on the characteristics of common configuration mistakes, attacks, and management failure scenarios. They strike a balance between simplicity and the wide range of possible capabilities. These two APIs make it possible to recover from many situations where remote NEs are overloaded and unresponsive.

Through the process management API, a MS can command the MMS to return a list of the processes running on a NE, kill a particular process, change a process priority, start a process, or reboot the NE. When the process management API is invoked on a MS for a NE, a special MMS frame that carries the parameterized process management command is sent to the NE and interpreted by the MMS running on that NE. For example, when the destination NE receives a kill command with a process id parameter, the MMS kernel module running on the destination NE iterates through the kernel process table and sends a kill signal to the intended process. While extremely simple, in practice these capabilities are the primitives that operators and IT staff commonly use to mitigate problems and restore service.

The MMS packet filtering API allows IP data plane packet filters to be installed directly via the MMS without first obtaining a shell to run a user space application (in contrast to `iptables` [46] invocation, for example). When the packet filtering API is remotely invoked, a packet filter rule is sent from a MS to a NE. The MMS on the target NE directly communicates the rule to the packet filtering kernel module, for example `netfilter` [46], without competing with any user space applications for resources.

The security provisions of the MMS ensure these APIs can only be invoked by a valid MS, and the MS software itself can validate that the MS operators have the rights to perform the tasks.

In Section V, we demonstrate the use of these APIs to restore liveness under resource exhaustion conditions.

[4]Our MMS prototype provides two additional APIs: (a) domain-name resolution and dynamic registration, and (b) an overlay service running on MSs that enables management applications on NEs to communicate with each other and with external networks. These APIs further enhance the utility of the MMS.

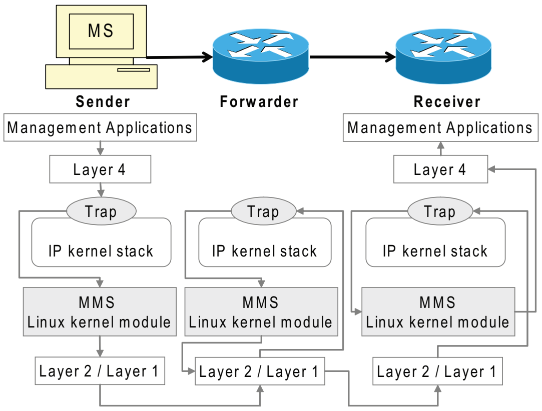

Fig. 2. High-level overview of the MMS implementation.

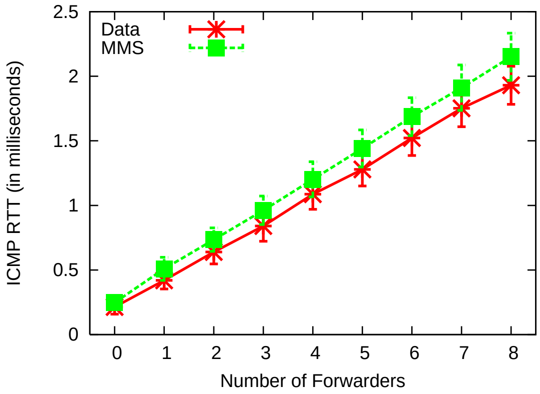

Fig. 3. Comparison of round-trip delay of ICMP packets using MMS channel versus using regular IP data channel.

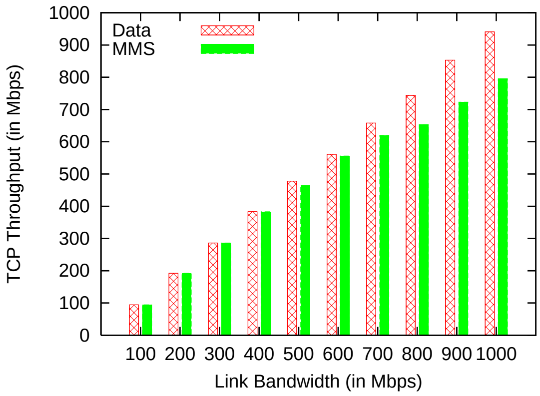

Fig. 4. Comparison of TCP throughput using MMS channel versus using regular IP data channel.

2) MMS API for Communication: There are a large number of existing network management tools that use the Internet Protocol for communication, such as SNMP pollers (e.g., MRTG, Cricket), remote scripting tools (e.g., rancid, expect), and PlanetLab administration tools. To maximize backward compatibility with existing management tools, the MMS provides a "virtual management LAN" abstraction. Specifically, when a MS is plugged into a network, the MMS presents a virtual management LAN that includes the MS and all authenticated NEs. Each node in the virtual management LAN is assigned a unique MMS management address. We intentionally make the management address the same length (32 bits) as an IPv4 address so that existing management applications can send messages to and receive messages from a management address as if using an IP address.

Inside the kernel, the MMS intercepts any packets sent over the virtual management LAN, encapsulates these packets into MMS frames, and transports the packets via MMS source routes.

G. MMS Implementation

Our MMS implementation is a Linux loadable kernel module, and it is introduced into the kernel network stack of the MS and NEs as shown in Figure 2. The MMS traffic is captured by a trap in the network stack and by-passes layer-3 IP processing completely. On the MS, traffic sent by a management application is injected into the MMS; the traffic is forwarded by the MMS on intermediate NEs, and is delivered via the MMS to the application running on the receiver NE.

The system consists of 21K lines of C code. Almost 17K lines of code are from the GNU MultiPrecision (GMP) library used to support cryptographic mechanisms. With additional engineering work, we could strip out the many unneeded functions from the library and reduce the code size.

IV. Performance Evaluation

In this section, we evaluate the delay and throughput overhead introduced by the secure forwarding mechanisms in MMS, the convergence speed of MMS routing in response to failures, and the speed of the recursive authentication mechanism used to authenticate NEs during initial network bootstrap.

The results show that the MMS has excellent performance, and it is practical to deploy the system.

Low forwarding overhead - We first measure the end-to-end delay and throughput overhead introduced by the MMS.

To measure the delay overhead, we connect nodes with 1 Gbps Ethernet links to form a linear chain topology. The sender and receiver exchange ICMP packets. We vary the hop count between the sender and receiver and compare round trip delays for ICMP packets carried by the MMS and by the regular IP data channel. Figure 3 shows that the round trip delays increase linearly with hop count, and the latency added by the MMS is less than 0.1 milliseconds per hop.

We measure the throughput overhead of MMS using a three-node chain topology, with a MS as the sender, one NE as the forwarder, and a second NE as the receiver. We use iperf [47] to measure the TCP throughput between the MS and the receiver. Using Emulab's configuration ability, we vary the bandwidth of the links connecting the three nodes. Figure 4 shows that the throughput difference between the MMS and the regular IP data channel becomes noticeable only after link bandwidth increases to 400 Mbps, and the best TCP throughput the MMS achieves is 800 Mbps.

Investigating further, the performance degradation is due to the encryption and decryption operations involved in using onion-encrypted source routes[5]. Nevertheless, the overhead

[5]Our implementation uses the "twofish" cipher with 128-bit keys.

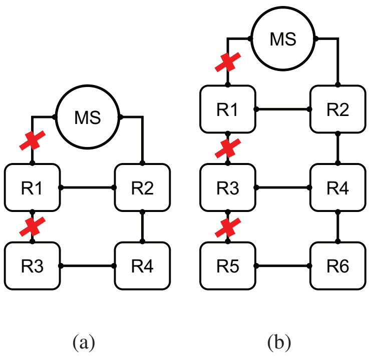

Fig. 5. Topology of the resiliency experiment. In (a), R1 simultaneously loses two links, and its initial LSAs to the MS are lost; MS detects failure of the link to R1 and it informs R1 to re-route through R2; LSA from R1 gets through allowing MS to re-compute and push a new route to R3. In (b), three links fail at the same time. The MS restores R1's route, receives LSA from R1, restores R3's route, receives LSA from R3, and finally restores R5's route.

Time line	Event
0 ms	R1 detects link failure & sends LSA to MS via route R1-MS, but LSA is lost
2 ms	MS detects the failure of link R1-MS, re-computes paths, and instructs R1 to use route R1-R2-MS
500 ms	R1 re-sends its LSA using the new route R1-R2-MS
505 ms	MS ACKs R1's LSA, re-computes paths, and instructs R3 to use route R3-R4-R2-MS

TABLE I
TIME LINE OF EVENTS TRIGGERED BY CONCURRENT LINK FAILURES IN FIGURE 5(A).

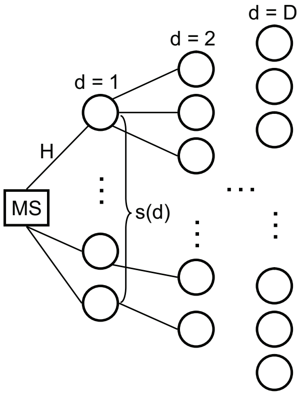

Fig. 6. Model for computing secure channels setup time. NEs are grouped by their hop-count distances from the MS. d stands for the hop-count distance of a group, H is hop latency, $s(d)$ is the number of NEs in the group d hops away from the MS.

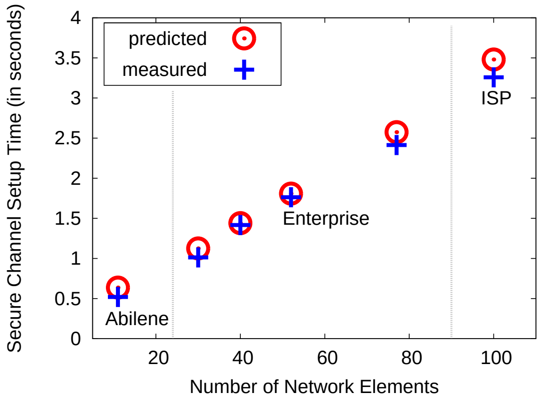

Fig. 7. Predicted secure channel setup times (plotted as circles), and measured setup times for real topologies (plotted as crosses).

imposed by the encryption is not large, and the security assurances made possible by onion-encrypted source routes outweigh the overhead.

Resilient routing - During network failures, NEs send LSAs to MSs and MSs re-compute and push out updated onion-encrypted source routes to NEs. When multiple failures occur simultaneously, some LSAs might fail to reach the MS. To address this issue, the MMS requires NEs to keep sending LSAs until an acknowledgment from the MS is received or the MS's soft-state is timed out. To evaluate the MMS's ability to maintain working communications in the presence of network failures, we construct the scenario as shown in Figure 5(a). In this scenario, two links fail at the same time and the failure of link R1-R3 cannot be immediately propagated to the MS as neither end of the failed link has a working route to the MS. Table I shows a timeline of the steps taken during re-convergence. The MS first detects the failure of link R1-MS and commands R1 to re-route using R2. When R1's LSA reaches the MS and notifies it of the failure of link R1-R3, the MS obtains an accurate view of the network and repairs R3's route.

In this case, one LSA retransmission is needed to update the MS with an accurate view of the network. Since the LSA retransmission timeout is 500 ms, it takes about 500 ms for the MMS routes to re-converge. We can recursively construct scenarios where more LSA retransmissions are needed. For example, two rounds of retransmissions are needed for the scenario in Figure 5(b) to re-converge. Recall that in Section III-C4, we proved that the MMS eventually re-converges even after multiple failures.

Fast secure-channel setup - When a new MS is brought up, it first authenticates its direct neighbors and then recursively authenticates the network as described in Section III-C3. To estimate how long this process will take in networks of different sizes, we first develop a simple model of the authentication process and validate the model using experimental data. We then use our model to predict the time required to establish secure channels in large networks.

Consider Figure 6. Given a network of n nodes, we divide the nodes into groups based on their hop-count distance to the MS. We define the nodes in group d to be the nodes d hops away from the MS and the number of nodes in this group to be $s(d)$. We define D as the maximum d; H as the hop latency; C_{node} as the time for a node to answer a challenge from the MS; and C_{MS} is the time for the MS to verify an answer. In our model, nodes in group d are challenged after all nodes in group $d-1$ have been verified, and the time cost for authenticating nodes in group d includes the MS sending challenges to the nodes, the nodes answering the challenges, and the MS verifying the answers. Let the time when the MS is brought up be time 0 and $t(d)$ be the time when nodes in group d have been verified, we have

$$t(d) = t(d-1) + d \times H + C_{node} + d \times H + s(d) \times C_{MS}$$

Solving for $t(D)$

$$t(D) = D \times (D+1) \times H + D \times C_{node} + n \times C_{MS}$$

The importance of this equation is that it highlights that nodes with the same hop-count distance to the MS can compute in parallel, resulting in the term $D \times C_{node}$ and implying that the time to authenticate will not be significantly affected even if network elements have slower CPUs than the MS and $C_{node} >> C_{MS}$. As shown in the equation, $t(D)$, the time the MS finishes authenticating and establishing secure channels to all nodes, is dominated by the term $n \times C_{MS}$ which grows linearly with the number of network nodes owing to the fact that the single MS has to verify answers from all nodes. And $t(D)$ is subjected to an offset bounded by the network diameter and average round-trip delay.

We conduct experiments to measure MMS channel setup time using three different types of topologies. The first is the Abilene backbone topology [48]; the second is an ISP backbone topology (AS 3967) derived from Rocketfuel [49] data; the third is a set of production enterprise network topologies used in [40]. Our measurements show that on the 3 GHz PC acting as the MS, C_{MS} is 27 milliseconds, and on the 800 MHz PCs serving as NEs, C_{node} is 45 milliseconds. Figure 7 plots the predicted and measured channel setup time for each topology. As shown, the measured times fit our analytical result.

According to the equation we deduced and experimentally validated, a new MS plugged into a network with one thousand NEs will take only about 30 seconds to build secure channels to all NEs.

V. Case Studies

To demonstrate the effectiveness of the MMS mechanisms, we give examples of how the MMS solves concrete problems that arise in network management. We implement these scenarios on Emulab [50] to illustrate these examples.

A. Self-Optimization in Wireless Mesh Networks

Wireless links can be asymmetric and links can have unpredictable packet loss rate. To show the MMS's effectiveness in wireless mesh networks, we experiment with an emulated network.

Figure 8 shows an emulation of a wireless mesh network using Emulab. The MS and the NEs run on PCs with a 3GHz CPU, running the Linux 2.6.12 kernel. They are richly connected to each other to emulate a mesh topology. The Emulab traffic-shaping nodes are employed to induce 40 percent packet loss between NE-1↔NE-4 and an asymmetric simplex-link is setup between MS←NE-3.

When the MS and the NEs are first brought up, the MS detects its immediate neighbor NE-3 and tries to authenticate it using the asymmetric link but fails. Meanwhile the surrounding nodes of NE-3 get authenticated to the MS, and provide

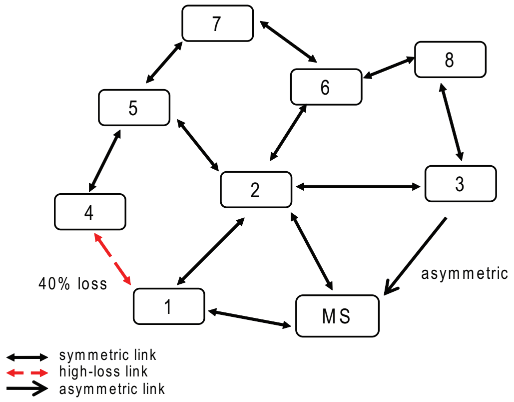

Fig. 8. An emulated wireless mesh network. The experiment shows MMS operating over lossy and asymmetric links that are common in wireless mesh networks.

alternate paths that the MS can use to reach and authenticate NE-3. Once all the NEs are authenticated, the MS has the full topology of the network and computes a source-route for NE-3 that avoids the asymmetric link. It takes 300ms to install a route on NE-3 that avoids the asymmetric link.

MMS handles lossy links in a similar way. It uses link quality estimates to detect the links with high packet loss. In our experiment, the link between NE-1 and NE-4 is induced with a 40 percent packet loss. When the MS and the NEs are brought up, the MS may authenticate NE-4 via the lossy link or through its other neighbors. Meanwhile the NEs use periodic HELLO messages to track the packet loss between their neighbors by measuring the time-gap between individual HELLO messages. Once all the NEs have been authenticated to the MS, the NEs start reporting the packet loss estimates to the MS via the LSAs. The MS uses these estimates as link weights in its network topology. When the MS receives the LSAs from NE-1 and NE-4, it detects the poor quality of the NE-1↔NE-4 link and re-computes source routes for NE-4 to avoid using the lossy link. In the experiment, it takes 500ms to detect the lossy link and route around it.

B. Recovery from Control and Management Plane Overload

A router's control and management planes run a variety of applications: routing daemons, traffic monitors, intrusion detection/prevention systems, and SNMP agents. Software bugs, network operation errors and network attacks (e.g., DoS, worms) can cause applications to consume excessive computing resources and can even render a router unreachable or unable to respond to remote management commands. For example during the breakout of the Slammer worm [51], many routers and switches became unresponsive. This was because the Slammer worm generated an enormous amount of packets with class D IP multicast addresses, and many routers and switches processed such multicast packets using their control plane CPUs [52]. As a result, routers' CPUs and memories were overwhelmed, forcing operators to physically visit the affected devices to install packet filters to block the worm traffic. This dramatically increased the time required to get the network back under control.

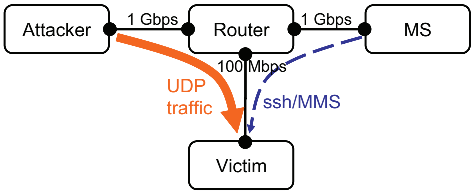

Fig. 9. Topology of the experiments in case study A. *Attacker* sends UDP packets to *Victim*; *MS* tries to establish a `ssh` session with *Victim* over MMS.

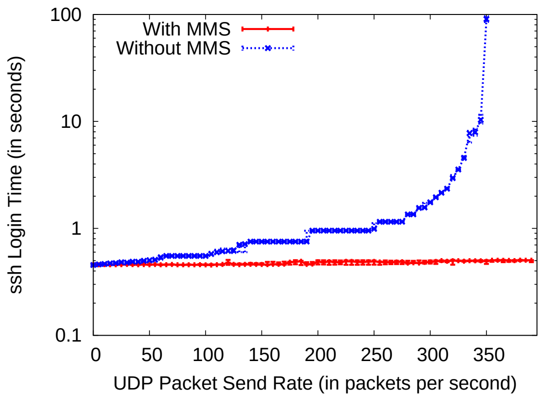

Fig. 10. `ssh` login time versus UDP packet rate in the `Snort` experiment. Each login is attempted 100 times and the first quartile, median, and third quartile of the login times are plotted. The Without MMS plot ends at 350 packets per second, because beyond that rate all of the attempted logins fail. The With MMS plot represents the sum of the time for invoking the packet filtering API and completing the ssh login.

In situations where the control and management planes are threatened by resource starvation, the MMS mitigates the threat through its packet filtering and process management APIs.

Using packet filtering API - Typically an operator installs packet filters by changing router configuration files or issuing shell commands such as `iptables` [46]. Ironically, under situations when the control/management planes are overloaded due to abnormal traffic and the deployment of packet filters is most desperately needed, it can be difficult to secure enough computing resources to change and commit the configuration or to launch the shell commands. The MMS APIs, however, provide a solution.

We demonstrate the benefits of the MMS using a real-world example based on `Snort`. `Snort` is an open source network intrusion detection/prevention system widely used in enterprise networks. When run in the inline mode, it holds packets in a user space queue and inspects them to make accept/drop decisions based on a set of rules. Unfortunately, when `Snort` (run in the inline debug mode) encounters bursty UDP packets, `Snort` can consume an excessive amount of resources and starve other applications[6]. We conduct an Emulab experiment to measure the impact of such starvation. We create a network as shown in Figure 9, where we run `Snort` version 2.4.3 on the Victim node with a 600 MHz CPU and the Linux 2.6.12 kernel and we send UDP packets from the Attacker node to the Victim node at increasing rate. Figure 10 shows the time it takes to `ssh` login to the Victim from the MS. We can see that without the MMS, `ssh` login becomes impossible when the UDP packet rate is merely above 350 pps because `ssh` is starved and times out. In contrast, with the MMS, `Snort` barely impacts `ssh` login. This is because the MMS provides a live communication channel, through which the MMS packet filtering API can be remotely invoked to block UDP packets, and then a `ssh` login via the MMS channel can be successfully completed. In such critical situations, the MMS can mean the difference between maintaining remote manageability or losing it completely.

[6] The problem exists on Linux kernels older than version 2.6.14.

Using process management API - Even when there is no malicious traffic, application software bugs can cause resource exhaustion. Anecdotally, it is known that certain bugs in the SNMP agents running on a tier-one provider's Alcatel 1630 switches had caused severe CPU overload on the switches when they received bursty SNMP queries. The problem persisted for minutes and the switches eventually shutdown. The consequence was that thousands of customers lost their local telephone services for half an hour and the provider had to report the incident to the Federal Communications Commission.

We conduct an experiment to emulate the scenario in the above example. We use the same network topology as in the previous experiment (Figure 9). We inject a bug into a SNMP agent (`snmpd`) so that it enters an infinite loop when it receives a certain SNMP query. We run this buggy SNMP agent with a high priority on the Victim node. When the SNMP bug is triggered, we find that `ssh` login to the PC from the MS becomes impossible because it times out.

The MMS process management API solves this problem. Through the live MMS communication channel, the MMS process management API can be remotely invoked to lower the priority of the mis-behaving `snmpd` process, and a `ssh` login can then be completed normally within one second. Again, in this situation, the MMS can mean the difference between maintaining remote manageability or losing it completely.

VI. Conclusions

Robust autonomic network management starts with robust support for management plane communications. We have argued for an autonomic network-layer foundation for management plane communications. Through designing, implementing and experimenting with the MMS, we have demonstrated the feasibility of such an autonomic network-layer foundation. We find that the strong security features in our fully functional implementation do not significantly hurt performance, even when run on commodity hardware. The latency and throughput performance will meet the requirements of many demanding management applications. We have also realized that in practice, a management communication subsystem can be under threats of compute and memory resource starvation. The MMS includes special recovery APIs that can be extremely useful in practice. The MMS software is freely available at `http://100x100.jot.com/mms`, and it readily supports higher-layer autonomic management systems.

References

[1] R. Sterritt, "Autonomic computing," *Innovations in Systems and Software Engineering*, vol. 1, no. 1, pp. 79–88, 2005.

[2] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[3] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Advanced Engineering Informatics*, vol. 19, no. 3, pp. 181–187, 2005.

[4] B. Jennings et. al., "Towards autonomic management of communications networks," *IEEE Communications Magazine*, vol. 45, pp. 112–121, October 2007.

[5] J. Strassner, N. Agoulmine, and E. Lehtihet, "FOCALE–A Novel Autonomic Networking Architecture," in *first Latin American Autonomic Computing Conference (LAACS), July*, 2006.

[6] A. Tizghadam and A. Leon-Garcia, "AORTA: Autonomic network control and management system," in *Computer Communications Workshops, 2008. INFOCOM. IEEE Conference on*, pp. 1–4, 2008.

[7] A. Konstantinou, *Towards Autonomic Networks*. PhD thesis, COLUMBIA UNIVERSITY, 2003.

[8] S. Balasubramaniam, K. Barrett, J. Strassner, W. Donnelly, and S. van der Meer, "Bio-inspired Policy Based Management (bioPBM) for Autonomic Communication Systems," in *7th IEEE workshop on Policies for Distributed Systems and Networks*, 2006.

[9] S. Davy, K. Barrett, S. Balasubramaniam, S. van der Meer, B. Jennings, and J. Strassner, "Policy-Based Architecture to Enable Autonomic Communications–A Position Paper," *Proceedings of IEEE CCNC, special session on Autonomic Communications, Las Vegas, USA, January*, 2006.

[10] J. Suzuki and T. Suda, "A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 2, pp. 249–260, 2005.

[11] H. Tianfield, "Multi-agent based autonomic architecture for network management," in *Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on*, pp. 462–469, 2003.

[12] Y. Cheng, R. Farha, M. Kim, A. Leon-Garcia, and J. Won-Ki Hong, "A generic architecture for autonomic service and network management," *Computer Communications*, vol. 29, no. 18, pp. 3691–3709, 2006.

[13] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, *et al.*, "Dynamic connectivity management with an intelligent route service control point," in *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, pp. 29–34, ACM New York, NY, USA, 2006.

[14] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai, "Tesseract: A 4D network control plane," in *Proc. Networked Systems Design and Implementation*, April 2007.

[15] M. Casado, M. Freedman, J. Pettit, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proc. ACM SIGCOMM*, 2007.

[16] "OpenFlow Switch Consortium." http://openflowswitch.org.

[17] J. Moy, "OSPF Version 2." RFC 2328 (Standard), April 1998.

[18] D. Oran, "OSI IS-IS Intra-domain Routing Protocol." RFC 1142 (Informational), February 1990.

[19] G. Malkin, "RIP Version 2." RFC 2453 (Standard), November 1998. Updated by RFC 4822.

[20] LAN/MAN Standards Committee of the IEEE Computer Society, "IEEE Standard for Local and metropolitan area networks: Medi a Access Control (MAC) Bridges - 802.1D," 2004.

[21] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable ethernet architecture for large enterprises," in *SIGCOMM*, 2008.

[22] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "ROFL: routing on flat labels," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 36, pp. 363–374, ACM New York, NY, USA, 2006.

[23] B. Ford, "Unmanaged internet protocol: taming the edge network management crisis," *SIGCOMM Computer Communications Review*, vol. 34, no. 1, pp. 93–98, 2004.

[24] F. Dressler, "Self-Organization in Ad Hoc Networks: Overview and Classification," *University of Erlangen, Dept. of Computer Science*, vol. 7.

[25] E. Gelenbe, R. Lent, and A. Nunez, "Self-aware networks and QoS," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1478–1489, 2004.

[26] G. Di Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research*, vol. 9, no. 2, pp. 317–365, 1998.

[27] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, and C. MIT, "Energy-efficient communication protocol for wireless microsensor networks," in *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, p. 10, 2000.

[28] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks," *IEEE TRANSACTIONS ON MOBILE COMPUTING*, pp. 366–379, 2004.

[29] C. Perkins, E. Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," 2003.

[30] D. Johnson, D. Maltz, J. Broch, *et al.*, "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks," *Ad Hoc Networking*, vol. 5, pp. 139–172, 2001.

[31] R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. Addison-Wesley Professional, 2000.

[32] B. Kumar, "Integration of security in network routing protocols," *ACM SIGSAC Review*, vol. 11, no. 2, pp. 18–25, 1993.

[33] B. Smith, S. Murthy, and J. Garcia-Luna-Aceves, "Securing Distance-Vector Routing Protocols," in *NDSS*, February 1997.

[34] L. Zhou and Z. Haas, "Securing ad hoc networks," *Network, IEEE*, vol. 13, no. 6, pp. 24–30, 1999.

[35] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti, "Secure pebblenets," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pp. 156–163, ACM New York, NY, USA, 2001.

[36] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[37] S. Cheung, "An efficient message authentication scheme for link state routing," in *13th Annual Computer Security Applications Conference*, pp. 90–98, IEEE Computer Society, 1997.

[38] R. Hauser, A. Przygienda, and G. Tsudik, "Reducing the Cost of Security in Link State Routing," in *Symposium on Network and Distributed Systems Security (NDSS 97)*, pp. 93–99, 1997.

[39] Y. Hu, D. Johnson, and A. Perrig, "SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, 2003.

[40] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, August 2004.

[41] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do Internet services fail, and what can be done about it?," in *Proc. USENIX Symposium on Internet Technologies and Systems*, 2003.

[42] Z. Kerravala, "Configuration management delivers business resiliency." The Yankee Group, Nov 2002.

[43] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Information Hiding*, pp. 137–150, 1996.

[44] I. Avramopoulos and J. Rexford, "Stealth Probing: Efficient Data-Plane Security for IP Routing," in *USENIX Annual Technical Conference*, May 2006.

[45] X. Zhang, H. Chan, A. Jain, and A. Perrig, "Bounding packet dropping and injection attacks in sensor networks," Tech. Rep. CMU-Cylab-07-019, Carnegie Mellon, 2007.

[46] "netfilter/iptables." http://www.netfilter.org.

[47] "Iperf – The TCP/UDP Bandwidth Measurement Tool." http://dast.nlanr.net/Projects/Iperf.

[48] "Abilene Backbone Network." http://abilene.internet2.edu/.

[49] N. Spring, R. Mahajan, and D. Wetheral, "Measuring ISP topologies with RocketFuel," in *Proc. ACM SIGCOMM*, August 2002.

[50] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. Operating Systems Design and Implementation*, pp. 255–270, December 2002.

[51] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer worm," *IEEE Security and Privacy*, vol. 1, pp. 33–39, July–August 2003.

[52] G. Travis, E. Balas, D. Ripley, and S. Wallace, "Analysis of the "SQL Slammer" worm and its effects on Indiana University and related institutions," *Technical report, Advanced Network Management Lab, Indiana University*, February 2003.