

To Automate or Not to Automate: On the Complexity of Network Configuration

Sihyung Lee Tina Wong Hyong S. Kim
Carnegie Mellon University
{sihyunglee,tinawong,hskim}@cmu.edu

Abstract—Configuring a network is a low-level, device-specific task. Many have compared it to writing a distributed program in assembly language, reserved only for highly experienced network operators. Automation has been proposed by researchers and industry as the solution to problems in network configuration. However, there is a certain amount of resistance from the operator community against automation. On the one hand, operators do desire a way for network-wide configuration. On the other hand, they still like to have access and control to details, to ensure flexibility and for debugging. In this paper, we attempt to answer the question “How should we automate network configuration?” by studying where the complexity lies in network configuration. With an operational perspective, using data from three different types of production networks, we analyze the configuration files from these networks over the span of up to two years. Our analysis shows that the majority of changes to these files are a few lines each and made frequently. We found that routing, especially its policies, constitute a significant portion of the configuration files, as well as modifications to them. We then present complexity models to measure network-wide risk, impact and duplication of routing policies in network configuration. We show that risk and impact tend to grow over time, and the duplication factor is high. Based on the results of our analysis, we propose ways to automate the complex parts of network configuration.

I. INTRODUCTION

It is well known that networks are difficult to manage and operate. Companies spend more resources on the daily management and operations of their networks than on new IT services. In particular, configuring a network can be a daunting task. Usually, it is performed manually, by logging on to and manipulating each router separately. There are commercial products that help to track changes in the configuration and keep router software up-to-date, but most of these products only work in networks with devices from a single vendor, which is not always a practical requirement. On top of that, there can be hundreds of devices, thus hundreds of configuration files, in a network, each with thousands of commands. A change in one device can potentially affect other devices, or even the whole network. Often, multiple devices need to be re-configured to make a relatively minor change in the network.

Studies on networks have shown that misconfigurations are common and can significantly affect the correct operations of networks. Mahajan et al. [8] observe that misconfigurations in BGP routers can lead to unintended production or suppression of BGP routing announcements in up to 1% of the global routing table. Wool [10] conducted a quantitative study on 37 enterprise firewalls, and found that all of them have some form

of misconfigurations. Our own studies [7] on router configuration files confirm this observation. As a network evolves, its configuration become even more difficult to understand, extend and debug. Patches are sometimes put into configuration files during firefighting to temporarily deal with a problem, and forgotten and left in place after the pressure of the situation is lifted. Personnel changes mean that configuration files are edited by multiple engineers with different backgrounds and working styles. Also, because of the low-level nature of router configuration commands, the same high-level goal can be achieved in multiple ways in the configuration.

Automation has been proposed as a solution to problems in network configuration [4], [3], [5]. The goal in automation is to allow a human to specify a high-level intent for its network, and the network along with all its devices are configured automatically by the network management software. While attractive, some have pointed out that automation may not always be the answer. Brown and Hellerstein [2] argues that automation can sometimes increase the costs of IT operations if the task being automated does not need to be repeated often, and the overhead of creating and executing the automation software is high. HCI studies in systems administrations have shown that automation can hide information necessary during decision making, thus it should be designed carefully [6]. Blindly automating all tasks that are simple and intuitive can hurt network operations in the long run, because an engineer can start to lose context of what is going on in a network.

In this paper, we aim to answer the question “How should we automate network configuration?” by studying where the complexity lies in network configuration. § II contains background information on network configuration. § III describes our datasets. § IV discusses our complexity model and the application of the model onto the datasets. The model characterizes four different aspects of network configuration: the size of the configuration per each function, the frequency and size of the configuration changes over time, the degree of dependencies, and the degree of similarities among network elements. Based on the results of the analysis, we propose ways to automate network configuration in § V. We then conclude in § VI.

II. NETWORK CONFIGURATIONS

In this section, we give brief background on network configuration. A network configuration is a distributed set of commands such that each device has a subset of commands

```

1  router bgp 10
2  neighbor base peer-group
3  neighbor base remote-as 20
4  neighbor 20.1.1.1 peer-group base
5  neighbor base prefix-list pf_base in
6  neighbor base route-map from_base in
7  neighbor base route-map to_base out
8
9  ip community-list deny-list permit 10:666
10
11 ip prefix-list pf_base permit 20.1.1.0/24
12
13 route-map from_base permit 10
14 set local-preference 30
15 set community 10:100 10:200
16
17 route-map to_base deny 10
18 match community deny-list

```

Fig. 1. Excerpt of a Cisco router configuration file.

called a configuration file. A configuration file can be divided into different segments, each of which instructs how each segment of the device should operate, which protocol should be running, and the options allowed by the protocol. Examples of these different segments include interface, access control, intra-domain routing, inter-domain routing, packet filtering, Quality-of-Service parameters, traffic engineering, and general management functions.

Each router vendor has its own proprietary configuration language. Some configuration features are vendor-specific and not available in another vendor. In the rest of this paper, we use Cisco IOS terminology and syntax, but we have applied both our methodology and implementation for Cisco IOS and Juniper JUNOS configuration languages. Figure 1 is a factitious excerpt of a router configuration file. We use it to illustrate how routing protocols and policies are defined.

Lines 1-7 configure the BGP routing process. The number 10 is the local autonomous system (AS) number. In line 2, the command `peer-group base` creates a group called `base`. `peer-group base` facilitates the application and modification of routing policies on a set of neighbors. Line 3 associates the AS number 20 to `base`, which tells us whether the BGP session is with external or internal neighbors. Line 4 assigns the BGP neighbor with IP address 20.1.1.1 to the peer group `base`. Line 5 applies a prefix list called `pf_base` to all BGP sessions of `base`. `pf_base` is defined in line 11. The keyword `in` specifies the prefix list is applied to incoming advertisements. Therefore, only 20.1.1.0/24 is accepted from the peer group `base`. Lines 6-7 apply the import policy `from_base` and export policy `to_base` to the BGP sessions in `base`, which are defined in lines 13-15 and lines 17-18, respectively. `from_base` accepts all routes and assigns them a local preference value of 30. It also tags the routes with two communities 10:100, and 10:200. `to_base` denies routes which carry the community tag listed in the community list `deny-list`, defined in line 9, which corresponds to the community 10:666. Consequently, routes marked with 10:666 are not advertised to BGP neighbors of `base`. Community list `deny-list` and prefix list `pf_base` can be referred to by other import or export policies. Likewise,

Network name	Time-span	Num. of routers total (IOS,JUNOS)	Num. of LOC (min,max)
Net-DC	Jun 04-Mar 06	44 (37,7)	(230,3060)
Net-HPR	Jun 04-Mar 06	6 (6,0)	(455,3638)
Net-UCB	Mar 06-May 06	67 (65,2)	(92,1688)

TABLE I
SUMMARY OF NETWORKS USED IN EVALUATION. LOC STANDS FOR LINES OF COMMANDS.

import policy `from_base` and export policy `to_base` can be reused by multiple peers if the peers apply similar policies.

III. DATASETS

Our dataset is comprised of configuration files from a regional network access provider (Net-DC), a state-wide research network (Net-HPR), and a large university campus network (Net-UCB). Net-DC serves a large base of universities and research institutions. It peers with hundreds of commodity peers and buys services from multiple upper tier providers. Net-HPR is a smaller research network providing advanced services for experimental application users. Net-UCB is a stub network that buys service from a single provider. Information related to the networks is summarized in Table I. Two of the networks include both Juniper and Cisco routers and our study considered both types of routers. We collected the router configuration files from Net-UCB over a period of 3 months and the ones from Net-DC and Net-HPR for 21 months. Daily file snapshots were captured for Net-DC and Net-HPR, and bi-weekly snapshots for Net-UCB.

Even though we analyzed only three networks, we believe that the dataset is representative of existing networks and covers different architecture types. Our data include the four types of relationships between autonomous domains: customer-to-provider, provider-to-customer, peer-to-peer and sibling relationships.

IV. COMPLEXITY MODEL

This section presents our network configuration complexity analysis.

A. General Patterns

We first identify the most commonly utilized commands in the configuration files of our datasets. We use the Lines of Code (LOC) metric, which has long been used to measure software complexity by the software engineering community. The number of LOC in our case counts the number of commands and parameters in a configuration file. We found that interface definitions and routing protocols, especially routing policies, have substantially more LOC than other segments of network functions. In large size files, we found that the section related to the configuration of BGP routing policies can contribute to up to 66% of a file. In particular, commands related to route tagging (in Cisco IOS, `community`, `community-list`, `community members`, `ip route tag`) are predominant in the three networks. Commands related to route filtering (in Cisco

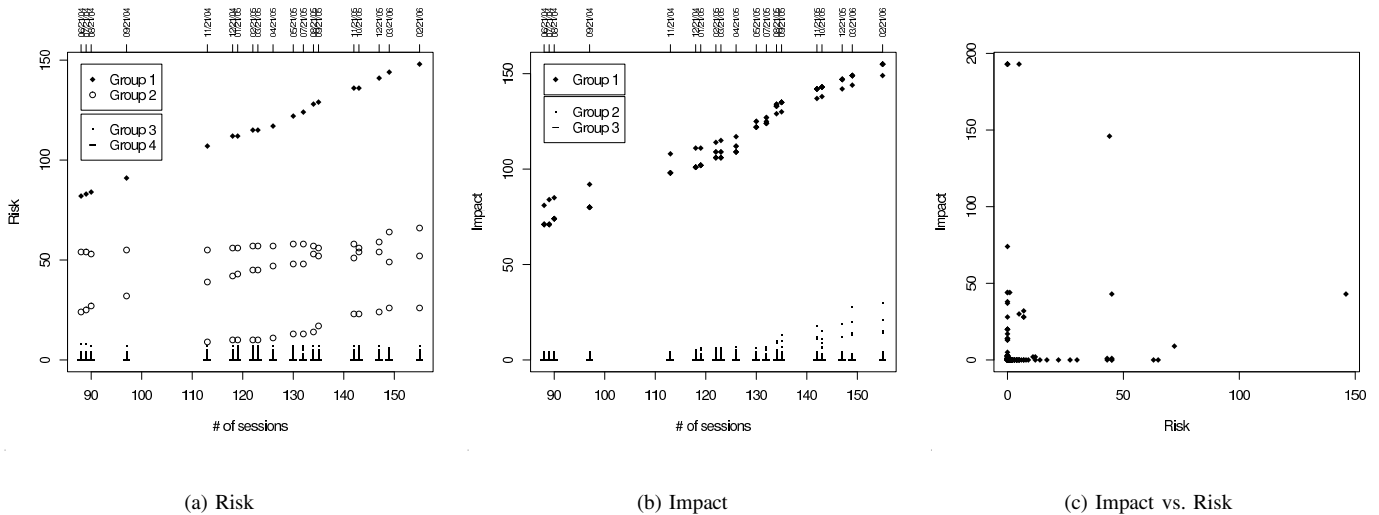


Fig. 5. Complexity of communities used in Net-DC with its commodity peers in (a) and (b), and with all its neighbors in (c).

c. The risk \mathcal{R} and impact \mathcal{I} of a community c are defined as, respectively:

$$\mathcal{R}(c) = \Omega\{b : b \in B \wedge p_{set}(c, M_{in}(b))\} \quad (1)$$

$$\mathcal{I}(c) = \Omega\{b : b \in B \wedge p_{match}(c, M_{out}(b))\} \quad (2)$$

Figure 4 shows an example of the two metrics. The network includes five routers, $\{R1, R2, R3, R4, R5\}$, and uses two communities, \mathcal{P} and \mathcal{C} . \mathcal{P} is set from two routers, $\{R1, R2\}$, and matched on three routers, $\{R3, R4, R5\}$. Consequently, there are two locations where \mathcal{P} can be mistakenly set on route advertisements and such mistakes can affect route redistribution from three routers. The risk and impact of \mathcal{P} are therefore 2 and 3. The other community \mathcal{C} can be set from all five routers. Thus, the risk=5 is greater than \mathcal{P} . However, the impact=1 is less since \mathcal{C} is matched by only one router, $R2$.

Figures 5 (a, b) depict the risk and impact associated with the communities used in the sessions connecting Net-DC with its commodity peers over the span of 21 months. In Figure 5 (a) (resp. b), each dot represents a community. The y-axis represents the number of BGP sessions that sets (resp. matches) that specific community. The top x-axis shows the dates of configuration snapshots, while the bottom x-axis illustrates the scale of the number of BGP sessions between Net-DC and its peers. For example, in the snapshot dated 11/21/04, there were about 113 sessions between Net-DC and its peers.

Figure 5 (a) shows four groups of communities:

- Group 1 consists of a single community, present in most BGP sessions between Net-DC and its peers. This community is used to mark the routes learned from peers. Its risk increases over time as the number of peers grows.
- Group 2 comprises of a number of communities indicating the source of the routes. More specifically, they designate the routers where the routes are learned. Their

respective risk also increases with the number of BGP sessions with each router.

- Group 3 contains communities with lower risk. These communities typically uniquely identify the peers (at the level of AS) for routes customization.
- Group 4 indicates a risk level of 0. These communities correspond to the tags that are matched but not set. They comprise the communities that are set to routes originated internally or set in external networks.

Figure 5 (b) shows the impact of the communities used in sessions between Net-DC and its peers. Three main clusters of communities stand out:

- Group 1 consists of two communities. They are used to specify the routes to filter and the ones to advertise to all the peers. The impact of these two communities increase with the number of peering networks as they need to be present in all corresponding BGP sessions.
- Group 2 is a set of communities used for customization. Certain routes from different customers are advertised to specific classes of peers but not others.
- Finally, Group 3 indicates an impact of 0. We correlated some of these communities between the routing policies in Net-DC and Net-UCB and found that some of these communities are used for inter domain routing. Set in Net-DC, the communities are matched in Net-UCB.

The graph representing the risk and impact of the communities used with customer networks present similar patterns.

Finally, Figure 5 (c) represents the impact and risk of all the communities used in Net-DC for the latest snapshot (March 2006) of the configuration files. It highlights the existence of two sensitive communities. One presents a risk of 50 and an impact of 150 while the second represent a risk of 150 and an impact of 50. Both communities serve to coordinate routes between a set of customers and a set of peers.

D. Duplication Patterns

We investigate the degree of duplication in configuration files. We first start by studying the frequency of appearance of a same community list and prefix list definitions among routers. We find that up to 91% of the defined community lists and 54% of the defined prefix lists may appear in at least two routers. These numbers show that same definitions of both community lists and prefix lists are being re-used across routers. Also, 13 community list and 15 prefix list definitions were present in more than 25% of the routers in one of our network. As such, it appears that large fractions of routers re-use common definitions.

We introduce a duplication metric to assess the degree of similitude among routing policies defined in the routers of a network. Routing policies are typically defined as a sequence of conditions and actions: routes matching a set of conditions receive the specified actions. Each router vendor supports a wide range of conditions (“match statements”) and actions (“set statements”). We focus on four primitives: prefix list, AS path, community and local preference. These four primitives are the most commonly used in the match and set statements in our analysis.

Our duplication metric models a routing policy definition in the following way. We convert each command line of a routing policy into a token, taking into consideration the direction of the routing policy (in, out) and the action performed on the route (e.g. permit, deny). In other words, a token represents an action in a routing policy. We call a set of tokens (i.e., actions) shared by at least k sessions, a pattern. As an example, the routing policies of Figure 1 are converted into the following five tokens:

```
in_permit_set_localpref_30
in_permit_set_comm_10_100
in_permit_set_comm_10_200
out_deny_match_comm_10_666
```

We define $k = \max(0.05 \times \Omega(B), 2)$, where $\Omega(B)$ is the total number of sessions. The number of patterns is indicative of the degree of similitude between routing policies. The size of the clusters represents the number of sessions presenting a specific pattern. Finally, the size of each pattern illustrates the number of common actions in the shared policies.

The analysis reveals many patterns in each network. Net-DC presents more than 400 patterns. The patterns themselves can be large, including up to 40 common actions. Analysis of these patterns reveals that most of the actions present in large patterns are related to BGP communities. In many instances, groups of actions related to communities are performed together.

Figure 6 further presents the distribution of the discovered patterns for Net-HPR. We found that:

- 1) Most patterns are shared by small clusters of eBGP sessions. However, because of the large number of eBGP sessions, a cluster size of 10% still includes more than 20 eBGP sessions.
- 2) A number of patterns are present in more than half of

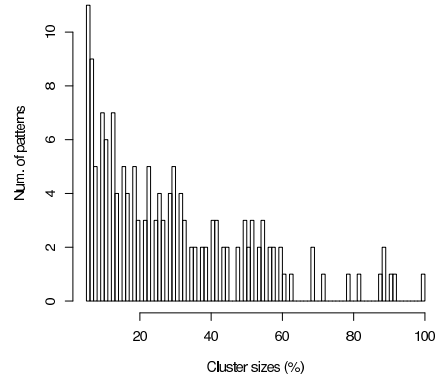


Fig. 6. Duplication of routing policies across Net-HPR.

the eBGP sessions. In Net-HPR, some routing policies are applied to all eBGP sessions. These actions are related to the filtering and distribution of routes through communities.

The number of discovered patterns and the number of BGP sessions using the same pattern suggest that similar routing policies are applied across large subsets of BGP sessions. This required degree of consistency in turn implies that the risk of misconfiguration is elevated and exacerbated by current network management techniques in which each router is configured manually and separately.

V. WHAT TO AUTOMATE?

Small “tweaks” (15 lines or less) constitute a substantial portion of configuration changes in our networks. Moreover, single line changes range from 16% to 37% of all changes. These incremental changes could also be from an operator slowly rolling out a significant change to ensure continuous availability of the network. In either case, this finding has a number of implications. First, automated network management software should provide peepholes to low-level details of a network, even if the common case is for the software to hide these details. Second, it is neither worthwhile nor desirable to automate all segments of network configuration. The flexibility of using individual commands for testing and debugging should remain.

We found that routing policy configurations impose risk and impact on the network. To facilitate the configuration of complex routing policies, the BGP community attribute was introduced by IETF. It is highly expressive, and allows an operator much flexibility in its use. However, community values are typically defined in an ad-hoc manner and coordinated across routers and autonomous domains manually [11]. The complexity of BGP community usage (or any form of route tagging and filtering) may not be apparent during configuration or examination of configuration files. This complexity can grow over time, as we have shown, especially when a network expands in size and function. The risk and impact complexity metrics defined in this paper can help to identify routing policy configurations in a network that should be carefully

maintained. More importantly, an operator can evaluate the complexity of a routing policy configurations in terms of these metrics, and simplify it if possible, before deployment. This network audit can be easily automated to help in the maintenance and management of a network's configuration files.

We found that high degrees of duplication exist in routing policy configurations. The duplicated patterns of routing policy configurations observed in our complexity model point themselves to automation. Note that the duplicated patterns represent subsets of a router's complete routing policy. In other words, each router's complete routing policy can be unique, but portions of the policy can be shared (thus duplicated) across many routers in a network. This means an automation engine may not be able to simply duplicate one router's policy onto a new router in a network. The commonly implemented approach is to parametrize the automation engine – it asks the operator to provide information about the configuration being automated. However, this approach has the drawback discussed in [2], in which the overhead of executing the automation engine is higher than the savings it provides. The main observations in this paper point to another approach. During configuration, the network management software can automatically perform data mining to find duplicated patterns related to the configuration task and presents them to the operator. This provides the operator visibility to and control of low level details, essential in incremental modifications, while allowing the operator to easily copy-and-paste configuration commands from elsewhere in the network.

A. Intent-based Network Configuration

Researchers have observed that it is not obvious to translate a simple high-level intent into low-level implementation of configuration commands. The configuration commands are designed by router vendors, with routing protocols and router processing in mind, and do not necessarily have the appropriate constructs for network engineers to specify their intent. Many (for example, [9], [1]) have proposed configuration languages and systems to combat this problem. Users specify their intent in these high-level languages and the systems automatically translate them into low-level commands.

Our analysis results provide several insights to the design of intent-based network configuration languages and systems. We found that configuration changes are made incrementally and frequently. Intent-based configuration should allow operators to specify their intent not only in the spatial manner but temporally as well. This means intent-based languages should provide constructs that describe step-by-step deployment or function roll-out, for example. We observed that network configuration can increase in complexity over time. Intent-based systems should provide ways to make sure originally specified intent remain the case over time, and different intents do not conflict with one another. Our results show that routing policy configurations can particularly benefit from high-level abstraction available through intent-based configuration. Instead of using low-level commands such as route tagging and

filtering, the corresponding intent, such as “do not propagate routes from peer A to peer B” should be explicitly configured.

VI. CONCLUSIONS

We characterize the complexity of the configurations from five production networks in four different dimensions: the size of the configuration by each segment, the size and frequency of changes over time, the degree of dependencies, and the degree of similarities across the network. We find that routing policies represent a dominant part of the configurations. The routing policies mostly address the route tagging and filtering. The commands related to these features are frequently modified, almost daily. Interestingly, 80% of modifications involve 15 lines of the configurations or less. This suggests that most modifications are either small or incremental changes. Based on these observations, we study the routing policies in terms of their dependencies and similarities across the network. We find that route tags have a high degree of dependencies. A small mistake on the tags impact the route advertisements to more than 100 peers. We also find that more than 400 routing policy patterns are common across 20-100% of eBGP sessions. We draw the following conclusions from the results. First, operators use individual commands to make small changes in daily operations. These commands will be used despite the degree of automation. Second, we should be able to describe and automate policies that need to be deployed incrementally. Third, the automation can perform periodic audits and classify network elements by highlighting ones with high impact and risk. The operators can thus prioritize the level of responses to these classified network elements accordingly. Finally, the automation can data-mine common patterns and present the patterns to the operator as possible configurations. The patterns can also be used to ensure that the common policy is configured consistently across the network.

REFERENCES

- [1] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A Novel Firewall Management Toolkit. In *ACM Transactions on Computer Systems*, 2004.
- [2] A. B. Brown and J. L. Hellerstein. Reducing the Cost of IT Operations – Is Automation Always the Answer. In *HotOS X*, 2005.
- [3] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The Cutting EDGE of IP Router Configuration. In *HotNets-II*, Boston, MA, November 2003.
- [4] A. Feldmann and J. Rexford. IP Network Configuration for Intradomain Traffic Engineering. *IEEE Network Magazine*, 2001.
- [5] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. Automated Provisioning of BGP Customers. *IEEE Network*, 2003.
- [6] D. Hrebec and M. Stiber. A survey of system administrator mental models and situation awareness. In *SIGCPR*, 2001.
- [7] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb. Minerals: Using data mining to detect router misconfigurations. In *ACM Sigcomm Workshop on Mining Network Data*, Sep 2006.
- [8] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In *Sigcomm*, August 2002.
- [9] A. Mayer, A. Wool, and E. Ziskind. Fang: A Firewall Analysis Engine. In *IEEE Security and Privacy*, May 2000.
- [10] A. Wool. A Quantitative Study of Firewall Configuration Errors. *IEEE Computer*, June 2004.
- [11] M. Yannuzzi, X. Masip-Bruin, and O. Bonaventure. Open Issues in Interdomain Routing: A Survey. *IEEE Network Magazine*, Nov. 2005.