Computer Sciences Department

A Case for Complexity Models in Network Design and Management

Theophilus Benson
Aditya Akella
Dave Maltz

Technical Report #1643

August 2008

A Case for Complexity Models in Network Design and Management

Theophilus Benson, Aditya Akella
Dept. of Computer Sciences
University of Wisconsin
Email: {tbenson,akella}@cs.wisc.edu

Dave Maltz
Microsoft Research
Email: dmaltzmicrosoft.com

***Abstract*—Anecdotal evidence and intuition suggest that an operator's ability to manage a network decreases as the network becomes more complex. However, there is currently no way to systematically quantify how complex a network's design is nor how complexity may impact network management activities. In this paper, we develop a suite of *complexity models* that describe the routing design and configuration of a network in a succinct fashion, abstracting away details of the underlying configuration languages. Our models, and the *complexity metrics* arising from them, capture the difficulty of configuring specific control and data plane behaviors in various routers. They also enable measurement of the inherent complexity of reachability constraints that a network implements via its routing design. Our models simplify network design and management by facilitating comparison between alternative designs for a network. To demonstrate their value, we use the metrics to perform a comparative study of the complexity of five different networks, including three university networks and two enterprise networks.**

I. Introduction

Experience has shown that the high complexity underlying the design and configuration of enterprise networks generally leads to significant manual intervention when managing networks. While hard data implicating complexity in network outages is hard to come by, both anecdotal evidence and intuition suggest that more complex networks are more prone to failures, and are difficult to upgrade and manage.

Today, there is no way to systematically quantify how complex an enterprise configuration is, and to what extent complexity impacts key management tasks. In this paper, we develop a family of *complexity models* that describes the complexity of the design and configuration of an enterprise network in a succinct fashion, abstracting away all the details of the underlying configuration language. We designed the complexity models to have the following characteristics: (1) They align with the complexity of the mental model operators must use when reasoning about their network—networks with higher complexity scores are harder for operators to manage, evolve or reason about correctly. (2) They can be derived automatically from the configuration files that define a network and its design. This means that automatic configuration tools can use the metrics to choose between alternative designs when, as frequently is the case, there are several ways of implementing any given policy.

The models we present in this paper are targeted toward the Layer-3 design and configuration of enterprise networks. As past work has shown [12], enterprises employ diverse and unique routing designs. Routing design is central both to enabling network-wide reachability and to limiting the extent of connectivity between some parts of a network.

We focus on modeling three key aspects of routing design complexity: (1) the complexity behind configuring network routers accurately, (2) the complexity arising from identifying and defining distinct roles for routers in implementing a network's policy, and (3) the complexity of the policies themselves (e.g. whether policies are conservative or permissive).

Configuration Complexity. To model the complexity of configuring a network's routing design, we use the *referential dependence* graph. This models dependencies in the definitions of routing configuration components, some of which may span multiple devices. We mine the graph to compute key metrics such as the number of atomic units of routing policy in a network, the configuration dependencies within and across units, and the number of units to which each router belongs.

Router Roles. We identify the implicit roles played by routers in implementing a network's policies. Networks become more complex to design and manage as the number of different roles increases or as routers simultaneously play multiple roles in the network. Our algorithms automatically identify roles by finding routers that share similar configurations. The algorithms also identify the roles played by routers in forwarding packets between different portions of a network. Using this, we are able to identify whether or not a network organizes its routing in a tiered fashion, the number of such tiers, and the number of routers in a tier.

Reachability Constraints and Implementation. We quantify the impact of the reachability and access control policies on the network's end-to-end connectivity. Networks that attempt to implement conservative policies that control which hosts can communicate are more complex to engineer and manage than more permissive networks. However, a network's policies are frequently not in any machine-readable form, and they cannot be directly read from the network's configuration. Our paper explains how the complexity of the policies can be automatically extracted by extending the concept of *reachability sets* first introduced by Xie *et al.* [15]. Reachability sets quantify the set of packets that a collection of network paths will allow based on the packet filters access control rules and routing/forwarding configuration in routers on path. By measuring the *volume* of the reachability set, which estimates the total number of packets contained in the set, we can expose how conservative the policies of a network are and also highlight the constraints imposed by control- and data-plane mechanisms. We develop algorithms based on firewall

rule-set optimization to compute reachability set volumes.

In addition to developing these metrics, another key contribution of our paper is **an empirical study of complexity** of network designs. We have applied the complexity models to the configurations of three university networks and two commercial enterprises. We find that the networks vary significantly in their routing designs, in the complexity of their configuration, the reachability policies implemented and the conservativeness of the policies. We are able to rank the networks in terms of their complexity. We find that while some networks define a small set of roles for their routers (making design and configuration very simple), others organize them in multiple groups containing a few routers each. We also find that while an unrestrictive network can be implemented with little complexity, operators may sometimes introduce complexity into their configuration as a means acheiving greater control over certain aspects of the network. Finally, we did not observe any correlation between the type of the network (University vs. Private network) and the overall conservativeness. Our empirical study shows that the metrics make it possible to directly compare distinct network configurations and argue about which design simpler and easier to manage.

This paper has 8 sections. Section II provides additional background and motivates the design of our metrics. In Section III, we discuss referential dependence and related complexity metrics. Section IV describes our approach to classifying routers in terms of their filtering roles. In Section V, we discuss complexity models pertaining to reachability policies. Section VI describes our empirical results. We present related work in Section VII and conclude in Section VIII.

II. Background

A. Controlling the Complexity of Networks

To ease the burden of managing and evolving their networks, operators use several common-sense strategies when designing and configuring their networks. In essence, the complexity measures we define in this paper quantify how well a network adheres to these strategies, and our evaluation shows how widely they are used.

Uniformity. To the extent possible, operators attempt to make their networks and configurations as homogenous as possible. Special cases not only require more thought and effort to construct in the first place, but often require special handling during all future network upgrades. The notion that uniformity makes networks simple pervades the definition of all our complexity metrics.

Roles. To limit the number of special cases operators must cope with, they often define a small number of archetypal configurations which they then reuse any time that special case arises. We call these archtypes *roles*. In Section IV, we provide a more formal definition of roles and outline algorithms for identifying them by looking for commonality in the configurations of the routers.

Tiered Structure. Operators often organize their routers into tiered layers to control the complexity of their network topology. For example, defining some routers to be border routers that peer with other networks, some routers to be core routers that are densely connected, and the remaining routers as edge routers than connect hosts.

Short Dependency Chains. Routers cannot be configured in isolation, as frequently one router's configuration will not behave correctly unless the configurations on other routers are consistent with it. We define this to be a dependency between those configuration lines. The set of all dependencies among the configuration files in the network forms a graph. Operators attempt to minimize the number and diameter of connected components in this graph. This is because making a change to one configuration file but not updating all the other configurations in that connected component will introduce a bug. In Section III, we discuss how to mine these dependencies. We also discuss metrics which quantify the extent of dependencies.

B. Overview of a Configuration File

```
1   interface Vlan901
2    ip address 128.2.0.225 255.255.255.248
3    ip access-group 9 in
4    ip pim sparse-mode
5    ip ospf priority 220
6   !
7   router ospf 1
8    router-id 128.2.1.133
9    passive-interface default
10   no passive-interface Vlan900
11   no passive-interface Vlan901
12   network 172.18.0.0 0.0.255.255 area 0
13   network 172.30.0.0 0.0.255.255 area 0
14   distribute-list in 9
15  !
16  access-list 9 deny   10.0.0.0 0.255.255.255
17  access-list 9 deny   172.16.0.0 0.15.255.255
18  access-list 9 deny   192.168.0.0 0.0.255.255
19  access-list 9 deny   169.254.0.0 0.0.255.255
```

Fig. 1. A sample configuration file.

The configuration file for a Cisco device consists of several types of stanzas, each encapsulating a different piece of the router's functionality. In Figure 1, we show a simple configuration file consisting of the three most relevant classes of stanzas: interfaces in lines 1-5, routing protocol in lines 7-14, and packet filtering in lines 16-19. Packet filters are widely known as ACLs due to the common "access-list" syntax for defining them (in Cisco's IOS, for example). The functionality exhibited by a router can be explained by the interactions between various instances of the identified stanzas.

Ingress filtering, i.e., preventing local hosts from sending traffic with IP addresses that cannot belong to them, has become a popular way to combat IP-address hijacking. Networks implement ingress filtering by defining a packet filter for each interface and creating a reference to the appropriate ACL from the interfaces. Line 3 exemplifies the the commands an operator would use to setup the appropriate references.

The purpose of any layer 3 device is to provide network wide reachability by leveraging Layer 3 routing protocols. Network wide reachability can be implemented by adding a routing stanza and making references between the stanza and the appropriate interfaces. Lines 16-19 declare a simple routing stanza with lines line 10 making a reference between this routing protocol and the interface defined earlier. Even in this

simple case, the peer routing protocol stanza on a neighboring device must be configured consistent with this stanza before routes can propagate routes between the devices and through the network. More complex reachability constraints can be imposed by controlling route distribution using ACLs. Line 14 is a filter used to control the announced received from the peer routing process on a neighboring router.

VLANs. VLANs are widely used in enterprises, but greatly complicate the behavior of the network by providing an alternative means for packets to travel between hosts that is independent of the layer-3 configuration. Most of the models we discuss for routing design must account for VLAN usage in a network. To provide the necessary background, we briefly discuss how typical VLAN usage and configuration.

In typical usage, each port on a switch is configured as layer-2 or layer-3. For each layer-3 port there is an interface stanza in the configuration file describing the properties of the port. Each layer-2 port is associated with a VLAN n, which we denote as V_n. The switches use trunking, spanning tree, and bridging protocols to ensure that packets received on a layer-2 port belonging to VLAN V_n can be received by every host connected to a port on V_n on any switch.

Layer-2 VLANs interact with layer-3 mechanisms via virtual layer-3 interfaces — an interface stanza not associated with any physical port but bound to a specific VLAN. Packets "sent" out the virtual interface are sent out the physical ports belonging to the VLAN using the rules of transparent bridging, and packets received by the virtual interface are handled using the layer-3 routing configuration. As the same VLAN can have a virtual interface on multiple routers, packets can flow into and out of the VLAN at multiple places in the layer-3 topology.

III. Referential Dependence

In the next few sections, we gradually develop our complexity metrics and models for routing design. In what follows, we develop a set of metrics which measure the complexity of *configuring* the routing design. These metrics measure the extent of referential dependencies in network configuration.

Referential Dependency Graph. The *dependency graph* captures references between stanzas in a configuration file (intra-file dependencies) as well as across stanzas in different configuration files (inter-file). Intra-file references are explicitly stated in the file. Examples of such references include the links in line 10 from a routing stanza to an interfaces, line 14 a routing stanza to an ACL, and finally in line 3 from an interface to the routing stanza. Inter-file dependencies are created when multiple routers refer to the same network object (e.g., a VLAN or subnet). For example, when the same subnet is mentioned in the configurations of several routers, it implies a relationship among those routers that the operator must take into account when configuring either. To capture the inter-file links, we use basic objects like VLANs and subnets as hints of links between stanza on different configuration files.

We use a two-step approach to parse configuration files and create a "configuration dependency graph":

1. Parsing, Symbol Table Creation. We parse each configuration file using a grammar we created which groups related configuration lines together into stanzas. Cisco documentation lists the commands that can appear within each stanza; Juniper identifies stanzas with $\{\ldots\}$. Using the grammar, the parser identifies the tokens in the configuration file that indicate a dependency between stanzas, and records these tokens in a symbol table along with the stanza in which they were found and whether the stanza defined the token or referred to it. For example, the access-list definitions in lines 16-19 of Figure 1 define the token ACL 9 and line 3 adds a reference to ACL 9. Our parser handles ACLs, interfaces, subnets, VLANs, and routing policies. The parser has rules to handle special configurations like line 9 that define a default behavior. In these special cases, the rules populate the symbol table with the appropriate references; for line 9, the rules adds references to interfaces into the symbol table.

2. Creating Links. In the linking stage, we create reference edges between the stanzas based on the entries in the symbol table. The table entries are used to create unidirectional links from the stanzas referencing the labels to the stanza declaring the label. Because every stanza mentioning a subnet or VLAN is both declaring the existence of the subnet or VLAN and referencing the subnet/VLAN, we create a separate node in the reference graph to represent each subnet/VLAN and create bidirectional references to it from stanzas that mentions it.

Metrics. Since our focus is on the routing design, we abstract a sub-graph specific to routing from the dependency graph. We define "complexity metrics" based on the sub-graph.

Using the referential graph, we first identify "routing instances" in the network [12]. A routing instance is a collection of routing processes of the same type in a network (e.g. all OSPF processes, or all BGP processes) which are all configured to be adjacent pairwise. There could exist multiple routing instances of the same type (e.g. OSPF instances); typically, these are configured to not distribute routes into each other directly. The referential dependency graph can be used to derive these adjacencies by tracing relationships between routing processes across subnets.

Each routing instance is an "atomic block" reflecting how the operators have organized the network's routing substrate to implement policy. The first complexity metric we employ is thus the *number of routing instances* in a network. The greater the number of instances the more difficult it is to track and configure the overall routing policy in a consistent manner.

Alongside this, we model the *organization of devices* across routing instances: consider two networks with the same number of devices and similar number of routing instances. The network where a large fraction of routers are configured to be part of a single instance, with small numbers routers being part of special routing instances, is simpler than the network where routing instances span multiple overlapping sets of devices each. To track this, we compute the *mean and the median number of routing processes* configured on each router.

Finally, we measure the *average complexity of configuration* for each router. For a router R, we use R's referential graph

to count the number of reference edges E_{P_R} which are required to fully configure a routing process P_R. In particular, E_{P_R} accounts for "reference chains" which are sequences of configuration stanzas that refer to one another. We compute the mean of $Total\_routing\_conf_R = \sum_{P_R} E_{P_R}$ across all routers R. The higher the value of this metric, the greater the configuration required to correctly set-up routing. In a similar manner, we can define the average complexity of configuration of a routing instance. This is the sum total of the number of reference edges required to correctly configure the routing processes pertaining to the instance on each individual routers. As before, we track reference chains within each router.

IV. Router Roles

When operators create a network, they typically start by defining a base set behaviors (e.g. filtering behaviors) that will be present across all routers and interfaces in the network. They then specialize the behavior of routers and interfaces as needed to achieve the objectives for that part of the network, for example, adding rate shaping to the dorm subnets, and additional packet filters to protect the administration subnets. Designers often implement these behaviors using *configuration templates* [4]. They create one template for each behavior, and the template specifies the configuration lines needed to make the router provide the desired behavior. Since the configuration might need to be varied for each of the routers, template systems typically allow the templates to contain *parameters* and fill in the parameters with appropriate values each time the template is used. For example, the template for an ingress filter might be as shown in Figure 2, where the ACL restricts packets accepted by interface 3 to those originiating from the subnet configured to the interface. The designer creates specific configuration stanzas for a router by concatenating together the lines output by the template generator for each behavior the router is supposed to implement.

```
interface III
   ip access-group 5 in
   ip address AAA SSS
access-list 5 permit AAA SSS
access-list 5 deny any
```

Fig. 2. Example of a configuration template

A common application of templates is to generate router ACLs which may apply at the control or data planes. Although templates simplify configuration of ACLs, it is quite possible that the ACLs must be modified by hand as the network evolves over time. Future operators who need to manage the network must start with the actual configuration stanzas themselves, not the templates that originally created them.

We hypothesize that we can work backwards from the text of the ACL configuration stanzas to retrieve the templates that created them. By doing so, we can measure the following issues related to the difficulty of configuring the filtering behaviors: how many distinct filtering behaviors are defined in the network routers? How many routers implement each behavior? Are there differences from the behavior template?

Copy-Paste Detection. We identify filtering behaviors that are shared across routers using a "copy-paste" analysis that looks for similar configuration stanzas on different routers.

We build this functionality on top of CCFinder [11], a tool that has traditionally been used to identify cheating among students by looking for text or code that has been cut and paste between their assignments. Briefly, the tool converts code portions to a standard form to detect copy-pasted code portions that have different syntax but have similar meaning.

We found that CCFinder by itself does not identify templates of the sort that may used in packet filter configuration (e.g., Figure 2). To discover templates, we first apply *generalization* that replaces the command arguments that may vary with wild card entries – for example, IP addresses are replaced by the string "IPADDRESS". Our implementation uses the grammar of the configuration language to identify what parameters to replace, and handles IP addresses, link weights, VLAN and interface names, ACL numbers, etc..

In addition to finding filtering templates that are shared across routers, we also use CCFinder to locate stanzas that are *identical* between routers. We refer to stanzas that are based on the same template as *shared-template filters*, and stanzas that are identical as *clone filters*.

Metrics. We define complexity metrics that capture the mental load placed on operators by the use of shared stanzas when configuring Layer-3 filtering. The metrics can be extended in a straightforward fashion to cloned filters.

The most important metric is the *number of behaviors*. Each template found through copy-paste analysis represents one behavior. As the number of behaviors increase, the basic complexity of the network increases.

Second, we model the *uniformity* among devices in terms of the behaviors defined on them. If all devices in the network exhibit the same behaviors because they all have the same shared-template or clone stanzas, then once an operator understands how one router filters control or data, he or she will understand how all the routers function. If devices contain different subsets of behaviors, each device will require individual study to understand the subset of behaviors.

We capture both uniformity and number of behaviors by identifying the *shared template-device set* of a behavior. This is the set of devices on which the configuration template for that behavior is present. We write the device set for a shared-template stanza as $ST_i = \{D^i_1, D^i_2, \ldots, D^i_{k_i}\}$ where the D^i_j represent a router that contains a configuration stanza generated from shared-template i. After obtaining the shared template-device sets, we scan them to identify identical sets. If two different shared-template stanzas are present on exactly the same set of routers, then the stanzas can be considered to have arisen from a single, larger template. The stanzas are merged and one of the device sets is discarded. The final number of distinct device sets which remain is the number of shared template behaviors, STB. To track uniformity, we simply compute the median and mean numbers of devices in the device sets. In a similar fashion, we can define clone device sets, the number of "cloned behaviors", CB, and the uniformity based on the clone filters.

V. Reachability Policies & Implementation

The common approach to implement reachability policies in networks is to classify hosts (or users) into groups and services, and then define reachability constraints that govern the groups' access to the services. A network's routing design plays a crucial role in implementing these constraints. In particular, the routing set-up can be used to influence reachability in two ways: (1) control plane restrictions, which limit whether or not routes exists between two subnets of a network (the subnets cannot communicate if no routes are present); and (2) data plane restrictions, which filter (i.e., drop) packets that match specified attributes. Upgrading or changing an enterprise's Layer-3 reachability constraints is challenging because both data and control plane functionality must be changed in a consistent manner across multiple routers. This is particularly difficult in networks where the reachability constraints themselves are highly sophisticated, meaning that users are divided into multiple small groups that differ in the constraints that apply to them.

We argue that based just on the network configuration files, we can model: (1) the reachability policies that a network has implemented on its end-to-end Layer-3 paths, (2) whether the constraints are conservative or permissive, and how this varies across network paths and network locations; and (3) the contribution of network control and data planes, and the role played by different routers, in constraining reachability.

Our models are based on the common framework of *reachability sets*, which we describe next.

A. Reachability Sets and Routing

Conceptually, a reachability set is the set of packets that a network will permit to travel between two points.

The reachability set for the path between two routers A and B, denoted by $\mathcal{R}_{All}(A, B)$, is the set of all packets that can originate on any of A's interfaces, traverse the $A \rightarrow B$ path, and leave via any of B's interfaces. Thus, the path's reachability set includes packets that are "sinked" by B (i.e. the packets are destined for subnets directly attached to B) as well as those that B forwards on to downstream routers.

We compute the reachability set of a network path using the following three steps: (1) compute valid forwarding paths between the network routers; (2) calculate the reachability set for each individual interface; and (3) compute reachability sets for end-to-end paths by intersecting the reachability sets for all interfaces along the path. In computing the reachability steps we consider three separate yet interacting mechanisms: control-plane mechanisms (such as routing protocols), data-plane mechanisms (such as packet filters on interfaces), and Layer-2 mechanisms (such as VLANs).

To implement the above steps we leverage ideas from Xie *et al.*'s models for static reachability in IP networks [15]. However, our approach differs from Xie *et al.*'s in three key ways. First, Xie *et al.* derive abstract set-theoretic representations that allow them to compute lower and upper bounds on the true reachability of the network. In contrast, we compute a single valid forwarding state for the network, and we estimate reachability constraints and their inherent complexity from this state. Second, we alter the representation of reachability sets using techniques from firewall rule-set optimization. These changes allow us to accurately compute the key attributes of reachability sets, such as the volume of the set, even for networks that have highly complex reachability constraints. Third, as mentioned above, our approach takes into account how VLAN usage impacts reachability in an enterprise network. Prior work does not model the effect of VLANs.

Next, we describe reachability set computation. Then, we define complexity metrics pertaining to reachability policies.

Routing Simulation. Our goal is to measure the complexity of the network's reachability policies, and doing so requires computing the reachability set between router pairs. The reachability set between a pair of routers depends on what path packets traveling between the routers will take, and this, in turn, is determined by the routing protocols run in the network.

Simulating a run of all the Layer-3 routing protocols within the network to compute the exact paths that packets will take through the network is challenging because of the diversity and complexity of protocol implementations. However, to compute the reachability sets, and hence the complexity metrics, it is sufficient to compute one single set of paths that are valid and internally consistent. A path between two routers is valid if it is among the paths the real routing protocols might produce, and the paths are internally consistent if there are no black holes that would not exist in the real network.

The paths in a network are determined by the contents of the routers' forwarding tables, or FIBs. Each FIB consists of entries that specify the interface out of which packets to a each destination subnet should be sent. To compute a set of valid and internally consistent paths, we begin by using the method of Xie *et al.* [15] to compute a *superset* of the actual contents of the FIB for each router by analysis of the router configuration files. For each destination subnet, a superset FIB lists *all* of the interfaces out of which the routing protocols might decide to send packets to that destination.

From the superset FIBs we construct a set of internally consistent paths. We provide a summary of the algorithm we use in the interest of space. For each subnet s directly connected to device d, create a Breadth First Search tree t rooted at d: (1) For each device d' with a link to d, check the FIB of d' for a route to destination s where the next hop is d. If such a route exists, then store d' as a child of d in the BFS tree t. If multiple such routes exist, pick the most specific route to simulate longest prefix matching in d'. Remove from d''s FIB all other entries that point to s. (2) Recursively execute step (1) on each of the leaves in t. The FIBs that result from the above algorithm contain a set of valid and internally consistent paths from every router to every reachable subnet.

B. Reachability Set Calculation

First we discuss the reachability set for a single interface and then discuss how to compute the set for an entire path.

Data Plane Reachability. For each interface i on a device d we define the set of packets that the data plane mecha-

nisms allow in that interface ($\mathcal{D}^{in}(i,d)$) and out that interface $\mathcal{D}^{out}(i,d)$). In and out are defined separately as the router configuration allows separate ACLs to be applied to incoming and outgoing packets.

Control Plane reachability. A device d will not send packets out an interface i unless its FIB contains an entry that directs packets to some destination out that interface. We define the control plane reachability $\mathcal{C}^{out}(i,d)$ as the union of all packets that d's FIB directs out interface i.

Reachability for a Path. To compute the reachability set $\mathcal{R}_{All}(A,B)$ of a path $A \rightarrow B$, we combine the per-interface sets for interfaces on the path. We first compute the following subsets: (1) For A, we compute the $Entry$ set which is the set of all packets that can potentially enter A (reflecting the inbound ACLs). In fact, we allow hosts on subnets attached to A to send packets with arbitrary source IP, port and protocol fields. (2) For B, we compute the $Exit$ set which is the set of all packets that can potentially leave B's interfaces (reflecting B's forwarding table and per-interface outbound ACLs). (3) For intermediate routers, we compute the intersection of the set of packets entering the device and those leaving it (reflecting inbound and outbound ACLs and the device's forwarding entries). The overall set $\mathcal{R}_{All}(A,B)$ is simply the intersection of $Entry$, $Exit$ and the intermediate sets.

Computing the reachability set to or from a VLAN requires an extra step, as there may be multiple virtual interfaces in the network where traffic can enter or leave the VLAN. We first compute the sets to and from each of the VLANs virtual interfaces as described above, and take the union of the sets to compute the reachability set to/from the VLAN as a whole. This works as VLANs are typically policy-free, meaning that packets are not filtered as they traverse the VLAN. If VLAN policy mechanisms become common, our approach must be extended to model the policy enforcement points.

C. Computing with Reachability Sets

Computing the properties of the reachability sets requires us to calculate intersections and unions of sets of packets, and these sets can be very large. To work efficiently with these sets, we represent each set as a linear series of rules like those used to define ACLs in the router configuration language, that is, a sequence of permit and deny statements that specify attributes of a packet and whether packets having those attributes should be allowed or forbidden, where the first matching rule determines the outcome.

ACL Optimization. Before computing any operation on reachability sets, we first optimize the ACL representation of the sets. This allows us to efficiently compute set operations.

ACL optimization is the process of taking a group of ACLs G and reducing them to an equivalent group G' which has the same functionality as G, but with two additional properties: (1) no two ACLs in G' have an overlap, and (2) the number of ACLs in G' is minimal. ACL optimization is commonly employed to reduce the number of rules and dependence across rules in firewalls and to make packet filtering more efficient. We employ standard techniques from firewall rule-set optimization to optimize ACLs [2], [7].

ACL Unions. The union of two ACLS is the set of packets that both accept. Given two optimized ACLs we merge the rules to create one ACL and then we optimize the resulting ACL to obtain the Union.

ACL Intersections. The intersection of two ACLs is the set of packets that both ACLs accept. Given two optimized ACLs a and b, we take each rule from a and compute its intersection with each rule in b. Each rule defines a range of allowed values for each dimension of the packet filter (e.g., source address). The intersection of two rules is a new rule that allows only the values from overlap between the two inputs. After computing the intersection of rules from a and b, we compute the union of the resulting rules and again apply ACL optimization.

D. Models and Metrics

To this point we have defined the reachability set of a network path between two routers. Next we describe metrics based on the computed sets that summarize the reachability constraints implemented by a network.

Most importantly, for the path between each pair of network routers, we compute the *conservativeness* of the path. Conservativeness estimates the restrictions on the communication between subnets attached to either end of the path. Second, we compute the *forwarding ratio* for a network path, which measures the extent to which different network routers contribute to end-to-end forwarding, and in turn to enabling end-to-end reachability. To compute either metric, we must first compute the *volume* of a reachability set.

Volume. The volume of a reachability set is the number of packets in that set. Assuming reachability decisions are made using the connection 5-tuple, we can think of a reachability set as a collection of regions in a 5 dimensional space with axes of source and destination address, source and destination ports, and protocol number. For example, a single `permit` ACL rule defines a hypercube in this space, as the rule specifies a range of values for each of the five dimensions (if a dimension is not mentioned in the rule, the range is assumed to be the minimum to the maximum allowed value).

As described earlier, our final reachability sets are composed of optimized ACLs. In an optimized ACL, the rules are non-overlapping, so the number of packets permitted by the ACL is the sum of the number of packets allowed by the ACL's permit rules. Since each rule defines a hypercube packet space, the number of packets permitted by a rule is found by multiplying out the number of values the rule allows on each dimension.

Conservativeness. We characterize the conservativeness of the network's policies applied to the path from $A \rightarrow B$ as the set of packets that can enter the network at A, travel to B, and that B "sinks", that is, packets whose destination address belongs to subnets directly attached to B. We denote this reachability set as $\mathcal{R}_{Sink}(A,B)$. The maximum size of this subset is $Max_{Sink} = 2^{32} \times 2^{|B|} \times 2^{16} \times 2^{16} \times 2^{8}$, where $|B|$ is the total number of IP addresses in the subnets attached to B. When first sent, the source addresses on the packets

could take any of the possible 2^{32} values — if mechanisms, such as ingress filters, are configured in the network to drop packets with illegal source addresses, this will be reflected during computation of $\mathcal{R}_{Sink}(A, B)$.

We define the conservativeness of a network path to be the ratio of the volume of $\mathcal{R}_{Sink}(A, B)$ to the value of Max_{Sink}. The closer this value is to 0, the more conservative is the network path. As we will see, networks with more conservative paths are more complex and more difficult to maintain.

To estimate the complexity of the control-plane alone, we recompute the reachability sets while ignoring data plane filters. We denote this set as $\mathcal{R}^{Control}_{Sink}(A, B)$ and use the ratio $\frac{\mathcal{R}^{Control}_{Sink}(A,B)}{Max_{Sink}}$ to evaluate the constraints imposed by the control plane on the $A \rightarrow B$ path.

Forwarding Ratio. Our second metric captures the extent to which a network router participates in forwarding end-to-end traffic. We define this metric for a path $A \rightarrow B$ as $\frac{\mathcal{R}_{Sink}(A,B)}{\mathcal{R}_{All}(A,B)}$, which is the fraction of A's packets "sunk" by subnets attached to B's interfaces, relative to the total number of A's packets leaving B's interfaces (including those sunk by B). We refer to this as the *forwarding ratio* for the $A \rightarrow B$ path. If the ratio is 1, then B does not forward traffic from A any further. If not, then B plays a role in forwarding A's packets to the rest of the network. Comparing the forwarding ratios for different network level paths allows us to identify the distinct roles different routers play in enabling end-to-end reachability.

VI. Empirical Study of Complexity

A. Enterprise Configuration Data

We analyzed the routing configurations of five US-based networks, all of which use Cisco routers. Three of the networks are large US universities: Univ-1 (12 routers) , Univ-2 (19 routers), and Univ-3 (24 routers). Two are commercial enterprises: Enet-1 (83 routers) and Enet-2 (10 routers).

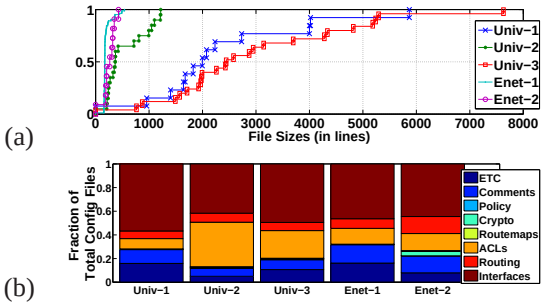


Fig. 3. Figure (a) displays the distribution of files in the various networks studied. Figure (b) shows the router functionality that accounts for the lines in the configuration files.(a)

Figure 3(a) plots the distribution of configuration file sizes for the five networks. The networks cluster into two groups: Univ-2, Enet-1, and Enet-2 consist of relatively small files, with 50% of their files being under 300 lines, while 90% of the files in Univ-1 and Univ-3 are over 1000 lines. As we will see, configuration file size is not a good predictor of network complexity, as Univ-1 is among the most complicated networks and Univ-3 among the simplest.

Figure 3(b) breaks down the lines of configuration by type. It shows networks differ significantly in the fraction of their configurations devoted to ACLs and routing stanzas. Univ-1, Enet-1 and Enet-2 spend the same proportion of configuration lines on routing stanzas and ACLs, while Univ-2 and Univ-3 define proportionately more ACLs than routing stanzas. Interface definitions, routing stanzas, and ACL definitions—the key building blocks for defining layer-3 reachability—account for over 60% of the configuration in all networks.

B. A Summary of Complexity Metrics

To briefly review, we define three metrics to measure routing configuration complexity: Configuration Complexity, which counts the number of links between routers in the configuration dependency graph and measures the complexity of configuring routing; and Shared Template and Behaviors, which count the number of routers with stanzas having the same pattern and measure the number of roles in the network. We define two additional metrics to measure the inherent complexity of a network's reachability policy. Conservativeness ($\frac{\mathcal{R}_{Sink}(A,B)}{Max_{Sink}}$) is the fraction of packets allowed by a path. Forwarding Ratio ($\frac{\mathcal{R}_{Sink}(A,B)}{\mathcal{R}_{All}(A,B)}$) measures the fraction of packets a router receives that it will forward, and is used to classify routers into tiers.

Next, we delve into the differences between the networks in terms of their design and complexity.

C. Complexity of Routing Configuration

First we study the complexity of the routing configuration. Our observations are summarized in Table I and column 8 provides a rank order of the complexity of the networks.

We envision that network designers or computer-assisted design tools could use our complexity metrics to compare alternate designs for given network. To convert the various complexity metrics we define into a single rank order we compare each pair of networks on each metric. If all metrics show one network is more complicated than the other, we order them appropriately. When metrics disagree we rank the networks of equal complexity. We ignore network size. The discussion below highlights several interesting comparisons and what can be learned from them.

Table I column 3 shows the number of routing instances for each network. The more routing instances, the more complex the network as each routing instance requires a distinct routing policy that the network operator must keep track of. To evaluate the complexity introduced in defining the routing policy, we look at the properties of the networks' routing referential dependency graphs.

We first compare the configurations of Univ-1 and Enet-2. The two networks have similar sizes, being the smallest networks in our collection. However, they differ vastly in their complexity. With only 2 more devices, Univ-1 contains many more routing instances than Enet-2. Furthermore, the definition of Univ-1's routing instances requires 2.5 times more references to other parts of the configuration as compared to Enet-2.

Network	# Routers	# Routing instances	Mean #Ref links per router	Mean ref links per instance	Median, Mean, Max # routers per instance	Median, Mean, Max # routing processes per router	Overall rank
Univ-1	12	14	41.75	35.8	1, 1.8, 12	2, 2.5, 4	4
Univ-2	19	3	8.3	58.3	3, 7.7, 19	1, 1.1, 3	3
Univ-3	24	1	4.1	99	24,24, 24	1,1, 1	2
Enet-1	83	10	7.5	62	1.5, 9.3, 71	1, 1.2, 3	4
Enet-2	10	1	1.6	16	7, 7, 7	1, 0.7,1	1

TABLE I
REFERENTIAL COMPLEXITY OF ROUTING CONFIGURATION. THE NETWORKS ARE RANKED FROM SIMPLEST (RANK 1) TO THE MOST COMPLEX.

Network	# Routing instances	# Data plane ACLs per instance		# Control plane ACLs per instance	
		Median	Mean	Median	Mean
Univ-1	14	2	3.6	2	2.5
Univ-2	3	6	51	0	1
Univ-3	1	20	20	0	0
Enet-1	10	2	3.3	2	2.8
Enet-2	1	6	6	0	0

TABLE II
ACL USAGE IN CONFIGURING ROUTING IN DIFFERENT NETWORKS. WE COUNT THE NUMBER OF DISTINCT ACLS USED.

N/w	# Rtrs	Shared template behaviors			Cloned behaviors			Overall rank
		Number	Device set size		Number	Device set size		
			Median	Mean		Median	Mean	
Univ-1	12	7	2	4.43	8	1	3.63	2
Univ-2	19	19	2	5.75	25	1	3.44	5
Univ-3	24	2	12.5	12.5	33	1	2	4
Enet-1	83	5	3	34.2	9	2	19.66	3
Enet-2	10	2	1.5	1.5	2	1.5	1.5	1

TABLE III
FILTERING BEHAVIORS IN THE DIFFERENT NETWORKS: DATA AND CONTROL PLANE FILTERS COMBINED.

Univ-2 is the next largest network with 19 routers. Although Univ-2 and Enet-2 both have few routing instances, Univ-2 has more complex routing policy as it requires a larger number of reference links per router (5X - see column 4) and per routing instance (4.5X - see column 5). Comparing Univ-2 against Univ-1, the number of routing instances is 5X smaller for Univ-2 (3 vs 14). Also, the average router in Univ-2 participates in fewer routing instances compared to Univ-1.

Our final network, Enet-1, has 83 routers. Univ-1 and Enet-1 contain an equivalent number of routing instances (14 and 10 respectively). However, Enet-1 requires on average 62 links to define each instance (column 5), and each of these instances is spread over 9 devices on average (column 6). Based on these metrics, we can predict that making routing-related changes will require manipulation of several routers.

In contrast to Enet-1, Univ-1 requires half as many links to maintain its instances (36) which are spread over fewer devices on average (2). However, when we look at the router level configuration requirements, we observe Univ-1 is more complex than Enet-1. In Univ-1, the average router belongs in more routing instances (2.5 vs 1.2, column 7) and requires many more reference links to maintain the routing configuration (42 vs 7.5 in column 4). Thus, despite their difference in size, the two networks have equally complex routing configuration.

D. Filtering Roles

We now examine the control and data-plane filters (i.e., ACLs) in the five networks. ACLs are important because data-plane ACLs directly drop packets; control-plane ACLs indirectly stop packets by dropping routing information; and the patterns of ACLs a router uses allows us to classify its role. Table II shows the number of *unique* ACLs that are configured per routing instance in the five networks. An ACL is unique if and only if it no other ACL contains the same sequence of deny and permit statements.

Interestingly, the networks in our data set can be categorized by their ACL usage pattern. Univ-2, Univ-3 and Enet-2 make little use of control plane filters but have extensive data-plane ACLs. In contrast, Univ-1 and Enet-1 use significant numbers of control plane filters but relatively few data-plane ACLs. The first set of networks defines fewer routing policy units (i.e. routing instances) than the second set. Thus, the first set depends primarily on data-plane mechanisms to define and control reachability in Layer-3, while the second set of networks employs a mixture of control- and data-plane mechanisms.

Next, we compare the networks according to the number of different roles they contain. Summarizing the discussion below, we find that networks become more complex as the number of roles increases, as measured by cloned and shared-template behaviors. The complexity increases further as the number of devices in each behavior *decreases*, as this fragments the routers into many small groups that the operators have to track and maintain identical filtering configuration within. However, our metrics still enable an automatic, direct comparison of the complexity of networks that are very distinct in terms of their size and routing design.

Table III shows the number of shared-template and cloned filtering behaviors in the five networks. Enet-2 has just two shared template behaviors and two cloned behaviors, making it the simplest and most regular of the five networks. Univ-1 and Enet-1 have similar numbers of shared and cloned behaviors. However, the number of devices per behavior is different: Enet-1 averages 20 devices per cloned behavior, compared to the 3.6 devices for Univ-1.

Univ-3 is a good example of a complex network with a large number of roles (two shared template and 33 cloned behaviors), each of these roles involving only a small number of routers. This forces the operators to be exceptionally careful when applying updates to the network, as the update's effect must be verified against each role.

E. Inherent Complexity of Policy

In this section, we show how our complexity metrics alone can be used to deduce the structure, function, and policies of a network. This is a dramatic improvement over the state of the art, where a human operator has to skim through the thousands of lines of router configuration files in order to piece together a picture of how the network functions.

Name	Num Routers	Conservativeness Metric			Control Plane Restrictions			Rank
		Mean	stdev	min	Mean	stdev	Min	
Univ-1	12	0.98	0.08	0.72	0.98	0.08	0.72	2
Univ-2	19	0.46	0.28	0.25	0.99	0.02	0.92	4
Univ-3	24	0.99	0.01	0.96	0.99	0.02	0.97	1
Enet-1	83	0.33	0.1	0.06	0.99	0.00	0.99	5
Enet-2	10	0.9	0.3	0.26	1	0	1	3

TABLE IV
CONSERVATIVENESS OF INHERENT REACHABILITY POLICY. NETWORKS ARE RANKED FROM OPEN (RANK 1) TO MOST CONSERVATIVE.

1) Conservativeness: First, we compare the different networks according to the conservativeness metric for the paths within each network. Our observations are summarized in Table IV. For each network, we show the mean conservativeness across all network paths, along with the standard deviation and minimum value of the conservativeness metric.

We note that our networks fall into two distinct classes: open networks with mean conservativeness ≥ 0.9, and closed networks with mean conservativeness of ≤ 0.5.

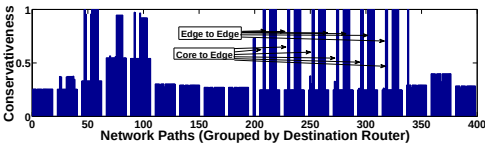
In the two conservative networks, Univ-2 and Enet-1, the average router-pair permits 46% and 33% of all possible packets, respectively, between subnets attached at either end. Referring back to Table I, we see that both networks implement very distinct control plane functionality to achieve roughly similar restrictions on reachability: While Univ-2 defined very few routing instances, Enet-1 made heavy use of control plane functionality (using 10 routing instances). From table II, we also see that the two networks differ significantly in their usage of control plane ACLs, with Univ-2 using almost none. However, both networks have one aspect in common: they are heavy users of data-plane filters (Table II).

The two most open networks, Univ-1 and Univ-3 allow unfettered access for over 98% of the packets sent between hosts in the network (on average). Although the average path in Enet-2 seems open (0.9), a significant number of paths impose serious restrictions. This is indicated by the minimum (0.26) and standard deviation (0.3) of the conservativeness metrics.

Univ-1, Univ-3 and Enet-2 are not overly conservative. However, they each use control plane functionality in a different manner to create the openness. As we saw in Table I, Univ-3 and Enet-2 use a single routing instance where Univ-1 uses 14 routing instances. We found that one main instance in Univ-1 was used to distribute routes between network devices, and another distinct instance was defined on each device. These instances announce selected routes for the locally connected subnets to the main routing instance for redistribution to other routers in the networks. Thus, to achieve the same reachability goals as Univ-3 and Enet-2, operators in Univ-1 trade routing configuration simplicity for finer grain control over redistribution of routes.

Although some networks are conservative, not all paths may be equally restrictive. Similarly, networks which are reasonably open may place severe restrictions on some paths. Configuring network-wide reachability accurately is a non-trivial task in such networks, since the lack of uniformity requires constant attention to ensure changes do not alter any of the different constraints. In our analysis, we found Enet-2 to be an example of a network with the latter style of policies, and Univ-2 to be an example of the former. To highlight this, Figure 4 displays the conservativeness for various router-pairs in Univ-2 and Enet-2. The paths in Univ-2 have widely varying restrictions. For example, paths from core to edge devices, as well as paths between pairs of edges (we discuss router roles further below) place no restriction on reachability. In the case of Enet-2, we note that paths from a single source router are all restricted to filter out 75% of the packets. This router has an egress filter which prevents it from forwarding packets to the space of Private IP addresses hosted by other routers in the network. All other paths are permissive. Thus, one can argue that Enet-2 has a simpler network-wide reachability policy.

(a) Univ-2



(b) Enet-2

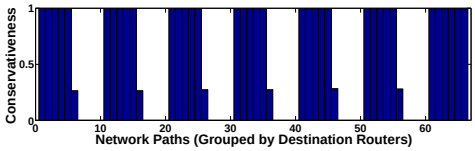


Fig. 4. Conservativeness profiles for Univ-2 and Enet-2. The paths in each network are grouped by the destination router.

Name	# Rtrs	Mean	stdev	min
Univ-1	12	0.88	0.23	0.14
Univ-2	19	0.45	0.45	0.001
Univ-3	24	0.87	0.30	0.02
Enet-1	83	0.15	0.19	0.03
Enet-2	10	0.31	0.1	0.15

TABLE V
FORWARDING RATIO IN DIFFERENT NETWORKS.

2) Role of the Control Plane: Next, we examine the role that control plane functionality plays in imposing reachability constraints in the five networks. We focus our analysis on Univ-2, Enet-1 and Enet-2, which had non-trivial values for the conservativeness metrics. We refer to Table IV.

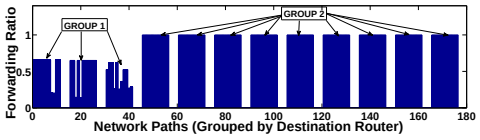
We first note that control plane functionality plays no role in constraining reachability in Enet-2 (the mean conservativeness metric for the control plane is 1). Thus, the small amount of restrictions placed by Enet-2 on its paths (the overall conservativeness metric averages 0.9) are via data plane filters.

Similarly, although Univ-2 is very conservative overall, we find that the control plane again plays little or no role in imposing the constraints (the conservativeness ratio for the control plane is 0.99). As shown in Table II, Univ-2 makes very heavy use of data plane filters.

3) Forwarding Ratio: Finally, we examine the forwarding ratios to identify the forwarding roles of routers in different networks and the number of distinct classes of forwarding roles. Our observations are summarized in Table V.

Enet-2 has low forwarding ratios overall: the maximum forwarding ratio itself is just 0.4 (not shown) and the minimum is 0.15. Thus, we can deduce that all routers in Enet-2 play roughly identical forwarding roles and there is no real distinction of core vs edge routers.

(a) Univ-1



(b) Univ-2

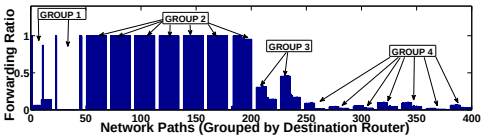


Fig. 5. Forwarding profiles for Univs 1, 2. The forwarding profile consists of the network level paths for the devices, group by the destination router.

Univ-1 and Univ-3 exhibit very similar forwarding ratio profiles, with high means of 87% and 88% respectively and moderately equivalent standard-deviations. The profile for Univ-1, with a high mean and relatively low standard-deviation of 23%, represents a network with a simple topology. A simple topology consisting of two levels of hierarchy, the first layer with core routers sinking at least 14% of the received traffic while forwarding the remaining 86% to the edge devices on the second layer. Univ-2 and Enet-1 have common forwarding ratio profiles, with means below 50%, and standard-deviations that are comparable to the mean. These networks seem to have significant diversity the roles played by routers, even when compared to Univ-1. In particular, the two networks seem to have relatively fewer routers which play the role of edge devices. To illustrate this point, in Figure 5, we compare the forwarding ratios for different paths in Univ-1, Univ-2. We note that, as mentioned above, Univ-2 contains roughly about 4 classes of devices: the edge (Group 2), the core (Group 1), intermediate-core (Group 4), and intermediate-edge(Group 3). Univ-1 consists of a two layered architecture with three core routers and nine edge routers, respectively labeled Group 1 and Group 2 in figure 5.

VII. Related Work

Models. To the best of our knowledge, our's is the first attempt at quantifying the complexity of enterprises. The notion of "complexity" has been explored in domains such as Software Engineering [13] and the metrics are similar to our referential dependency metrics. Recently, Ratnasamy has proposed that protocol complexity be used in addition to efficiency [14] to compare network protocols. Just as Ratnasamy's metrics help choose the right protocol, our metrics help pick the right network design.

Prior research has also developed abstract models of router configuration. In [12], Maltz et al., introduced abstractions for representing a network's routing design. As mentioned in Section III, we borrow from [12] the idea of a routing instance and use it as a way to group routing protocols. Our referential dependence graph is similar to the abstractions used in [4], [6]. Unlike [4], [6] our abstraction spans beyond the boundaries of a single device which allows us to define the complexity of network-wide configuration.

Measurement. In [12], the authors analyze the configurations of several networks and highlight different practices in routing designs. In contrast, we model complexity of routing design and understand the contribution of the reachability policies and router mechanisms toward complexity. Garimella et al.[8] examine the layer 2.5 structure of a campus network and discuss interactions with routing. Unlike [8], our study focuses on the paths generated by the layer 3 and layer-2.5 routing. Also, while Garimella highlight the causes and effects of inefficiencies in the paths themselves, we analyze the high-level reachability constraints on network paths.

Several studies have considered how to make network management simpler by building ground-up support (see [5], [9], [3]). We hope that our study can inform such ideas on clean slate alternatives. Finally, we believe that our metrics fits nicely with existing tools for configuration management such as AANTS [1] and OpenView [10], and can aid operators in making informed changes to their network configurations.

VIII. Conclusions

In this paper, we develop a set of complexity models that describe the routing design and configuration of a network in an abstract, succinct fashion. These models capture the difficulty involved in configuring the routing design, and they enable the automatic identification of roles each device plays in a network. We also develop models that describe the conservativeness and overall complexity of the policies implemented by a network within its routing framework.

We apply these metrics to conduct a unique empirical study of the complexity of five US-based networks. Using our metrics, we are able to reverse-engineer key design decisions made by network designers in implementing network-wide reachability constraints. Our empirical study demonstrates that the metrics make it possible to directly compare two distinct network configurations and argue about which is simpler and easier to manage. We believe that our complexity models can be integrated into automated configuration tools and used to compare design alternatives, as well as directly assist network operators understand and verify the properties of their networks as they conduct management tasks such as network upgrades and network-wide configuration changes.

References

[1] Authorized Agent Network Tool Suite (AANTS). http://www.doit.wisc.edu/network/upgrade/faq/aants.asp.
[2] S. Acharya, J. Wang, Z. Ge, T. Znati, and A. Greenberg. Simulation study of firewalls to aid improved performance. In *ANSS '06*.
[3] H. Ballani and P. Francis. CONMan: A Step towards Network Manageability. In *Proc. of ACM SIGCOMM*, 2007.
[4] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The cutting EDGE of IP router configuration. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, Nov. 2003.
[5] M. Casado, M. Friedman, J. Pettitt, N. Mckeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *SIGCOMM '07*.
[6] X. Chen, Z. M. Mao, and J. van der Merwe. Towards automated network management: network operations using dynamic views. In *INM '07*.
[7] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *INFOCOM 2000*.
[8] P. Garimella, Y.-W. E. Sung, N. Zhang, and S. Rao. Characterizing vlan usage in an operational network. In *INM '07*.
[9] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM Sigcomm CCR*, 2005.

[10] Hewlett-Packard. Enterprise Management Software: HP OpenView. http://h20229.www2.hp.com/.
[11] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28(7):654–670, 2002.
[12] D. A. Maltz, J. Zhan, G. Xie, G. Hjalmtysson, A. Greenberg, and H. Zhang. Routing Design in Operational Networks: A Look from the Inside. In *ACM SIGCOMM*, 2004.
[13] T. McCabe and C. Butler. Design Complexity Measurement and Testing. *Communications of the ACM*, 32(12), 1989.
[14] S. Ratnasamy. Capturing Complexity in Networked Systems Design: The Case for Improved Metrics. In *HotNets*, 2006.
[15] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *Proc. IEEE INFOCOM*, Mar. 2005.