CONMan: 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. Further, there is a general lack of research on fundamentals and an increasing reliance on temporary “band-aids”. We argue that the difficulty of network management can partly be attributed to a fundamental flaw in the existing architecture: protocols expose all their internal details and hence, the complexity of the ever-evolving data plane encumbers the management plane. Guided by this observation, in this paper we explore an alternative approach and propose Complexity Oblivious Network Management (CONMan), a network architecture in which the management interface of data-plane protocols includes minimal protocol-specific information. This restricts the operational complexity of protocols to their implementation and allows the management plane to achieve high level policies in a structured fashion. Apart from building the CONMan interface of a few protocols and a management tool that can achieve high-level configuration goals based on this interface, our preliminary experience with applying this tool to real world VPN configuration indicates the architecture’s potential to alleviate the difficulty of network management.**

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 [22] 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 have been aggravated by the lack of research on fundamentals. Instead, there is an increasing reliance on temporary “band-aids” that have evolved piecemeal as needed. While this has allowed a number of flaws to creep into the way we manage networks, in this paper we focus on one specific shortcoming:

> *Today, protocols and devices expose their internal details leading to a* **deluge of complexity** *that burdens the management plane.*

For instance, it is not uncommon for a network device to have thousands of manageable objects. A review of SNMP MIB modules found more than 13,000 MIB objects in IETF MIBs alone [37]; MIBDepot [55] lists 6200 MIBs 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 [43]. Encumbering the management plane with all this complexity leads to these problems:

- *Perception differs from reality*. Management applications need to effectively reverse engineer the capabilities and the functionality of protocols and devices from their detailed MIBs. The low-level and non-intuitive nature of these parameters makes this task difficult, if not impossible [32].
- *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 [25].
- *Fragmentation of tools*. Since devices and their exposed details keep evolving at a frantic pace, management applications tend to lag behind the power curve [30]. 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 [34] 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 is highly inter-dependent. These dependencies are not reflected in the existing set-up; thus, when a low-level value changes, the appropriate dependent changes don’t always happen [32]. Instances of improper filtering because the

address assigned to some machine changed, or the application was started on some other port are very common. Recent work details the challenges involved in tracking such dependencies in the existing set-up [13] and gives examples of how failure to track them leads to problems in large networks [24].

These shortcomings indicate that an (extreme) alternative worth exploring is to confine the operational complexity of protocols to their implementation. As a matter of fact, we observe that almost all data-plane protocols share some very basic characteristics that should, in theory, suffice for the management of the network. Guided by this observation, we adopt a more modest approach and argue that:

> *The management interface of data-plane protocols should contain as little protocol-specific information as possible.*

This allows all data-plane protocols to have a generic yet simple management interface. In this paper we present the design and implementation of a network architecture, *Complexity Oblivious Network Management* (CONMan), based on this principle. In CONMan, 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. To this effect, this paper makes the following *contributions*:

- We present the detailed design of a network architecture that minimizes the protocol-specific information in the management interface of data-plane protocols. We also present protocol-independent configuration primitives that can be used to interact with this interface and hence, manage the network.
- We describe the implementation of the management interface of a few protocols in compliance with the proposed architecture.
- We detail the implementation of a management application that, given the abstraction of the protocols and devices in the network, can achieve high-level configuration goals using the aforementioned primitives.
- The paper presents the use of CONMan in a real-world configuration scenario (VPN configuration) to highlight its advantages over the status quo. Further, we also use a naive but hopefully informative metric to compare the protocol agnosticity of CONMan configurations against today's configurations in three different scenarios (GRE tunnels, MPLS LSPs and VLANs).

Note that CONMan doesn't reduce the total system complexity; it only attempts to correct the skewed division of functionality between management done inside the managed device and that done outside the managed device. While the fact that management applications don't have to deal with myriad protocol details reduces their burden, protocols still need various low-level details in order to operate. With CONMan, it is the protocol implementation that uses the high-level primitives invoked by the management applications and out-of-band communication with other protocols to determine these. This, in effect, puts the responsibility for detailed understanding of protocol operation on the protocol implementor. Since the protocol implementer requires this knowledge in any event, this seems to be a smarter placement of functionality.

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 need conform to our proposal. Thus, while the management interface gives the appearance of protocol modularity, the protocol implementation itself may be modular or monolithic. In other words, a *non contribution* of this paper is the notion of implementation modularity.

Finally, we believe that the approach presented in this paper has value for all aspects of network management. While we briefly comment on some of these such as the ease of tracking dependencies and debugging errors with CONMan in place, the primary focus of this paper is basic configuration. Further, while our implementation and evaluation efforts serve as a sanity check for the proposal, they represent merely a first stab at an alternative approach towards network management. However, the fallacies of the existing architecture and the importance of alleviating them to improve network manageability are not disputable. In this context, we hope that our proposal would stimulate discussion about structured management of networks and hence, serve as a step towards the holy grail of *self managing networks*.

II. CONMan Architecture

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 globally unique, topology independent identifier (*device-id*) that can carry cryptographic meaning (for example, by hashing a public key). 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>.

A. *Management Channel*

As mentioned in section I, the piecemeal evolution of management approaches has resulted in many flaws in the way we manage networks. One such flaw is that the existing *management plane depends on the data plane* [7,17]. 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. In recent work, Greenberg et. al. [17] discussed the implications of this dependency loop and proposed a technique for achieving a self-bootstrapping, operationally independent management plane. While such management plane independence can be established using a few other approaches (for instance, a more generalized and self-bootstrapping version of the separate management network that is used by some large ISPs), we agree with their basic hypothesis and in this paper assume the presence of a *management channel*. This management channel should be independent of the data-plane, should not require any pre-configuration and should allow devices in the network to communicate with the NM. However, we do not dictate if the management channel operates or does not operate over the same physical links as used by the data-plane. Some implications of keeping this channel configuration-free and other related issues are detailed in section V.

B. *Overview*

Our approach derives from two key observations: First, the main purpose of a network is to provide paths between certain applications on certain hosts while preventing certain other applications and hosts from using those paths.[1] Second, we observe that most data-plane protocols have some basic characteristics whose knowledge should suffice for managing the aforementioned paths and hence, for network management in general. For instance, most protocols have the ability to connect to certain other protocols, to switch packets, filter packets, queue packets and so on. We believe that it is these basic characteristics that should serve as the narrow waist for Internet's management plane. Consequently, the management plane only maps the high-level communication goal into the path through the network (i.e. which protocols should connected and how) and the protocols themselves figure out the low-level parameters that they need to operate.

In our proposal, we try to capture these basic characteristics using a generic abstraction called the *Module Abstraction* – all protocol modules in CONMan self-describe themselves using this abstraction. To this effect, 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 (figure 1). Thus, the

[1]Of course, this is a simplification since the paths must perform adequately, have certain security properties, etc. but the basic argument still applies.

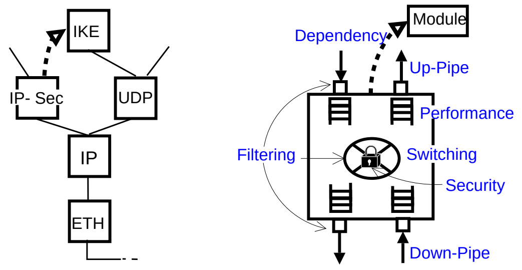


Fig. 1. Modules, pipes, and dependencies form a graph that describes the operation of a device (in particular) and the network (in general). The figure on the right denotes the major components of the *module abstraction*. Note that some modules may not require all elements of the abstraction to describe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

abstraction describes what the protocol is capable of (*potential*) and what it depends on (*dependencies*). Further, the module can be configured to operate in a certain fashion (*actual*) by manipulating its abstraction using the *CONMan primitives*. Such modeling of protocols using a generic abstraction decouples the data and the management plane so that they can evolve independently of each other.

Each device in the network uses the management channel to inform the NM of its physical connectivity, all modules that it contains, and their respective module abstractions. 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. 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.

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 operate. The NM can then use the management channel to invoke the appropriate CONMan primitives and configure the modules accordingly. Thus, the NM can configure the entire network from the ground up with (almost) no protocol-specific knowledge. We believe that such as approach would ameliorate a lot of the problems afflicting network management today.

C. *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, 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 data-modules self-describe themselves using the abstraction shown on the right in figure 1. Below we briefly comment on the components of this 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* pipes 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 pipes through which packets travel between the modules. Of these, the NM can create up-down pipes. It cannot create physical pipes, but can discover and enable them. Also, pipes have identifiers which the NM can use to refer to them. Applications can also lead to creation of pipes. For example, a HTTP-client initiating a TCP connection may lead to up-down pipes between the following module pairs: {HTTP-client, TCP}, {TCP, IP}, and {IP, ETH}. The NM can dynamically discover and potentially disallow such pipes.

Modules are 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, each pipe is associated with one or more *peers modules*. For example, the peer module for a down-pipe of a TCP module would be the remote TCP module to which the down-pipe ultimately leads to. Also, each module is associated with a set of *peerable-modules*. For example, the peerable-modules for a TCP module are {TCP} while the peerable-modules for a HTTP-server module are {HTTP-client}.

In effect, the notion of pipes abstracts away the details that protocols need for basic operation. Given a connectivity goal, the NM simply builds the corresponding path by creating pipes while the modules determine the low-level parameters. For instance, creating a down pipe from an IP module to an ETH module might be a part of establishing IP connectivity between two hosts and may cause the IP module to communicate with its peer IP module through the management channel to exchange the MAC address of the ETH module below it.[2] Apart from communication with peer modules, modules may need more help in determining the low-level parameters – they express these as *dependencies* that need to be satisfied before the pipe can be created.

2) Switch: Switches capture the ability of modules to pass packets between up, down and physical 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 $\Rightarrow$ phy], [phy $\Rightarrow$ up] and [phy $\Rightarrow$ phy] switching (eg. Ethernet module). A module advertises its switching capabilities. The NM uses this and the information about the connectable-modules of each module to build a *potential connectivity* graph for the network. As we show in section III-C, this allows the NM to determine what paths are and are not possible. For instance, the ETH module in a Layer-2 switching device advertises that it can do [phy $\Rightarrow$ phy] switching and so, can be used by itself along a path between two devices that the NM is trying to connect. As a contrast, the ETH module in a router would not have [phy $\Rightarrow$ phy] switching capability and so, the NM must use it in conjunction with the IP module on the router.

When incorporating a module as part of a path, the NM must direct the module as to how packets must be switched between the pipes just created – this is the *actual* switch configuration. Of course, it is not necessary that there be a one-to-one mapping between the pipes. Instead, incoming packets on a pipe may be switched to one of many other pipes and hence, switches may have state which conditions how packets are switched. This switching state can be determined by the module through interaction with its peer module. For instance, the NM, as part of establishing an IP-IP tunnel, may direct an IP module to switch packets between up-pipe P1 (to another IP module) and down-pipe P2 (to the underlying ETH module). The creation of pipe P1 and P2 and the actual switch rule causes the three modules to interact with their peers and determine the parameters needed for a low-level routing rule such as `ip route to 204.9.169.1 dev eth1 nexthop 204.9.168.1`. Alternatively, it also possible that the switching state is generated by control protocols and this is exposed as part of the module abstraction. Section II-F discusses these alternatives.

3) Filters: The filter abstraction allows modules to describe whether and how they can filter packets. Filter rules are described in terms of other abstracted components: pipes, devices, modules or even module types. 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 (such as addresses and port numbers). This process and other related issues are detailed in section II-E.

4) Performance: Unlike the components above, 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

[2]Note that ARP achieves this in the existing set-up and even with CONMan, the IP module could just as well rely on ARP for the peer's MAC address.

Name	Caller	Callee	Description
showPotential	NM	MA of device	Sec. II-D.1
showActual	NM	MA of device	Sec. II-D.1
create, delete	NM	MA of device	Sec. II-D.1
conveyMessage	Module (Source)	Module (Destination)	Sec. II-D.1
listFieldsAndValues	Module (Inspecting)	Module (Target)	Sec. II-E

TABLE I
FUNCTIONS THAT ARE PART OF THE CONMAN ARCHITECTURE

encompass most of the IP performance metrics proposed by IETF [40]; 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 in terms of these metrics. 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. Instead of exposing the low-level options and 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.

Similarly, modules may also advertise their explicit 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, this paper does not detail the modeling of performance trade-offs and enforcement in CONMan– we refer the interested reader to [4].

5) 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's dependency on IKE). However, security is another aspect that is not discussed in the paper.

*D. Network Manager (***NM***)*

The management channel allows devices in the network to communicate with the NM. Each device uses this to inform the NM of its physical connectivity, thus allowing the NM to determine the network topology. Beyond this, the NM must configure the network and debug network problems when they do arise.

1) Network Configuration: Given the network *potential*, the NM can achieve high level network configuration goals

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 pipes	Information about the physical 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; Is the switch state that governs the switching 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 (not explained)	Apart from the classification based on which perf. can be enforced, the module advertises one of these: (1) Queuing and Shaping capabilities (2) Service classes on offer
Security (not explained)	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

simply by creating and deleting pipes and module components. The following primitives capture the NM's interaction with the devices in the network as part of network configuration. Table I shows these and other CONMan primitives offered by the NM and the modules themselves.[3]

(a). *showPotential ()* allows the NM to determine a device's capabilities. The device returns a list of modules with their abstractions. The type of information returned for each module is shown in table II.
(b). *showActual ()* allows the NM to determine the state of modules in a device. The state of each module includes state for all the pipes, the switch, filters, 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.
(c). *create ()* and *delete ()* allow the NM to create and delete pipes, filter-rules, switch-rules and performance enforcement state (queuing structures or service classes). The *showPotential ()* function provides the NM with all the information it needs to create and delete components.

The NM needs no protocol specific knowledge to use these primitives. For instance, it can create up-down pipes simply by satisfying their dependencies and invoking the *create* function. For instance, consider a NM creating a pipe between an IP module and an underlying GRE module. In terms of today's configuration, this amounts to creating a new GRE tunnel which requires a number of low-level

[3]We do not give details of the CONMan API. However, we do show the use of these primitives in section III.

parameters to be specified. With CONMan, it is the GRE module that coordinates these parameters with its peer GRE module. For instance, the modules may exchange the tunnel key values to be used, so the NM does not need to know the notion of keys. Since the management channel allows the modules to communicate *only* with the NM, the NM provides:
(d). *conveyMessage ()* allows modules to convey messages to each other through the NM (see detailed example in section III-B).

2) Debugging: In CONMan, modules can check protocol parameters with their peers and may actually send packets over the data plane to detect and localize faults. Such self-testing may be periodic, event-driven or may be invoked by the NM. Note that this is not novel per se; layer-1 and layer-2 modules in optical networks already have such functionality [20]. Further, Microsoft is currently working on adding a similar functionality to help users debug the Windows network stack [38]. The self-testing ability of protocol modules can simplify debugging since the NM can debug problems in communication between two application modules by tracing and testing the sequence of modules and pipes between them. However, our focus on basic configuration implies that the CONMan debugging primitives are not described here.

E. Hiding Complexity

Much of the reduction in management plane complexity 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 - “drop packets from module <IP,B,y> and going to <FOO,C,z>” (where FOO is an application module with up-down pipes to TCP). The protocol 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 “drop 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 dependent state in different modules when some low-level value in a given module changes. To ensure this, the NM tracks the dependencies between component identifiers (that have been resolved) and opaque 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, 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 [15]. This is best done by the NM having explicit knowledge of how to assign IP addresses (as DHCP servers do today). Similarly, tasks like regular expression matching in HTML do not seem amenable to abstraction and should be done by specialized NMs such as Intrusion Detection Systems. Further, there are cases such as P2P protocols where protocol designers don’t want to provide the protocol values since they don’t want to be filtered. Thus, there are scenarios where the NM will have to deal with protocol-specific details.

F. Control Modules

Many data-plane protocols rely on externally generated state for their operation. Today, this may be provided manually as part of the protocol configuration. Alternatively, control-plane protocols can 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.

In CONMan, data modules can generate this state by interacting with their peer modules based on the create/delete primitives invoked by the NM. While this follows from the general CONMan philosophy, there are cases where such an approach poses challenges regarding the scalability, robustness and responsiveness of the network.

Alternatively, even in CONMan, we may rely on control protocols for the low-level state. However, 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 relying on control modules 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.

One way to address some of these problems is to let the NM perform the function of the control protocols whereby it uses 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 [17] 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 state should be generated by the protocols themselves against the ones in which existing control protocols should be used against the ones in which the control protocols should be replaced is an important research question. However, in order to explore the limits of our proposal (i.e. what can be captured and what cannot be captured), this paper (rather naively) ignores the existence of control protocols. Instead, our implementation involves the protocol modules generating the low-level details and the implications of this are discussed in section III-C.3.

III. Implementation

CONMan does not necessitate any changes to the way data plane protocols operate. The modeling of protocols as modules can be implemented as wrappers around existing implementations. We have implemented four protocol modules (GRE, MPLS, IP, ETH) as user-level wrappers around the corresponding existing protocol implementation in Linux (kernel 2.6.14). In the first part of this section (section III-B), we use the establishment of GRE tunnels as an example to detail our GRE module implementation. We also implemented a NM that understands the CONMan abstraction and implements the CONMan NM primitives. To illustrate the operation of CONMan in a real-world management scenario, we used this NM for configuring provider-provisioned Virtual Private Networks (VPNs). We describe the NM implementation and compare CONMan VPN configurations with configurations today in the second part of this section (section III-C).

A. Management Channel

The testbed used for the examples described below comprised of Linux-based PCs operating as end-hosts and routers with Ethernet as the connecting medium. All the PCs were equipped with a separate management NIC and connected to a separate network that served as the management channel for our experiments. Communication between the protocol modules and the NM was done through UDP-IP over this management channel. Note that this is not ideal since the management channel had to be pre-configured. However, we also implemented a straw-man version of a management channel that can operate on the same underlying physical network used by the data plane using the techniques proposed by the 4D project [17] for their discovery/dissemination plane. Here, the protocol modules and the NM 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. Most importantly, no pre-configuration is needed for such a management channel.

	Parameter	Value
i	Name	<GRE,device-id,module-id>
ii	Up.Con-Modules (Connectable-Modules)	IPv4
iii	Up.Dependencies	Performance Trade-offs to be specified
iv	Down.Con-Modules	IPv4
v	Down.Dependencies	None
vi	Physical pipes	None
vii	Peerable-Mod.	GRE
viii	Filter	Nil
ix	Switch	[Up $\Rightarrow$ Down],[Down $\Rightarrow$ Up]
x	Perf Reporting	Number of recieved and transmitted packets on each up and down pipe
xi	Perf Trade-Offs	{[Jitter, Delay] Vs [In-order delivery] | Up-pipe} {[Loss-Rate] Vs [Error-Rate] | Up-pipe}
xii	Perf Enforcement	Nil
xiii	Security	Nil

TABLE III
Abstraction exposed by our GRE implementation

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 [52]) and traffic engineering [53]. 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 [58]). 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 [12] is looking at exactly such tunnel configuration problems.

Here, we use the GRE protocol to elucidate the establishment of tunnels in 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*). We focus on GRE with IPv4 as the underlying delivery protocol - $GRE\text{-}IP$. Consequently, each tunnel is characterized by a source and a destination IP address. Besides this, GRE also involves key'ing of tunnels - the source and the destination must agree on the key for the tunnel to operate correctly. Configuring such a $GRE\text{-}IP$ tunnel today requires the management plane to provide the IP addresses of the tunnel end-points, the key values, whether to use sequence numbers or not (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, and whether to use path-mtu-discovery or not.

We have implemented a GRE module conforming to the CONMan architecture. As mentioned earlier, our implementation is based on the Linux GRE kernel module with a user-level wrapper that confines the protocol-specific details to the implementation and exposes a generic abstraction to the NM. This abstraction is shown in table III and some of the entries are explained below:
ii). Ideally, GRE can carry any payload protocol and hence,

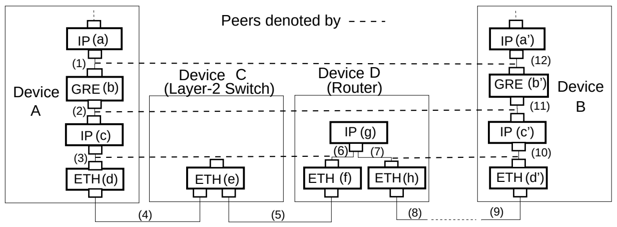


Fig. 2. GRE-IP tunnel between devices **A** and **B** - the NM needs to build the path labeled from (1) to (12).

there should not be any restriction on the modules that the GRE module can connect 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.
iii). To create an up pipe, the NM needs to specify the performance trade-offs (see k below) that apply to pipe.
iv,v). The module is restricted to having IPv4 as the tunneling protocol with no explicit dependencies.
ix). The module can switch packets between an up-pipe and a down-pipe. The switching state is generated by the module on its own.
x). The underlying Linux implementation provides limited performance reporting: the number of packets transmitted and received on each up and down pipe.
xi). The module offers the following trade-offs: For a given 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 GRE 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.

To illustrate the relevant implementation details, we now describe how a NM can use CONMan primitives to achieve the following **low-level goal**:

Configure the path between the IP modules <IP,A,a> and <IP,B,a'> labeled as (1) through (12) in figure 2.

This is equivalent to creating a GRE-IP tunnel between devices **A** and **B** in the existing set-up. **In CONMan, the human manager is not aware of such low-level goals or the notion of pipes and switches or the CONMan script shown below**. Instead, he/she specifies a high-level goal that the NM maps to such a low-level goal. The next section describes an example high-level goal and our implementation of the mapping process. This mapping process informs the NM which modules along the path are peers of each other – in figure 2, the dashed line between pipes labelled (1) and (12) indicates that modules *a* and *a'* are peer modules for these pipes (as are modules *b* and *b'*). We also assume that the NM has, as part of the mapping process, invoked the *showPotential* primitive at these devices and hence, is aware of the CONMan abstraction for all the modules involved; for instance, table III shows the abstraction exposed by the module <GRE-IP,A,b> (or *b* for short). The other modules have similar abstractions that are not shown here. This equips the NM with all the information it needs to

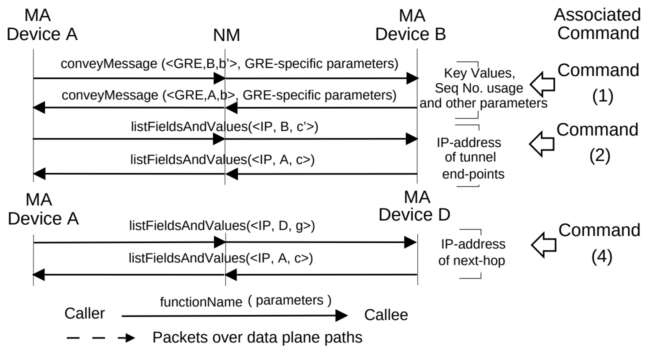

MA Device A
NM
MA Device B
Associated Command
conveyMessage (<GRE,B,b'>, GRE-specific parameters)
conveyMessage (<GRE,A,b>, GRE-specific parameters)
Key Values, Seq No. usage and other parameters
Command (1)
listFieldsAndValues(<IP, B, c'>)
listFieldsAndValues(<IP, A, c>)
IP-address of tunnel end-points
Command (2)
MA Device A
MA Device D
listFieldsAndValues(<IP, D, g>)
listFieldsAndValues(<IP, A, c>)
IP-address of next-hop
Command (4)
Caller
functionName (parameters)
Callee
Packets over data plane paths

Fig. 3. GRE-IP Tunnel establishment between devices **A** and **B** - the management plane is simplified by ensuring that protocol complexity is restricted to protocol implementation.

create the appropriate pipes and switch state. As a contrast, some manual must be read (either by the implementor of the management application or the system administrator) to gain the equivalent knowledge while configuring $GRE\text{-}IP$ tunnels today.

With this information at hand, the NM can build the segment of the path in device **A** (i.e. $a \Rightarrow b \Rightarrow c \Rightarrow d$) using the following script. Note that a similar script needs to be invoked to build the rest of the path.

```
(1). P1 = create (pipe, <IP,A,a>, <GRE,A,b>,
          <IP,B,a'>, <GRE,B,b'>,
          trade-off: order delivery,
          trade-off: error-rate)
(2). P2 = create (pipe, <GRE,A,b>, <IP,A,c>,
          <GRE,B,b'>, <IP,B,c'>, None)
(3). create (switch, <GRE,A,b>, P1, P2)
(4). P3 = create (pipe, <IP,A,c>, <ETH,A,d>,
          <IP,D,g>, <ETH,D,f>, None)
(5). create (switch, <IP,A,c>, P2, P3)
(6). create (switch, <ETH,A,d>, P3, P4)
```

In the script, command (1) creates pipe P1 between the IP module *a* and the underlying GRE module *b*. The fourth and fifth arguments in the command specify the peer IP (*a'*) and GRE (*b'*) modules *for the pipe being created*. Further, the NM satisfies the dependency for creating an up pipe for a GRE module by specifying that it desires in-order delivery of packets and low error-rate. These choices would be based on high-level performance goals specified by the human manager. Similarly, commands (2) and (4) create pipes P2 and P3. Through command (3) the NM specifies that GRE module *b* should switch between pipes P1 and P2. Similarly, commands (5) and (6) configure the switch in modules *c* and *d* respectively.

The simple and structured process described above is all the configuration that the NM needs to do. It is the protocols that incorporate the complexity of determining the low-level parameters. Each module, based on the commands invoked by the NM, interacts with its peer module through the management channel to determine the required protocol specific parameters – this process is briefly described below and illustrated in figure 3.

On invocation of command (1) and the corresponding command on device **B**, modules *b* and *b'* use the *conveyMessage* primitive to exchange the GRE-specific

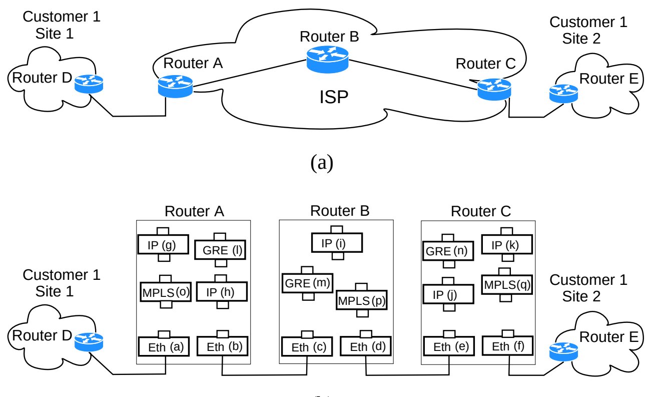


Fig. 4. Experimental set-up emulating an ISP and customer sites for the VPN configuration. Figure 4(b) shows the network map and the modules seen by the NM prior to configuration.

parameters needed for connectivity between them. These include the GRE key values in each direction, the use of sequence numbers, etc. Note that some of these parameters are based on the trade-off decisions specified by the NM. For example, the NM, as part of command (1), opts for in-order delivery. This causes modules *b* and *b'* to negotiate the use of sequence numbers for the GRE tunnel between them. Similarly, on invocation of command (2), IP modules *c* and *c'* figure out the IP addresses of the tunnel end-points by determining each other's IP address through the use of *listFieldsAndValues*. Command (3) causes the GRE module *b* to generate the actual Linux command to configure the GRE tunnel, the parameters for this command already having been determined:

```
ip tunnel add name gre-P1-P2 mode gre remote
204.9.169.1 local 204.9.168.1 ikey 1001 okey 2001
icsum ocsum iseq oseq
```

Similarly, command (4) causes IP modules *c* and *g* to exchange their IP addresses while command (5) causes IP module *c* to generate the low-level routing rule in device **A** such that packets to device **B** are routed through **D**:

```
ip route add to 204.9.169.1 via 204.9.168.2 dev
eth2
```

Note that our implementation of the IP module relies on ARP for IP-to-MAC mapping and this is exposed in its abstraction. Alternatively, it is possible to imagine command (4) causing ETH modules *d* and *f* to exchange their MAC addresses.

C. Virtual Private Networks (VPNs)

VPNs are commonly used to connect geographically distributed enterprise sites across the Internet while offering security and performance comparable to connecting the sites across a dedicated network. As the name suggests, a "provider-provisioned VPN" involves the ISP that provides connectivity to the enterprise sites configuring and maintaining the VPN [3]. We implemented a NM that can be used for such VPN configuration. In the interest of brevity, the discussion below focusses on the configuration sub-task

Module	Connectivity and Switching
<ETH,A,a>	Up: {IP, MPLS}, Down: None, Phy: to C1-S1, Switching: [Phy ⇒ Up],[Up ⇒ Phy]
<ETH,A,b>	Up: {IP, MPLS}, Down: None, Phy: to <ETH,B,c>, Switching: [Phy ⇒ Up],[Up ⇒ Phy]
<MPLS,A,o>	Up: {IP}, Down: {ETH}, Phy: None, Switching: [Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down]
<IP,A,g>	Up: {IP, GRE}, Down: {IP, GRE, MPLS, ETH}, Phy: None, Switching:[Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down],[Up ⇒ Up]
<IP,A,h>	Up: {IP, GRE}, Down: {IP, GRE, MPLS, ETH}, Phy: None, Switching:[Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down],[Up ⇒ Up]
<GRE,A,l>	Up: {IP}, Down: {IP}, Phy: None, Switching: [Down ⇒ Up],[Up ⇒ Down]

TABLE IV
CONNECTIVITY AND SWITCHING CAPABILITIES OF THE MODULES IN DEVICE A.

of an ISP trying to ensure that traffic between two sites *S1* and *S2* of a customer *C1* is isolated from other traffic. A complete VPN configuration involves doing the same for all pairs of sites of each customer needing VPN support. Figure 4(a) shows the relevant part of the set-up in our lab with five Linux hosts (labeled *A*-*E*) serving as two of the ISP's edge routers in different POPs (*A* and *C*), the ISP's core router (*B*) and customer C1's routers at site S1 (*D*) and site S2 (*E*). Specifically, the NM aims to achieve the following **high-level goal specified by the human network manager**:

> *Configure connectivity between sites S1 and S2 of customer C1.*

This is equivalent to the following high-level goal in CONMan terminology:

> *Configure connectivity between the customer-facing interfaces <ETH,A,a> and <ETH,C,f> (see figure 4(b)) for traffic between C1-S1 and C1-S2.*

Ideally, C1-S1 and C1-S2 should be high-level identifiers that get mapped to the IP prefixes for the two sites through communication between the NM of the ISP and the NM of customer C1. However, this paper is restricted to management in a single domain and hence, we provide the NM with this information. Further, we assume that the NM has already assigned IP addresses to the IP modules and is aware of IP address domains. As explained later in the section, this knowledge is used by the NM in both the way it finds paths through the network and the CONMan commands it generates to configure these paths. We admit that this is case of NM using protocol-specific information. As mentioned earlier, while it is possible to abstract IP addresses, their ubiquity and scarcity combined with the impact of address assignment on routing scalability suggests that it makes engineering sense to let the NM be aware of them and this is what we chose for our implementation.

1) NM Implementation: All devices in the ISP's network (routers *A*, *B*, *C*) inform the NM of their physical connectivity through the management channel. Given the aforementioned goal, our NM implementation invokes *showPotential* at these devices to determine the abstraction for the modules in these devices. Thus, the NM has a network map akin to the one shown in figure 4(b). This also provides the NM with information about how the modules can be connected

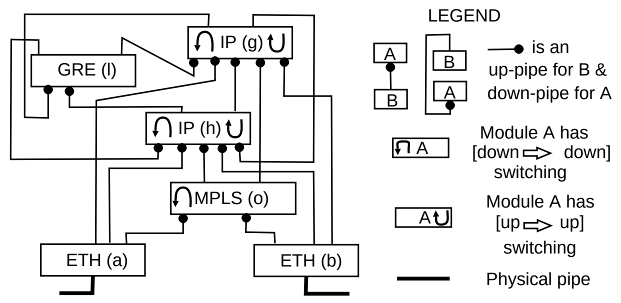

LEGEND
GRE (l)
IP (g)
IP (h)
MPLS (o)
ETH (a)
ETH (b)
is an up-pipe for B & down-pipe for A
Module A has [down ⇒ down] switching
Module A has [up ⇒ up] switching
Physical pipe

Fig. 5. Potential Connectivity sub-graph for device A.

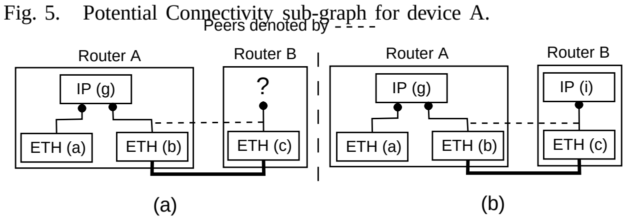


Fig. 6. Options explored by the NM's path finder.

to each other and how they can switch packets (shown in table IV). Based on this, the NM constructs a graph of *potential connectivity* with modules as "nodes" and up-down and physical pipes as "edges". Figure 5 shows the device **A** part of this graph.

The NM also includes a *path-finder* component that can find all paths between any two modules in such a graph. To do so, the component traverses the graph in a depth-first fashion while avoiding cycles. Further, we made two modifications to the traversal: First, the NM knows that a module encapsulates packets in a protocol header when using [up ⇒ down] and [up ⇒ phy] switching; for example an ETH module adds an Ethernet header to packets that it sends out onto a physical pipe. Similarly, a module decapsulates packets when using [down ⇒ up] and [phy ⇒ up] switching. A module processes the packet header but doesn't remove or add headers when using [phy ⇒ phy], [down ⇒ down] and [up ⇒ up] switching. The traversal keeps track of such encapsulation and decapsulation by the modules along the path and hence, restricts itself to paths that are "sane" in the protocol sense. For instance, assuming that the path shown in figure 6(a) is the path already traversed, this rule implies that the next module should be able to decapsulate or process an IP header and hence, the only possible next module is the IP module in device B, <IP,B,i>. This also allows the NM to determine modules that are peers of each other; in the path above, <ETH,B,c> decapsulates the encapsulation put in by <ETH,A,b> and hence, they are peers.

Second, the NM knows the address domains various IP modules belong to. For instance, <IP,A,g> is assigned an address from C1-S1 domain (because <IP,A,g> represents the virtual router connected to C1-S1) while <IP,B,i> is assigned an address from the ISP's domain. The traversal uses this information to rule out invalid paths. For instance, the path shown in figure 6(b) is an invalid path as it makes IP modules *g* and *i* peers even though they are in different address domains.

For the given goal, the NM directs the path-finder to find paths between modules <ETH,A,a> and <ETH,C,f>. We were expecting the NM to generate the following three paths (we only show the module-id for each module along the path):

(a). Using IP-IP tunnel: *a, g, h, b, c, i, d, e, j, k, f.*
(b). Using GRE-IP tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f.*
(c). Using MPLS: *a, g, o, b, c, p, d, e, q, k, f.*

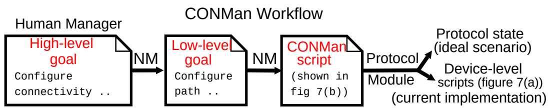
However, the NM generated six more paths: IP-IP over MPLS, GRE-IP over MPLS, IP-IP over MPLS only between A and B, IP-IP over MPLS only between B and C, GRE-IP over MPLS only between A and B, and GRE-IP over MPLS only between B and C.[4] While this suggests that we should use more aggressive pruning rules for our traversal, it also shows that the NM can determine the various ways of achieving a high-level goal given the capabilities of the devices in the network. As a contrast, today it is the human managing the network that relies on RFCs and device manuals to determine the options available. Obviously, such an exercise can be misleading (as was the case when we manually determined the paths for the example network).

The NM now needs to be able to choose amongst the paths based on high-level directives and/or other metrics. We implemented a very simple algorithm that minimizes the total number of pipes instantiated in the routers. This is, in some sense, akin to minimizing the amount of state on the routers and the communication overhead on the NM. For the scenario in question, the MPLS-based path and the IP-IP tunnel are the best options (our NM implementation prefers the MPLS-based path because the MPLS abstraction mentions that it offers good forwarding bandwidth). We can also think of more sophisticated metrics such as the performance capabilities of the modules along the path or satisfying security constraints. Moreover, while the ability to choose amongst possible configurations without protocol-specific knowledge is critical to the CONMan argument, this is an area that we haven't explored in any detail.

Once a path is chosen, the NM then automatically generates the script of CONMan primitives needed to create the path. This translation of a low-level path creation goal to CONMan primitives was explained in the previous section while the complete workflow is shown in the top part of figure 7(a).

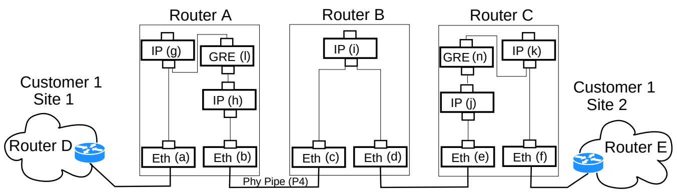
2) Comparing to the status quo: For each path in the example above, we directed the NM to generate the CONMan primitives needed to create the path. These primitives were invoked at the modules in the devices (routers A, B and C) to configure them. Since the modules are implemented as wrappers around existing protocol implementations, they in turn generate the device-level scripts from the CONMan

[4]Typically, ISPs use MPLS-over-MPLS [36] or MPLS-over-GRE [44] for VPN support. Both these configurations are not supported by the Linux hosts used for our experiments and hence, the NM cannot propose these paths.



```
#!/bin/bash
# Insert the GRE-IP kernel module
(1) insmod /lib/modules/2.6.14-2/ip_gre.ko
# Create the GRE tunnel with the appropriate key
(2) ip  tunnel add name greA mode gre remote 204.9.169.1
local 204.9.168.1 ikey 1001 okey 2001 icsum ocsum
iseq oseq
(3) ifconfig greA 192.168.3.1
# Enable Routing
(4) echo 1 > /proc/sys/net/ipv4/ip_forward
# Create IP routing from customer to tunnel
(5) echo 202 tun-1-2 >> /etc/iproute2/rt_tables
(6) ip rule add to 10.0.2.0/24 table tun-1-2
(7) ip route add default dev greA table tun-1-2
# Create IP routing from tunnel to customer
(8) echo 203 tun-2-1 >> /etc/iproute2/rt_tables
(9) ip rule add iff greA table tun-2-1
(10) ip route add default dev eth1 table tun-2-1

(11) ip route add to 204.9.169.1 via 204.9.168.2 dev

eth2
```

(a) Configuration “Today”



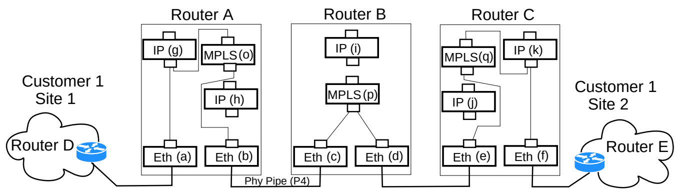
```
(1). P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
(2). P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>,
<GRE,C,n>, trade-off: in-order delivery, trade-off: error-rate)
(3) create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
(4) create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
(5). P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>,
<IP,C,j>, None)
(6). create (switch, <GRE,A,l>, P1, P2)
(7). P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>,
<ETH,B,c>, None)
(8). create (switch, <IP,A,h>, P2, P3)

(9). create (switch, <ETH,A,b>, P3,P4)
```

(b) CONMan configuration (**algorithmically generated by the automated NM**)

Fig. 7. VPN connectivity between sites S1 and S2 of customer C1 through a GRE-IP tunnel between routers A and C.

```
#!/bin/bash
# Instantiating MPLS kernel modules
modprobe mpls
modprobe mpls4
# MPLS LSP for traffic from S2->S1
mpls labelspace set dev eth2 labelspace 0
mpls ilm add label gen 10001 labelspace 0
KEY-S2-S1=‘mpls   nhlfe add key 0 mtu 1500 instructions
nexthop eth1 ipv4 192.168.0.1 | grep key | cut -c
17-26‘
mpls      xc add ilm_label gen 10001 ilm_labelspace 0
nhlfe_key $KEY-S2-S1
# MPLS LSP for traffic from S1->S2
KEY-S1-S2=‘mpls nhlfe add key 0 mtu 1500
instructions push gen 2001 nexthop eth2 ipv4
204.9.168.2 | grep key | cut -c 17-26‘
echo 1> /proc/sys/net/ipv4/ip_forward
ip          route add 10.0.2.0/24 via 204.9.168.2 mpls
$KEY-S1-S2
```

(a) Configuration “Today”



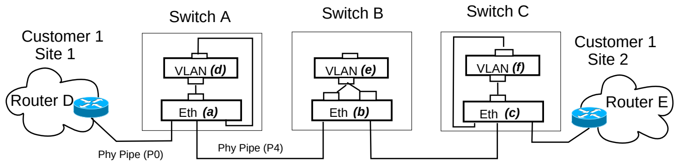
```
  P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
  P1 = create (pipe, <IP,A,g>, <MPLS,A,o>, <IP,C,k>,
<MPLS,C,q>, None)
  create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
  create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
  P2 = create (pipe, <MPLS,A,o>, <ETH,A,b>, <MPLS,B,p>,
<ETH,B,c>, None)
  create (switch, <MPLS,A,o>, P1, P2)

  create (switch, <ETH,A,b>, P2, P4)
```

(b) CONMan configuration (**algorithmically generated by the automated NM**)

Fig. 8. VPN connectivity between sites S1 and S2 of customer C1 using a MPLS LSP through router A, B and C.

```
# put module0 port 9 into VLAN22
# ensure MTU is set properly
set vlan 22 name C1 mtu 1504
set vlan 22 gigabitethernet0/9
# ensure module 0 port 7 is access port
interface gigabitethernet0/7
switchport access vlan 22
switchport mode dot1q-tunnel
exit
vlan dot1q tag native

end
```

(a) Configuration “Today” on Cisco CatOS



```
  P1 = create (pipe, <ETH,A,a>, <VLAN,A,d>, <ETH,C,c>,
<VLAN,C,f>)
  P2 = create (pipe, <VLAN,A,d>, <ETH,A,a>, <VLAN,B,e>,
<ETH,B,b>)
  create (switch, <ETH,A,a>, [P0, Tagged ⇒ P1])
  create (switch, <ETH,A,a>, [P1 ⇒ P0])
  create (switch, <VLAN,A,d>, P1, P2)

  create (switch, <ETH,A,a>, P2, P4)
```

(b) CONMan configuration (**algorithmically generated by the automated NM**)

Fig. 9. VPN connectivity between sites S1 and S2 of customer C1 through VLAN tunneling between switches A and C.

primitives. It is the management plane that needs to generate these device-level scripts with today's setup. Below we compare the configurations for two of these paths: the *GRE-IP* and the *MPLS* path.

Figure 7(a) shows a Linux configuration snippet at router A that establishes a GRE tunnel to router C and carries traffic between sites S1 and S2 of customer C1. As a contrast, the desired module connectivity and the CONMan commands invoked by the NM at router A to achieve this are shown in figure 7(b). These commands were explained in section III-B. Similarly, figures 8 shows the Linux and CONMan configuration snippet needed to establish the MPLS path.

Note that while our testbed capabilities constrained us to Layer-3 VPNs, some ISPs establish VPN connectivity at Layer-2. This is typically achieved using Ethernet-over-MPLS or PPP-over-L2TP. Recently, VLAN tunneling has been proposed as another means of doing so [45] and as the use of Ethernet in wide-area networks increases, this could be a future VPN technology. Consequently, we also present the Cisco CatOS and CONMan configuration snippet to establish a VLAN tunnel in figure 9.

The figures show that configuration today requires the management plane to specify a lot of low-level details. As a result, **it is difficult to build management applications that automatically generate these configurations**. Instead,

	GRE		MPLS		VLAN	
	T	C	T	C	T	C
Generic Commands	1	2	1	2	3	2
Specific Commands	6	0	6	0	4	0
Generic State Var.	9	21	6	18	3	14
Specific State Var.	11	2	8	2	5	1

TABLE V

COMMANDS AND STATE VARIABLES IN THE SCRIPTS TODAY (T) AND WITH CONMAN (C). THE TABLE AND THE SCRIPTS ARE COLOR/FONT CODED; FOR INSTANCE, THE FIRST OCCURRENCE OF A "GENERIC COMMAND" IN EACH SCRIPT APPEARS IN RED/ITALICS AND SO ON.

many management applications provide a better user interface and/or some syntactic sugar to the human manager (this is useful in itself). Even with these applications, the human manager still needs to provide the specifics and this leaves the door open for many kinds of errors; for instance, some error possibilities in figure 7(a) include not configuring device A as a router (command 4), 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 (commands 5-9), configuring the tunnel end points with the wrong key values (command 2), using tunnel end point IP addresses that are wrong or do not have IP connectivity between them (command 2), etc.

The CONMan scripts do not appear any-less-fragile. However, as mentioned earlier, **the human manager doesn't need to see, much less write, these scripts**. All the identifiers in the script, such as the module and device identifiers, are exposed by the devices themselves and learnt by the NM through *showPotential*. Further, there is very little protocol-specific information in CONMan scripts and hence, **an automated NM can generate the commands and other details algorithmically without incorporating protocol-specific knowledge**. Also, the similarity in the CONMan scripts for three completely different protocols can be seen as retrospective (yet relevant) evidence of CONMan decoupling the management plane from data-plane evolution. As a matter of fact, the VLAN tunneling scenario above is a good example of how, with CONMan in place, the same management logic can deal with new data-plane technologies as and when they arise.

To quantify the protocol-agnosticity of CONMan, we counted the number of protocol-specific commands and state variables in the scripts. Table V shows that today's scripts have far more protocol-specific commands and state-variables. As mentioned earlier, the instances of protocol-specific state variables in CONMan scripts (such as *C1-S2* representing the IP prefix for customer1-site2 on line (3) of figure 7(b)) result from the fact that our current effort is restricted to management in a single domain. On the other hand, CONMan scripts have more generic state-variables. This is an outcome of both the verbose nature of the existing CONMan primitives and the fact that CONMan requires the NM to specify a lot of well-structured and systematically learnt generic information which the protocol modules then use to determine the protocol parameters.

	GRE	MPLS	VLAN
Messages Sent	3n+2	3n-2	3n-2
Messages Received	2n+2	2n-1	2n-1

TABLE VI

MESSAGES SENT AND RECEIVED BY THE NM OVER THE MANAGEMENT CHANNEL.

While we admit that these represent very coarse metrics, we see this as a naive yet important step towards quantifying the advantages of having management applications generate CONMan primitives instead of device-level configuration.

Apart from easing configuration, CONMan also simplifies the debugging of errors by a NM. For example, in case of the VPN example, errors like a wire getting cut off or a port on a line card not working will show up in the topology map that the NM maintains. By tracing the 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 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 when NM asks the IP module to self test its connectivity to its peer. 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. However, such debugging capabilities have not been implemented yet and are part of future work.

3) Messaging Overhead: An important concern regarding our approach is the amount of communication overhead imposed on the NM. Table VI shows the number of management messages sent and received by the NM in three scenarios assuming n routers along the path (for our lab set-up, n=3). For instance, to set up a GRE tunnel, the current NM implementation sends commands to each router along the path (n sent), conveys messages between the two GRE modules in devices A and B (2 sent, 2 received), conveys messages between the two IP modules in devices A and B (2 sent, 2 received) and conveys messages between the IP modules in each pair of consecutive devices along the path (2n-2 sent, 2n-2 received). The table shows that, at least in the VPN configuration scenario, the messaging overhead scales linearly with the diameter of the network. However, the more important question is the overhead resulting from a large number of such and other configurations.

On a more general note, there are scenarios where the CONMan approach leads to obvious scaling and robustness concerns. For instance, an extreme scenario would be one where the NM configures modules across the network whenever an application initiates a connection. Note that in such a set-up, the message overhead imposed on the NM(s) would be similar to that imposed on domain controllers in the SANE project [9] and one could use their results to claim that even this can scale. However, for a lot of tasks, the NM can use existing control protocols. For instance, our current path-finder could easily be modified to use a hierarchical two-step traversal wherein the first step finds paths between devices that have been pre-established using a routing algorithm while the next step finds the complete module-level path given the device-level path. Apart from this, CONMan would certainly benefit from many of the proposals to improve the scalability of automated agents within today's SNMP framework [16,27,33]. Further, as we discuss in section V, the NMs themselves may do specialized jobs and hence, scale by divide-and-conquer.

IV. Related work

There is a tremendous amount of past work in network management, the most relevant of which we briefly cite here. On the commercial side, SNMPLink [34] lists many existing management tools, from low-end tools like packet analyzers (eg, Wireshark [51]), traffic monitors (eg, MRTG [39]), and SNMP agents (eg, ITM [8]) to high-end managers like OpenView [47].

Zeroconf [57] (and similar efforts like UPnP [50], DLNA [56], etc.) enable "local communication in networks of limited scale" without any configuration [18]. CONMan is more general (even though the examples in this paper focus on enterprise and ISP networks) but there are networks, such as ad-hoc networks, that we don't deal with. Further, with CONMan, the human manager does need to specify a configuration goal, albeit at a high-level. However, there are a number of Zeroconf features, such as address auto-configuration using link-local addresses, that CONMan could gain from. Policy-based management [19] tries to reduce the amount of intricate knowledge required by human managers by allowing management of QoS [2,35] and security [41] based on high-level policies. There are efforts in both research [54] and industry [46–49] with the similar goals. While steps in the right direction, some entity still has to map these policies to the individual device configurations. The complexity of this translation was the major impediment in the adoption of policy-based networking [21]. CONMan has similar objectives since human network managers only specify the configuration goals that are to be achieved and don't deal with low-level commands (not even CONMan primitives).

CONMan says nothing about how data-plane protocols should be implemented. However, there is the vast body of literature that does deal with protocol implementation, i.e. through abstractions [1], specification languages (Estelle, LOTOS, SDL [42]), implementation languages [11,28,29], and modularization (Click [23], [6]).

The 4D proposal [17] 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. Recently, there has been a spurt of research detailing the reasons for outages and anomalies in IP backbones [26,31], Internet services [32] and BGP routing [14,30]. These studies point to configuration errors as a major culprit. CONMan can reduce these errors, particularly the ones impacting data plane operation. The basic idea behind CONMan was proposed in earlier work [5]. Finally, we believe that CONMan can simplify the cross-layer database and interface proposed in [24], the dependency discovery proposed in [13], and indeed may provide the basis for the Knowledge Plane objectives laid out by Clark et. al. [10].

V. Discussion and Future work

In this paper we have presented a network architecture that is amenable to management. Implementation of a few protocols according to the CONMan model and their use in VPN configuration scenarios shows that the approach is worth considering. Though it is too early for us to claim that the abstraction presented here suffices for all data plane protocols and for tasks beyond basic configuration, we do not envision the module abstraction expanding much beyond its current state. As with OSs where we rely on *ioctls* and special-purpose interfaces for things that cannot be accomplished with the file system interface, in cases where protocol features are not captured by the abstraction (some were mentioned in the paper but we hope they will be few and far between), the low-level parameters will have to explicitly be set. Hence, we allow for the possibility of management applications accessing low-level details and provide the relevant hooks. However, we necessitate that any direct changes to the low-level details be appropriately reflected in the protocol's CONMan abstraction.

Our current attempt has focussed on a single NM managing a given network. However, multiple NMs may exist. Primary and secondary NMs will be needed for robustness. 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. Different domains will have their own NM and these may need to communicate.

These possibilities present a number of challenges such as the need for scoped management channels, extending the management channel beyond a single domain, the possibility of conflicting configurations and so on. Consequently, the notion of a management channel needs more thought. However, we would still like to keep the management channel as simple as possible so we don't run into the problem of managing the management channel. Further, the robustness and scalability questions regarding this channel suggest that it should only be used as the basis for low-level configuration. Higher-level management tasks should then rely on the data-plane for communication.

The NM design requires more work – both on the user-side (example, coming up with good high-level goals or even a language for such goals) and on the network-side (example, ensuring that the translation process can scale to large network). Another important question is 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. However, there is still a lot of work to be done before we can worry about the widespread adoption of CONMan and hence, the path of least resistance towards a manageable, much less a self-managing network.

REFERENCES

[1] M. B. Abbott and L. L. Peterson, "A language-based approach to protocol implementation," in *Proc. of ACM SIGCOMM*, 1992, pp. 27–38.

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

[3] L. Andersson and T. Madsen, "RFC 4026 - Provider Provisioned Virtual Private Network (VPN) Terminology," March 2005.

[4] Anonymized, "Complexity Oblivious Network Management," Available upon request, Tech. Rep., 2006.

[5] ——, "CONMan - Taking the Complexity out of Network Management," in *Workshop paper – Available upon request*, 2006.

[6] E. Biagioni, "A structured TCP in standard ML," in *Proc. of ACM SIGCOMM*, 1994.

[7] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform ," in *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[8] Carsten Schmidt, "Interface Traffic Monitor Pro," http://software.ccschmidt.de/.

[9] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," in *Proc. of Usenix Security*, 2006.

[10] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proc. of ACM SIGCOMM*, 2003.

[11] T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe, "Finally, a Use for Componentized Transport Protocols," in *Proc. of the Fourth Workshop on Hot Topics in Networking*, 2005.

[12] A. Durand and T. Narten, "IETF Tunnel Configuration BOF (tc)," Mar 2005, http://www.ietf.org/ietf/05mar/tc.txt.

[13] P. B. et. al., "Discovering Dependencies for Network Management," in *Proc. of workshop on Hot Topics in Networks*, 2006.

[14] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[15] B. Ford, "Unmanaged Internet Protocol: taming the edge network management crisis," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, 2004.

[16] G. Goldszmidt, Y. Yemini, and S. Yemini, "Network management by delegation: the MAD approach," in *Proc. of the conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, 1991.

[17] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communications Review*, October 2005.

[18] E. Guttman, "Autoconfiguration for ip networking: Enabling local communication," *IEEE Internet Computing*, vol. 5, no. 3, 2001.

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

[20] J. Lang, Ed., "RFC 4204 - Link Management Protocol (LMP)," Oct 2005.

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

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

[23] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, August 2000.

[24] R. R. Kompella, A. Greenberg, J. Rexford, A. C. Snoeren, and J. Yates, "Cross-layer Visibility as a Service," in *Proc. of workshop on Hot Topics in Networks*, 2005.

[25] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, " IP Fault Localization Via Risk Modeling ," in *Proc. of 2nd Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[26] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental Study of Internet Stability and Backbone Failures," in *Proc. of Symposium on Fault-Tolerant Computing (FTCS)*, 1999.

[27] K.-S. Lim and R. Stadler, "Developing Pattern-Based Management Programs," in *Proc. of Conference on Management of Multimedia Networks and Services (MMNS)*, 2001.

[28] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica, "Implementing Declarative Overlays," in *Proc. of ACM SOSP*, 2005.

[29] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan, "Declarative Routing: Extensible Routing with Declarative Queries," in *Proc. of ACM SIGCOMM*, 2005.

[30] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. of ACM SIGCOMM*, 2002, pp. 3–16.

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

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

[33] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE/ACM Trans. Netw.*, vol. 36, no. 7, 1998.

[34] Pierrick Simier, "SNMPLink," www.snmplink.org/Tools.html.

[35] R. Rajan, D. Verma, S. Kamat, E. Felstaine, , and S. Herzog, "A policy framework for integrated and differentiated services in the internet," *IEEE Network Magazine*, vol. 13, no. 5, September 1999.

[36] E. Rosen and Y. Rekhter, "RFC 4364 - BGP/MPLS IP Virtual Private Networks (VPNs)," February 2006.

[37] J. Schonwalder, "Characterization of SNMP MIB Modules," in *Proc. of International Symposium on Integrated Network Management*, 2005.

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

[39] Tobias Oetiker and Dave Rand, "MRTG : Multi Router Traffic Grapher," http://mrtg.hdl.com.

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

[41] D. Verma, "Simplifying Network Administration using Policy based Management," *IEEE Network Magazine*, March 2002.

[42] G. von Bochmann, "Usage of Protocol Development Tools: The Results of a Survey," in *Proc. of Conference on Protocol Specification, Testing and Verification*, 1987.

[43] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, and G. Hjalmtysson, "Routing design in operational networks: a look from the inside," in *Proc. of ACM SIGCOMM*, 2004, pp. 27–40.

[44] E. R. Y. Rekhter, R. Bonica, "Use of PE-PE GRE or IP in BGP/MPLS IP Virtual Private Networks," draft-ietf-l3vpn-gre-ip-2547-05, February 2006.

[45] "CISCO 802.1Q Tunneling," http://www.cisco.com/univercd/cc/td/doc/product/lan/c3550/1219ea1/3550scg/swtunnel.htm.

[46] "CISCO Network Management Products," http://www.cisco.

com/en/US/products/sw/netmgtsw/index.html.
[47] "HP OpenView," www.openview.hp.com/.
[48] "IBM's Autonomic Computing," http://www-03.ibm.com/autonomic/.
[49] "Microsoft Dynamic Systems Initiative," http://www.microsoft.com/windowsserversystem/dsi/default.mspx.
[50] "UPnP Forum," http://www.upnp.org/.
[51] "Wireshark: A Network Protocol Analyzer," http://www.wireshark.org/.
[52] "Arbor PeakFlow-SP : DoS protection," , January 2006, www.arbornetworks.com.
[53] "Cisco IP Solution Center Traffic Engineering Management," , January 2006, www.cisco.com/en/US/products/ps6163/.
[54] "IBM Research: Policy-based Networking," , Dec 2006, http://www.research.ibm.com/policy/.
[55] "SNMP MIB Search Engine," , January 2006, www.mibdepot.com.
[56] "Digital Living Network Alliance," Jan 2007, http://www.dlna.org/.
[57] "Zeroconf Working Group," Jan 2007, http://www.zeroconf.org/.
[58] "Management Newsgroups," Jan 2006, news:fa.netbsd.tech.net,news:comp.dcom.sys.cisco,news:microsoft.public.isa.vpn,andnews:comp.os.linux.networking.