Complexity Oblivious Network Management

A step towards network manageability

Hitesh Ballani, Paul Francis
Cornell University
{*hitesh, francis*}*@cs.cornell.edu*

***Abstract*— Networks are hard to manage and in spite of all the so called holistic management packages, things are getting worse. We argue that this is an outcome of two fundamental flaws in the existing architecture: the management plane depends on the data plane and the complexity of the ever-evolving data plane encumbers the management plane. Consequently, addressing these flaws can make the network amenable to management. In this paper, we present Complexity Oblivious Network Management (CONMan), a network architecture in which the management plane does not depend on the data plane and all data plane protocols expose a generic management interface. This restricts the operational complexity of protocols to their implementation and allows the management plane to achieve high level policies in a structured fashion. Our preliminary experience with building the CONMan interface of a couple of protocols and using them for real world management tasks indicates the architecture's potential to alleviate the management troubles of the Internet.**

I. Introduction

IP networks are hard to manage. Network management (installation, configuration, provisioning, monitoring, testing, debugging) requires detailed knowledge of many different network components, each with its own management interface. To cope, network managers rely on a host of tools ranging from sophisticated centralized network management packages to home-brewed scripts and elementary tools such as ping and traceroute. For instance, our organization uses half a dozen different tools, commercial and public domain, and has over 100K lines of scripts for managing the switch and router infrastructure alone (not including email, servers, DNS, DHCP, billing, etc.). In spite of their ever increasing sophistication, management tools seem to be waging a losing battle which is shown by rising management costs and network downtime. A recent survey [19] showed that 80% of the IT budget in enterprises is devoted to maintain just the status quo - in spite of this, configuration errors account for 62% of network downtime.

We believe that the management troubles of the Internet are a consequence of two shortcomings of the existing network architecture. First, the existing *management plane depends on the data plane*. For example, SNMP operates on top of the data plane and hence, management protocols rely on the correct operation of the very thing they are supposed to manage. This dependency loop is a fundamental flaw that leads to:

- *Manual pre-configuration*. The network (at least the lower levels) needs to be manually brought up before the management applications can talk to the devices in the network. Given the complexity of some of these low level protocols, routing especially, significant manual and therefore, error-prone configuration must be done before they can operate correctly [5], [16]. Also, the disconnect arising out of the fact that the management application is not responsible for a complete, bottom-up configuration of the network is a common cause for errors and misconfigurations [16].
- *Poor failure-mode operation*. The network manager may not be able to communicate with network equipment when a failure occurs, which is when the manager is most needed [9].

Second, the fact that protocols expose their internal details leads to a *deluge of complexity* which burdens the management plane. For example, it is not uncommon for a network device to have thousands of manageable objects. MIBDepot [47] lists 6200 MIBs (Management Information Base) from 142 vendors for a total of nearly a million MIB objects. A single router configuration file can consist of more than 10,000 command lines [41]. Encumbering the management plane with all this complexity leads to:

- *Perception differs from reality*. Management applications need to effectively reverse engineer the capabilities and the functionality of protocols and devices from their detailed MIBs containing lots of internal parameters. The low-level and non-intuitive nature of these parameters makes this task difficult, if not impossible [30].
- *Error-prone configuration*. Network configuration involves mapping high-level policies and goals to the values of protocol parameters. Since management applications don't have an understanding of the underlying network in the first place, they often resort to a cycle of setting the parameters and correlating events to see if the high level goal was achieved or not. Apart from being haphazard, the noise in measurements and correlations is often the root-cause of misconfigurations and related errors. The inability to understand the network's operation also makes debugging these errors very difficult [22].
- *Fragmentation of tools*. Since devices and their internal details keep evolving at a frantic pace and these details are exposed to management applications, such applications tend to lag behind the power curve [27]. Additionally,

the inability of standard management interfaces (IETF MIBs) to keep pace with data plane development has led to a plethora of vendor specific MIBs and even vendor specific management applications and has put us in a situation where *no one management approach* suffices. For example, SNMPLink [32] lists more than 1000 management applications, many of them being vendor specific command line or HTML-based tools. Hence, the Internet management plane doesn't have anything analogous to the IP thin waist around which the Internet data-plane is built.

- *Lack of dependency maintenance*. Management state throughout the network tends to be riddled with protocol parameters and low-level values. These need to be tracked so that management state can be updated when the values change but the current management plane does not do so [30]. Instances of improper filtering because the address assigned to some machine changed, or the application was started on some other port are very common. [21] details many examples of how failure to track such dependencies leads to problems in large networks.

All these shortcomings point to the two principles that, we believe, should form the basis of a manageable network architecture:

(a). *Operationally independent, self bootstrapping* management plane. The management plane should be operationally independent of the data plane and should be able to bootstrap without any pre-configuration (this was proposed as part of the 4D project [16] and is extended in this paper). This would allow for a management plane that can configure a network from the ground up. Also, the ability to manage the devices as long as they have physical connectivity has implications for all facets of network management, especially the ability to operate in the face of failures.

(b). A *single, simple management interface* for all data plane protocols. The operational complexity of protocols should be confined to their implementation and they should express the information needed for managing them through a simple management interface. This puts the responsibility for detailed understanding of protocol operation on the protocol implementor while reducing the burden on management applications. Since the protocol implementer requires this knowledge in any event, this seems to be a smarter placement of functionality.

In this paper we present the design and implementation of a network architecture, *Complexity Oblivious Network Management* (CONMan), that follows these principles. CONMan includes an extension of the 4D configuration-free management channel. This channel is established and maintained separately from the data plane and hence, serves as a substrate for the rest of the management framework. All protocols and devices express their capability and their functionality using a generic abstraction. This allows the management plane to understand the potential of the underlying network, to configure it in line with the desired high-level policies and to fix it when something breaks, without being encumbered by the details of the protocol/device implementation. Having a fixed interface between the management plane and the data plane also allows for independent evolution of the two. Hence, this paper makes the following *contributions*:

- We propose a set of architectural principles that a network conducive to management should be based on.
- We present the detailed design of a network architecture following these principles.
- The paper also describes the implementation of the management interface of two protocols in compliance with the proposed architecture.

CONMan does not change the operation of data plane protocols nor does it dictate the way they are implemented – only the management interface of each protocol should conform to our proposal. Consequently, while the paper talks about individual protocol modules and the components of their management interface, the underlying implementation may not be modular. For example, a monolithic network stack that exposes the appropriate abstraction for all its protocols fits just as well into our proposal. Hence, a *non contribution* of this paper is the notion of implementation modularity.

While we admit that our experience with modeling protocols according to CONMan is rather limited, the importance of ensuring network manageability is not disputable. In this context, we hope that our proposal would stimulate a discussion about structured management of networks and hence, serve as a step towards the holy grail of *self managing networks*.

II. CONMan Architecture

We have designed a network architecture, CONMan, based on the principles described in section I. Here we describe the architecture in detail.

A. Terminology and Overview

Our architecture consists of *devices* (routers, switches, hosts, etc.) and one or more *network managers* (NMs). A NM is a software entity that resides on one of the network devices and manages some or all of them. Each device has a identifier (*device-id*) and an internal *management agent* (MA) that is responsible for the device's participation in the management plane. The device-id is globally unique, topology independent and can even carry cryptographic meaning (for example, by hashing a public key). All protocols and applications in devices are modeled as *protocol modules*. While the rest of the paper talks about a device performing management tasks, in actuality it is the device's MA that is responsible for these.

CONMan achieves the first principle by borrowing techniques used by 4D for its discovery and dissemination plane [16]. We briefly describe the set-up of our management plane here and refer the reader to [16] for details. All devices and NMs can send management frames to their directly connected neighbors over the (almost) raw physical link. This allows devices to determine their physical connectivity. Beyond this, each NM periodically floods the network with beacons

which gather device-ids of the devices they propagate through and hence, provide each device with a path to the NM in question. This allows the devices to send source-routed frames to the NM. In turn, the NM can now get back to these devices and thus, this forms the NM's *management channel*. Note that the management channel does not require any pre-configuration and is completely independent of the data plane paths in the network.

As mentioned earlier, a given network can have more than one NM. Also, the basic notion of a management channel as presented above requires various extensions, for instance to allow communication between NMs belonging to separate networks. We defer discussion of these extensions to section II-F. For ease of exposition, the following discussion focusses on a network under the control of a single administrative entity with just one NM.

In order to satisfy the second principle, protocol modules self describe themselves using a generic abstraction - this is the *Module Abstraction*. The driving idea behind our abstraction is to identify the basic characteristics that virtually all protocols share. Consequently, we model every protocol module as a node with connections to other nodes, certain generic switching capabilities, certain generic filtering capabilities, certain performance and security characteristics, and certain dependencies. This abstraction captures what the protocol is capable of (*capabilities*) and what it depends on (*dependencies*). Also, the module can be configured to achieve the desired *functionality* by creating and deleting various components of the abstraction. Finally, modeling all protocols using a generic abstraction decouples the data and the management plane so that they can evolve independently of each other. Hence, the module abstraction provides the narrow waist around which our management-oriented architecture is designed.

Together, the management channel and the module abstraction allow the NM to manage the network based on high-level policies and goals in a structured fashion. Each device uses the management channel to inform the NM of its physical connectivity, all modules that it contains and their respective module abstractions. This provides the NM with the real picture of the network - it does not need to reverse engineer numerous low-level and non-intuitive parameters. The module abstraction allows the NM to understand exactly how packets may flow (or not flow) through a given module and hence, from application to application.

Given the network's real picture and the high-level goals and policies that need to be satisfied, the NM builds a graph of modules in various devices that satisfy these. This graph captures how each module should function and hence, how each module should be configured. The NM then configures the modules accordingly through the management channel. Thus, the NM can configure the entire network from the ground up with a minimum of protocol-specific knowledge. We believe that such as approach would ameliorate most of the problems afflicting network management today.

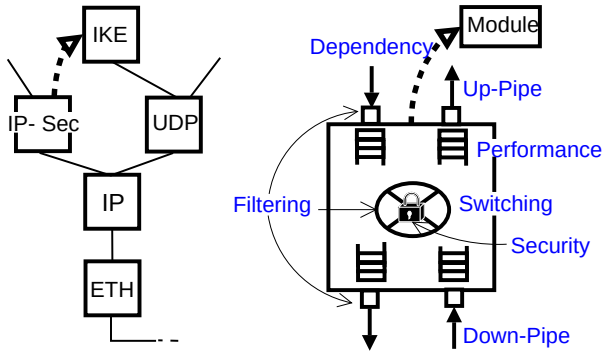

Fig. 1. Modules, pipes, and dependencies form a graph that describes the operation of a device (in particular) and the network (in general). The figure on the right shows the major components of the *module abstraction*. Note that some modules may not require all elements of the abstraction to describe themselves.

B. Module abstraction

There are two kinds of modules: data plane modules and control plane modules. Examples of data plane modules (or data modules for short) include TCP[1], IP, Ethernet, while examples of control plane modules (or control modules for short) include routing algorithms and negotiation algorithms like IPSec's IKE or PPP's LCP and NCPs.

Data modules connect to each other to carry data packets. These connections are called pipes. Control modules also connect to data modules using pipes for delivery services. Data modules may require the use of a control module; we refer to this as a dependency. For instance, in Figure 1, the IPsec module has a (data plane) pipe to IP, and has a dependency on IKE, which in turn has a pipe to UDP. Ultimately, modules, pipes, and dependencies form a graph that in some sense describes the operation of the network.

The actual module abstraction, as shown on the right in figure 1, tries the capture the capabilities, the dependencies and the functionality of protocol modules in generic and abstract terms. Below we comment on pipes and the components of the module abstraction.

1) Pipes: *Up* and *Down* pipes connect modules to other modules above and below themselves in the same device. Such pipes are point-to-point only. Point-to-point pipes are modeled as unidirectional (and usually come in pairs), though for simplicity we present them as bidirectional. The actual network links are modeled as *Physical Across* pipes (or across pipes for short) and can be point-to-point or broadcast. Hence, the *path* between two modules in two different devices is the sequence of up-down and physical across pipes through which packets travel between the modules. Also, pipes have identifiers which the NM can use to refer to them.

Physical across pipes either exist or don't; devices discover them by exchanging management frames with their neighbors over the raw physical links. As a contrast, the NM can actually create up and down pipes to construct paths between modules. Such pipes may have dependencies that need to be satisfied

[1]The paper does not provide citations or acronym definitions for standard protocols.

before they can be created. For example, a pipe may require other pipes to be created or switch state to be specified (see below) before it can be created. Applications can also lead to creation of pipes. For example, a HTTP-client initiating a TCP connection may lead to pipes between the following module pairs: {HTTP-client, TCP}, {TCP, IP}, and {IP, ETH}.

Up and down pipes are also associated with a list of *connectable-modules*. For example, the connectable-modules for the down pipe of a particular TCP module might be restricted to {IPv4, IPv6} implying that the TCP implementation in question can only operate on top of (have a down pipe to) IPv4 or IPv6.

While modules pass packets between up and down pipes, the end goal is to be able to communicate with modules in other devices. To capture this, we introduce the notion of *peer modules*. For example, the peer modules of a given TCP module are the TCP modules that this module has TCP connections to. Also, each module is associated with a set of *peerable-modules*. For example, the peerable-modules for a TCP module is restricted to {TCP} while the peerable-modules for a HTTP-server module may be restricted to {HTTP-client}.

2) Module Name: The *module-name* is a tuple <A,**X**,**y**> comprising of the protocol name A, the device-id X, the module-id y. Examples of protocol names include "IPv4", "RFC791", or even a URI (which might be useful for naming applications). The module-id is a locally unique number assigned to the module by the device MA. Given that the device-id is a globally unique number, the module-name uniquely identifies the module while the module-id allows multiple modules for the same protocol to exist within the same device.

3) Switch: Switches capture the ability of modules to pass packets between up, down and physical across pipes. A switch can be unicast or multicast and can have a small number of basic configurations: packets pass between down and up pipes ([down $\Rightarrow$ up] and [up $\Rightarrow$ down] switching, e.g. TCP module), [down $\Rightarrow$ down] switching (e.g. IP module with forwarding enabled), [up $\Rightarrow$ up] switching (e.g. IP module with loopback functionality), [up $\Rightarrow$ physical] and [physical $\Rightarrow$ up] switching (eg. Ethernet module). A module advertises the potential and actual configuration of its internal switch, thus allowing an NM to know what paths are and are not possible.

Of course, switches may also have switching state which conditions the switching. The module advertises if the switch state is produced locally through the protocol operation or needs to be provided externally. For instance, an up-pipe from TCP to an application is selected by the port number chosen by the application and hence, the [down $\Rightarrow$ up] switch state is created locally through the protocol operation. In other switches, such as those of IP or MPLS modules, the switching state is not generated out of the protocol operation and needs to be provided externally. This can either be calculated by some control module (e.g., BGP) or directly set by the NM (as proposed in 4D). The need for externally generated state is discussed in section II-E.

4) Filters: It is common for protocol modules like Ethernet or IP to filter, often based on deep packet inspection. The module abstraction captures this by allowing modules to advertise their filtering capabilities. The filter specification includes the classification based on which filtering can be done; for example, the module might be able to filter packets from or to specified IP, TCP and HTTP modules or even specific pipes of these modules. The filter might be per pipe, per direction (incoming or outgoing), or there might just be a module-wide filter such that each filter rule is applied to all traffic traversing the module. Note that the NM only needs to specify the module-names that need to be filtered - it is the protocol implementation that is responsible for determining the relevant protocol fields. This process and other related issues are detailed in section II-D.

5) Performance Reporting: Modules may report on the performance of their connectivity to each of their peer modules and their ability to do so is advertised. In our current abstraction, performance is reported in terms of six generic *performance metrics* - delay, jitter, bandwidth, loss-rate, error-rate, and ordering. These encompass most of the IP performance metrics proposed by IETF [38] (though in our architecture the metrics can be used by any module that has the ability to describe its performance, not just the IP module). Apart from this, a module may also report on other performance criteria associated with the abstraction components, for instance filter lookup or switching time, number of packets on various pipes, and so on.

6) Performance Trade-Offs: Most protocols, as part of their operation, can offer performance trade-offs. For example, many MAC layer protocols offer optional error correcting checksums which represent a trade-off between error-rate on one hand and bandwidth and delay on the other. Similarly, the amount of buffering done by the IP module provides a trade-off between loss-rate and delay/jitter. Instead of exposing the low-level options and the associated parameters, modules specify the trade-offs they can enforce. Just as with filters, the module might allow these trade-offs to be applied to specific traffic classes as specified by the names of modules or pipes and this too is advertised. Also, all the trade-offs are in terms of the generic performance metrics mentioned above. The NM, based on some high-level performance goals, can choose from the trade-offs offered by these modules who can then configure and coordinate the low level values.

While we would like to restrict our performance metrics to a small list that is closely related to the typical high-level policy goals, we realize that network evolution would certainly warrant addition to our current list. Similarly, though Internet management has been CONMan's focus, applying the framework to management of other networks such as ad-hoc networks brings to light new metrics and new trade-offs. For example, power is an important performance metric in such networks. And the use of ARQ by a layer-2 module is an example power trade-off - it represents a trade-off between power and delay on one hand and error-rate on the other. Hence, we envision that policies to be enforced by NMs

will also evolve with the network[2]. An NM with old policies (the ones used for wired networks) won't be able to take advantage of the power trade-offs on offer when managing an ad-hoc network. However, a reformulation of the policies to incorporate power would allow the NM to map power related high-level goals to module configurations.

7) Performance Enforcement: Satisfying performance requirements of applications by utilizing the performance enforcement capabilities of devices is an important part of network management. We envision that the network high-level policy would include application priorities specified by the user/administrator and performance requirements specified by applications. For example, a VOIP application module may specify that it requires x Kbps of bandwidth[3]. For legacy applications that do not advertise their requirements, the NM can use performance reports of other modules to determine the same. For example, the NM can determine the bandwidth requirement of a TCP-based application by asking the TCP module for the performance report of the pipe connecting the TCP module to the application module. In some cases, the administrator may have to provide these.

There seem to be two approaches for modeling the performance enforcement capabilities of modules. In the first approach, performance is modeled through queues and shapers. There are a limited number of queue and shaper types and configurations, and hence, modules advertise these. The NM uses application priorities, performance requirements and its own measurements to configure modules that are capable of performance enforcement. Note that even in this approach, the NM does not deal with low-level values such as DiffServ code points (assuming that the IP module's DiffServ capabilities are being used for performance) to identify a traffic class.

The second approach further abstracts the traffic management mechanisms offered by modules by allowing them to advertise the service classes they can provide. These service classes are tuples of performance metrics described earlier. For example, *<very low delay, very low jitter, very low loss, constant rate, very low error, good ordering>* might be one class while *<low delay, best-effort jitter, low loss, variable rate, low error, best-effort ordering>* may be another. This modeling is similar to the service class based DiffServ configuration [3], though we don't restrict ourself to DiffServ based performance enforcement. With this approach, the NM needs to map policy goals to the advertised service classes while it is the protocol implementation that maps service classes to queues and shapers (see [3] for a detailed description of how typical applications may be mapped).

These two approaches represent a trade-off between the amount of detail the NM has to handle and the flexibility in performance enforcement. While we are not sure about the performance model that should be part of the module abstraction, it might be the case that the choice is guided by the kind of network. The first approach might work better for networks with sophisticated performance requirements such as ISPs, while the second approach might be better for smaller networks such as enterprises and home networks.

8) Security: A module may have the means to ensure the integrity, authenticity or confidentiality (or some combination of the three) of its communication with any given peer. Such modules advertise their ability to establish secure communication. The state associated with these security features, for example the keying material, may be determined by the module through interaction with the peer module (example, SSL). In other cases, this state may have to be provided by an external entity and is advertised as a dependency (example, IP-Sec dependency on IKE).

[2]this does not necessitate NM re-implementation and so, is different from forcing the management plane to follow all device and protocol changes.

[3]the application may provide even more information, for example it requires x Kbps of *<very-low delay, very-low jitter, very-low loss>* bandwidth.

Name	Caller	Callee	Description
showPotential	NM	MA of device	Sec. II-C.1
showActual	NM	MA of device	Sec. II-C.1
create, delete	NM	MA of device	Sec. II-C.1
conveyMessage	Module (source)	NM	Sec. II-C.1
conveyMessage	NM	Module (destination)	Sec. II-C.1
test	NM	Module	Sec. II-C.2
listFieldsAndValues	Module (inspecting)	NM	Sec. II-D
listFieldsAndValues	NM	Module (target)	Sec. II-D

TABLE I

FUNCTIONS THAT ARE PART OF THE CONMAN ARCHITECTURE

C. Network Manager (NM)

The NM maintains a management channel that allows two-way communication between itself and the devices in the network. Each device informs the NM of its physical connectivity, thus allowing the NM to determine the network topology. Beyond this, the NM must determine both the existing configuration and the capabilities of the modules in the individual devices on the basis of which it can configure the network and debug network problems.

1) Network Configuration: Given the network *reality*, the NM can configure devices and achieve high level network goals simply by creating and deleting pipes and module components. The following functions capture the NM's interaction with the devices in the network as part of network configuration. Table I shows these and other functions offered by the NM, the MAs of devices, and the modules themselves.

- *showPotential ()* allows the NM to determine a device's potential configuration. The device returns a list of modules with their abstractions. The type of information returned for each module is shown in table II.
- *showActual ()* allows the NM to determine the actual connectivity and state of modules in a device. The state of each module includes state for all the pipes, the switch, filters, trade-offs, performance and security enforcement elements. Also returned is a report on the performance parameters. In effect, the NM is presented with the network reality - a module graph and associated information which allows it to understand how the device (and hence, the network) is or should be behaving. By contrast, in the

Parameter	What is advertised?
Name	<A,**x**,**y**>
Up and Down pipes	Information about up and down pipes such as connectable-modules, dependencies etc.
Physical across pipes	Information about the physical across pipes (if any) connected to the module
Peerable-Mod.	Set of modules that can be peers of this module
Filter	Classification based on which filtering can be done - this includes what can be filtered and where it can be filtered
Switch	Possible switching between up, down and physical pipes; the kind of switch state that governs the switching and if it is generated locally or needs to be provided externally
Performance Reporting	Performance metrics that are reported for the module's pipes, filters, switch etc.
Performance Trade-Offs	Traffic classes to which performance trade-offs can be applied and the possible trade-offs
Performance Enforcement	Apart from the classification based on which performance can be enforced, the module advertises one of these: (1) Queuing and Shaping capabilities (2) Service classes on offer
Security	Ability to secure communication with the peer modules. If the state needed for this is to be provided, it is advertised as a dependency.

TABLE II
MODULE ABSTRACTION; *showPotential* () DESCRIBES EACH MODULE USING THIS ABSTRACTION

current set up, the NM is presented with all kinds of MIB objects from which it must deduce network behavior.

- *create ()* and *delete ()* allow the NM to create and delete pipes, filter-rules, switch-rules, trade-offs to be enforced and performance enforcement state (queuing structures or service classes). Note that the *showPotential ()* function provides the NM with all the information it needs to create and delete components.

The NM needs very little protocol specific knowledge to use these primitives. For instance, it can create up and down pipes simply by satisfying their dependencies and invoking the *create* function. It is the protocol implementation that is responsible for all the protocol-specific exchange and configuration that needs to be done to actually install the state that instantiates the pipe. For example, establishing an up pipe for a GRE module amounts to creating a new GRE tunnel and hence, requires the module to communicate with its peer GRE module about the tunnel key values to be used. The GRE module advertises this need for peer coordination as a dependency. To facilitate the actual co-ordination between peer modules, the NM provides:

- *conveyMessage ()* allows modules to convey messages to each other through the NM (see detailed example in section III-B).

2) Debugging: In CONMan, modules provide a *test ()* function with which the NM can test connectivity of the module to any of its peer modules. Invoking the *test* function causes the module to check protocol specific parameters with the desired peer module through the management channel (using *conveyMessage*). Testing might also require modules to send packets to their peer over the data plane. Some existing protocol implementations, for example Cisco's GRE implementation [45], already have something akin to such a test function. Also, Microsoft is currently working on adding a similar functionality to help users debug the Windows network stack [35]. Here, a process is able ask protocol modules if they believe themselves to be healthy.

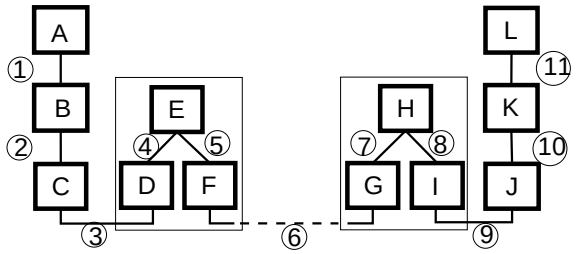

Fig. 2. Modules A and L are unable to communicate: the *test* function offered by modules allows the NM to perform a stackwide parameter check. This, coupled with the NM's ability to trace the path between the devices, makes root-cause analysis of the problem possible.

Given a problem in communication between two application modules, the NM can debug it by tracing and testing the sequence of modules and pipes between them. For instance, lets assume modules A and L in figure 2 are having problems communicating. To ensure that A and L agree on their protocol specific parameters, the NM invokes the *test* function at these modules. Beyond this, debugging of the problem devolves to the debugging of the modules and pipes along the path between A and L. The actual state of the modules (as given by *showActual*) and the *test* function offered by them allows the NM to trace and test modules B through K. Hence, the NM checks to see if module B forwards packets from pipe (1) to (2) and then uses its *test* function to test B's connectivity to module K (B's peer) and so on for all the modules involved. We think that through such a structured debugging approach, the NM can determine the root-cause of most network problems.

D. Component identifiers

The NM uses identifiers for the abstraction components such as module names, pipes identifiers etc. to specify traffic classes. Modules can then be directed to operate on these classes - filter them, switch them or even ensure some performance characteristics. For example, the NM can simply ask a module to filter packets between two given modules - "check if the packet is from module <IP,**B**,**y**> and going to <FOO,**C**,**z**>" (where FOO is an application module with up-down pipes to TCP). This ensures that the NM, while being opaque to protocol-specific fields, can trace the paths between applications and hence, can reason about its policies regarding a particular application-module.

It is the protocol module doing the filtering that is responsible for determining the actual protocol fields. For example, given the high-level specification above, the module determines that it needs to "filter packets from source address 128.19.2.3 and destined to address 20.3.4.5, port 592". In some cases, the inspecting module may know what fields and field values to check for on its own. But in other cases, it may not.

In CONMan, modules provide a *listFieldsAndValues ()* function. This allows other modules to query the target module for the low-level fields and field values corresponding to the identifiers associated with its components. Hence, in the

example above, the inspecting module can send queries to the target modules <IP,**B**,**y**> and <FOO,**C**,**z**> (via the NM), as well as to the modules below them, and ask those modules what field values it should be checking for.

Such an approach also allows for maintenance of network state dependencies - the need to update the relevant state in different modules when some low-level value in a given module changes. To ensure this, the NM maintains the dependencies between component identifiers (that have been resolved) and low-level fields. Also, the NM installs triggers in the target modules telling them to inform the NM when their low-level values change.

However, the use of component identifiers also raises a number of issues. First, frequent invocations of *listFieldsAndValues*() and maintaining all the dependencies between component identifiers and the low-level values can generate a lot of management traffic and create a bottleneck at the NM. While good engineering design involving caching of information and having stackable NMs may partly address this, it could be the case that the traffic and processing overhead represents an acceptable trade-off when seen in the light of the increase in network manageability.

Second, there are cases that do not seem to be easily handled by a NM using component identifiers. For instance, some filtering instances like matching on regular expressions in HTML are difficult to capture using component identifiers and must simply be explicitly set by some sort of a specialized NM such as an Intrusion Detection System. Similarly, making best-match hierarchical IP address inspection opaque is difficult [13]. Given the ubiquity of IPv4, the scarcity of addresses and the restrictions on assigning them, it makes a lot of engineering sense for the NM to have protocol-specific knowledge in this case and assign addresses by itself.

Third, the presence of translation devices such as NATs, NAPTs etc. in the network can invalidate the low-level values returned by a target module at the inspecting module. In our architecture, such translation devices advertise their translation functionality to the NM. Hence, the NM must trace the path(s)[4] from the target module to the inspecting module to see if any module translates the fields in question and if yes, ask the module for the translated value. The final translated value is then returned to the querying module.

Finally, the above example also doesn't work if some of the target modules are in a different domain, and therefore not accessible to the NM. The NM can realize this and in some cases, direct the inspection to be based on incoming interface. For example, to inspect packets from a customer network, the NM can ask for inspection based on the customer physical pipe instead of specifying the customer prefix. In other cases, the NM may need to explicitly specify IP addresses or other fields to identify modules. Communication between NMs in different domains can avoid this and is discussed in section II-F.

[4]the NM determines these paths using its knowledge of the network topology

E. Control Modules

Many data-plane protocols rely on externally generated state for their operation. In CONMan, data modules generate some of this state from the create/delete instructions given by the NM while some state can be determined through the interaction with peer modules. However, there are scenarios where this state cannot be generated by CONMan locally or through peer interaction, for instance the switch state for an IP module (commonly referred to as the IP forwarding table).

In the Internet, control-plane protocols generate some of the state required for data plane operation. For example, routing protocols generate the IP routing table. Similarly, LCP generates PPP configuration state. With CONMan, control modules do not fit into the generic module abstraction presented earlier. Instead, they advertise their ability to provide the state for certain data modules and the NM simply uses them. For example, the PPP module could advertise that it has a dependency on external state (say, X) and the LCP module advertises that it can satisfy dependency X. While such an approach suffices in some cases, there are also cases when the control module itself requires quite a bit of configuration. Also, the fact that the NM does not generate this state hinders its ability to understand related network operations and gets in the way of root-cause analysis. Finally, errors in control module operation cannot always be debugged by the NM. For example, the NM does not understand BGP and hence, cannot be expected to debug route flaps and the resulting prefix dampening.

In most of these cases, the NM can replace the control protocols and use some high-level goal to generate the required state itself. Of course, this implies that the state generation logic must be embedded into the NM. For example, the 4D research argues for the *replacement* of routing protocols, with the NM using its knowledge of the topology to set the switch state for IP modules in devices across the network. A characterization of the scenarios in which existing control protocols should be retained against the ones in which they should be replaced is part of our future work.

F. Multiple NMs

While the architectural description in the previous section focussed on a single NM managing the network, multiple NMs may exist in a network. Primary and secondary NMs will be needed for robustness (4D focusses on this scenario). In such a set up, one primary NM controls the network at any given time, and another takes over should the primary fail. However, we can also imagine multiple simultaneously operating NMs. One reason for this might be that NMs do specialized jobs. For example, one is responsible for tunnel creation while another monitors for security violations. Another reason might be that NMs are administratively nested. For example, a high-level NM creates VLANs, but each VLAN has its own NM. And if nothing else, competition between vendors would dictate that multiple NMs tussle for control [8], and therefore have to cooperate with each other. The possibility of multiple NMs implies that the architecture should allow for multiple

management channels. Also, different NMs may have different *scopes*, i.e. they might be allowed to manage a subset of the devices in the network.

In CONMan, a NM establishes its own management channel by flooding beacons and hence, multiple management channels can exist. One way to restrict each NM to its scope is to include a list of device-ids in the beacons that the NM generates. While such an approach might work in small networks such as home networks, other networks such as enterprises might require something more secure and scalable. A solution at the other end of the spectrum involves allowing the NM to flood the network but requiring them to produce cryptographic tokens for the devices they want to manage. This would require pre-configuring each device in the network with the network-wide public key. Given this, the human manager can generate capabilities allowing access to only a subset of the devices and hence, NMs can be securely restricted to desired scopes. Extending this, devices could be given their own keying material (say, a public-private key pair or a self certifying device-id [29]). This would allow authenticated (and encrypted, if desired) communication between devices and NMs and would allow devices to determine if a neighboring device is part of the same network.

With regards to the network configuration, having multiple NMs raises the possibility of conflicting configurations. While this is also the case with the existing management plane, the fact that NMs understand the abstraction exposed by all modules makes it easier to detect and resolve conflicts. One way to avoid conflicts in the first place would be to ensure that a module can only by managed by a given control NM and all other NMs needing to configure the module go through the control NM. In effect, this allows the control NM to avoid configuration mismatches. However, any approach that puts one NM in charge raises fears of vendor lock-in and hence, this needs to explored further.

Finally, we would like to ensure that each management channel follows the principle of least-privileges. Hence, devices should be restricted to communication with the NM when using the management channel. There is a large design space for addressing this goal with different assumptions, overheads and resulting solution properties. In our current proposal, we adopt a simple solution - devices only forward management-frames that are either sourced from or destined to the NM.

Of course, different domains will have their own NMs that may need to communicate with each other. In the past, this problem has been considered in the context of trouble reporting and diagnostics [36]. The trust relationship between neighboring domains can be used to allow for NMs in neighboring domains to communicate through their respective management channels. For example, an ISP could provide its subscribers with the relevant keying material at installation time. Using this, the NM of a home network can communicate with NM of the ISP without depending on the data path. Such a set-up would be very beneficial in providing customer support for low level network problems. Similarly, neighboring ISPs could exchange the keying material along with the other information that is currently exchanged during peering establishment. This would serve to reduce the reliance on communication through phones and mailing-lists when things go wrong. In most cases such domains represent competing entities and hence, the NM of one domain would only be allowed to communicate with the NM of the other domain. Other issues concerning the structure of management information shared across domains given the conflicting interests are important but beyond the scope of this paper.

However, this would not work for inter-NM communication across far separated domains. In these cases, the part of the communication between the two NM's outside their networks will be on top of the data plane (i.e. over IP). This means that there are a few failure modes that the NM cannot account for (for example, failures in the network between the domains), something that seems to be endemic of the distributed nature of the Internet.

III. Implementation

CONMan does not necessitate any changes to the way data plane protocols operate. The modeling of protocols as modules and the associated functions can be implemented as wrappers around existing implementations. Using this approach, we have implemented two protocols (GRE and IP) in compliance with the abstraction model presented in section II-B. In the rest of the paper, an existing protocol P_{old} that has been abstracted is referred to as P_{new}. Here, we touch upon our implementation efforts and walk through actual configuration scenarios. Due to space constraints, our IP protocol implementation (IP_{new}) is briefly mentioned in the tunneling example (section III-B) and the performance management example (section III-C) but is not described per se.

A. Management Channel

We have implemented a management channel that operates in a network comprised of Linux-based PCs operating as end-hosts and routers with Ethernet as the connecting medium. The MA on each device and the NM have the ability to send management frames encapsulated in ethernet frames. This was achieved through sockets of the SOCK_PACKET family that allow user-level processes to send raw ethernet frames. Having the management channel run over the link layer instead of the raw physical link avoids the need to change the Ethernet device driver but does imply that our implementation would treat Ethernet problems as lack of connectivity. Also, our implementation determines the MTU restrictions of the underlying medium and hence, fragments and reassembles management data to and from appropriate sized frames respectively. Apart from the use of management frames by devices to discover their neighbors, devices and NMs also participate in the build up of the management channel (briefly described in section II-A). This allows for two way communication between the MA of each device and the NM and hence, provides a substrate for the tunelling example that follows.

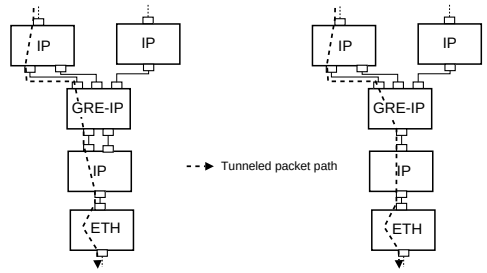

Fig. 3. Module connectivity in a device with three GRE tunnels to two different devices - the figure on the left shows the way GRE-IP modules are modeled in this paper while the figure on the right shows another possible model for GRE-IP modules

B. GRE tunneling

Tunneling is a tool present in the kit of most system administrators. Traditionally, tunnels have been used for both plain (IP-IP, GRE) and secure (IP-Sec) communication between two private networks. Lately, tunnels have also been used by ISPs for DoS-protection (ArborNetworks [44]) and traffic engineering (MPLS [46]). In spite of their widespread use, a look at most network management newsgroups suggests that tunnels pose many configuration and debugging problems (about 5-10% of the postings on [49]). For example, a simple-to-address yet very common problem is the tunnel end-points not agreeing on parameters such as addresses, keys etc. An IETF group [11] is looking at exactly such tunnel configuration problems.

Here, we use the GRE protocol to elucidate and contrast the establishment and debugging of tunnels in the existing and the proposed architecture. GRE is an encapsulation protocol that can be used to encapsulate any network protocol (*payload protocol*) in any other network protocol (*delivery protocol*). However, an actual implementation of GRE is highly dependent on the delivery protocol. We focus on GRE with IPv4 as the underlying delivery protocol - $GRE\text{-}IP$. Hence, each tunnel is characterized by a source and a destination IP address. Besides this, GRE also allows key'ing of tunnels - the source and the destination must agree on the key for the tunnel to operate correctly. Hence, configuring a $GRE\text{-}IP_{old}$ tunnel involves determining the following:

- IP addresses of tunnel end-points
- Key values - keying of the tunnels allows for differentiation of the tunnel traffic. The key is chosen by the receiver independent of the key in the other direction and must not conflict with other keys at the receiving end.
- Sequence number usage - sequence numbers help with in-order delivery of the tunneled packets and can be used independently in both directions.
- Other protocol specific values such as the tunnel TTL, the TOS field for the tunneled packets, whether to use the checksums or not, whether to use path-mtu-discovery or not.

1) $GRE\text{-}IP_{new}$ *:* We have implemented a $GRE\text{-}IP_{new}$ module conforming to the CONMan architecture. Our implementation is based on the Linux GRE kernel module with a user-level wrapper that presents the module details using the generic abstraction. This confines all decisions on how packets fields are used for demultiplexing and other protocol-specific details to the implementation. While the way demultiplexing is done reflects how the implementor chooses to divide functionality between various protocols and is not of any interest to the NM, it does impact the parameters that a module needs to determine when the NM creates and deletes pipes. Hence, we start by presenting some of these internal details.

In our current implementation, a single $GRE\text{-}IP_{new}$ module models all GRE over IP tunnels originating from a device. Since a GRE-IP tunnel is characterized by the IP addresses of the tunnel end-points and a key value in each direction, we chose to model the demultiplexing of incoming GRE packets as split into two phases. The IP module below the $GRE\text{-}IP_{new}$ module demultiplexes packets based on the source-destination address pair. Hence, the number of down pipes for the $GRE\text{-}IP_{new}$ module is equal to the number of distinct source-destination address pairs for tunnels originating from this device. The $GRE\text{-}IP_{new}$ module then demultiplexes packets from each down pipe into an up pipe based on the key value. Thus, each up pipe for the $GRE\text{-}IP_{new}$ module represents a separate GRE-IP tunnel originating from the device.

The left part of figure 3 shows the module connectivity with our current model for a device with three GRE-IP tunnels; two of these tunnels are to the same peer device (i.e. two of them have the same source-destination address pair)[5]. Hence, the $GRE\text{-}IP_{new}$ module has three up pipes representing the three tunnels and two down pipes representing the fact that these tunnels are to two distinct peer devices. The figure also shows the path for a packet that originates at the device and gets tunneled on the way out. There are other possible models for GRE-IP tunneling, such as one where the $GRE\text{-}IP_{new}$ module has just one down pipe. The figure on the right shows such a scenario - in such a model, the $GRE\text{-}IP_{new}$ module is responsible for both stages of demultiplexing. Hence, tunnel establishment in our model involves the $GRE\text{-}IP_{new}$ module determining the key values and the IP module determining the IP addresses of the end points (as explained later in this section). In the alternate model, the $GRE\text{-}IP_{new}$ module would have to be responsible for both.

Such internal details are not exposed to the NM. The abstraction exposed by our $GRE\text{-}IP_{new}$ module to the NM is shown in table III and some of the entries are explained below:

b). Ideally, GRE can carry any payload protocol and hence, there should not be any restriction on the modules that can be connected to using an up pipe. However, most implementations restrict the payload protocol to a well defined list of protocols - with our underlying Linux implementation, the only payload protocol possible is IPv4.

c). To create an up pipe, the NM needs to specify the name of the peer $GRE\text{-}IP_{new}$ module. In effect, this allows the

[5]the figure assumes that the IP modules above and below the $GRE\text{-}IP_{new}$ module are different, as the case may be in virtual routers. However, the same discussion applies even when the same IP module is above and below the $GRE\text{-}IP_{new}$ module.

	Parameter	Value
a	Name	<GRE-IPv4,**device-id**,**module-id**>
b	Up.Con-Modules (Connectable-Modules)	IPv4
c	Up.Dependencies	Peer GRE-IPv4 module
d	Down.Con-Modules	IPv4
e	Down.Dependencies	Nil
f	Physical Across pipes	Nil
g	Peerable-modules	GRE-IPv4
h	Filter	Nil
i	Switch	[Up $\Rightarrow_{\text{one, unicast}}$ Down | Auto] [Down $\Rightarrow_{\text{many, unicast}}$ Up | Auto]
j	Perf Reporting	Recieved and Transmitted packets on each up and down pipe
k	Perf Trade-Offs	{[Jitter, Delay] Vs [In-order delivery] | Up } {[Loss-Rate] Vs [Error-Rate] | Up}
l	Perf Enforcement	Nil
m	Security	Nil

TABLE III

$GRE\text{-}IP_{new}$ ABSTRACTION EXPOSED BY OUR IMPLEMENTATION

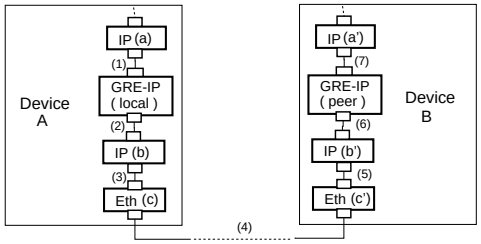

Fig. 4. GRE-IP tunnel between devices **A** and **B** - the relevant module connectivity is shown here.

module to coordinate various protocol specific values with the peer module.

d) The module is restricted to having IPv4 as the tunneling protocol.

e). No dependencies need to be satisfied for establishment of a down-pipe.

i). The module has the following switching capabilities: up pipe to a single down pipe and down pipe to many up pipes (the switch is still unicast i.e. any given packet from a down pipe is still switched to a single up pipe). The actual switching state is produced automatically by the module.

j) The underlying Linux implementation provides limited performance reporting: the number of packets transmitted and received on each up and down pipe. Performance metrics for connectivity to peer modules are not reported.

k). The module offers the following trade-offs: For a specified up pipe, it can trade-off delay and jitter for in-order delivery. The fact that this is attained by enabling sequence numbers whose use needs to be coordinated with the peer module is not exposed. Similarly, the module can trade-off loss-rate for error-rate for a specified up pipe through the use of checksums.

The NM can establish a GRE-IP tunnel by creating the requisite pipes. Consider the establishment of a tunnel between devices **A** and **B**. The relevant modules are shown in figure 4. The process by which the NM configures a tunnel is labeled as *Configuration by NM* in figure 5 and is discussed below. We only discuss the functions invoked at device **A**; the invocations at **B** are similar. Also, the description below is for the scenario when both **A** and **B** are in the same domain; establishment of cross-domain tunnels can be achieved in a similar fashion through inter-NM communication.

First, given its knowledge of the network topology, the NM determines if there is physical connectivity between devices **A** and **B**. Second, the NM invokes *showPotential* () at device **A** that allows it to determine the following:

- Name and abstraction for the ETH module at **A** that is physically connected to the ETH module at **B** - let this module be <ETH,**A**,**c**> (or *c*, for short). While we don't show the ETH module abstraction here, the connectable-modules for the up pipe of this module include IP. Hence, the NM finds an IP module that can be connected above *c*.
- Name and abstraction of the IP module at **A** that can be connected above *c* - let this module be <IP,**A**,**b**> (or *b*). Using the same logic as above, the NM needs a $GRE\text{-}IP_{new}$ module that can be connected above *b*.
- Name and abstraction of the $GRE\text{-}IP_{new}$ module at **A** that can be connected above *b* - let this module be <GRE-IPv4,**A**,**local**> (or *local*). The module abstraction exposed by *local* is shown in table III. It describes the possible up and down connectivity for *local* and the trade-offs it can offer. Using the same logic, the NM determines that module <IP,**A**,**a**> (or *a*) needs to be connected above *local*.

Hence, *showPotential*() equips the NM with the names of the modules that need to be connected to create the GRE-IP tunnel and the module abstractions for these. By contrast, some manual must be read (either by the implementor of the NM system or the system administrator) to gain the equivalent knowledge while configuring $GRE\text{-}IP_{old}$ tunnels.

Next, the NM connects the aforementioned modules as follows: *create* (*pipe*, *b*, *c*) creates pipe (3), *create* (*pipe*, *local*, *b*) creates pipe (2), and *create* (*pipe*, *a*, *local*) creates pipe (1)[6]. In all three cases, the NM needs to satisfy any dependencies that the pipes being created may have. For example, when creating pipe (1), the NM needs to satisfy the dependency of the up pipe of *local* by providing the name of the peer $GRE\text{-}IP_{new}$ module - in this case, the NM tells *local* that the peer module is *peer*. The creation of pipes (5) through (7) is similar.

The simple, structured and bottom-up process described above is all the configuration that the NM needs to do. On the other hand, the protocol implementations themselves have to incorporate the complexity of the tunnel establishment. Each module uses the high-level abstraction and interaction with the peer modules through the management channel to determine the required protocol specific parameters – this is shown as *Determining protocol details* in figure 5 and is detailed below. In case of the $GRE\text{-}IP_{old}$ configuration, it is the management plane (either an application or a human) that is expected to specify the parameters.

create (*pipe*, *local*, *b*) and *create* (*pipe*, *peer*, *b'*). The creation of pipes (2) and (6) causes modules *b* and *b'* to figure

[6]Note that each create command may not lead to the creation of an actual pipe, as the requisite pipe may already exist due to some previously configured tunnel. However, the NM need not be aware of this.

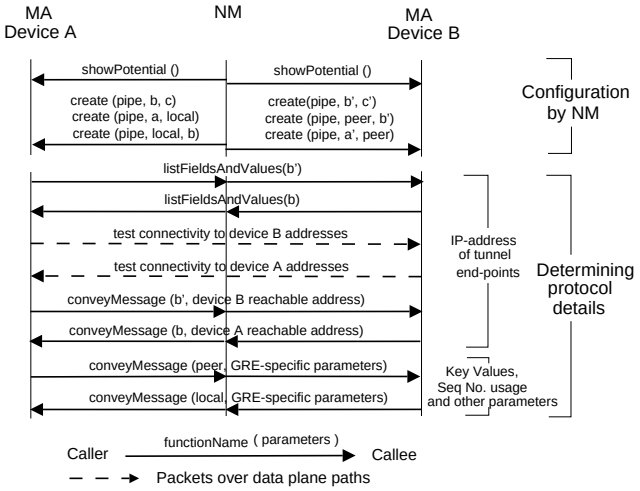

Fig. 5. GRE-IP Tunnel establishment between devices **A** and **B** - the management plane is simplified by ensuring that protocol complexity is restricted to protocol implementation.

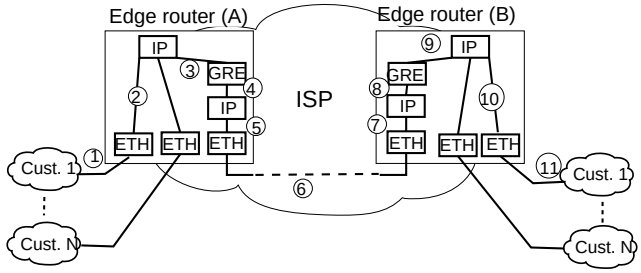

Fig. 6. GRE-IP tunnels for ISPs' customers - the figure shows the module connectivity that the NM aims to achieve

out the IP addresses of tunnel end-points. They determine each other's IP addresses through the use of *listFieldsAndValues* and actively send packets over the data plane to determine the IPv4 address pair that would allow bidirectional IPv4 connectivity between them.

create (*pipe*, *a*, *local*) and *create* (*pipe*, *a'*, *peer*). The creation of pipes (1) and (7) causes modules *local* and *peer* to exchange the GRE-specific parameters needed for connectivity between them. These include the key values in each direction, the use of sequence numbers, etc. Note that some of these parameters, such as the use of sequence numbers, may be guided by the trade-off decisions made by the NM.

2) Example configuration scenario: In order to highlight the advantage of CONMan over the present day management plane, we discuss a real world tunnel configuration example. ISPs often use GRE tunnels to carry traffic between distant customer sites. The tunnel provides traffic differentiation which can be used for the enforcement of performance, security, etc. Lets consider an ISP that wants to establish GRE tunnels between customers at two of its POPs. The customer sites at the first POP are connected to edge router **A** while the ones at the second POP are connected to edge router **B**.

In CONMan, the NM logic must map the high-level tunneling goal to what it can do - *building pipes and modules*. In order to configure a $GRE\text{-}IP_{new}$ tunnel for the customer-1, the NM must trace and build a path between the customer-1

```
#!/bin/bash
# Inserting the  GRE-IP kernel module
insmod /lib/modules/2.6.10-1/ip_gre.ko
# Creating the GRE module with the appropriate key
ip tunnel add name greA mode remote 128.84.223.112 local \
128.84.222.111 ikey 2001 okey 1001 icsum ocsum iseq oseq
ifconfig greA 192.168.1.3
# Enable routing
echo 1 > /proc/sys/net/ipv4/ip-forward
# Create IP routing state from customer  to tunnel
echo 202 tun-1-2 > /etc/iproute2/rt_tables
ip rule add iff eth0 table tun-1-2
ip route add default dev greA table tun-1-2
# Create IP routing state from tunnel to customer
echo 203 tun-2-1 > /etc/iproute2/rt_tables
ip rule add iff greA table tun-2-1
ip route add default dev eth0 table tun-2-1
```

(3, 4)
(1)
(2)

Fig. 7. Configuration script at **A** (ISP-facing IP : 128.84.222.111) - the script tunnels traffic from a customer network attached to *eth0* to **B** (ISP-facing IP : 128.84.223.112). The GRE-IP tunnel is key'ed, uses sequence numbers and checksums in both directions.

interfaces of devices **A** and **B** that goes through the respective GRE modules. This is the path labeled as (1) to (11) in figure 6. Tunnels for the other customers can be configured similarly. Our implementation of the NM builds various segments of the aforementioned path as follows:

(a). (1)-(2): Pipe (1) is a physical pipe while pipe (2) is created by the NM. The ETH module's abstraction shows that switching packets between (1) and (2) is its default operation and hence, does not require any configuration.

(b). (2)-(3): Pipe (3) is created by the NM. The IP module above the GRE module advertises that it has down-to-down switching capability and this must be configured by the NM. Hence, the NM configures the down-to-down switching at the IP module to achieve connectivity between pipes (2) and (3).

(c). (3)-(4),(4)-(5),(6)-(7),(7)-(8),(8)-(9): as described in the previous section, the NM achieves this by creating the relevant pipes while satisfying the necessary dependencies. In effect, this sequence of pipes represents customer-1's GRE tunnel.

(d). (9)-(10): same as (2)-(3).

(e). (10)-(11): same as (1)-(2).

As a contrast, configuring a $GRE\text{-}IP_{old}$ tunnel would require the management plane to specify all the low level configuration details. Figure 7 shows a Linux configuration snippet at the customer-facing border router **A** that establishes the tunnel for just one customer. Apart from the configuration being complex, it leaves the door open for many kinds of errors. Some such error possibilities have been marked in the figure: (1). not configuring device A as a router, (2). misconfiguring the underlying routing so that traffic from the wrong customer goes into a tunnel or the tunneled traffic is delivered to the wrong customer at the other end, (3). configuring the tunnel end points with the wrong key values and (4). using tunnel end point IP addresses that are wrong or do not have IP connectivity between them.

The CONMan model also allows the NM to debug tunnel errors. For example, errors like a wire getting cut off or a port on a line card not working show up in the topology map that

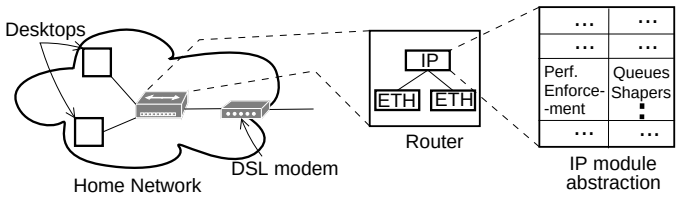

Fig. 8. A home network with desktops, a router and provider equipment (eg, DSL-modem). The router advertises its modules and the IP module abstraction includes its performance enforcement capabilities.

the NM maintains (via the protocol used for maintaining the management channel). By tracing the network and protocol graph, the NM may even report what applications are likely to be affected by such an error. On the other hand, errors like an invalid filter rule in the ISP's network that blocks IP connectivity between the tunnel end points will be detected when the NM inspects the state of the module with the filter rule. And errors like path MTU problems are detected by invoking the IP module's *test* function. Hence, the bottom-up knowledge of the network, the ability to understand the functionality of modules and the *test* functions provided by the modules allow the NM to find the root cause of most errors.

C. Other examples

We now discuss a few example management scenarios that CONMan can serve useful in – in each case, we very briefly mention the problem and a possible solution. First, consider a *basic configuration* scenario involving printer management in an enterprise. In the current set-up, users are frequently befuddled by the various protocols that they can use to connect to print servers, not to mention configuring the parameters used by these protocols. For example, our own organization supports printing using LPD, CUPS, Windows Active-Directory and IPP and maintains a detailed document (>200 lines) so that users, CS professionals in this case, can configure printing on their machines.

In CONMan, the Printer-Client module(s) on the user's machine advertises its connectable-modules, the user specifies his preferences (if any) for the print server to be used, the administrator specifies the restrictions on which machines can access which print server and the Printer-Server module on each print server advertises its connectable modules. With this information, the NM can connect pipes for the appropriate Printer-Client and the Printer-Server modules and hence, ensure basic printer connectivity. Beyond this, it is the protocol modules that negotiate their parameters - this might include authentication based on user credentials. Note that UPnP offers similar device management capabilities with zero user configuration. However, UPnP operates on top of the data plane and forces the control points (analogous to NMs in CONMan) to understand protocol and vendor specific details. This violates both principles described in section I. CONMan can ensure basic device discovery and connnectivity with much less effort on part of the management plane. Though, on a positive note, UPnP-enabled protocols are designed to coordinate parameters with their peer protocols which makes them easy to fit into the CONMan framework.

Second, consider a *performance management* scenario involving VOIP usage in a home network. Figure 8 shows such a home network with multiple desktops connected to the ISP's modem through a router. ISPs typically cap the upload bandwidth of their customers. Meanwhile, the user may want to ensure good call quality and web browsing experience regardless of the background traffic. Many companies already offer routers for home networks with in-built QoS enforcement capabilities for this very purpose [48]. In CONMan, the IP_{new} module in the router advertises its performance enforcement capabilities, the user specifies his priorities (and other information like the maximum number of simultaneous calls) and application modules specify their bandwidth requirements. For example, the VOIP application being used, lets say a Skype-client, may advertise that it produces traffic at a constant rate of 15 Kbps per call [50]. This information allows the NM to configure the performance enforcement capabilities of the router. Our IP_{new} implementation supports such configuration - currently, the NM restricts the output rate to the cap value and prioritizes traffic according to user/application specifications, all without knowing the details of how priority is enforced and how traffic from a particular application module is identified.

Third, consider a *basic debugging* scenario involving problems in accessing a web-site from a host. In CONMan, the NM debugs the problem by actually trying to configure various modules in the host to establish connectivity to the web-site in question. The NM directs the creation of pipes between the host's HTTP-client module, TCP module, IP module and ETH module and then invokes the respective *test* functions. A problem with DNS shows up in the creation of the down pipe for the HTTP-client module (which advertises that the creation of a down pipe has a dependency on DNS). The web server being down shows up when the *test* function for the TCP module is invoked. Routing and physical connectivity problems inside the host's domain are detected directly by the NM while external routing and physical connectivity problems show up when the *test* function for the IP module is invoked. While the NM may only be able to address some of these issues, the debugging done by the NM would certainly be useful when the system administrator is called in.

Finally, consider a *security management* scenario involving enforcement of a security policy in an enterprise network. The NM can configure and enforce policies such as an access matrix specifying what machines should be able to reach each other (by configuring filters in IP modules appropriately or by doing what 4D had proposed), what machines should have secure connectivity between them (by establishing IP-Sec tunnels between machines) and so on. The NM, given its network wide view, can also detect any violation of these policies. For example, a user bringing his home equipment (laptop, access-point, etc.) to work would show up as a new device and associated pipes that are against the specified

security policy[7].

IV. Related work

The heterogeneity in the size, technology and needs of domains making up the Internet has given rise to a large variety of management tools. These vary from low-end tools like packet analyzers (eg, Ethereal [42]), traffic monitors (eg, MRTG [37]), and SNMP agents (eg, ITM [6]) to highly sophisticated Network Management Systems (eg, OpenView [43]). SNMPLink [32] gives an extensive and categorized list of such management tools. The number and varied operation of these tools corroborates our claim that no single approach suffices for managing today's networks.

The research and the standards community have also tried to address the network's management needs. At the root of these is the notion of a management station controlling various devices through their management interfaces which formed the basis of management standards like SNMP and RMON [34]. Lately, a majority of the proposals have focussed on the improving the scalability, reducing the overhead and promoting reusability of components by doing some part of the management at the devices themselves. These include efforts like management by delegation [15], using mobile agents for management [31] and pattern-based management [24]. However, none of these address the problem arising out of the very detailed and hugely varying interfaces exposed by protocols and devices.

Policy-based management [17] tries to reduce the amount of intricate knowledge required by network managers by allowing management of QoS [33], [2] and security [39] based on high-level policies. While a step in the right direction, some entity still has to map these policies to the individual device configurations. The complexity of this mapping was the major impediment in the adoption of policy-based networking by major vendors and enterprises [18]. As a contrast, CONMan argues that the management plane should perform the much easier task of mapping policies to the generic abstraction exposed by devices. Also, all the commercial tools and standards discussed above suffer from having the management plane depend on the data plane.

The question of coming up with abstractions for protocols has been asked previously in the context of protocol implementation (Morpheus [1]), protocol specification (Estelle, LOTOS, SDL [40]) and fault localization in model-based systems (see [14] for examples). Click [20] presents a fine grained abstraction for the IP module as the basis for building flexible and configurable routers. A lot of work has also been done on modular implementations of protocols (eg, [4]). Recently, the P2 project proposed declarative and componentized implementations of routing protocols [26], overlays [25] and transport protocols [10]. This paper does not argue for implementation modularity. Rather, our goal was to capture the essence of most protocols for the purpose of manageability using a simple and generic abstraction and make these protocols, regardless of how they are implemented, expose it.

The 4D proposal [16] recognizes the complexity of the Internet's control and management plane and hence, argues for restructuring them. We were motivated by, among other things, 4D's decision plane and have proposed some management channel extensions to support multiple NMs. While 4D focusses on routing-related management issues, CONMan is a proposal for stack-wide configuration and debugging. As mentioned earlier, 4D's proposal for doing away with routing and switching protocols also fits into our framework, though we are not sure if this should be the case or if this should be done for other control protocols too.

Recently, there has been a spurt of research detailing the reasons for outages and anomalies in IP backbones [23], [28], Internet services [30] and BGP routing [12], [27]. These studies point to configuration errors as a major culprit. CONMan reduces the management plane's burden by restricting the complexity of protocols to their operation and hence, can reduce these errors, particularly the ones directly impacting data plane operation. We believe that our proposal has the potential to achieve the Knowledge Plane objectives laid out by Clark et. al. [7] - our current proposal presents the knowledge plane for individual domains and can be extended to an Internet-wide knowledge plane.

Kompella et. al [21] present a strong case for maintaining cross-layer dependencies and argue for a cross-layer database with a fixed interface (for the imported information) to do so. We agree with the need to maintain these associations and think that our abstraction can serve as such an interface. This would simplify both the database and the management applications that use it. Also, the authors suggest using SNMP and other measurement-based approaches to populate the database. We propose that the network devices provide the management plane with information about themselves through a management channel which is independent of the data plane.

V. Discussion and Future work

In this paper we have presented a network architecture that is amenable to management. Implementation of the GRE and IP protocols according to the CONMan model and their use in real world configuration scenarios shows that the approach has a lot of promise. Though it is too early for us to claim that the abstraction presented here suffices for all data plane protocols, we do not envision the module abstraction expanding much beyond its current state. Having a small module abstraction that is generic enough has a lot of benefits and we believe that expanding upon it will run into diminishing returns. In cases where protocol features (if any) are not captured by the abstraction, the protocol parameters will have to be set explicitly.

The ease-of-management achieved by our architecture has some security implications. One could argue that the complex and disjoint management set-up existing today reduces the impact of network security breaches. CONMan addresses the

[7] assuming that the device is incorporated in the management channel in the first place

root of the problem by allowing for specification of security policies that the network elements can enforce. This reduces the possibility of a break in, allows for detection when someone does break in and can even be used to trace back the attacker's actions (in a fashion similar to debugging of configuration errors). Also, our design decisions, such as restricting NMs to their scopes and making the management channel default-off, follow the principle of *least privileges* and should serve to address the security concerns. While an evaluation of the net impact of CONMan on network security is an important avenue of future research, we can at least claim the higher moral ground of not doing *security through obscurity*.

There are numerous other issues and challenges posed by our proposal. In the interest of brevity, we simply mention a couple of them: applying CONMan to networks other than the traditional wired networks, allowing for product differentiation in the face of our generic module abstraction, and having an incremental deployment model for CONMan. In response to the last issue, one could imagine an intermediate scenario whereby all protocols expose their abstraction as a meta-MIB which is then accessed and configured by the NM through SNMP (this does not satisfy the first principle presented in the paper and hence, does not reap all of CONMan's benefits). We believe that this can serve as a path of least resistance towards the widespread adoption of CONMan and hence, towards the ultimate goal of a manageable (or even, self-managing) network.

REFERENCES

[1] ABBOTT, M. B., AND PETERSON, L. L. A language-based approach to protocol implementation. In *Proc. of ACM SIGCOMM* (1992), pp. 27–38.

[2] AMIRI, K., CALO, S., AND VERMA, D. Policy based management of content distribution networks. *IEEE Network Magazine* (March 2002).

[3] BABIARZ, J., CHAN, K., AND BAKER, F. Configuration Guidelines for DiffServ Service Classes. draft-ietf-tsvwg-diffserv-service-classes-01, July 2005. IETF Draft.

[4] BIAGIONI, E. A structured TCP in standard ML. In *Proc. of ACM SIGCOMM* (1994).

[5] CAESAR, M., CALDWELL, D., FEAMSTER, N., REXFORD, J., SHAIKH, A., AND VAN DER MERWE, J. Design and Implementation of a Routing Control Platform . In *Proc. of 2nd Symp. on Networked Systems Design and Implementation (NSDI)* (2005).

[6] CARSTEN SCHMIDT. Interface Traffic Monitor Pro. `http://software.ccschmidt.de/`.

[7] CLARK, D. D., PARTRIDGE, C., RAMMING, J. C., AND WROCLAWSKI, J. T. A knowledge plane for the internet. In *Proc. of ACM SIGCOMM* (2003), pp. 3–10.

[8] CLARK, D. D., WROCLAWSKI, J., SOLLINS, K. R., AND BRADEN, R. Tussle in cyberspace: defining tomorrow's internet. In *Proc. of ACM SIGCOMM* (2002).

[9] COMER, D. *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture*. Prentice Hall, 2000.

[10] CONDIE, T., HELLERSTEIN, J. M., MANIATIS, P., RHEA, S., AND ROSCOE, T. Finally, a Use for Componentized Transport Protocols. In *Proc. of the Fourth Workshop on Hot Topics in Networking* (2005).

[11] DURAND, A., AND NARTEN, T. IETF Tunnel Configuration BOF (tc), Mar 2005. `http://www.ietf.org/ietf/05mar/tc.txt`.

[12] FEAMSTER, N., AND BALAKRISHNAN, H. Detecting BGP Configuration Faults with Static Analysis. In *Proc. of 2nd Symp. on Networked Systems Design and Implementation (NSDI)* (2005).

[13] FORD, B. Unmanaged Internet Protocol: taming the edge network management crisis. *SIGCOMM Comput. Commun. Rev. 34*, 1 (2004).

[14] GARDNER, R., AND HARLE, D. Methods and systems for alarm correlation. In *Proc. of Global Telecommunications Conference* (1996).

[15] GOLDSZMIDT, G., YEMINI, Y., AND YEMINI, S. Network management by delegation: the MAD approach. In *Proc. of the conference of the Centre for Advanced Studies on Collaborative research (CASCON)* (1991).

[16] GREENBERG, A., HJALMTYSSON, G., MALTZ, D. A., MEYERS, A., REXFORD, J., XIE, G., YAN, H., ZHAN, J., AND ZHANG, H. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communications Review* (October 2005).

[17] HALPERN, J., AND ELLESSON, E. The IETF Policy Framework Working Group. Online Charter. `http://www.ietf.org/html.charters/OLD/policy-charter.html`.

[18] JUDE, M. Policy-based Management: Beyond The Hype. *Business Communication Review* (2001), 52–56. `http://www.bcr.com/bcrmag/2001/03/p52.php`.

[19] KERRAVALA, Z. Enterprise Networking and Computing : the Need for Configuration Management. Yankee Group report, January 2004.

[20] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3 (August 2000), 263–297.

[21] KOMPELLA, R. R., GREENBERG, A., REXFORD, J., SNOEREN, A. C., AND YATES, J. Cross-layer Visibility as a Service. In *Proc. of fourth workshop on Hot Topics in Networks (HotNet-IV)* (2005).

[22] KOMPELLA, R. R., YATES, J., GREENBERG, A., AND SNOEREN, A. C. IP Fault Localization Via Risk Modeling . In *Proc. of 2nd Symp. on Networked Systems Design and Implementation (NSDI)* (2005).

[23] LABOVITZ, C., AHUJA, A., AND JAHANIAN, F. Experimental Study of Internet Stability and Backbone Failures. In *Proc. of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (FTCS)* (1999).

[24] LIM, K.-S., AND STADLER, R. Developing Pattern-Based Management Programs. In *Proc. of the 4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS)* (2001).

[25] LOO, B. T., CONDIE, T., HELLERSTEIN, J. M., MANIATIS, P., ROSCOE, T., AND STOICA, I. Implementing Declarative Overlays. In *Proc. of ACM SOSP* (2005).

[26] LOO, B. T., HELLERSTEIN, J. M., STOICA, I., AND RAMAKRISHNAN, R. Declarative Routing: Extensible Routing with Declarative Queries. In *Proc. of ACM SIGCOMM* (2005).

[27] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Understanding BGP misconfiguration. In *Proc. of ACM SIGCOMM* (2002), pp. 3–16.

[28] MARKOPOULOU, A., IANNACCONE, G., BHATTACHARYYA, S., CHUAH, C., AND DIOT, C. Characterization of Failures in an IP Backbone. In *Proc. of IEEE INFOCOMM* (2004).

[29] MAZIÈRES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. Separating key management from file system security. In *Proc. of the 17th ACM Symposium on Operating Systems Principles* (1999), pp. 124–139.

[30] OPPENHEIMER, D., GANAPATHI, A., AND PATTERSON, D. Why do Internet services fail, and what can be done about it. In *Proc. of USENIX Symposium on Internet Technologies and Systems* (2003).

[31] PHAM, V. A., AND KARMOUCH, A. Mobile Software Agents: An Overview. *IEEE/ACM Trans. Netw. 36*, 7 (1998).

[32] PIERRICK SIMIER. SNMPLink. `www.snmplink.org/Tools.html`.

[33] RAJAN, R., VERMA, D., KAMAT, S., FELSTAINE, E., , AND HERZOG, S. A policy framework for integrated and differentiated services in the internet. *IEEE Network Magazine 13*, 5 (September 1999).

[34] STALLINGS, W. *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*. Addison-Wesley Publishers, 1999.

[35] THALER, D. Automating Network Diagnostics to Help End-Users . , June 2005. `research.microsoft.com/events/smnsummit/Presentations/thaler.ppt`.

[36] THALER, D., AND RAVISHANKAR, C. An Architecture for Inter-Domain Troubleshooting. In *Proc. of the 6th International Conference on Computer Communications and Networks (IC3N)* (1997), p. 516.

[37] TOBIAS OETIKER AND DAVE RAND. MRTG : Multi Router Traffic Grapher. `http://mrtg.hdl.com`.

[38] UIJTERWAAL, H., AND ZEKAUSKAS, M. IP Performance Metrics (ippm). Online Charter, Jan 2006. `http://www.ietf.org/html.charters/ippm-charter.html`.

[39] VERMA, D. Simplifying Network Administration using Policy based Management. *IEEE Network Magazine* (March 2002).

[40] VON BOCHMANN, G. Usage of Protocol Development Tools: The Results of a Survey. In *Proc. of the Seventh International Conference on Protocol Specifi cation, Testing and Verifi cation* (1987), pp. 139–161.
[41] XIE, G., ZHAN, J., MALTZ, D. A., ZHANG, H., GREENBERG, A., AND HJALMTYSSON, G. Routing design in operational networks: a look from the inside. In *Proc. of ACM SIGCOMM* (2004), pp. 27–40.
[42] Ethereal : A Network Protocol Analyzer. `www.ethereal.com`.
[43] HP OpenView. `www.openview.hp.com/`.
[44] Arbor PeakFlow-SP : DoS protection. , January 2006. `www.arbornetworks.com`.
[45] CISCO GRE Keepalives. , January 2006. `www.cisco.com/en/US/tech/tk827/tk369/technologies_tech_note09186a008040a17c.shtml`.
[46] Cisco IP Solution Center Traffi c Engineering Management. , January 2006. `www.cisco.com/en/US/products/ps6163/`.
[47] SNMP MIB Search Engine. , January 2006. `www.mibdepot.com`.
[48] D-Link Wireless Broadband VoIP Router, Jan 2006. `http://www.dlink.com/products/?sec=1&pid=446`.
[49] Management Newsgroups, Jan 2006. `news:fa.netbsd.tech.net`, `news:comp.dcom.sys.cisco`, `news:microsoft.public.isa.vpn`, and `news:comp.os.linux.networking`.
[50] Skype Wikipedia entry, Jan 2006. `http://en.wikipedia.org/wiki/Skype`.