

Meta-Management System for GENI

GDD-06-37

GENI: Global Environment for Network Innovations

April 20, 2007

Status: Draft (Version 0.2)

Note to the reader: this document is a work in progress and continues to evolve rapidly. Certain aspects of the GENI architecture are not yet addressed at all, and, for those aspects that are addressed here, a number of unresolved issues are identified in the text. Further, due to the active development and editing process, some portions of the document may be logically inconsistent with others.

This document is prepared by the Backbone Working Group.

Editors:

David A. Maltz, *Microsoft Research*

Eugene Ng, *Rice University*

Zheng Cai, *Rice University*

Hemant Gogineni, *Carnegie Mellon University*

Hong Yan, *Carnegie Mellon University*

Hui Zhang, *Carnegie Mellon University*

Revision History:

Version	Changes log	Date
v0.1	Original version posted	12/01/06
v0.2		04/20/07

Meta-Management System for GENI

David A. Maltz Zheng Cai, T. S. Eugene Ng Hemant Gogineni, Hong Yan, Hui Zhang
Microsoft Research Rice University Carnegie Mellon University

April 20, 2007

1 Introduction

Management of the geographically distributed GENI facility is a complex problem. The current proposed management architecture envisions a GENI Management Core (GMC) that remotely manages a set of physical computing and network elements (*GENI components*). The GMC must support remote network management capabilities such as remote component diagnostic and reboot, network service migration, and network maintenance operations. These capabilities need to remain functional even when other software systems have failed. To realize these remote management and diagnostic capabilities, GENI requires a robust communication channel for management traffic that is independent of the operation and health of any other GENI services.

Leveraging the existing Internet network to provide a physically out-of-band channel between the GMC and the GENI components can only be a stop-gap solution to the problem because of the security risks, reliability problems, and management complexity of the existing Internet. As pointed out in the GENI Facility Security Design Document [1], the desire is for the management of GENI to not depend on the Internet:

Initially, we envision GENI management commands will be carried over the Internet for convenience (subjecting them to all the problems of Internet security and reliability); as GENI construction proceeds, however, it may be possible to move the GENI control plane to running over GENI itself, reducing the likelihood of such attacks.

1.1 Requirements for GENI Management Systems

We argue that GENI should be designed as a self-contained architecture without dependencies on other networks, where both GENI's management system and its data services run over the same physical network.

To achieve this end, the GENI design criteria should include the following three requirements.

- First, GENI should be designed such that a management channel is available to each GENI device so long as any GENI path exists from that device to a host that is part of the GMC.
- Second, GENI should adopt a single coherent approach to management connectivity to devices at all layers — including both routers and optical switches. GENI should also provide a uniform means by which a device discovers its neighbors at each layer.

- Third, GENI should support existing management applications by providing an IP bearer service between the hosts implementing the GMC and the GENI components.

1.2 A Meta-Management System for GENI

This document describes how these requirements can be realized through the creation of a *meta-management system* for GENI. We describe the motivating use-cases for the system, expand on the design requirements for the system, define the key features of the system, and present a system prototype as a proof-of-concept.

The meta-management system establishes and maintains a secure and robust communication channel between GENI components and the GMC as long as there is physical network connectivity. The key architectural feature of the system is the logical separation between the meta-management system and the rest of the control and management software components. The meta-management system is an extremely simple, self-contained software system. Its operation does not depend on any other software components or functions, such as a set of correct data-forwarding tables produced by the routing protocol, and it does not depend on complex configuration files, such as those required to bring up a router or routing protocol today.

1.3 Management Differences between PlanetLab and GENI

Comparing the management needs of PlanetLab [2] and GENI motivates the need for a meta-management system. In PlanetLab, a *node* is an end-host machine that is statically assigned “at least one non-shared (not NAT’ed) IP address” so that it is reachable via the Internet from the PlanetLab Consortium (PLC) management servers: the *slice authority* and *management authority* servers.

Compared to PlanetLab, GENI management is different in two major ways: (1) a GENI component may not be an end host, instead, it may be a *network element* (e.g., programmable router and switch) that itself routes traffic and affects network reachability. Rather than managing *end hosts* through a presumably healthy network, GENI needs to use a management network to manage *network components* that themselves also form the management network; (2) as a potential starting point for the “Future Internet”, GENI should *not* assume the existence of a second physically out-of-band management network, such as the Internet, that provides reachability from the management servers to all the GENI components.

The contributions of the proposed meta-management system will be twofold. First, it will provide GENI an independent, robust, and secure communication channel for network management. Second, it will show the feasibility of using a logically separate communication channel for remotely managing a network using the same physical network facility — addressing a critical chicken-or-the-egg problem in today’s Internet architecture.

2 Meta-Management Use Cases

As shown in Figure 1, the meta-management system manifests itself in the form of a *meta-management service layer* (MMSL) embedded into the software stacks of both the *component manager* (CM) that “exports a well defined, remotely accessible interface” for component management, and the *management authority* (MA) that “provides an operations center” [3]. The meta-management service layer is a substrate that management functions of the GMC will use as

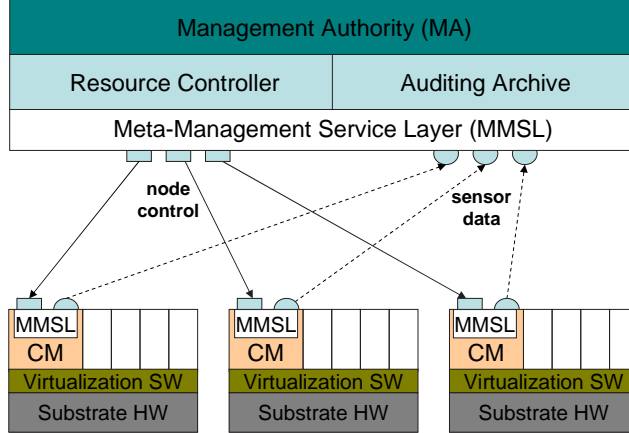


Figure 1: Role of the meta-management system.

a communication channel between MA(s) and CMs. We now show three use cases to illustrate the point.

Remote reboot: The MA can remotely reboot GENI components which include both end hosts and a variety of network elements such as routers and switches. To reboot a GENI component, the MA uses the meta-management service to send a reboot command to the component addressed with a component identifier¹.

The meta-management service disseminates the reboot command to the CM that controls the target component. After reading the command from its meta-management service layer the CM reboots the component. During the time the component is being rebooted, the MA should be able to reach and continuously manage the other physically connected GENI components, regardless of the disruption of data traffic caused by the reboot. Immediately after the component is booted up, its management channel should be restored. Compared with today’s PlanetLab remote reboot, GENI remote reboot (1) works on not only end hosts but also routers and switches; (2) does not rely on Internet (thus not subject to various Internet reliability and security problems) and does not require configuration of a management IP address on the target component.

Remote maintenance and upgrade: The MA can remotely upgrade software, even router software that routes data traffic (e.g., OSPF daemon), on a GENI component without disrupting the management traffic. The MA uses the meta-management service to send a **kill-process** command to the target component. The component receives the command from its meta-management service layer and performs the requested process shut down. After the shut down, although the other network elements stops sending data traffic to the component, the MA can still use the logically out-of-band meta-management channel to reach that component and install a newer version of the router software on it. After installation, the new router software will be activated and the component will rejoin the GENI network. During the upgrade procedure, from the data traffic routing’s perspective the routing component is taken down for maintenance and then brought up; while the routing of management traffic is uninterrupted.

¹Component identifiers are discussed in Section 3.1.4 of the GENI architecture overview design document [3]

Failure notification: When a network link or node fails, the meta-management service layer of the affected component(s) will detect the failure and notify the MA, where an up-call will be made by the MA's meta-management service layer to other applications sitting on the top — aiding development of applications that manage the overall GENI infrastructure. While failure of GENI components may interrupt meta-management communication channels, the meta-management system itself will restore the channels. Experimental results so far indicate this restoration will take less than a second.

3 System Requirements

The design of GENI meta-management system has been guided by an explicit set of requirements, which can be roughly grouped into three categories.

3.1 Functional Requirements

The first category concerns the basic functionalities of the system:

Run as a self-contained lightweight service: The operation of the meta-management system should not depend on the correctness of data traffic routing in the GENI network or the Internet. For example, even when the routing tables of all the GENI routers get corrupted due to misconfiguration or software bugs, the meta-management system should remain functional, allowing network operators to reach the affected routers and fix the problem.

Bootstrap with minimal configuration: The meta-management system should be able to bootstrap with as little pre-configuration as possible to minimize the chance of a configuration error. In contrast, today's IP routers are usually bootstrapped with a large amount of configuration information that includes IP addresses, routing protocols, routing policies, and packet filters. These configurations and the dynamic forwarding state they generate require a myriad of ad hoc scripts and systems in the management plane to validate, monitor, and update. The result is a complex, fragile, and failure-prone network [4]. For example, a mistake in configuring packet filters or routing policy (either during installation or run-time maintenance) may disrupt the data delivery service of a router and disable the communication between the management system and the router.²

Apply to heterogenous GENI network devices: The meta-management system should be applicable to all of the different types of network elements planned for use in the GENI environment. The only assumption the meta-management system makes is that link layer connectivity between physical neighbor nodes is automatically configured on device power-up.

Serve existing network management systems: The meta-management system should be made agnostic to the information it carries — it is not itself a complete control plane. Rather, it decouples the protocols that transport information to the GENI devices from the decision-making

²A commonly reported mishap among network operators is installing a packet filter that accidentally covers the IP address from which they are logged into the router, thereby immediately severing their management connection and forcing a fall-back to some other way to connect to the router.

logic that the control plane uses to determine what information to send. This decoupling will facilitate network researchers implementing their own custom control planes, as they can take advantage of the services offered by the underlying secure and robust meta-management communication framework.

3.2 Security Requirements

The next set of requirements concern the security properties:

Provide authenticated management commands to CMs: Provide a basis that CM's can use to validate that the management commands they receive have come from an authorized MA that they should obey. Reduce or eliminate the ability of attackers to prevent messages from an MA from reaching the intended CM.

Survive the compromise of network elements: The meta-management system should be capable of detecting and “failing off” compromised network elements to prevent them from launching further attacks or taking control of the GENI network.

Survive the compromise of a GMC MA: Even under the unlikely situation where one or more GMC MAs are compromised, the meta-management system should not allow the attacker to take over the whole GENI network.

Survive attacks exploiting GENI software vulnerabilities: GENI will run a wide range of software including management software providing GENI services and experimental software run by experimenters. The meta-management system should not be subject to security problems caused by bugs and vulnerabilities of other software, and the system itself should be kept as simple as possible to reduce vulnerabilities.

Survive DoS attacks against the GENI management infrastructure: Under DoS attacks, the meta-management system should continue carrying management commands from the GMC MA to the network elements.

3.3 Robustness Requirements

The final set of requirements concern the robustness of the meta-management system:

React to network changes quickly: Planned or unplanned network changes such as router maintenance and link failures can cause disruption to both data and management traffic. The meta-management system should be optimized to react to network changes quickly and make sure that management traffic can reach a network element as long as it is physically connected to a GMC MA.

Tolerate GMC MA failures: The meta-management system should provide built-in support for GMC MA redundancy such that it can survive the failure of a MA.

System Requirement	Design Element
Self-contained system	Independent meta-management routing (Section 4.1)
Minimal configuration	Lightweight routing (Section 4.1) and easy bootstrapping (Section 4.3)
Heterogenous network devices	Simple interface to devices (Section 4.5)
Compatible with existing management systems	Meta-management APIs; Provide IP service (Section 4.6)
Survive CM and MA compromise	Threat model and defense (Section 4.2)
Survive attacks against software vulnerabilities	Meta-Management system as an independent and thin service layer
Survive DoS attacks	Logically out-of-band meta-management channel
Robustness	Responses to network failures (Section 4.4)

Table 1: Design requirements for GENI meta-management system. References to design elements that address them are provided.

3.4 Requirements Summary

Table 1 summarizes these system requirements and refers to different meta-management system elements in our reference design that address the requirements.

4 System Design

This section provides a reference meta-management system design that aims to meet the above requirements. The design also serves as a feasibility study that explores and addresses the challenges for such a physically in-band and logically out-of-band meta-management system.

In later sections we describe a prototype we have constructed based on this reference design.

4.1 Management Traffic Routing

The core functionality of the meta-management system is to provide a self-managed communication channel between a few MAs and many CMs. The unique few-to-many traffic pattern might allow us to design a simple yet robust and secure protocol to independently route management traffic. We now describe a reference design for routing management traffic between MA(s) and CMs. We present the base protocol in this section and discuss the security enhancement in Section 4.2.

Each MA floods a *path explorer* message through the GENI network to find CMs. To ensure fast recovery from link failures, the path explorer is sent periodically (e.g., every 20 ms), and can also be triggered by topology updates. The flooded path explorer carries a unique identifier of the MA. When a CM receives the path explorer from one of its network interfaces, it appends its own component ID to the path explorer and floods the path explorer out its other network interfaces. It also caches the full MA-to-CM path so that it can later use the reverse path to reach the MA (we assume that all links are bidirectional). Path explorers carry sequence numbers so that CMs can avoid unnecessary re-flooding.

A CM sends link state advertisements (LSAs) to a MA the first time it receives a path explorer from that MA, or when there is a link state change (e.g., link failure). As mentioned above, when sending LSAs to a MA, a CM uses the reverse source route it recorded from the latest path explorer.

After collecting LSAs from all CMs, each MA forms a full view of the GENI network topology. The MA then uses a single source shortest path algorithm (e.g., Dijkstra) to compute paths to the CMs and use the paths to deliver management commands. On network failures, the MA receives

LSAs from affected CMs and re-compute the MA-to-CM paths using, for example, an incremental shortest path algorithm to bypass the failed links and/or nodes.

4.2 Threat and Defense

The GENI infrastructure will involve many geographically distributed components, making it critical that the platform survive and recover even if attackers are able to compromise individual pieces of the infrastructure. Further, it must be possible to carry out the initial response to a compromise from a remote location, as some parts of the infrastructure may not have technical staff available 24x7.

Specifically, we consider the following threat models and propose corresponding security enhancements to the base routing protocol described earlier.

4.2.1 CM Compromise

A basic security property we want the meta-management system to satisfy is that a compromised CM cannot order other CMs to do anything, and cannot forge management messages from other CMs. This can be achieved by having each CM to share a private/public key pair with a MA so that MA messages can be signed by the MA and be validated by the CMs. We discuss secret key generation and distribution in Section 4.3.

4.2.2 MA Compromise

As a logically centralized system, if a MA were compromised, it could order CMs to execute “bad” management commands and wreck havoc on the network. However, recovery is still possible. Other MAs can query the management commands executed by each CM and do sanity checks, allowing a compromised or misbehaving MA to be identified. As long as a quorum of MAs remain uncompromised they can generate a new a private/public key pair and remotely re-key CMs. Notice that while it can totally disrupt data traffic, a compromised MA *cannot* disrupt the management traffic between other MAs and the CMs. This is one of the benefits of having control traffic traversing a secured channel that is logically separate from paths traversed by data packets. Once re-keyed, the CMs will ignore the compromised MA.

As a point of comparison, with today’s data networks it would be hard to recover from a compromised management station as the compromised management station can block other backup management stations from reaching the network devices.

At the level of the control plane, the security of OSPF today is based on a single secret key stored in plain-text in the configuration file. If any network device is compromised, the key is compromised, and incorrect LSAs can be flooded through the network. The attacker could then DoS all the network devices by forcing them to continuously rerun shortest path computation or draw traffic to itself by forging LSAs from other routers indicating broken links.

4.3 Configuration and Bootstrapping

The meta-management system is designed to bootstrap with as little configuration as possible to minimize the chance of errors due to configuration mistakes. So long as this minimal configuration is persisted and protected from corruption, the meta-management system will be robust to arbitrary system crashes and reboots — this can be achieved using non-volatile storage within each node.

The entire configuration state for the meta-management service (MMS) on a CM consists of: a network certificate, and a MA certificate. When the MMS is first installed on a CM, a USB key storing the network certificate and a MA certificate is plugged into the device. The MMS generates (or gets from the CM) a component ID, which can be as simple as a randomly generated 128-bit number, and a private/public key pair. The network and MA certificates, the component ID, and the key pair are stored into a nonvolatile memory. The MMS then encrypts the information with the public key of the MA, and writes it back onto the USB key. When the USB key is eventually inserted into a MA, the MA will have a secret channel to each device and a list of the valid component IDs.

4.4 Robustness

With the proposed meta-management routing design, valid routes between each MA and every physically reachable CM can be found within twice the maximum propagation delay of the end of a series of link or device failures, regardless of the order in which the failures occur (this is roughly 200 ms, even for the largest terrestrial networks).

Additionally, if all MAs should somehow fail, a new MA can be introduced into the network, and the system will quickly enable communications between the new MA and the individual CMs. This same mechanism also allows a GMC MA to reach all CMs even when all CMs have failed.

To further enhance the robustness of the meta-management service, the MA pre-computes k alternate source routes for each CM. If a CM observes a physical link failure, it will use one of the backup source routes to communicate the link failure to the MA and at the same time re-establish a new default spanning tree source route with the MA. This way, the meta-management system can route around a failed link quickly. In the common case that not all k pre-computed paths are affected by the failure, the meta-management communication can be established very quickly. In the case that all k paths are affected by the failure, the dynamically generated path will enable re-establishment of meta-management communication as long as there is physical connectivity between the CM and the MA.

4.5 Device Interfaces

In order for heterogenous GENI network devices to run the meta-management service, they must expose certain software interfaces to the meta-management service layer. Basically, the meta-management software needs to detect available network interfaces and exchange data with neighbors. It's also preferable that traffic sent and received by a device can be prioritized. We do case analysis on different types of GENI devices:

Ethernet Devices: Ethernet switches or PCs with Ethernet cards can auto-detect Ethernet interfaces. For example, the widely supported POSIX API `ioctl()` can be used to list and retrieve information on all Ethernet interfaces. A broadcast or multicast MAC address can be used to send Ethernet frames to neighbors and a queuing discipline can be employed to give priority to the meta-management frames.

Electronic Circuit & Optical Switches: In addition to Ethernet devices, the GENI backbone will have other devices such as electronic circuit switches (SONET Digital Cross-connect) and optical switches. In general, such a device has a control processor that manages the physical

device. In addition, there is usually physical level support to de-multiplex control and management traffic. For example, SONET has the dedicated DCC channel and DWDM has dedicated optical supervisory channels (OSC). This allows the meta-management traffic to be differentiated from user data traffic on the link.

4.6 Management Software Interfaces

There are a large number of existing network management tools that use the Internet Protocol for transport, such as the PlanetLab administration tools, SNMP graphers (e.g., MRTG, Cricket) and remote scripting tools (e.g., rancid, expect). Even the current GENI documentation talks of management traffic as being sent across the IP network. In designing the meta-management system, however, we differentiate between the IP-based transport service expected by these mechanisms and the requirement for an “IP network” to provide them.

Enabling existing management tools to function across the meta-management system requires only that an IP network stack running on the MA be able to exchange IP packets with an IP network stack on each CM.

To achieve this, the meta-management system and the GMC work together to expose an “IP interface” to the management tools.

The GMC’s responsibility is assigning IP addresses to each CM. Once the meta-management bootstraps using its extremely minimal configuration, the GMC will then assign each CM a unique IPv4 address from private address space. Each CM will also auto-configure itself with unique IPv6 addresses.

The meta-management system appears to the network stack on the MA and each CM as an interface onto a LAN. The meta-management system provides translation between IPv4/v6 addresses and the matching devices’ MA/CM IDs (i.e., the equivalent of ARP). The meta-management system itself will then provide transport of IP packets by encapsulating them into meta-management frames and transporting them as normal.

5 A Meta-Management System Prototype

As a proof of concept, we build an MMS prototype as a Linux kernel module. It runs on distributed MAs and CMs to form a simple yet robust and secure substrate that provides the meta-management services we proposed in previous sections. We want to show that it is feasible to have a simple implementation that provides the MMS service, requires minimal dependency on the rest of the system, and supports existing management applications with no modifications.

Figure 2 gives a high-level overview of the MMS prototype we have built. Our MMS program is compiled into a Linux kernel module and it is hooked into the kernel network stack of the MA and CMs shown in Figure 2. On the MA, traffic sent by a management application is injected into the MMS infrastructure; the traffic gets forwarded by an CM as the forwarder, and then gets delivered to the application running on the receiver CM.

The subsections that follow describe the properties of our implementation, along with an overview of the services it provides. We then describe in more detail the implementation of the key services and show the life of a management packet as it traverses the network through the MMS infrastructure. Finally, we show how the MMS can be used in concert with VINI to provide an overall system like that required by GENI.

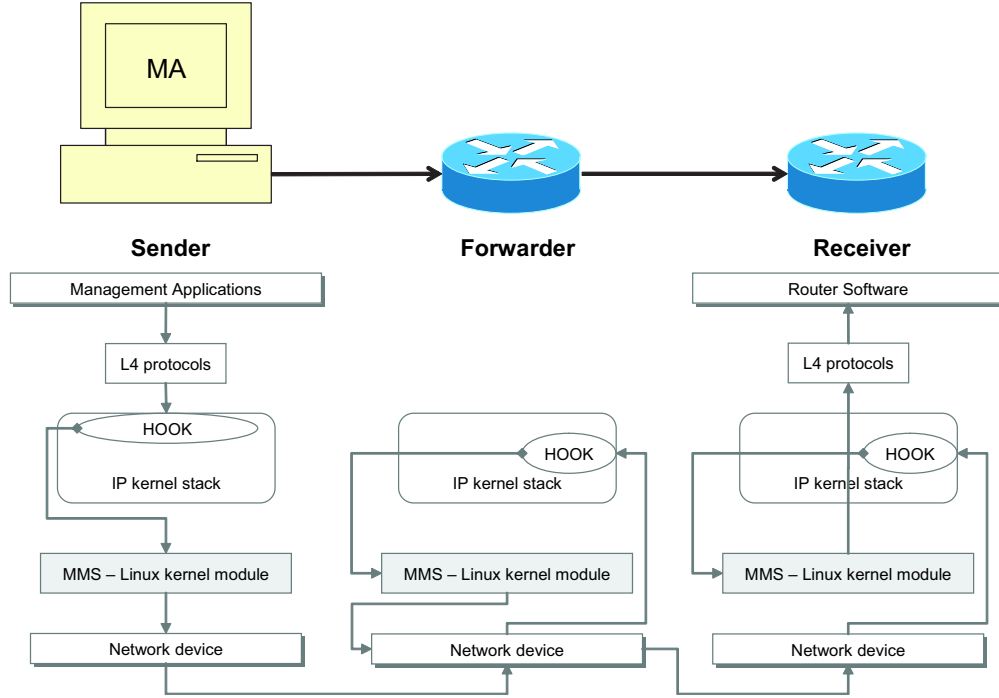


Figure 2: High-level Overview of the MMS prototype.

5.1 Features of the Prototype’s Design

In addition to meeting the design requirements described in Section 3, our prototype embodies two additional noteworthy goals.

Keep it simple and minimize dependency: As a software module that provides the “kernel” communication services, the MMS implementation should be kept as simple as possible. It should have a small code base and a small memory footprint. Our MMS prototype is implemented as a self-contained Linux kernel module with less than 2,200 lines of C code. We leverage the services provided by the Linux kernel as much as we can to avoid implementing sophisticated algorithms or using third-party libraries. For example, as discussed in Section 5.3.1, we use the Linux kernel cryptographic API [5] for encryption and decryption, rather than depending on third-party libraries such as OpenSSL [6].

Our MMS design enables a MMS node to self-bootstrap with an extremely small set of configuration state, as described in Section 5.3.2. In addition, the whole MMS network formed by MAs and CMs running the MMS has the capability to be booted up without complex configurations; Section 5.3.3 describes a simple recursive authentication protocol that enables this capability.

Serve existing network applications unmodified: There are a large number of existing network applications that use the Internet Protocol for transport. Enabling existing applications to function across the MMS requires that an IP network stack running on the MA be able to exchange IP packets with an IP network stack on each CM. To achieve this, the MMS needs to expose an “IP interface” to the applications. In our design, the MMS assigns each CM a unique IPv4 address

from a private address space. The MMS appears to the network stack on the MA and each CM as an interface onto a LAN. The MMS itself then provides transport of IP packets by encapsulating them into “MMS frames” and transporting them through the MMS system. Section 5.3.1 elaborates on the technical details of this design decision. To make the IP bearer service provided by the MMS complete, we also integrate domain name resolution into our prototype. However, we leverage an existing DNS server implementation (BIND [7]) rather than building our own in the MMS kernel module. The downside of this design is that when the DNS server fails for any reason, the MMS would not be able to resolve domain names, though applications using IP addresses are still functional. This is an example where we have to make compromised design choices to keep the MMS clean and simple.

5.2 Service Overview

In this section, we elaborate on the services provided by the MMS and highlights their benefits. For each service we list the key motivations for the existence of the service and summarize how the service meets those needs.

5.2.1 Interface Management Service

Protecting the management communication channels from being shutdown by accident: Human error is a major source of network failures [8]. In today’s network, in order to stop the data services on an interface, the interface is often shutdown. If an operator remotely logs in a CM and accidentally shuts down all the network interfaces, there would be nothing the operator can do to restore the management connection remotely. The MMS manages the network interfaces and prevents such human errors. The MMS provides a mechanism for data services shutdown without shutting down the interfaces so management communications can continue. This is a simple service, but still an important one.

5.2.2 Unified Management Communication Service

Providing management communication channels that do not depend on the data services: The MMS logically separates management communication channels from data communication channels so that they no longer share the same fate. Today’s routing processes such as OSPF and BGP compute and update routing tables, and the data packets are forwarded based on decisions made by those routing processes. Management traffic, however, is handled by the MMS from the time the traffic leaves the management application and enters the network stack to the time it is delivered by the MMS to the application running on the other side of the network. The MMS forwards management traffic using its own internal mechanisms. This architecture eliminates a whole class of potential problems caused by data services configuration errors or corruptions (e.g. the AS7007 incident [9], incorrect packet filter configurations, etc.).

The MMS also makes remote router maintenance easier to perform. Today, because management traffic is also carried by the IP service, the operator needs to maintain the IP service to the target router during maintenance. However, in order to avoid user data traffic from being affected, the operator needs to inflate the OSPF link weights of the target router to cause user data traffic to route around it. This is a lot of ad hoc guess work. With the MMS, the data services can simply be administratively shutdown, and remote maintenance can be performed over the MMS.

Exploiting all available physical connectivity to connect MAs to CMs: Today, IP network routers sometimes use the telephone network as a backdoor for management. Routers in the next generation Internet will likely use the legacy Internet as a backdoor for management, too. The design philosophy for the MMS is that as long as there is physical connectivity between an MA and an CM, the connectivity should be *transparently* leveraged to provide a management communication channel. When an MA sends a management packet to an CM, the MMS uses its own routing subsystem to find a path to reach the CM using all available connectivity, including any Internet tunnels. The independence of the MMS from data services allows the MMS to run over such backdoor links that do not carry data traffic.

Maintaining the health of management communication channels in the presence of network failures: The MMS is also a adaptive system that reacts quickly to network nodes or link failures and recomputes its internal routing decisions accordingly. As long as connectivity still exists between an MA and an CM, the MMS discovers and provides a management communication channel.

5.2.3 IP Bearer Service

Exposing an IP programming interface to management applications: There are a large number of existing network management tools that use the Internet Protocol for transport, such as the PlanetLab administration tools, SNMP graphers (e.g., MRTG, Cricket) and remote scripting tools (e.g., rancid, expect). Despite all the new services added by the MMS, the MMS needs to maintain backward compatibility to enable existing management tools to function without any change. For this reason, the MMS exposes an IP interface to applications, rather than creating a new API and forcing application developers to switch to it. The MMS assigns each MMS node an IP address (denoted as the *virtual management IP address*) which is not statically bound to any physical network device on the node. Management applications use the normal network API (e.g., Berkeley sockets) to send and receive packets, except that instead of passing the IP address of a physical network interface into the network API (e.g., `bind()`, `accept()`, `sendto()`), they invoke the API functions with the *virtual management IP address* of the MMS node.

Providing an address resolution service using the existing DNS protocol: The MMS allows each MA and CM to be configured with an easy-to-remember name (denoted as the *management name*). The MMS maps each MA and CM's *management name* to its *virtual management IP address* and it supports translation between *name* and *address* using the existing DNS protocol. The MMS module on each MA keeps track of the name-address mappings and acts as an address resolution server. When a *management name* needs to be resolved, a DNS request is sent to an MA. The MA looks up its mapping records and returned the *virtual management IP address* as a DNS record. Using the existing DNS protocol avoids adding a new API for MMS address resolution, i.e., the widely used library routines such as `gethostbyname()` can be borrowed unmodified to lookup MMS *management names*.

Proxy Gateway Service to Internet: An CM might or might not have public Internet interfaces. As part of the IP bearer service, the MMS should provide a proxy service to allow CMs that do not have public Internet interfaces to access Internet. There are scenarios where this can be

useful. For example, an CM might need to download software from external Internet hosts, or it might need to fetch security credentials from an external principle.

5.3 Design of the MMS Prototype

We now present the detailed design of the key services in our MMS prototype.

5.3.1 Logically Out-of-band IP Bearer Service

The management communication channels between MAs and CMs are physically in-band but logically out-of-band, meaning that management traffic and data traffic share the same physical links and devices but they are routed with different mechanisms and thus no longer share the same fate in presence of routing failures. In the meanwhile, to enable existing network management applications that use the Internet Protocol for transport to function across the MMS, the MMS exposes an IP interface.

To achieve the goal of providing out-of-band IP bearer service, we tackle the following design issues and technical issues when we build our prototype:

Accepting IP Packets from Management Applications: When an application sends a packet out, the MMS needs to determine whether it is a data packet that should use the data channel, or it is a management packet that is to be taken charge by the MMS. Our solution to this problem is to create an MMS management address space and assign each MMS node, CM or MA, a network-wide unique IP address allocated from that address space. In our prototype, we simply use a private IPv4 address space, for example 192.168.0.0/16, as the MMS management address space. When assigning an management address to an MMS node, we create a virtual network device [10] named *mm0* and associate it with the management address. The *mm0* virtual device is not statically bound to any physical network device, so that the MMS has the flexibility to dynamically select network device to send packets coming from *mm0*. When *mm0* is created and assigned an IPv4 address, the network kernel stack automatically inserts a routing table entry to redirect all packets destined to that network address space, 192.168.0.0/16 in the above example, to the *mm0* virtual device.

When a packet leaves the user space application and enters the kernel IP stack, it traverses the so-called netfilter [11] hooks that allow functions registered with a hook to capture, verdict and manipulate the packet. The MMS kernel module registers the `NF_IP_LOCAL_OUT` netfilter hook, so that whenever an IP packet is sent by a local application the packet is examined by an MMS function to check if it is a management packet. If the MMS function finds that the packet is pushed into the kernel from the *mm0* virtual device, it reports it as a management packet and accepts the packet into the management communication channel by passing it to another MMS function that processes management packets. Otherwise, the packet is passed on untouched and it is forwarded by the Linux kernel IP stack as normal.

From management applications' perspective, existing software that uses IP for network communication can run on the MMS unchanged. To leverage the MMS services, all they need is to use the management address when sending and receiving messages.

Once a packet is admitted by the MMS, it is routed through the MMS routing subsystem. Next we show the implementation of our MMS routing subsystem.

MMS Routing Subsystem: The *logically out-of-band communication channel* requirement forces MMS to have its own routing subsystem. In our prototyped MMS routing subsystem, we adopt onion source routing [12] as the base routing scheme, because source routing is a simple way to provide reachability between a few MAs and many CMs, and onion routing has nice property that fits well with the MMS security requirement, as we will discuss in Section 5.3.3. We note that Tesseract [13] also uses onion source routing to distribute routing tables computed from centralized decision servers to routers. However, the Tesseract dissemination plane is internally used merely for routing table distribution, whereas the MMS employs the mechanism to provide a variety of management services. Implementation wise, the Tesseract onion source routing is part of a user space application and it uses OpenSSL [6] library to encrypt and decrypt source routes, whereas the MMS routing subsystem is a Linux kernel module that leverages Linux kernel cryptographic APIs [5] for encryption and decryption.

The base protocol used in the MMS routing subsystem is flooding plus source routing: the MA initiates exploring paths to CMs by flooding a beacon that carries the MA's management IP address. When an CM receives the flooded beacon, it

1. caches the source route;
2. uses the source route to ack the beacon and send the MA the CM's management IP address;
3. appends its management IP address to the received beacon;
4. floods the beacon out its other interfaces.

Using this simple protocol, MAs learn routes to all CMs and all CMs learn routes to MAs. For an MA to send a packet to an CM, it puts the cached MA-CM route in the packet header, and the packet is source routed to the target CM, and vice versa. If an CM needs to talk to another CM, the MMS first redirects the packet to an MA, and the MA relays the packet to the target CM.

The protocol actually used in our prototype also involves onion encryption and decryption, which is discussed in Section 5.3.3.

Delivering IP Packets to Management Applications: When a packet is received, it is examined by an MMS function registered with the `NF_IP_PRE_ROUTING` netfilter hook to decide whether it is an MMS packet that has reached its destination. If that's the case, the MMS strips off the MMS header, restores the original packet, and hands it off to the IP kernel stack for local delivery.

5.3.2 Minimal-Configuration Bootstrap

The MMS is designed to bootstrap with as little configuration as possible to minimize the chance of errors due to configuration mistakes. In contrast, today's IP routers are usually bootstrapped with a large amount of configuration information that includes IP addresses, routing protocols, routing policies, and packet filters. These configurations and the dynamic forwarding state they generate require a myriad of ad hoc scripts and systems in the management plane to validate, monitor, and update. The result is a complex, fragile, and failure-prone network [4]. For example, a mistake in configuring packet filters or routing policy (either during installation or run-time maintenance)

may disrupt the data delivery service of a router and disable the communication between the management system and the router.³

For the MMS, the entire configuration state on an CM consists of a management IP address pre-assigned by an MA, and security certificates. This section focuses on management IP address configuration, and leaves the discussion of security related configurations to Section 5.3.3.

On power-up we create the virtual network device *mm0*, as discussed in Section 5.3.1, and `ifconfig mm0` with the node's management address.

After obtaining the management address, the MMS kernel module walks through a list of `net_device` pointers starting from `dev_base` to get the information and status of all the network devices attached to the CM or MA. Then it uses Linux `timer_list` data structure to create a hello timer and periodically sends out hello messages to exchange management IP addresses with neighbors, so it knows which network device to use to reach which neighbor. An MA also creates a beacon timer to periodically flood beacons to explore paths. Upon receiving a beacon or a beacon ack, a node updates its source route cache as described in Section 5.3.1.

After being authenticated by an MA (see Section 5.3.3 for authentication steps), an CM becomes part of the MMS network and starts to enjoy all reachability services provided by the MMS. That is to say, just like the BIOS for a computer, the MMS remotely turns on management communication channels at the very time an CM is turned on.

So long as the above minimal MMS configuration is persisted and protected from corruption, the MMS will be robust to arbitrary system crashes and reboots — this can be achieved using non-volatile storage within each node.

As the security of a computer program depends on the security of the whole OS stack and its underlay – the BIOS, network security has to be built bottom up starting from the system that brings up the network. Next, we discuss how the MMS boots up the network into a secure state and gives network management applications a clean foundation to run on top.

5.3.3 Security

An MMS empowered network might involve many geographically distributed components, making it critical that the infrastructure survive and recover even if attackers are able to compromise individual pieces of the infrastructure. Further, it must be possible to carry out the initial response to a compromise from a remote location, as some parts of the infrastructure may not have technical staff available 24x7. We designed the MMS with the following security objectives in mind.

- a. The network elements and MA should be able to bootstrap securely as long as their bootstrap configuration has not been tampered with;
- b. Individual network elements can be compromised but they should not be able to disrupt MMS routing paths that don't pass through them;
- c. Compromised network elements cannot use MMS forwarding channels to collude with each other and bring down the MMS forwarding infrastructure;
- d. Network elements should not be privy to information in the MMS packets that reveals the topology of the network or be able to identify and target critical resources in the network;

³A commonly reported mishap among network operators is installing a packet filter that accidentally covers the IP address from which they are logged into the router, thereby immediately severing their management connection and forcing a fall-back to some other way to connect to the router.

- e. Transfer of management commands between MA and the CMs should be secure;

To send data to a MA, a CM uses the encrypted source route it recorded from a beacon sent by that MA. When an upstream CM receives the message, it decrypts the source-route using its secret key. This reveals the management IP address of the next hop CM along the path to the MA. By successive decryption of the source route by the CMs, management packets are delivered to the MA. Since the MA knows the secret-key of every CM, it can construct an onion-encrypted route to any CM it desires.

For the MMS, we borrow this concept of onion-encrypted source routes to *securely forward* MMS packets across CMs and the MA. The encryption of the source routes prevents any intermediate CMs from inferring topology information from MMS packet headers. Also any attempt at tampering the source-route in an MMS packet prevents the next hop CM from correctly forwarding the packet to the destination. Hence compromised CMs cannot communicate with arbitrary CMs unless they are explicitly allowed to do so by the MA.

Securing the MMS: The routing subsystem is the heart of the MMS and it needs to be bootstrapped securely in order to prevent MMS from being compromised. To support this the security model of MMS incorporates:

- a. Secure authentication of network elements under hostile network conditions;
- b. Secure bootstrap of MMS routing subsystem by installing signed source routes on CMs;

When the MMS is first installed on an CM, a USB key storing a network certificate and an MA certificate is plugged into the device. The MMS generates a private/public key pair. The network and MA certificates, and the key pair are stored into a nonvolatile memory. The MMS then encrypts the information with the public key of the MA, and writes it back onto the USB key. When the USB key is eventually inserted into a MA, the MA will have a secret channel to each device. To ensure that only CMs with valid security keys are allowed to communicate using the MMS, every CM's MMS must authenticate itself with the MAs before its messages can be forwarded by the MMS. The secure authentication protocol used by MMS is illustrated in Figure 3. The MMS facilitates authentication by establishing an *authenticated path* between the MMS modules of MAs and CMs over which additional authentication information can be exchanged. The MAs initiate the authentication process by sending a challenge to those directly connected CMs. If a CM answers the challenge correctly by demonstrating it has a valid public/private key pair, the CM is authenticated and the MA creates a pair-wise secret key for that CM to use for subsequent communication.

Once a secure path to the MA is established, the authenticated CM uses the path to send an LSA (Link State Advertisement) to the MA to inform it of its neighboring CMs. The MA builds a topology graph by aggregating the adjacency data from the LSAs. When the MA learns of a new CM, it computes a signed source to the new CM using the network topology and uses it to send a challenge to the CM. Again, the new CM responds to the challenge via its authenticated neighboring CM. Once the CM is authenticated, the MA again creates a pair-wise secret key as well as a signed source route for the newly authenticated CM.

Using this protocol, CMs recursively authenticate themselves with the MAs. In the process, every valid CM receives a signed source route to use to communicate with the MAs. These source routes together form a spanning tree rooted at each MA, and they provides the default secure

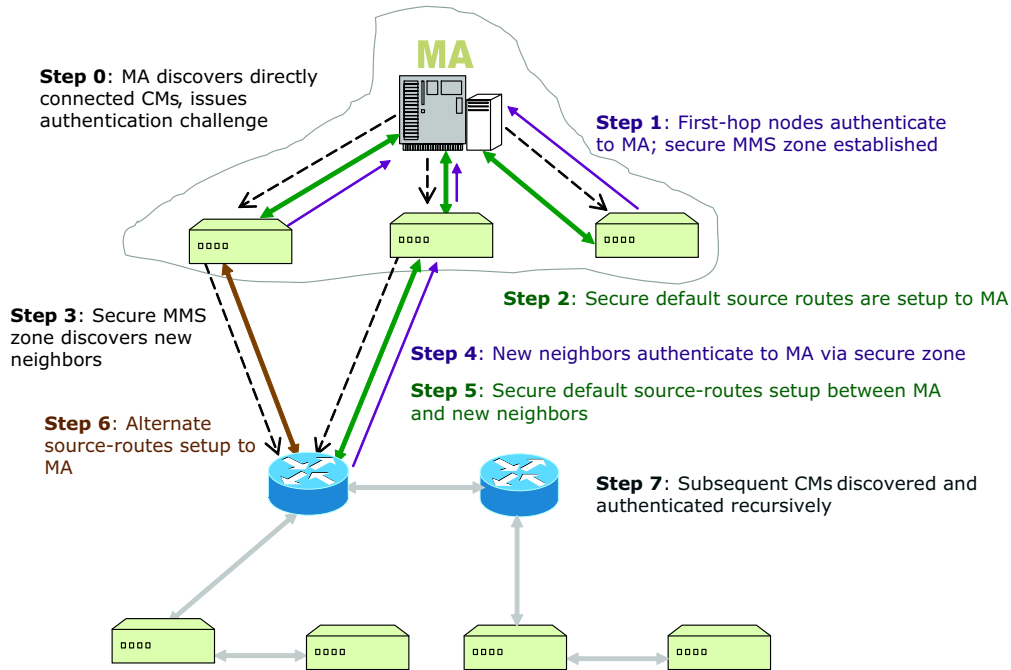


Figure 3: Recursive MMS Authentication.

communication channel between the MA and all CMs. A compromised CM that lies adjacent to an authenticated CM cannot authenticate itself to the MA, and as a result all neighbors of this CM cannot authenticate themselves to MA via the compromised CM. This ensures that the authentication process remains secure in the presence of compromised CMs.

At the end of the protocol, all valid CMs would have securely authenticated themselves to the MA and at the same time installed signed source-routes that securely bootstrap the MMS's routing subsystem. The pair-wise secret key on each CM enables the MA to encrypt management commands and securely tunnel them to the CMs using the MMS.

5.3.4 Name Resolution and Internet Proxying

Besides the basic IP bearer service, the MMS also provide name resolution and Internet proxying services.

DNS Support: MMS allows CMs and the MA to have their own domain names. All CMs and the MA are assigned a a DNS name for their mm0 virtual network device along with an management IP address. CMs and MA can use this domain name to talk to each other. The below example illustrates how MMS enables the DNS service for both intra-MMS and also Internet hosts.

1. The MA runs a DNS name service (bind version 9.3.4 [7]) when it boots up. The MMS sets up the zone files for the name service and initially configures its cache with just one node name-IP address mapping, that of the MA (ma.geni.org ↔ 10.0.0.100).

2. As part of the MMS routing-subsystem, the MMS running on the MA sends out periodic beacons. The MMS now includes the IP address of the MA in these beacons so that CMs R1, R2 and R3 know how to reach the DNS service.

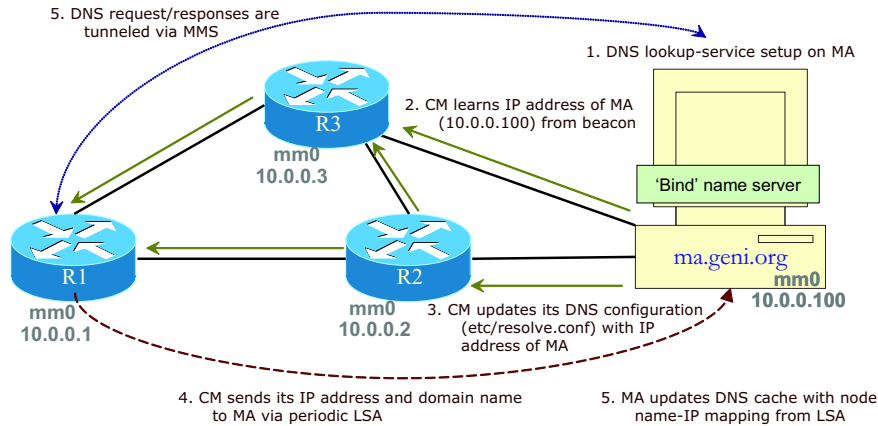


Figure 4: Domain Name Resolution using MMS

3. When the CMs R1, R2, R3 receive the beacon they update their configuration files with the IP address of the MA.

4. As part of the MMS routing-subsystem, the MMS running on the CMs sends out Link State Updates (LSAs) to the MMS of MA. We modify these LSAs so that they also carry the domain name and IP address of that particular CM. For instance when R1 comes up, the LSA it sends out also contains its name and IP address (R1.geni.org \leftrightarrow 10.0.0.1). Note: This avoids the need to preconfigure the nodeIP-DNS name mappings on the MA.

5. When the MMS of MA processes each LSA, it updates the cache of name service with the node name - IP address mapping for the node that sent the LSA. This enables the MA to have a consistent and frequently updated cache of DNS entries.

6. The applications running on CMs can now talk to each other using their MMS domain names seamlessly. When the application on R1 wants to talk to R2, it uses the DNS name of R2 (R2.geni.org). The OS running on R2 generates a DNS request for resolving the IP address of R2.geni.org and sends it to name server (10.0.0.100). Since the IP address of the name server lies in the 'management IP address' space, all DNS requests sent to that address are tunneled via the MMS interface (mm0) to the MA. The name service on the MA looks up for the IP address of R2 in cache populated by MMS, and generates a reply for R1 as 'R2.geni.org is 10.0.0.2'. The reply is tunneled back to R1 via MMS and the OS on R1 learns the IP address of R2. All packets sent to R2.geni.org are hence forwarded to 10.0.0.2.

7. The name service on MA is configured to forward all DNS requests to an external DNS server (dns.geni.org) if it fails to resolve the name-address mappings by itself. This enables R1, R2 and R3 to also talk to hosts outside MMS domain (for example, Internet) without any additional configuration.

Internet Proxying: An CM can use an MA as a gateway to reach Internet. After an CM is booted, a network operator can configure the CM from an MA through the MMS no matter if the CM has access to Internet, thanks to the MA-CM reachability provided the MMS. If a network operator decides to use an MA as a proxy for the CM to reach some or all Internet destinations, she or he can configure the CM's routing table to redirect packets with corresponding destination addresses to the *mm0* virtual device. Since those packets now enter the network stack using the

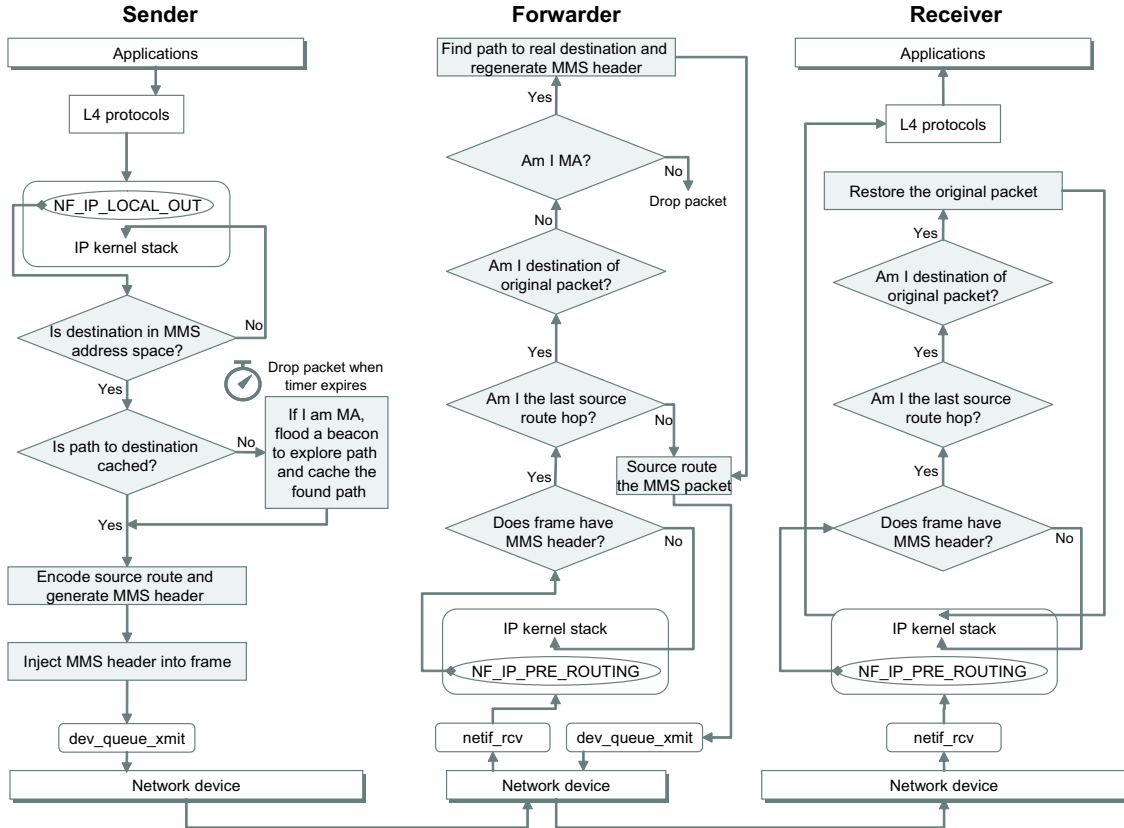


Figure 5: Life of a Management Packet through the MMS.

`mm0` virtual device, the MMS takes over the job of forwarding those packets and delivers them to an MA, where the MMS uses the NAT service provided by Linux iptables [11] to proxy the packets.

5.4 Life of a Management Packet

Figure 5 shows the life of a management packet traversing the network through the MMS infrastructure. Also illustrated in Figure 5 is the interaction among major MMS functional blocks and their relationship with the kernel network stacks and applications. All MMS components are shown as shadowed boxes in the diagram.

Below are the steps of sending out a management packet:

1. The management application opens a TCP or UDP or raw socket, sets the destination address to be a *management IP address*, and sends the packet using one of the socket `send` APIs.
2. The layer four protocols process the packet and calls `ip_queue_xmit` to pass the packet on to the IP layer.
3. Using the `NF_IP_PRE_ROUTING` netfilter hook, an MMS function intercepts the packet and checks the packet to determine if the packet should be transported by the MMS.

4. If the packet comes from *mm0* it is accepted into the MMS infrastructure.
5. The MMS routing subsystem maintains a table that contains all currently valid paths to other nodes; if such a path is not found for the given packet's destination, the MMS starts a timer and tries to fetch a valid path to reach the destination before the timer expires.
6. Once the path is located, the MMS constructs a header including meta information about the packet and an encrypted source route, and inserts the created header to the packet.
7. The encapsulated packet now has an MMS header and is thus called an **MMS packet**.
8. Based on the path information, the sender sends the MMS packet to one of its neighbors.

After a packet is encapsulated and pushed into the network by the MMS, it is forwarded by MMS nodes between the sender and receiver. The forwarding steps are as following:

9. The layer three network stack receives the encapsulated MMS packet from a network device queue.
10. Using another netfilter hook, `NF_IP_PRE_ROUTING`, the MMS parses the packet header to know if it is getting an MMS packet.
11. The source route is decrypted to find the next hop towards the destination, and then the `current_hop` pointer in the source route is modified and the packet is re-encapsulated and forwarded.
12. Note that an CM can use an MA to proxy to other CMs or the Internet, so for an MA, it needs to take care of that situation even when the `current_hop` pointer points to itself when the MMS packet arrives.

Eventually, the packet arrives at the real receiver node:

13. The MMS function registered with the pre-routing netfilter hook is invoked to look at the packet header and decide whether it is an MMS packet.
14. If it is determined that the packet has reached its final destination, the MMS strips the MMS header, restores the packet and lets it continue its journey up to layer four and further to the application.

5.5 Case Study: Using MMS to Control VINI

We conduct Emulab [14] experiments to demonstrate controlling VINI [15] with MMS. VINI is a virtual network infrastructure that enables multiple virtual nodes to co-locate on a single physical CM and run experimental network protocols. VINI sets up virtual topologies between the virtual nodes using UDP tunnels, and uses Click [16] to forward packets between them. The flexibility to inject exogenous events (like link failures) into the system and the ability to operate under real traffic conditions make VINI an interesting case study for the MMS. In order for VINI and other virtual network infrastructures to function, they require support from the underlying infrastructure for (a) bootstrapping, (b) creating a virtual environment for the experiment to run, and (c) monitoring the experiment.

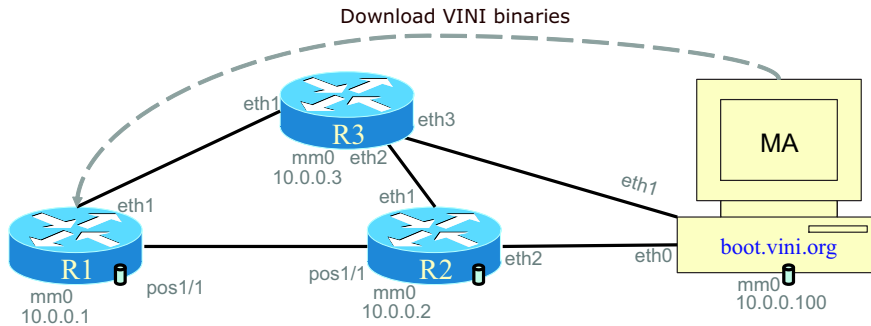


Figure 6: Bootstrapping of VINI using MMS

VINI consists of a number of networking tools developed by researchers for enabling virtual network infrastructures. Among them, it includes the Click router for data forwarding, the XORP package for routing protocols, and User Mode Linux (UML) for setting up the virtual environment on the CMs.

Bootstrapping: In order to bootstrap VINI, the binaries of the above tools should be uploaded to all CMs in the experiment. To do this, CMs must first connect to the experiment control center(s). MMS sets up this connectivity to the control center when the CMs first boot and maintains it throughout the duration of an experiment. Both internal network links and Internet tunnels (where available) are leveraged to make the forwarding path to the control center robust to network failures. Once nodes are reachable from the control center, the experimenter can make use of the IP bearer service exported by MMS to download the software images and initialization files into the CMs involved in the experiment. Since the IP bearer service is abstracted by a virtual network device that has a management IP address, the experimenter can use existing file transfer applications like scp/ftp to download the software to the nodes. Additionally, a quorum security framework can be used by the CMs to verify the validity of the downloaded software before it is installed.

The example below demonstrates the bootstrapping scenario for VINI over MMS:

1. R1 boots up and tries to register itself with the MA
2. R1 starts off by making a DNS query for the boot server boot.vini.org which is sent out through mm0 interface
3. mm0 tunnels the request to MA via the MMS routing subsystem. MMS routing subsystem uses beacons from MA to build secure source routes to MA
4. MA receives DNS request from its local mm0 interface, sends a reply back through mm0
5. R1 receives the name resolution of boot.vini.org as 10.0.0.100
6. R1 sends registration request to MA (10.0.0.100) via mm0
7. MA registers R1, and downloads the VINI software binaries and configuration files onto R1

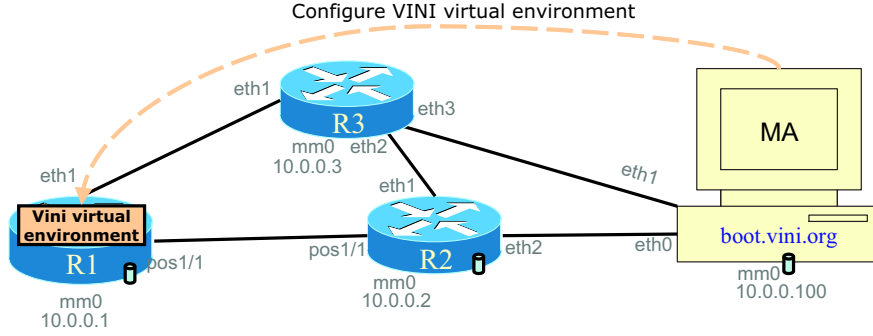


Figure 7: Setting up VINI virtual environment using MMS

Setting up virtual environment: The second step in running a VINI experiment is to bring up the virtual UML environment on the VINI nodes. This requires loading of VINI configuration information into the nodes from the control center. Once the virtual environment is brought up, VINI sets up UDP tunnels between CMs as per the user-specified configuration files to simulate the desired network topology. The overlay routing software is also brought up to enable forwarding over the virtual topology. Since these routing protocols run inside the VINI virtual environment they cannot be used by the experimenter to bring-up the UML environment. Hence the experimenter needs an independent routing infrastructure that can be used to reach the CMs and bring up UML and the overlay routing processes. MMS serves as this independent underlying routing infrastructure. With the help of the MMS-routing subsystem and the IP bearer service it exports, the experimenter can remotely login into the CMs and initialize UML. Once the virtual environment is setup successfully, the CMs are ready to run any custom experiment over the virtual network. Again by leveraging the IP bearer service of MMS, the experimenter can download the software images or configuration files for his custom experiment into the virtual environment on the CMs and bring up the experiment remotely.

The example below demonstrates the setup of VINI virtual environment using MMS:

1. R1 registers itself with the MA successfully, downloads the VINI software binaries
2. MA remote-logins into R1 using ssh via mm0. ssh packets are routed via the MMS routing subsystem between R1 and MA
3. MA configures the UML environment on R1; it also sets up the configurable parameters of XORP and Click as per user specified topology
4. MA executes commands on R1 to bring up the UML environment and hence the virtual topology

Monitoring the experiment: Finally, once the VINI experiment is successfully brought up, the control center may need to continuously monitor the health of the virtual CMs participating in the experiment. The management applications that control the experiment can use the IP bearer service to communicate the status of the experiment to the control center. This also facilitates the injection of link/node failure commands from the control center into the CMs running the VINI experiment.

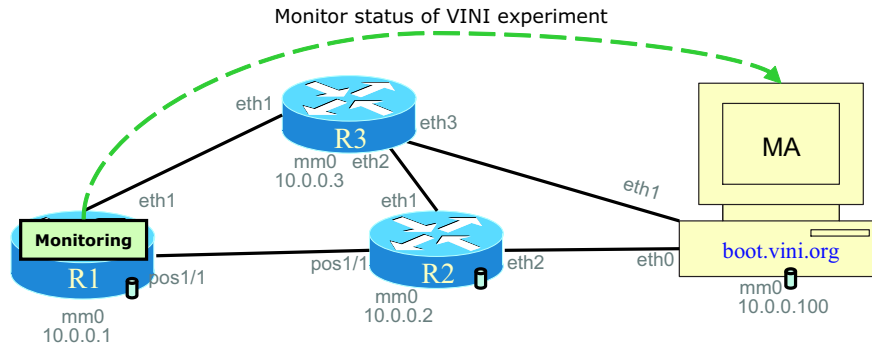


Figure 8: Monitoring of VINI experiment via MMS

The example below demonstrates the monitoring of VINI experiment using MMS:

1. R1 registers itself with MA; virtual environment is brought up on R1
2. MA remote-logs into the virtual environment of R1 using the management IP address (10.0.1.1) of the R1's UML management interface. The packets are tunneled back and forth between R1 and the MA via the mm0 interface by the MMS routing subsystem
3. MA monitors the XORP routing tables on the virtual environment of R1 by remotely executing XORP status commands
4. MA remote-logs into the node R1 using its management IP address (10.0.0.1) and monitors the Click forwarding table

References

- [1] T. Anderson and M. Reiter, *GENI Facility Security*. Distributed Services Working Group, September 2006. GENI Design Document 06-23.
- [2] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman, "PlanetLab Architecture: An Overview," Tech. Rep. PDN-06-031, PlanetLab Consortium, May 2006.
- [3] L. Peterson and J. Wroclawski, *Overview of the GENI Architecture*. Facility Architecture Working Group, September 2006. GENI Design Document 06-11.
- [4] 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.
- [5] J. Morris, "The Linux Kernel Cryptographic API." <http://www.linuxjournal.com/article/6451>.
- [6] M. Cox, R. Engelschall, S. Henson, and B. Laurie, "The OpenSSL Project." <http://www.openssl.org/>.
- [7] "ISC BIND." <http://www.isc.org/sw/bind/>.
- [8] D. R. Kuhn, "Sources of failure in the public switched telephone network," *Computer*, vol. 30, no. 4, pp. 31–36, 1997.
- [9] V. J. Bono, "7007 Explanation and Apology." <http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html>.
- [10] C. Benvenuti, *Understanding Linux Network Internals*. O'Reilly, 2005.
- [11] "netfilter/iptables." <http://www.netfilter.org/projects/iptables>.

- [12] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Information Hiding*, pp. 137–150, 1996.
- [13] 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*, 2007. To appear.
- [14] 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.
- [15] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," in *Proc. ACM SIGCOMM*, September 2006.
- [16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Computer Systems*, August 2000.