CONMan: Taking the Complexity out of Network Management

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

ABSTRACT

Network management is difficult, costly, and error prone, and this is becoming more so as network complexity increases. We argue that this is an outcome of two fundamental flaws in the existing architecture: the management plane depends on the data plane, and network device management interfaces are varied, complex, and constantly evolving. 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 simple 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.

Keywords: Network management, Management plane, Data plane, Protocol abstraction.

1. 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. In spite of this, management costs continue to rise. A recent survey [11] 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. When there is a low-level failure, the management tools may not be able to communicate with the network. This dependency loop also increases the need for (error-prone) manual configuration, since low-level protocols must be configured before SNMP can operate [3, 8].

The second shortcoming is the fact that network devices expose their raw internal details, leading to a *deluge of complexity* which burdens the management plane. For example, it is not uncommon for a network device to have hundreds of manageable objects. A study of SNMP MIB modules found more than 13,000 MIB objects in IETF MIBs alone [21]. A single router configuration file can consist of more than 10,000 command lines [27]. It takes considerable expertise and sophistication on part of the network manager to understand what effect each of these detailed objects has on the network.

In addition to the inevitable errors that this complexity engenders, the pace at which these details evolve makes it difficult for any one management application to stay current. As a result, vendors produce management solutions for their own products, and network managers rely on low-level scripts to bridge the gap. Consequently, *no one management approach* suffices, leading to a plethora of tools, and further increasing the complexity of the management problem. For example, SNMPLink [19] lists more than 1000 management applications, many of them being vendor specific command line or HTML-based tools. Thus, the Internet *management plane* doesn't have anything analogous to the IP thin waist around which the Internet *data-plane* is built.

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 as long as physical connectivity exists. This was proposed as part of the 4D project [8]. We embrace and extend 4D.

(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.

With a goal towards stimulating discussion and new thinking about network management, this paper describes a network management architecture, *Complexity Oblivious Network Management* (CONMan), that follows these principles. CONMan includes 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.

Note that 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. Thus, while the management interface gives the appearance of protocol modularity, the protocol implementation itself may be modular or monolithic.

2. CONMan ARCHITECTURE

2.1 Terminology and Overview

Our architecture consists of *devices* (routers, switches, hosts, etc.) and one or more *network managers* (NMs). Though there are a number of issues associated with operating multiple NMs, in this paper we assume a single NM. Many such issues and the way CONMan deals with them are detailed in [2]. 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 globally unique, topologically independent identifier (*device-id*) that may carry cryptographic meaning. Each device also has an internal *management agent* (MA) that is responsible for the device's participation in the management plane. 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. All protocols and applications in devices are modeled as *protocol modules*. Each protocol module has a name as well as an identifier that is unique within the device. Examples of module names include "IPv4", "RFC791", or even a URI (which might be useful for naming applications). Thus, modules can be uniquely referred to using tuples of the form <module Name,**module-id**,**device-id**>.

CONMan achieves the first principle by borrowing techniques used by 4D for its discovery and dissemination plane [8]. As long as there is physical connectivity, 4D provides a robust *management channel* that allows NMs and devices to discover and communicate with each other. Devices can communicate indirectly by passing messages through an NM. For lack of space, we omit the details of how this is done and refer the reader to [8].

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.

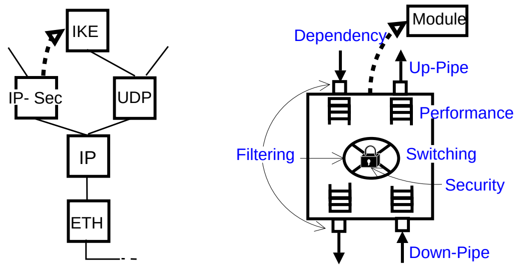


Figure 1: Modules, pipes, and dependencies form a graph that describes the operation of a device (in particular) and the network (in general). The figure on the right shows the major components of the *module abstraction*. Note that some modules may not require all elements of the abstraction to describe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

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.

2.2 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

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

that in some sense describes the operation of the network. The actual module abstraction, as shown on the right in figure 1, tries to capture the capabilities, the dependencies and the functionality of protocol modules in generic and abstract terms. In the following, we briefly describe the components of the module abstraction: pipes, switches, filters, performance, and security.

2.2.1 Pipes

Up and *Down* pipes connect modules to other modules above and below themselves in the same device, and are point-to-point. Physical links are *Across* pipes, and can be point-to-point or broadcast. The *path* between two modules in two different devices is the sequence of up-down and physical across pipes through which packets travel.

Because they are physical, the NM cannot create across pipes per se, but can discover and enable them. On the other hand, the NM can create up and down pipes to construct paths between modules. The NM can also discover and potentially disallow up and down pipes created by other means. For example, a HTTP-client initiating a TCP connection results in a pipe between the HTTP-client module and the TCP module which will be discovered by the NM. Each module lists with which other modules it may have up/down pipes. Each module also lists what it considers to be its *peer* modules. For instance, TCP modules are peers with each other, and HTTP-client modules are peers with HTTP-server modules. Note that pipes may have dependencies that need to be satisfied before they can be created or enabled. For example, a pipe may require other pipes to be created or switch state to be specified (see below) before it can be created.

2.2.2 Switch

Modules use the *switch* abstraction to self-describe how packets potentially or actually flow between pipes. Each switch entry can be unicast or multicast, and can have a small number of basic configurations: packets pass between down and up pipes ([down ⇒ up] and [up ⇒ down] switching, e.g. TCP module), [down ⇒ down] switching (e.g. IP module with forwarding enabled), [up ⇒ up] switching (e.g. IP module with loopback functionality), [up ⇒ physical] and [physical ⇒ up] switching (eg. Ethernet module). Within this basic configuration, more detailed switching information may be configured by the NM, or via other means, for instance a routing protocol.

2.2.3 Filters

The *filter* abstraction allows modules to describe whether and how they can filter packets. Filters are described in terms of other abstracted components: pipes, module types, device, or specific modules. Note that in configuring a filter, the NM only needs to specify the component names or identifiers that need to be filtered - it is the protocol implementation that is responsible for determining the relevant protocol fields.

2.2.4 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).

Name	Caller	Callee	Description
showPotential	NM	MA of device	Sec. 2.3.1
showActual	NM	MA of device	Sec. 2.3.1
create, delete	NM	MA of device	Sec. 2.3.1
conveyMessage	Module (source)	NM	Sec. 2.3.1
conveyMessage	NM	Module (destination)	Sec. 2.3.1
test	NM	Module	Sec. 2.3.2
listFieldsAndValues	Module (Inspecting)	NM	Sec. 2.4
listFieldsAndValues	NM	Module (Target)	Sec. 2.4

Table 1: Functions that are part of the CONMan architecture

2.2.5 Performance

Unlike the above components, which are quite specific in nature, performance is harder to specify and manipulate. 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 [25] (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). Additional metrics, such as power, can be added as needed.

Modules and pipes report on their performance with these metrics. They can also advertise the ability to 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. The NM, based on some high-level performance goals, can choose from the trade-offs offered by these modules which can then configure and coordinate the low level values. Similarly, modules may also advertise their performance enforcement capabilities, for example their ability to queue or shape packets or their ability to enforce certain service classes. The NM can then use this to satisfy network-wide performance goals. Due to space constraints we do not discuss how performance enforcement in modeled in the CONMan abstraction – [2] details this and describes a real-world performance management example.

2.3 Network Manager (NM)

Using the two-way communications channel provided by 4D between itself and the devices, the NM discovers the physical topology of the network, as well as the actual and potential logical protocol structure. This section describes the set of primitive functions used by the NM and device MAs to configure and debug the network.

Parameter	What is advertised?
Name	<Module-name,**Module-ID**,**Device-ID**>
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
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 2: Module abstraction; ***showPotential*** **() describes each module using this abstraction**

2.3.1 Network Configuration

The following functions capture the NM's interaction with the devices in the network as part of network configuration. Table 1 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 2.

- *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 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. 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, which in turn 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 2.5).

2.3.2 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 [30], 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 [23]. Here, a process is able ask protocol modules if they believe themselves to be healthy.

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. We think that through such a structured debugging approach, the NM can determine the root-cause of most network problems.

2.4 Hiding Complexity

Much of the complexity reduction of CONMan comes from the fact that the NM operates in terms of the abstract components, while the protocol modules themselves translate these into concrete protocol objects.

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). The inspecting module itself is responsible for determining the actual protocol fields. For example, given the high-level specification above, the inspecting 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". 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.

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. To address this, 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.

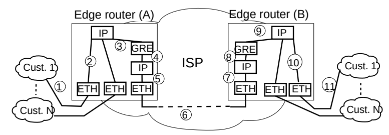
Note that not all detailed protocol values can be or should be determined by the protocols themselves. For instance, it appears difficult to expect IP modules to chat among themselves and assign IP addresses. This is best done by the NM having explicit knowledge of how to assign IP addresses (as DHCP servers do today). Similarly, some filtering instances such as matching on regular expressions in HTML are best left to specialized NMs such as Intrusion Detection Systems. Where to draw the line between what to abstract and what to make explicit is part of the design challenge of CONMan.

Finally, many data-plane protocols rely on externally generated state for their operation. In the Internet, control-plane protocols generate some of this state. 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 above. Instead, control modules 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 is has a dependency on external state (say X) and the LCP module advertises that it can satisfy dependency X. However, this approach does not address the issue of the configuration required by the control modules. It also hinders the ability of the NM to do root-cause analysis since the NM does not understand the operation of the control modules nor does it understand the state generated by them. For example, the NM does not understand BGP and hence, cannot be expected to debug routing flaps and the resulting prefix dampening. Alternatively, CONMan can possibly be used to replace some existing control protocols, such as DHCP and LCP, especially those that operate within a single domain. Indeed, the 4D authors argue that centralized configuration of IP routing tables and filters via the 4D management channel should replace existing decentralized routing protocols like OSPF. A characterization of the scenarios in which existing protocols need to be retained against the ones in which they should be replaced is part of our future work.

2.5 CONMan in action

We now use a GRE-IP tunnel configuration example to elucidate the advantages of CONMan over the status quo. A GRE-IP tunnel is characterized by a source and a destination IP address and a key value - the source and the destination must agree on the key for the tunnel to operate correctly. Hence, configuring a GRE-IP tunnel involves determining the IP addresses of the tunnel end-points, the key values, whether to use sequence numbers (sequence numbers help with in-order delivery of tunneled packets) and other protocol specific details such as tunnel TTL, the TOS field for tunneled packets, whether to use checksums or not, whether to use path-mtu-discovery or not.

In the current set-up, the management plane must specify all these low level configuration details. For example, consider an ISP's router (A) that needs to tunnel all traffic from a customer facing interface to another router (B) through a GRE-IP tunnel (as shown in figure 2). The figure also shows a configuration snippet needed to achieve this on a Linux router "Today". Apart from the configuration being complex, it leaves the door open for many kinds of errors. Some such error possibilities have been marked with labels adjacent to the snippet: (a). not configuring the host as a router, (b). 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, (c). configuring the tunnel end points with the wrong



Configuration at router A "Today"

```
 #!/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\      (c,d)
128.84.222.111 ikey 2001 okey 1001 icsum ocsum iseq oseq
 ifconfig greA 192.168.1.3
 # Enable routing                                               (a)
 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                     (b)
# 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
```

Configuration at router A with "CONMan"

```
create (pipe-2)
create (pipe-3)
create (switch-state,pipe-2,pipe-3)
create (pipe-4)
create (pipe-5)
```

NOTE: create (switch-state,pipe-2,pipe-3) creates IP module switch state to switch from pipe-2 to pipe-3

Figure 2: GRE-IP tunnels for ISPs' customers - the figure shows the module connectivity that the NM aims to achieve. Also shown is a configuration script on a Linux router needed to achieve this "Today" and what the management plane would need to do with "CONMan" in place.

key values and (d). using tunnel end point IP addresses that are wrong or do not have IP connectivity between them.

As a contrast, with CONMan, the NM logic maps the high-level tunneling goal into what it can do – *building pipes and modules*. The NM uses the *showPotential* and *showActual* primitive at A and B to determine the relevant modules and their abstraction. It then maps the aforementioned goal of tunneling packets between customers to the task of building a path between the customer interfaces of devices **A** and **B** that goes through the respective GRE modules. This path labeled as (1) to (11) in figure 2. The NM realizes that pipes (1) and (6) are physical pipes and hence, only needs to build pipes (2) through (5) at router A. It does so by invoking five primitives (shown in the figure) at the appropriate modules of router A. Note that similar invocations are needed at router B. The module abstractions inform the NM of the dependencies that need to be satisfied when invoking these primitives. For example, the GRE abstraction (not shown here in the interest of brevity) states that the creation of an up pipe has a dependency on the peer module. Hence, when the NM creates pipe-3, it needs to inform the GRE module in router A of the peer GRE module in router B. This allows the GRE modules in A and B to use the *conveyMessage* function to communicate and coordinate various protocol specific values. For instance, it is these GRE modules that coordinate the key values to be used by the tunnel and the NM does not have to deal with any such protocol-specific details. The modules similarly coordinate other details like sequence number usage based on other abstract information provided by the NM when invoking these primitives. Hence, while this example leaves out a lot of details, it does capture the essence of how most network management operations would be simplified by having CONMan in place.

3. RELATED WORK

There is a tremendous amount of past work in network management, the most relevant of which we briefly cite here, and none of which focuses on hiding management complexity inside protocol implementations as CONMan does. On the commercial side, SNMPLink [19] lists many existing management tools, from low-end tools like packet analyzers (eg, Ethereal [28]), traffic monitors (eg, MRTG [24]), and SNMP agents (eg, ITM [4]) to high-end managers like OpenView [29]).

A number of proposals ([7][18][14]) are aimed at dealing with the complexity of management interfaces like SNMP and RMON [22], but do not attempt to reduce complexity per se. Policy-based management [9] standardizes a common high-level policy framework for management of QoS ([20, 1]) and security [26], but again does not reduce complexity per se, a fact that has been an impediment of the adoption of this approach [10].

The 4D proposal [8] 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 discovery and dissemination plane. In order to allow for multiple NMs, we have also proposed some extensions to the 4D management channel which are detailed in [2].

Recently, there has been a spurt of research detailing the reasons for outages and anomalies in IP backbones [13, 16], Internet services [17] and BGP routing [6, 15]. 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 impacting data plane operation. Finally, we believe that CONMan can simplify the cross-layer database and interface proposed in [12], and indeed may provide the basis for the Knowledge Plane objectives laid out by Clark et. al. [5].

4. FUTURE WORK

In this paper we have outlined a network architecture called CONMan that is amenable to management. While CONMan shows promise, it is certainly too early for us to claim that the abstraction presented here suffices for all data plane protocols. There are many challenges to overcome, including:

First there is the question of even how to evaluate CONMan, given that network management is a pervasive and human-intensive activity. There are issues of scale and robustness associated with the 4D management channel and centralized NM approach. There are issues of crossing administrative domain and addressing domain (NAT) boundaries. There are issues of dominion where multiple NM's try to manage the same equipment. While we believe that CONMan will allow network managers to get a better handle on network security, the CONMan architecture also introduces new security concerns, where an attacker may get control of the NM. Finally, there are issues of how to deploy CONMan. It is likely to share IPv6's conundrum: namely that complexity has to be increased over the short-term in order to arrive at reduced complexity over the long-term. In spite of this list of issues, we believe that CONMan represents an interesting, novel, and promising approach to network management.

5. REFERENCES

[1] Amiri, K., Calo, S., and Verma, D. Policy based management of content distribution networks. *IEEE Network Magazine* (March 2002).

[2] Ballani, H., and Francis, P. Complexity Oblivious Network Management: A step towards network manageability. Tech. Rep. cul.cis/TR2006-2026, Cornell University, Ithaca, NY, US, 2006.

[3] 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).

[4] Carsten Schmidt. Interface Traffic Monitor Pro. `http://software.ccschmidt.de/`.

[5] 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.

[6] 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).

[7] 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).

[8] 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).

[9] Halpern, J., and Ellesson, E. The IETF Policy Framework Working Group. Online Charter. `http://www.ietf.org/html.charters/OLD/policy-charter.html`.

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

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

[12] 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).

[13] 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).

[14] 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).

[15] Mahajan, R., Wetherall, D., and Anderson, T. Understanding BGP misconfiguration. In *Proc. of ACM SIGCOMM* (2002), pp. 3–16.

[16] Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C., and Diot, C. Characterization of Failures in an IP Backbone. In *Proc. of IEEE INFOCOMM* (2004).

[17] 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).

[18] Pham, V. A., and Karmouch, A. Mobile Software Agents: An Overview. *IEEE/ACM Trans. Netw. 36*, 7 (1998).

[19] Pierrick Simier. SNMPLink. `www.snmplink.org/Tools.html`.

[20] 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).

[21] Schonwalder, J. Characterization of SNMP MIB Modules. In *Proc. of International Symposium on Integrated Network Management* (2005).

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

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

[24] Tobias Oetiker and Dave Rand. MRTG : Multi Router Traffic Grapher. `http://mrtg.hdl.com`.

[25] Uijterwaal, H., and Zekauskas, M. IP Performance Metrics (ippm). Online Charter, Jan 2006. `http://www.ietf.org/html.charters/ippm-charter.html`.

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

[27] 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.

[28] Ethereal : A Network Protocol Analyzer. `www.ethereal.com`.

[29] HP OpenView. `www.openview.hp.com/`.

[30] CISCO GRE Keepalives. , January 2006. `www.cisco.com/en/US/tech/tk827/tk369/technologies_tech_note09186a008040a17c.shtml`.