Fault Management Using the CONMan Abstraction

Hitesh Ballani and Paul Francis
Cornell University, Ithaca, NY
Email: {hitesh, francis}@cs.cornell.edu

Abstract—**Fault management in networks is difficult. We argue that a major contributor to the difficulty of debugging network faults is the sheer volume of semantically anemic details exposed by protocols. Unlike past approaches that try to cope with the deluge of information exposed, in this paper we explore how to reduce and structure the management information exposed by data-plane protocols and devices to make them more amenable to fault management. To this effect, we delineate two conditions that the management interface of data-plane protocols should satisfy: it should provide a structured description of protocol reality and it should support what we call a "conservation of bytes" invariant.**

Based on this, we propose an architecture wherein data-plane protocols expose management information satisfying these conditions. This allows management applications to detect, localize and (possibly) resolve faults in a structured fashion. We discuss the detection of a representative set of real-world faults to illustrate our approach. We implemented these fault management features into three protocols and built a management application that uses the features to debug faults. Apart from serving as a proof of concept, this exercise indicates that our proposal does indeed simplify debugging of a large fraction of network faults.

I. Introduction

An important part of managing networks is the ability to *detect*, *localize* and *resolve* faults. Fault management in networks today varies from manual probing using simple tools such as ping and traceroute to sophisticated packages such as OpenView [23] and SMARTS [22] that are often used by ISPs and large enterprises. While these packages are certainly useful tools in the hands of experienced administrators, debugging faults in networks still remains a black art relying heavily on the domain knowledge of experts.

We believe that a major contributor to the difficulty of fault management in networks is the **sheer volume of semantically anemic details exposed by protocols to the management plane**. This impacts debugging as follows:
(a). Difficulty of detection. Today, protocol MIBs (Management Information Base) often contain myriad counters to indicate statistics such as the number of packets sent and received, packets dropped due to different kinds of errors, etc. Management applications typically set SNMP traps on these counters to generate alarms that indicate possible problems. However, the error counters tend to have protocol intricacies embedded in them and hence, it is difficult to come up with meaningful and robust thresholds for them without protocol and network details. The resulting spurious or redundant alarms represent undue overhead for human managers and have resulted in a number of proposals to address the problem [5,17,21]. Further, many *silent failures* are not reflected in these counters and hence, may go undetected [14].
(b). Difficulty of localization. The complex nature of the management interface of protocols implies that it is difficult for management applications to understand the precise operation of the underlying network. This, in turn, impedes structured localization of faults. Instead, commercial network fault management systems like [22,23] rely on rule-based correlation of network alarms. Such an approach, like any expert system, is restricted by the domain knowledge encoded into the application rules [16]. Further, when combined with the rapid pace of development and deployment of protocols and devices, this implies that fault management systems tend to lag behind the power curve [14].

Also, failures of protocols, devices and even links tend to be inter-dependent. However, the web of dependencies is often confined to the mind of the human administrator managing the network [1] or to manually maintained databases [16]. This lack of dependency maintenance impedes automated localization of network faults [9,15].

As we detail in section VIII, a number of recent proposals have tried to cope with the deluge of information exposed by protocols and devices through novel management plane algorithms. Here we explore the alternative tact of addressing the aforementioned problems in the data-plane itself by reducing and structuring the management information exposed by protocols. Specifically, we argue that in order to be amenable to fault management, the management interface of protocols should satisfy two conditions:

1) *Structured description of protocol reality.* The management interface should detail the protocol operation, connectivity and dependency in a structured and protocol independent fashion. This would allow management applications to understand the network operation and hence, debug faults that disrupt proper network operation without domain-specific knowledge.
2) *Allow for invariant checks.* "Conservation of bytes" is a fundamental and trivial invariant governing the operation of protocols and states that the inflow and outflow of bytes into a module, a pair of connected modules and by extension, the entire network should match up. Network faults may result in the violation of this invariant and hence, management applications should be able to use the information exposed as part of the management interface of protocols to verify if the invariant holds for them.

Today's protocol MIBs don't satisfy these conditions. The MIBs are littered with low-level details that makes it very difficult for management applications to understand the impact of data-plane events on the protocol's operation (condition (1)). Further, it may seem that given these myriad details and counters, condition (2) would be satisfied and it would be possible for management applications to verify the byte-conservation invariant for some, if not all, entities. To check this, we studied the MIBs of five protocols (Ethernet [10], IP [11], GRE [27], TCP [19], MPLS [25]) and in spite of the use of protocol-specific information to decipher the meaning of the exposed variables, we found that in none of the five cases did we have enough information to perform the trivial check of matching up the inflow and outflow of bytes through a given protocol.

Guided by this, we propose a network architecture wherein the management interface of data-plane protocols satisfies these conditions. We satisfy the first condition by borrowing and extending the protocol abstraction proposed as part of the CONMan project [2]. To satisfy the second condition, we require protocols to expose generic and mutually consistent counters that are structurally related to each other. The fact that the management application understands how they are related allows it to use them to satisfy the byte-conservation invariant. To this effect, this paper makes the following contributions:

- We define *conservation of bytes*, an invariant that holds across all data-plane protocols and hence, can be used to detect faulty protocol operation in a protocol-oblivious fashion.
- We detail the information that devices and protocols should expose as part of their management interface so as to satisfy both the aforementioned conditions.
- We illustrate how management applications can utilise the information exposed by protocols to detect and localize *connectivity faults*, i.e. faults that impact the connectivity between devices.
- Finally, we modified the kernel-level implementation of three Linux protocols (IP, GRE and Ethernet) in accordance with our proposal and implemented a management application that can use the aforementioned features to detect faults that may afflict the operation of these protocols. We inject real-world faults into a testbed network and detail the use of this management application to detect the faults.

II. CONMan Overview

The first condition requires that management applications be able to understand the network operation based on the management interface of protocols. The CONMan project proposed a management interface for protocols that allows just this. Hence, the protocol features discussed in this paper build upon the CONMan proposal. This section provides a very brief overview of CONMan that suffices for the discussion in the rest of the paper - the interested reader is referred to [2] for more details. The CONMan architecture and by extension, the architecture in this paper, consists of *devices* (routers, switches, hosts, etc.) and one or more *network managers* (NMs). A NM is a software entity that manages some or all of the network's devices. Each device in the network has a *management agent* (MA) that is responsible for interacting with the protocols in the device on one hand and the network's NM on the other. CONMan assumes that the network has a management channel that is used by the NM for communicating with the devices. This management channel may or may not operate over the same physical links as used by the data-plane.

The key insight behind CONMan is that most data-plane protocols have some basic characteristics that should suffice for their management and it captures these basic characteristics using a generic abstraction called the *Module Abstraction*. In the abstraction, each protocol is modeled as a *protocol module* with *pipes* connecting it to other modules, generic *switching* capabilities, generic *filtering* capabilities, *performance* and *security* characteristics and certain *dependencies*. This protocol abstraction was used by the CONMan proposal for network configuration. In this paper, we assume that the network being managed is configured using CONMan in the first place. Beyond this, we extend the abstraction to allow the NM to debug network faults.

Of the various abstraction components, we briefly describe "pipes", "switches" and "dependencies" here because of their relevance to the discussion later. *Up-Down* pipes connect modules to other modules above and below themselves in the same device and can be created by the NM. The actual network links are modeled as *Physical* pipes. Further, each pipe is associated with one or more *peer 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.

A module's *switch* captures the module's ability to pass packets between up, down and physical pipes. These switches can have a small number of basic configurations. For instance, passing packets between down and up pipes ([down $\Rightarrow$ up] and [up $\Rightarrow$ down] switching; eg. TCP module), [down $\Rightarrow$ down] (eg. IP module with forwarding enabled), [up $\Rightarrow$ up] (eg. IP module with loopback functionality) represent a few of these configurations. Finally, protocol modules may depend on control-plane protocols for their operation and they express these as *dependencies*. For instance, the IP module relies on ARP for a peer's MAC address and hence, each down-pipe for the IP module is said to depend on the connectivity between the corresponding ARP modules. This dependency is exposed by the module in its abstraction.

As far as configuration is concerned, human managers in CONMan only specify *high-level goals* to be achieved by the management plane. Each device in the network uses the management channel to inform the NM of its physical connectivity, the modules it contains and their abstractions. This provides the NM with the real picture of the network. Given this picture and the high-level goals, the NM builds a graph of inter-connected protocol modules in various devices that would satisfy the goals. This graph, hereon referred to as the *connectivity graph*, captures how the modules should be connected and how each module should operate. The NM can then invoke the appropriate CONMan primitives (for instance, to create pipes, switch rules, filter rules, etc.) at various modules and it is the modules themselves that determine the low-level parameters necessary for their operation.

III. Fault Classification

As in CONMan, we assume that a network's human manager only specifies the high-level goals to be achieved by the network. Hence, we define a *fault scenario* as a situation wherein the network being managed does not satisfy specified high-level goals. Such a scenario can result from one or more *primary faults* each of which, in turn, may lead to zero or more *dependent faults*.

A network's high-level goals can vary a lot in complexity. For the sake of concreteness, this paper assumes that the high-level goal for the network is specified in the form of a *connectivity matrix* that indicates the devices (or, the protocols and applications in these devices) that should or should not be able to communicate with each other. Given that the connectivity matrix is the network's goal, any scenario that violates the matrix is a fault scenario. Consequently, the fault scenarios discussed in the rest of the paper directly impact the connectivity between devices.

Note that high-level goals may be more sophisticated and hence, there can be other kinds of faults. For instance, the goal may include the the performance parameters that the network paths must satisfy. Actually, poor performance in a correctly connected network is a very common fault scenario. While we discuss such faults in section IX, the majority of this paper restricts itself to debugging of connectivity faults.

Further, network faults can be classified based on their root-cause [13] into (a) *Hardware faults*, (b) *Software faults*, (c) *Power faults*, (d) *Configuration faults* and (e) *External faults*. The techniques presented in this paper focus on intra-domain faults and hence, don't apply to fault family (e). Since human managers in CONMan do not write low-level configuration scripts, the possibility of configuration faults resulting from human errors (which represent the dominant fraction of faults today [13]) in CONMan is already very low. However, errors in the CONMan software itself can lead to erroneous configuration. Hence, the techniques presented in the paper apply directly only to faults of type (a), (b), (c) and some fraction of class (d).

IV. Protocol Features for Fault Management

In this section, we detail the operation of devices and the information that should be exposed by protocol implementations as part of their management interface to satisfy the conditions specified in section I.

A. Connectivity Reports

As mentioned in section II, devices in the network probe their neighbors over the data-plane and periodically inform the NM of their physical connectivity over the management channel. This provides the NM with the network topology. A device losing power or other faults afflicting the physical connectivity between devices are thus reported to or can be inferred by the NM.

B. Protocol Abstraction Reports

Protocols expose information about their operation to the management plane using the CONMan abstraction. Over time, as network events happen, a protocol's low-level parameters can change. The protocol implementation should ensure that any changes to its low level parameters or its operational status that impact the protocol's abstraction are appropriately reflected in the abstraction exposed and reported to the NM. Below we give examples of this for a couple of CONMan abstraction components:

(a). Pipes: An up-down pipe represents the association between the two protocol modules it connects. Any change in the status of the protocols that impacts this association should cause the status of the corresponding pipe to change. For example, in CONMan, the down pipe from a GRE module to the underlying IP module represents a GRE-IP tunnel. Thus, any event that leads the GRE tunnel interface to go down should cause the deletion of the corresponding down pipe and the NM to be informed of this change. Similarly, the down pipe between a VLAN module and the underlying ETH module represents the membership of an Ethernet port in a VLAN group. Hence, the elimination of a port from a VLAN group should be notified to the NM as the deletion of the corresponding pipe.

Note that we are not claiming that faults like the ones above cannot be detected today. Of course, management applications in the existing setup can set SNMP traps to get notifications regarding the state of the tunnel interfaces, VLAN membership and so on. However, such notifications lack semantics about what they entail for the protocol and network operation. This implies that the management application needs to be embedded with the meaning and implications of protocol-specific events such as VLAN group membership changes. As a contrast, the pipe abstraction allows us to capture what such events mean for the flow of packets in a protocol-oblivious fashion. Specifically, the events above are reported to the NM as pipe deletions and it can determine what this means for the network operation.

(b). Switches: A module's switch rules specify how it sends packets from one pipe to other pipes. The NM should be informed of any changes to these rules. For instance, the inability of a Linux host to "forward" packets at the IP layer (`/proc/sys/net/ipv4/ip_forward`=0) implies that the host's IP module cannot switch packets from one down pipe to the other (no [down $\Rightarrow$ down] switching) and this should be reported to the NM.

These abstraction reports ensure that the NM has knowledge of the network's current connectivity graph.

C. Protocol Counters

Not all protocol parameters are reflected in the protocol's abstraction. An undesirable change in such parameters may disrupt the traversal of packets without showing up in the abstraction. For instance, each Ethernet port on a device is represented by an up-down pipe between the device's ETH module and the underlying PHY module. However, the NM is not informed of low-level events, such as a change in the port's duplexity even though the change may cause a duplexity mismatch with the Ethernet port at the other end and hence, affect the device's Ethernet connectivity.

Further, a hardware or a software bug might cause packets to be dropped without any adverse settings of the low-level parameters. This entails that the management plane should be able to *account* for the flow of bytes through individual protocols. Specifically, we note that most such faults violate a fundamental and trivial invariant regarding the operation of protocols: the "conservation of bytes" invariant which essentially says that all bytes entering, consumed by and exiting an entity should match up. The "entity" here could be a single protocol, a pair of connected protocols, a sequence of connected protocols and by extension, the entire network. The NM should be able to verify these invariants.

One way to achieve this is for each protocol to maintain simple counters that can then be used to quantify the inflow and outflow of bytes through it. We also note that the components of the CONMan abstraction provide a very good framework for these counters since the counters can be associated with individual components and this would allow the NM to understand how the counters are related to each other. Hence, we propose that protocol modules should expose the following generic counters as part of their abstraction:

1) *Snd* (S_i) and *Rcv* (R_i) counters indicating the number of bytes sent and received by the module on each pipe i.
2) *Snd-proto* (Sp) and *Rcv-proto* (Rp) counters indicating the number of protocol generated bytes sent and received by the module. Protocol-generated bytes include protocol headers, retransmissions, overhead due to operations like encryption, bytes in additional copies of packets sent when multicasting and broadcasting and any other bytes that are generated as part of the protocol operation. The counters can be negative; for instance, when the module compresses data. These counters ensure that our proposal does not place any restrictions on protocol design while allowing the management plane to account for the flow of bytes in a protocol oblivious fashion.
3) *Snd-residual* (Sr) and *Rcv-residual* (Rr) counters. Up-down pipes between modules get created and deleted over time. For instance, each up pipe for a TCP module represents a TCP connection and hence, it exists as long as the connection exists. Residual bytes account for the bytes sent and received on deleted pipes and ensure that modules only need to maintain pipe-specific state for active up-down pipes. Hence, when a pipe *i* is deleted, the S_i and R_i counters are deleted too while the residual counters are modified as follows: Sr += S_i and Rr += R_i.
4) *Dropped* (D) counter indicating the number of bytes dropped by the module. For instance, packets that have been corrupted due to a duplex mismatch on an Ethernet port and hence, are dropped by the ETH module contribute to the module's *Dropped* counter. Note that dropped packets may not always represent an error scenario; for example, filtered packets also contribute to these counters.
5) *Buffered* (B) counter indicating the number of bytes buffered by the module.
6) *Snd-dep-error* ($S_{i,j}$) and *Rcv-dep-error* ($R_{i,j}$) counters indicating errors afflicting dependency j of pipe i. For instance, an IP module has one down pipe for each of its next-hops (i.e. any device that can be reached within one IP hop). Further, as mentioned earlier, a down-pipe for the IP module has a dependency on connectivity between the ARP module of the host and that of the next-hop. When the host's IP module is not able to forward packets to a next-hop due to ARP failure, the dropped packet, apart from being added to the IP module's $Dropped$ counter, also contributes to the *Snd-dep-error* counter for the dependency of the corresponding down-pipe on ARP.

Protocol modules periodically report these counters to the management agent of their device. However, for a module's counters to be comparable to each other, they must represent the protocol state at a given point of time and hence, there is a need to synchronize these counters. This is discussed in section VII.

V. Fault Management

The network's management plane, including the NM and the management agents (MAs) on the individual devices, use the aforementioned protocol features to detect network faults. This section describes such detection while we illustrate it through example faults in the next section.

A. Detection by management plane

1) Faults appearing in the connectivity graph: The first kind of faults detected by the NM are the ones that appear in the connectivity graph itself. Network devices send both periodic and event-based reports regarding their connectivity and the abstraction for their protocols. Thus, after any event, the NM modifies its connectivity graph accordingly and can detect any faults that adversely impact the desired connectivity.

2) Faults violating byte-conservation invariant: The management plane can verify if the byte-conservation invariant holds for individual protocol modules to detect hardware and software bugs that cause packets to get dropped without being accounted for in one of the counters. To this effect, the MA of a device ensures that for each protocol module, the total number of non-header bytes received (after accounting for the dropped and buffered packets) is the same as the total number of non-header bytes sent. Thus, for each protocol module, the MA checks to ensure that the following equation is satisfied and if not, it informs the NM.

$$\begin{array}{ccc} \text{(Total bytes received -} & = & \text{(Total bytes sent +} \\ \text{Dropped bytes - Protocol} & & \text{Buffered bytes - Protocol} \\ \text{specific bytes received)} & & \text{specific bytes sent)} \\ \sum_i (R_i) + Rr - D - Rp & = & \sum_i (S_i) + Sr + B - Sp \end{array}$$

Of course, it may be possible for a fault, apart from causing packets to be dropped, to result in the module's *Snd/Rcv* counters to not be updated or be updated improperly such that the consistency equation still holds. To account for this, the MA checks for byte-conservation across adjacent modules. Given that it has already verified that bytes are conserved by individual modules, it can verify conservation for a pair of connected modules by checking that for each pipe between the modules in question, all bytes sent by one module are received by the other and vice versa. Specifically, the MA compares Snd and Rcv counters for the modules at the end of each up-down and physical pipe and informs the NM if they don't match. This check ensures that unless the bug impacts the Snd and Rcv pipe counters at multiple modules, it is detected.

3) Dependent faults: The NM uses dependency error counters as an indicator of a fault resulting from a separate *root fault*. Specifically, when a module reports an increment in the value of $S_{i,j}$, the NM knows that a fault afflicts dependency j of pipe i for the module in question. The same applies to the $R_{i,j}$ counter.

B. Localization by management plane

By making protocol modules expose information that can be used by the management plane to detect faults, we also ensure that the device and the protocol responsible for the detected fault is either apparent or trivial to localize. First, for power faults and any other faults afflicting the physical connectivity between devices, the NM can use its knowledge of the network topology and correlate updates from different devices to localize the fault to a single physical link or the two

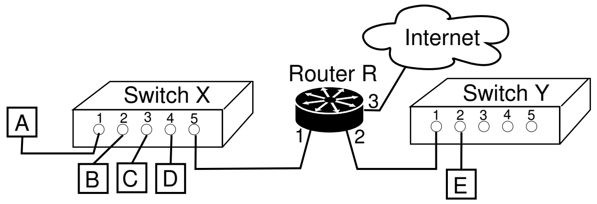

Fig. 1. A small test network used to illustrate the example faults.

	A	B	C	D	E	Internet
A		✓	✓	✗	✗	✓
B	✓		✓	✗	✓	✓
C	✓	✓		✗	✗	✓
D	✗	✗	✗		✓	✗
E	✗	✓	✗	✓		✓
Internet	✓	✓	✓	✗	✗	

TABLE I
CONNECTIVITY MATRIX TO BE ENFORCED FOR THE TEST NETWORK IN FIGURE 1 - ENTRIES IN THE MATRIX DENOTE IF CONNECTIVITY SHOULD (✓) OR SHOULD NOT (✗) BE ALLOWED.

devices the link connects. Second, a fault detected based on the abstraction of a module, including the violation of invariants, is caused by the module itself and hence, there is no need for localization. Finally, in case of dependent faults, the NM may need to do additional work to determine the root fault. Such localization is explained in the next section.

C. Resolution by management plane

The underlying cause for faults in categories (a)-(d) cannot be addressed in an automated fashion. For instance, a software fault will have to be reported to the network's human manager and will need to be resolved by the device vendor. However, once it detects and localizes a fault, the management plane can attempt to mitigate the fault's impact on the network's high-level goals.

VI. FAULT MANAGEMENT EXAMPLES

We now explain the proposed fault management techniques through a few examples. We used debugging manuals of various vendors, bug reports and other sources to collect a set of eleven real-world faults that are representative of fault types (a)–(d) and are listed below. We illustrate these faults using the small test network shown in figure 1. The network comprises of a router, a couple of switches and five end-hosts. Table I shows the connectivity matrix to be achieved in this network. We also assume that the network has an automated network manager (NM) that configures the network elements using CONMan such that the matrix is satisfied. Specifically, the NM creates VLANs on switch X, creates the routing and filtering rules on router R and configures the end hosts using CONMan primitives. The relevant part of the module-level connectivity graph for the configured network is shown in figure 2 (the figure does not show the NM or the management channel).

(a). Hardware Faults.

Fault H1: Wire cut. The physical wire connecting host A to switch X gets cut off.

Fault H2: Jammed Line Card. The line card of router R corresponding to interface 2 jams up and fails to dequeue packets [14].

Fault H3: Non-functional ports. Under heavy traffic, some ports of switch Y (a Catalyst 5000) stop transmitting frames [24].

(b). Software Faults.

Fault S1: VLAN partitioning. A software bug in switch X causes individual VLANs to be partitioned into isolated segments. In the past, such a bug impacted Cisco Catalyst 4000 switches [24].

Fault S2: Duplex mismatch. A bug in the NIC driver of host B leads to a duplex mismatch between the Ethernet interface of host B and the port of switch X to which it is connected. This, in turn, results in performance issues, intermittent connectivity, and loss of communication [24].

Fault S3: Configuration wipe. Rebooting switch X causes it to lose all existing VLAN configuration. Thus, when the switch boots up, hosts A, B and C are able to access host D since they are no longer in separate VLANs. Such configuration wipe problems afflicted a model of Linksys switches [26].

(c). Power Faults.

Fault P1: Power supply. A problem with the power supply of router R causes it to shut down.

(d). Configuration Faults.

Fault C1: Improper filtering. The NM aims to block connectivity between hosts A and E by configuring an IP filter rule in router R that filters packets between interface 1 and interface 2. However, due to a software error in the NM implementation, it does not account for the fact that the rule also (inadvertently) blocks packets from host B to E even though such packets should be allowed. Note that this is a configuration fault since the data-plane software is correct, it is just configured incorrectly.

Fault C2: MTU problem. The MTU of the Ethernet interface of host A does not account for the VLAN tags added by switch X. Thus, MTU-sized packets originating at host A and with an IP "Don't Fragment" flag set are dropped at X.

Further, since the fault at an element can also lead to faults in the operation of protocols dependent on the element in question, we have **dependent faults:**

Fault D1: ARP Failure. Fault S1 may disconnect host A and B even though they are in the same VLAN. This, in turn, can cause ARP queries issued by host A for host B to fail and thus, impact IP-level connectivity between A and B.

Fault D2: DHCP Failure. Fault P1 shuts down router R and hence, the DHCP server for the network. If host A requires a new IP address while router R is down, its request will fail and this will impact A's IP connectivity.

A couple of aspects of these faults deserve comment. First, we don't claim these faults to be representative of all possible network faults. Instead, they include examples of the different kinds of internal faults and serve to illustrate how the CONMan management plane deals with network faults. Second, protocols in networks today are already designed to be resilient against some of the example faults. For instance, routing and switching algorithms account for faults like P1, H1 and H3 and ensure that the data plane is routed around these. As a contrast, most of the other example faults involve implementation errors that, while infrequent, may not lead to alarms in the existing framework.

A. Detection and Localization

We discuss how the NM detects and localises these example faults.

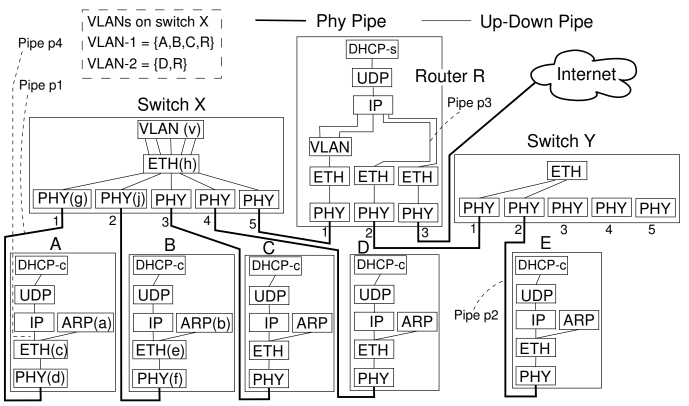

Fig. 2. Connectivity graph for the test network showing how the modules in various devices are connected so as to satisfy the desired connectivity matrix.

1) Faults appearing in the connectivity graph: H1, H3, P1, S3, D2.

– Fault H3: Lets assume that heavy traffic causes port 2 of switch Y to stop transmitting frames. Depending on the precise nature of the fault, the PHY module corresponding to port 2 may or may not realise that pipe p2 is inoperational. However, the MA of switch Y is unable to probe host E and informs the NM of this change in the switch's connectivity. Based on this connectivity update, the NM determines that a fault afflicts the connectivity of switch Y to host E. Given the physical nature of the problem, the NM is unable to resolve it. However, the NM can determine the fault's impact; any paths in the connectivity graph that include pipe p2 are affected and hence, none of the other hosts will be able to access host E. Faults H1 and P1 are detected similarly by the NM based on connectivity updates.

– Fault S3: When switch X reboots and comes back up, it informs the NM of the abstraction for the modules in it including how the modules are connected. The switch's loss of VLAN configuration is apparent in the abstraction of the ETH and VLAN modules since the VLAN module is no longer connected to the underlying ETH module. The NM can resolve the fault by reinvoking the CONMan commands to create up-down pipes between the VLAN and the ETH module in switch X which, in turn, recreates the VLAN configuration.

2) Faults violating byte-conservation invariant: H2 and S1.

– Fault H2: When router R's IP module receives a packet, it increments the Rcv counter for the pipe corresponding to the packet and queues the packet for further processing. However, the fault causes the module to fail to dequeue some of these packets and this is not reflected in the module's $Dropped$ counter. Hence, the number of bytes received is more than the bytes sent. This inconsistency is detected by router R's MA and the NM is informed of the fault.

It is possible that the fault also impacts how the module's other counters are updated. For instance, apart from not dequeuing packets off pipe $p3$, the module may not reflect the packets in the R_{p3} counter. Thus, byte-conservation holds for the module. Even so, the fault would be detected when the MA checks for $(S_{p3})_{ETH} = (R_{p3})_{IP}$ and finds that the number of bytes sent by the ETH module onto p3 $\{(S_{p3})_{ETH}\}$ does not match up with the number of bytes received by the IP module on p3 $\{(R_{p3})_{IP}\}$.

3) Dependent Faults (not already detected): D1.

– Fault D1: In figure 2, host A should be able to directly reach host B since they are part of the same VLAN. In CONMan, such connectivity is captured through a down pipe (P4) for the IP module of host A that has the IP module of host B as its peer. Further, the module's abstraction includes the fact that down pipe p4 has a dependency requiring that the corresponding ARP modules in hosts A and B (labeled as a and b in the figure) be able to reach each other. Thus, increments for the $S_{p4,1}$ counter of host A's IP module provide the NM with an indication of a fault along the path from module a to b. The NM already knows the sequence of modules that make up the path between a and b, labeled as *a, c, d, g, h, v, h, j, f, e, b* in figure 2. To localise the fault, the NM queries these modules' abstraction and finds that the invariant does not hold for module *v*. This implies that the root fault afflicts module *v*. Indeed, the root fault that led to fault D1 is fault S1 that impacts the VLANs on switch X. Note that this fault may already have been detected by the MA of the switch and the NM informed about it.

B. Remaining faults

1) Detection by modules: Faults that either cause changes in the connectivity graph or involve bugs that cause packets to be dropped abnormally are detected by the NM. However, protocols can also drop packets due to a number of valid factors and these packets are reflected in the *Dropped* counter exposed in the protocol's abstraction. Determining if the dropped packets indicate a fault requires protocol-specific details. However, we are averse to embedding such protocol-specific information in our management plane.

Consider fault S2 – the duplex mismatch between host B's NIC and switch X's port results in a high number of Ethernet frame collisions and hence, malformed frames. These are reflected in the *Dropped* counter of the PHY modules representing host B's NIC and switch X's port. However, collisions of Ethernet frames can also occur due to other reasons. For instance, if both the NIC and the switch port are operating in half duplex mode, some number of collisions are bound to happen. Specifically, the Cisco switch debugging manual states that a switch port should observe almost no collisions when the ports to it are in full duplex mode while a 1% collision rate is acceptable when any of the ports is in half-duplex mode [24]. Hence, for the NM to set a threshold on the number of collisions that is acceptable, it would have be aware of protocol-specific details such as the notion of port "duplexity" and its implications.

In most such cases, the protocol module itself has all the necessary information to determine thresholds for the error counters that represent an anomaly and hence, indicate the presence of a fault. In the example above, the PHY module of a device is aware of its duplex configuration and can use the management channel to determine the duplex mode of its peer module. Thus, the module can determine if a given number of collisions represent an anomaly on its own. We propose that such *self-detection* be embedded into the protocol implementation. Consequently, when data-plane protocols drop packets due to some kind of (non-dependency, non queue-overflow and non-filtering) fault, apart from reflecting this in the *Dropped* counter, they either try to resolve the fault on their own and if not, indicate the fault to the NM that raises an alarm for human intervention. Fault C2 can similarly be detected by

Faults ⇒	Avoided	Detected by CONMan				Not detected by CONMan
	Human Errors	Connectivity Graph Faults	Invariant Violation Faults	Dependency Counter Faults	Protocol Parameter Faults (the ones that don't appear in abstraction)	Remaining Faults (external faults, many control-plane faults and bugs in CONMan software)
Detection by	-	NM	MA	NM	Protocol Module	Humans/Applications
Detection timescale	-	Machine timescale (seconds)				Humans timescale (hours to days)
Localization	-	For dependent faults, NM localizes root fault	Not needed	NM localizes the root fault	Not needed	NM (future work)
Resolution	-	NM routes around the fault except the ones that can be resolved by modifying the abstraction of the faulty module (eg. S3)			Protocol Module attempts resolution	NM routes around the fault
Example.	-	P1, H1, H3, S3, D2	H2, S1	D1	S2, C2	C1

TABLE II
DEBUGGING FAULTS

the module itself.

2) Complimentary techniques: Table II summarizes how different kinds of faults are detected, localized and resolved in our architecture. While the presented approach does simplify fault management for the management plane, not all network faults can be detected with it. First, the framework described above is restricted to the devices within a domain. Second, the CONMan abstraction does not apply to control-plane protocols and hence, faults resulting from the specifics of control protocols may not be detected. For example, the NM may rely on a routing protocol for routing between the devices and any faults resulting from the operation of the routing protocol will not be detected unless they explicitly appear in the abstraction of the IP modules whose switch rules are being set by the routing protocol.

Finally, faults in the CONMan component of both the data and the management plane are unlikely to be detected. For instance, in the case of fault C1, the filter rule erroneously blocks connectivity between hosts B and E and while the fault (the invalid filter rule) does appear in the abstraction, it wouldn't be detected by the NM that configured the rule in the first place.

In these instances, we have to rely on user and application alarms for detection of faults. Consequently, such faults would be detected on human timescales. Further, such detection entails additional localization logic, for instance the cross-layer traceroute functionality proposed as part of [6] to determine the protocol that drops the packets between a given pair of application modules.

VII. IMPLEMENTATION

In this section, we discuss our implementation of CONMan protocols and a NM and detail the use of the NM to detect faults in a test network.

A. Management channel and Connectivity updates

The testbed used for the experiments described in this section comprises of Linux-based PCs operating as end-hosts and routers with Ethernet as the connecting medium. Given CONMan's reliance on the presence of a management channel that allows for communication between the devices in the network and the network's NM, we implemented a straw-man management channel that can operate on the same underlying physical network as used by the data plane using the techniques proposed by the 4D project [8].

The MA on each device periodically sends raw Ethernet frames as data-plane beacons to discover the device's immediate neighbors and track the device's physical connectivity to them. The MA then informs the NM of its neighbors over the management channel and sends "connectivity updates" when the device's connectivity changes. Consequently, the NM can generate the *current* network topology.

B. Protocol Implementation

We modified the kernel-level (Linux 2.6.14) implementation of 3 protocols – IP, Ethernet, and GRE – to build features required for fault management. We also built a user-mode MA that interfaces with these protocol modules and the NM. Below we briefly mention the features supported by the CONMan protocol modules and illustrate these with appropriate examples:
– *Abstraction Updates.* The protocol modules expose their CONMan abstraction and appropriately reflect any changes to their low-level parameters that impact the module's abstraction as abstraction updates. For instance, in case of the IP module, up pipes represent the module's association with different transport protocols. Hence, as the Linux GRE-IP module (`ip_gre.ko`) is inserted and deleted from the kernel, the up pipe from the IP module to the GRE module is created and deleted. In case of the GRE module, a down pipe represents the module's association with an underlying network protocol. Thus, a change in the status of a GRE-IP tunnel is reflected in the corresponding down pipe.
– *Dependencies.* The pipes of a module may have dependencies. These are exposed as part of the module's abstraction. For instance, the IP module has two dependencies for its down pipes. First, an IP address must be assigned to the module before its down pipes can be created. Second, a down pipe requires connectivity between the ARP module of the device and that of the pipe's peer. Our implementation ensures that any change in the abstraction due to these dependencies is reported. For the example above, lack of an IP address for the IP module causes its down pipes to be deleted.
– *Counters.* Each protocol module maintains the counters discussed in section IV-C. Our implementation of the counters is similar to the way protocols already maintain and export various SNMP statistics regarding their operation. Periodically, the MA queries the modules in its device for their counters and checks if the invariant holds. Note that the fact that each of the three protocol implementations we considered reside within a single thread in the Linux kernel implies that no special synchronization is needed to ensure that the counters from a given module are in a consistent and comparable state when they are exported.

However, when comparing counters across modules, synchronization is needed. For instance, before comparison, the *Snd*/*Rcv* counters exported by a module for each of its pipes need to be synchronized with the counters from the module at the other end of the pipe. To achieve this, when the MA queries a GRE module for the counters of a down pipe to an underlying IP module, the GRE module effectively sends its counter values to the IP module over the pipe in question as a marked packet. The IP module recognizes this marked packet, retrieves the values of the counters at the GRE end of the pipe and removes the packet from the data stream. Finally, the IP module sends the counters for both ends of the pipe to the MA.

C. NM Implementation

We also implemented a NM that maintains a connectivity graph for the network and changes this graph as its gets connectivity and abstraction updates. The NM can thus search for faults by checking whether the connectivity matrix is satisfied by current connectivity graph. Also, in case of dependent faults, the NM tries to localize the problem by querying all the modules along the suspect path for their status.

D. Fault Management in a test network

In order to validate our implementation, we used our NM for debugging faults in a test network with several hosts running our CONMan protocols. This network is the same as shown in figure 1 with PC-routers in place of the two switches. We injected the faults described in section VI into the network. The lack of switches in the network and other limitations imply that we were unable to reproduce fault scenarios H2, H3, S1, S3, C2 and D1. However, we did introduce a few alternate faults:

Fault H2'. To emulate fault H2, we forced the IP protocol to discard a fraction of incoming packets after they have been received and the *Rcv* counter for the incoming pipe incremented.

Fault H2". We modified fault H2' such that packets were dropped even before they were added to the *Rcv* counter.

Fault S3'. Fault S3 involves loss of VLAN configuration on switch X. We emulate a similar fault involving the loss of GRE configuration at host A that brings down the GRE-IP tunnel from host A to host E.

Overall, we injected eight faults into the network: H1, H2', H2", S2, S3', P1, C1, D2. The results of this exercise were along expected lines with all faults except C1 being detected and localized:
– Faults H1, S3', P1, and D2 showed up in the connectivity graph and were detected by the NM.
– Faults H2' and H2" were detected by the NM based on violation of the byte-conservation invariant.
– Fault S2 was detected by the ETH module using self detection.
– Fault C1 was not detected.

We admit that such injection of synthetic faults is a very limited validation exercise. However, we think that it provides tangible evidence that CONMan does indeed reduce the burden on management applications since the NM, working with just a connectivity graph, pipes and other abstraction elements, is able to debug the faults in an algorithmic fashion. Further, our NM is mostly protocol agnostic with the notion of IP addresses being used by the IP module being the only protocol-specific information embedded in the implementation.

To contrast this with the existing set-up, we studied how management applications today would debug such faults (of course, there might be other ways of detection).
– Faults H1, P1. Assuming that the network is running an intra-domain routing protocol, the management application will have to track routing updates to detect the fault. For instance, for OSPF, the management application will have to listen and interpret "Link State Updates". However, if the topology permits, the routing protocol will route around the fault on its own.
– Fault S2, S3' and D2 would require that management applications be hard-coded with various events they need to track. These events include:

- S2 causes the Ethernet driver on our Linux PCs to generate an error in a log file stating "eth0: Transmit error, Tx status register 82. Probably a duplex mismatch ...".
- S3' generates an SNMP trap for deletion of the ifEntry (interface entry) corresponding to the tunnel interface in the ifTable (interface table) of the Interface MIB of the device.
- D2 leads to an error in a log file stating "... No DHCPOFFERS received. No working leases in persistent database ...".

Apart from the protocol details required to track these events in the first place, the fact that the management plane doesn't understand how protocols are connected to or dependent on each other implies that it is difficult to determine the impact of a fault even when it is detected.
– Fault H2', H2" and C1 would be very hard to detect based on protocol MIBs today. Instead, the faults would probably be detected based on user complaints or application performance.

VIII. Related work

There has been a tremendous amount of work towards network fault detection, the most relevant of which we mention here. We discussed both commercial [22,23] and research [5,17,21] fault management systems in section I. While most of these focus on data-plane events, recent efforts [20] have tried to incorporate control-plane information into the correlation process. Another approach to debug network faults is to use a network model to reason about normal and abnormal network behavior [4,7]. Instead of constructing a model out of diverse and detailed protocol interfaces, we argue for the protocols to model themselves so as to be amenable to management.

An important aspect of fault management is the dependencies between protocols and devices. The notion of SRLGs capture such dependencies at the optical layer and have been combined with IP fault notifications to detect low-level network faults [12,16]. Sherlock [1] discovers the dependency graph governing the network while both [1] and [9] use such a graph to detect and localize network faults. We require that protocols and devices themselves report dependency and status information. The byte-conservation invariant has been used in many proposals to prove protocol properties, for instance, the WATCHERS protocol [3] used it to detect compromised routers. We propose a novel use of this invariant for fault detection.

IX. Discussion

Scalability. Perhaps the most important concern regarding the architecture presented in this paper is its scalability with network size and complexity. In a network with a lot of devices and a number of protocols per device, the total number of protocol modules being managed can easily grow to a substantial amount. Some aspects of our design, such as the fact that the invariant checks are done by a device's MA itself, are guided by this concern. However, we would like to point out that this problem is not qualitatively different from the scalability of today's management set-up. Today, protocols expose detailed MIBs riddled with protocol-specific parameters and other low-level counters. Management applications typically register for the events they wish to be notified for and the same can be done with our approach. Further, proposals to improve the scalability of automated agents within today's SNMP framework [18] apply directly to our management plane.

Beyond connectivity faults. This paper is restricted to faults that impact connectivity, i.e. packets either go through or not. Other faults may impact performance, may involve the load on devices or might even concern the security of the communication provided. While challenging, we think that the fault management approach presented here can be extended to these faults and we intend to pursue this in the future. For instance, the CONMan abstraction of a protocol contains information about basic performance parameters, including the latency, jitter and bandwidth of various pipes and statistics about the module's queues that could be used by the NM to detect violation of high-level goals that include performance and load constraints.

Management-plane faults. Having the management channel independent of the data-plane would seem to suggest that a fault afflicting the data-plane doesn't impact the management channel. However, the management channel may share the underlying physical links with the data-plane. Further, implementation realities might introduce dependencies between the management and the data-plane. For instance, in order to avoid changing the Ethernet driver on devices, our implementation of the management channel operates on top of Ethernet. Hence, in our implementation, faults affecting the physical links and the Ethernet layer impact the management channel. However, the NM can trivially detect such faults based on the lack of management plane updates from one or more devices and can use its knowledge of the network topology to localize the fault. More insidious are faults that afflict the management software itself and their detection was discussed in section VI-B.

Evaluation. Finally, a crucial question is how to quantify the advantages of our approach over the status quo. The main benefit of the proposed abstraction and fault management techniques is that they reduce the burden on management applications by making it easier to detect faults in an algorithmic fashion. Such "ease-of-debugging" and "management application complexity" is difficult to measure but is critical for our argument. A few metrics that we are currently considering for such an evaluation include the fraction of faults detected, the communication overhead imposed, detection times, total number of parameters needed, or even the lines of code of management applications.

To summarise, this paper describes a management interface for data-plane protocols that it is conducive to debugging of faults. We have shown how this interface can be used to detect synthetically introduced faults in protocols. This represents a very simple illustration of the proposed approach and there are many questions that remain unanswered. In spite of this, we believe that this paper represents a promising first stab at the basic idea of making the data and the management plane equal partners in the challenging task of network fault management.

Acknowledgements

This work was partially supported by NSF Grants 0338750 and 0626978.

References

[1] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proc. of ACM SIGCOMM* (2007).

[2] Ballani, H., and Francis, P. CONMan: A Step towards Network Manageability. In *Proc. of ACM SIGCOMM* (2007).

[3] Bradley, K. A., Cheung, S., Mukherjee, B., Olsson, R. A., and Puketza, N. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. In *Proc. of IEEE Security and Privacy (Oakland)* (1998).

[4] Brugnoni, S., Bruno, G., Manione, R., Montariolo, E., Paschetta, E., and Sisto, L. An Expert System for Real Time Fault Diagnosis of the Italian Telecommunications Network. In *Proc. of the IFIP Symposium on Integrated Network Management* (1993).

[5] Chao, C. S., Yang, D. L., and Liu, A. C. An automated fault diagnosis system using hierarchical reasoning and alarm correlation. *J. Netw. Syst. Manage. 9*, 2 (2001).

[6] Fonseca, R., Porter, G., Katz, R., Shenker, S., and Stoica, I. X-Trace: A Pervasive Network Tracing Framework. In *Proc. of USENIX/ACM NSDI* (2007).

[7] Forman, G., Jain, M., Mansouri-Samani, M., Martinka, J., and Snoeren, A. Automated whole-system diagnosis of distributed services using model-based reasoning, 1998.

[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] Gruschke, B. Integrated event management: Event correlation using dependency graphs. In *Proc. of Workshop on Distributed Systems: Operations and Management* (1998).

[10] J. Flick. RFC 3635 - Definitions of Managed Objects for the Ethernet-like Interface Types, Sep 2003.

[11] K. McCloghrie. RFC 2011 - SNMPv2 Management Information Base for the Internet Protocol using SMIv2, Nov 1996.

[12] Kandula, S., Katabi, D., and Vasseur, J.-P. Shrink: a tool for failure diagnosis in IP networks. In *Proc. of ACM SIGCOMM workshop on Mining network data (MineNet)* (2005).

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

[14] Kompella, R., Yates, J., Greenberg, A., and Snoeren, A. C. Detection and Localization of Network Blackholes . In *Proc. of IEEE Infocom* (2007).

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

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

[17] Liu, G., Mok, A., and Yang, E. Composite events for network event correlation. In *Proc. of Integrated Network Management* (1999).

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

[19] R. Raghunarayan. RFC 4022 - MIB for the TCP, Mar 2005.

[20] Roughan, M., Griffin, T., Mao, Z. M., Greenberg, A., and Freeman, B. IP forwarding anomalies and improving their detection using multiple data sources. In *Proc. of the ACM SIGCOMM workshop on Network troubleshooting (NetT)* (2004).

[21] Wu, P., Bhatnagar, R., Epshtein, L., Bhandaru, M., and Shi, Z. Alarm correlation engine (ACE). In *Proc. of NOMS* (1998).

[22] EMC Smarts Family. http://www.emc.com/products/software/smarts/smarts_family/.

[23] HP OpenView. www.openview.hp.com/.

[24] Troubleshooting Cisco Catalyst Switches. , April 2007. http://www.cisco.com/warp/public/473/46.pdf.

[25] Juniper MPLS MIB, Dec 2007. http://www.juniper.net/techpubs/software/junos/junos42/swconfig-install42/html/snmp-mibs3.html.

[26] Linksys Community Forum, Apr 2007. http://forums.linksys.com/linksys/board/message?board.id=Switches&message.id=400.

[27] Wellfleet GRE MIB, Dec 2007. http://www.oidview.com/mibs/18/Wellfleet-GRE-MIB.html.