# Analysis of Operator Errors in Routing Policy Configurations

Sihyung Lee

*Seoul Women's University, South Korea*
*sihyunglee@swu.ac.kr*

***Abstract***

*Routing policy configurations are one of the crucial element of network configurations since they deal with a network's connectivity, quality of service, and security. However, the languages and user interfaces used to configure routing policies are not well suited to network operators' needs. This often leads to configuration errors and lengthens the time taken to resolve problems. To better understand the causes of this problem, we analyze configuration errors that we collected in four production networks over an eight-month period. We also learn from network operators, who provide feedback about the features that cause mistakes and delays. We observe that current routing policy configuration management have four major problems, including both technical and usability problems: (i) the large number of obsolete and irrelevant configurations, (ii) subtle interactions with multiple relevant technologies, (iii) the overlapping, complex set of configuration options, and (iv) insufficient support for the efficient reuse of common configuration segments. Based on this observation, we propose a set of guidelines for creating more usable configuration management.*

***Keywords:*** *Network Management; Network Configuration; Routing Policy Configuration; Design Guidelines; Usability Analysis*

## 1. Introduction

Configuring routing policies in a network is crucial for network operations and needs to be performed accurately and quickly. Routing policies control the primary functions of the Internet, which include providing connectivity between different nodes, reducing delay by distributing Internet traffic over multiple paths, and disallowing illegitimate traffic for an improved level of security. The process of configuring routing policies must be done quickly since changes are made frequently, often more than hundreds of times in a week. At the same time, this process has to be performed with extreme caution since a simple configuration error can cause cascading failures across an entire infrastructure, leading to global connectivity disruptions and heavy financial losses. For example, a routing policy misconfiguration in AS (Autonomous System) 9121 brought down tens of thousands of networks for more than ten hours [1].

Even though configuration of routing policies needs to be accomplished quickly and accurately, this process is inherently labor-intensive and highly error-prone [2, 3]. The main reason for these difficulties is known to be the low-level and distributed nature of routing policy configuration. This low-level nature results in configurations that are larger than thousands of lines of commands in each router. Also, network configurations are decomposed in multiple routers as in distributed programs, and these distributed pieces are dependent upon one another. Due to these dependencies, the configuration of a single router commonly affects the behavior of multiple routers. Therefore, such a configuration requires a complex plan that involves more than a

single router. For example, to configure the high-level routing policy "*allow our network to carry traffic from network A towards network B*", we first need to map this policy to a set of low-level commands ("*receive routes from network B*", "*attach a tag T to these routes*", "*pass any routes with the tag T*", and "*re-advertise a route to network A if the route has the tag T*"). A different subset of these low-level commands should then be properly configured in each of the nodes that the traffic may go through. This simple example demonstrates the length and complexity of configuring routing policies.

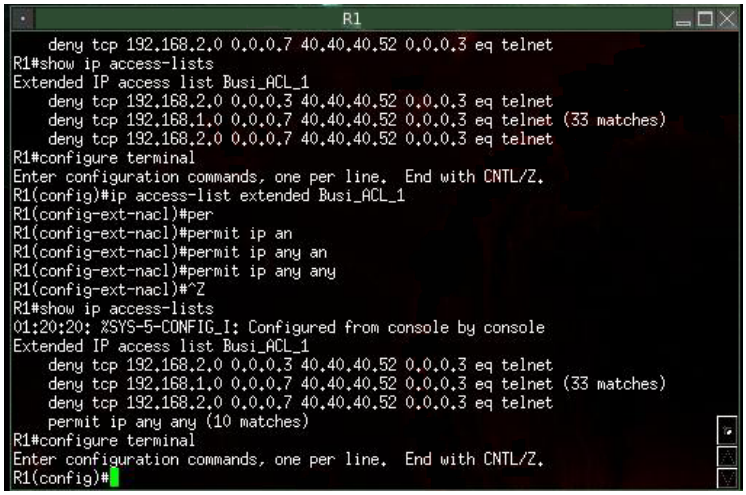



**Figure 1. Command Line Interface Showing Cisco IOS Configuration Commands**

Previous works [4-7] propose high-level configuration management for routing policies, but none fully addresses the challenge of configuring routing policies nor are their proposed methods widely used. High-level management would allow network-wide goals to be specified in a central place. Such high-level management would also present routing policies in a high-level form, and this would reflect an operator's high-level goal, rather than relying on low-level configuration commands, which implement the goal. However, most of the proposed high-level management requires manual interpretation and description of many low-level details, such as IP address blocks, BGP (Border Gateway Protocol) communities, and local preference values. Hiding these low-level details is difficult for two main reasons. First, certain parameters cannot be automatically determined unless the Internet architecture and protocols change. For example, a new IP address block is assigned to a network by the Internet Assigned Numbers Authority (IANA) and is manually configured in the network. Second, user studies show that network operators want to be able to access low-level details even in the presence of high-level configuration management [8]. This preference for low-level details stems from the desire to be able to control details. For example, being familiar with low-level details helps operators quickly debug a problem if it turns out to be related to low-level details.

Our work complements previous approaches in that it is aimed at discovering various aspects of current configuration management that contribute to the complexity of routing policy configuration, as opposed to focusing on the well-known distributed, low-level nature of the configuration management. Based on these problems, we

develop a set of guidelines for configuration management that better supports network operators' needs. These guidelines can be used for both low-level and high-level management. In particular, we focus on inter-domain routing policy configurations, one of the largest, highest impact, and the most complex parts of network configurations [9]. We identify the problems by analyzing more than three hundred real configuration errors, which are collected in four production networks over an eight-month period. We also analyze the daily snapshots of the configurations to study the order in which these configurations are erroneously modified. This helps more precisely identify the reasons why the errors are made. The identified problems are then confirmed by the network operators.

From our misconfiguration analysis, we observe four main aspects of routing policy configuration management that significantly increase operational errors and lengthen problem resolution time: (i) a large number of configurations that are obsolete or irrelevant, (ii) subtle interactions among multiple aspects, (iii) the large and complex set of configuration commands, and (iv) insufficient modularity support. Based on these observations, we propose a set of improvements which can guide the design of future routing policy configuration management.

The remainder of this paper is structured as follows. We contrast our work with related works in Section 2 and provide background information on inter-domain routing and routing policy configuration in Section 3. We then describe our dataset and the methodologies of our misconfiguration analysis in Section 4. In Section 5, we present the four major problems with current routing policy configuration management and propose a set of design guidelines that help the management mitigate these problems. Each of the four problems is illustrated with real examples that are extracted from the configurations that we analyzed. We then discuss a few other issues that are raised by the network operators in Section 6 and conclude in Section 7.

## 2. Related Work

Several past studies, such as Mahajan's work [2], rcc [3], NetScope [10], NetPiler [11], and Minerals [12], propose methods that detect errors in routing policy configurations. These studies identify the decentralized and low-level nature of routing policy configurations as being primarily responsible for misconfigurations. In addition to these reasons, our work further investigates detailed reasons as to why misconfigurations are prevalent. Benson, et. al., [13] discuss complexity models of routing policy configurations and report the existence of outdated configurations as a complicating factor. This previous work focuses on the complexity of a network's routing design, whereas our work focuses more on the complexity of routing configuration framework. The network operator community publishes a list of features that are error-prone, discouraging the use of these features (e.g., `redistribute` command) [14]. These commands can serve as additional examples of large and complex commands, as shown in Section 5.3.

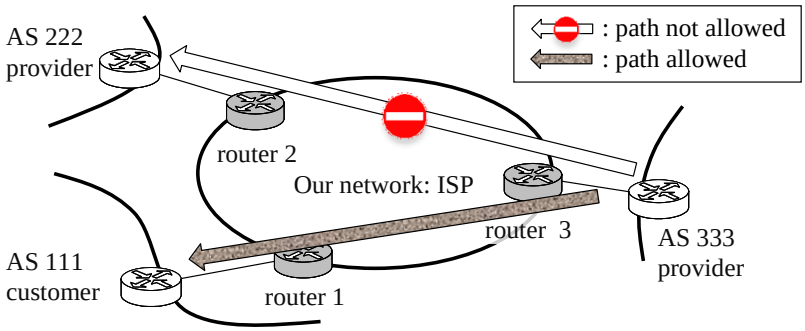



**Figure 2. Example of Routing Policies that Establish Business Relationships**

Harbor, et. al., [8] present a list of guidelines for the development of a general system administration interface, unlike our study which is focused on routing policy configuration systems for ISPs and enterprise networks. However, many guidelines presented in [8] are still valuable and can be used to improve routing policy configuration management. For example, configuration management needs to support communication among multiple network operators since they often collaborate for the same large workflow.

## 3. Background

In this section, we give a brief overview of inter-domain routing and routing policy configuration.

Inter-domain routing enables Autonomous Systems (ASes) to communicate with one another across the Internet, as opposed to intra-domain routing, which is limited to communication within an AS. An AS is a unit of network that is controlled by a single administrative entity, such as a university or a business enterprise. Inter-domain routing works by first exchanging reachability information, called routes, among different ASes and by then learning about routes to other parts of the Internet. The current standard inter-domain routing protocol is BGP (Border Gateway Protocol) [15], which mainly defines two types of information: (i) the attributes carried by a route, such as a destination IP address and an origin AS, and (ii) the rules that determine the best route based on routes' attributes when multiple routes exist for the same destination. One unique attribute that BGP routes contain is the AS-path, a list of ASes that a route has traversed. BGP prefers routes with a shorter AS-path to those with a longer AS-path.

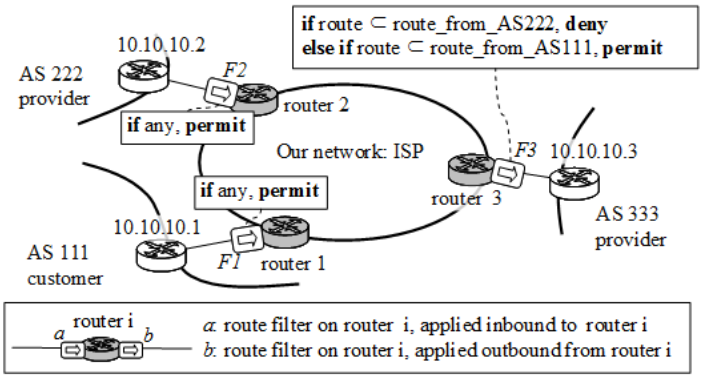
Although BGP can exchange routes among ASes autonomously, network operators often override the default behavior of BGP by using routing policies. Routing policies refer to a set of rules that influence the selection of routes and eventually the paths taken by data traffic. The two main goals of routing policies are to engineer traffic (i.e. to control where traffic flows) and to establish business relationships [16]. Traffic engineering moves traffic flows from one path to another, either to balance the load on the paths or to prefer a path with a higher level of quality. For example, a routing policy may prefer a route that goes through an AS that is known to have the shortest delays, even though the route has a longer AS-path. Establishing a business relationship may prevent a route from being announced to a neighbor AS so that the network carries only authorized traffic. For example, if a network announces a route received from one provider AS to another provider AS, the network has to carry traffic between the

providers without being paid, which is undesirable. We can avoid such situations by not announcing routes from one provider to any other providers. Figure 2 illustrates an example of routing policies that establish business relationships. The oval in the middle of the diagram is the network that we administer. This network possesses the three shaded routers, routers 1 through 3. The network also has three neighbor ASes: AS 111, AS 222, and AS 333. A straight line between an internal router and a neighbor (external router) means that the two routers exchange BGP routes. The depicted routing policy is related to the traffic that enters the network from AS 333 in the right of the diagram and then exits the network to either AS 111 or AS 222 in the left. As shown in the diagram, the routing policies allow traffic from AS 333 to AS 111, but they disallow traffic from AS 333 to AS 222 since both AS 333 and AS 222 are providers of the network.

A routing policy is implemented by configuring route filters in the corresponding routers. For example, the business relationship for providers is implemented by applying a router filter in the outbound direction of the connection with each provider so that routes from other providers are not advertised. A route-filter has a structure similar to the *if-then-else* chain of a programming language. An *if-clause* describes the routes to which the policy is applied, in terms of the attributes of the routes. A *then-clause* is followed by actions on the routes, such as `permit` (to allow the routes) and `deny` (to disallow the routes). These actions may also include commands that manipulate the attributes of the routes. Figure 3 illustrates an example of route filters that implement the routing policies, as shown in Figure 2. These policies can be implemented by announcing routes from AS 111 to AS 333 and not announcing routes from AS 222 to AS 333. The rounded rectangles, *F1* through *F3*, are the route filters. The arrow in a route filter denotes the direction where the route filter is applied. For example, *F1* is applied to router 1 for routes arriving at router 1 from AS 111. Each route filter has its actual content in a rectangle connected with a dashed line. For example, router 1 and 2 accept any routes from AS 111 and AS 222, respectively. All of the three internal routers exchange routes with one another to share all of their routes. Therefore, routers 1 and 2 will eventually send the routes from AS 111 and AS 222 to router 3. Router 3 will then advertise the routes from AS 111 towards AS 333 but filter out the routes from AS 222.

Figure 3(b) presents actual configuration commands in Cisco IOS (Internetwork Operating System) syntax for route filter *F3* configured in router 3. Cisco IOS's configuration language is the most dominant and popular language at the time of writing, and many other vendors have a configuration language similar to that of Cisco IOS. Lines 01-02 define the neighbor router in AS 333. This neighbor has the IP address, `10.10.10.3`. Line 3 applies route filter `TO_AS333` to the neighbor. `route-map` refers to a route filter in IOS. This filter corresponds to *F3* and is defined in lines 04-09. Lines 04-06 are equivalent to the first *if-then* clause in *F3*, "**if** route $\subset$ route_from_AS222, **deny**". In line 04, `deny` indicates that a route matching the condition of the *if-then* clause is disallowed. On the contrary, `permit` indicates the route is allowed. Number `10` after `deny` is the sequence number of the *if-then* clause. When there are multiple *if-then* clauses in an *if-then-else* chain, the *if-then* clauses are compared in ascending order of the sequence numbers. In line 05, `match` is followed by a condition. This condition stands for "if the AS-path of the route matches the condition defined in AS-path list 2". AS-path list 2, defined in line 11, describes the set of routes that are received from AS 222 in terms of the AS-path attribute. Line 06 indicates the end of the first *if-then* clause. Likewise, lines 07-09 are equivalent to the second if-

clause "**if** route ⊂ route_from_AS111, **permit**". The two subcomponents, AS-path lists 1 and 2, can be reused in other route filters. The route filter `TO_AS333` can also be reused for other neighbors.




(a) Implementation of the routing policies, as shown in Fig. 2. The neighbor routers in ASes 111, 222, and 333 are assigned IP addresses, 10.10.10.1, 10.10.10.2, and 10.10.10.3, respectively.

***Router 3***

```
...
01  neighbor 10.10.10.3 remote-as 333
02  neighbor 10.10.10.3 description ProviderInAS333
03  neighbor 10.10.10.3 route-map TO_AS333 out
...
04  route-map TO_AS333 deny 10
05   match as-path 2
06  !
07  route-map TO_AS333 permit 20
08   match as-path 1
09  !
...
10  ip as-path access-list 1 permit ^111
11  ip as-path access-list 2 permit ^222
...
```

(b) Configuration commands used to configure the route filter *F3* in router 3, as shown in (a). These commands are written in the Cisco IOS syntax.

**Figure 3. Implementation of the Routing Policies, as Shown in Figure 2**

For simplicity, the example of routing policy configurations concerns only unidirectional traffic. However, Internet traffic typically flows in both directions, so we need additional configurations for the other direction which is not shown in the example. The configurations become even more complex with a large number of neighbor ASes, which is often more than a thousand in a large network and which may apply disparate routing policies for traffic between each other. These large configurations are frequently modified over time by multiple operators [9], increasing even more the complexity of routing policy configurations.

**Table 1. Summary of Dataset**

| Network | Size | # external networks connected per router[a] | # filters, LoC[b] | # errors analyzed |
|---|---|---|---|---|
| ISP 1 | Large | (3, 45, 66) | (3982, 18407) | 99 |
| ISP 2 | Medium | (11, 22, 25) | (228, 3378) | 48 |
| ISP 3 | Large | (3, 17, 202) | (449, 23021) | 161 |
| Univ. 1 | Large | (3, 3, 7) | (449, 1286) | 46 |

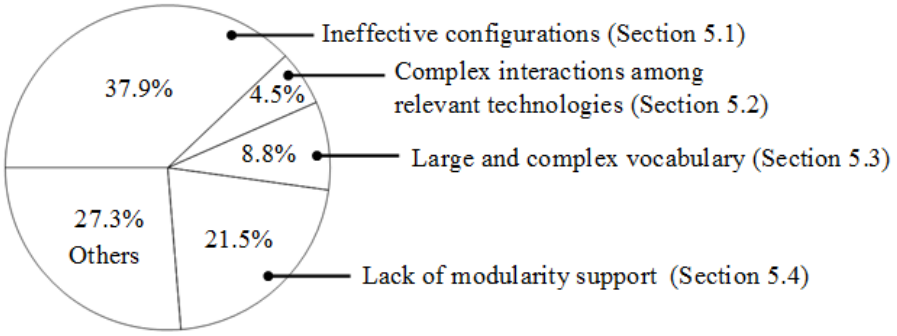
[a]The three numbers denote the 10th-percentile, median, and the 90th-percentile values, respectively.
[b]The LoC corresponds to # of configuration commands related to inter-domain routing policies.

## 4. Data Description and Analysis Method

We analyze common errors that often appear in routing policy configurations in order to understand why the errors are made and also to find a way to prevent these errors. In particular, we focus on the identification of dangerous features in routing policy configurations which cause the misconfigurations and therefore should be improved.

The misconfigurations are collected over an eight-month period from four production networks, including one nationwide ISP, two regional ISPs, and a university network. The characteristics of these four networks are summarized in Table 1. Due to the proprietary information of these networks, we only characterize the networks into small, medium, and large: a small network has less than 10 routers; a medium network has between 11 and 50 routers; and a large network has over 50 routers. Even though we analyze four networks, we believe that the dataset is representative of existing networks and shows important characteristics of routing policy configurations for the following reasons. First, our data covers different architecture types, including the four types of relationships between autonomous domains: customer-to-provider, provider-to-customer, peer-to-peer and sibling relationships. Second, as shown in Table 1, the networks are highly connected to other networks, up to 200 networks in a single router. For this reason, routing policy configurations are a major portion, comprising up to 70% of the configurations (e.g., more than a thousand lines of commands in a router).

We analyze a total of 354 configuration errors. Many of these errors represent critical problems – some of the errors allow private addresses to be advertised externally, and others lead to connectivity loss for an extended period of time by filtering out intended routes. The errors are identified by inspecting the configurations when potential problems are observed through monitoring and trouble tickets. In addition to the configuration errors, we also have access to the network operators as well as the configurations from the four production networks. The configurations in these networks are in both Cisco IOS and Juniper JUNOS syntax. We do not expect to see strikingly different results with other configuration languages since the types of options and the level of details are similar across existing configuration languages.




| Problems with Routing Policy Configuration Management | Design Guidelines |
|---|---|
| Configurations contain ineffective configurations, configurations that do not play any role in a network's functionality. Ineffective configurations can be either outdated, defined for future use, or the result of misconfigurations. They often misdirect users to read irrelevant configurations (Section 5.1). | Give options to present only effective and active configurations, to remove obsolete configurations, and to highlight erroneous configurations for correction. |
| Routing policy configurations are related to the configurations of many other technologies, such as firewalls, VPNs, NATs, QoS, MPLS tunnels, and VLANs. Users need to navigate through these multiple configuration segments while inferring and memorizing complex relations (Section 5.2). | Expose a single unified interface through which the multiple technologies can be managed without needing to consider their complex interactions. |
| Configuration options include a huge, complex, and ambiguous set of configuration commands. A number of different implementation choices exist. This variety leads to a diverse style of configurations, and it often confuses network operators as to the meanings of different options (Section 5.4). | Define a set of simple and usable configuration options that unify overlapping functions but still provide the flexibility that network operators need. Allow configuration options that are more intuitive to users and that are less error-prone. |
| The reuse of common configurations is not fully supported. The same policy can be reused within the same device, and the policy can be reused only in its entirety, not partially. The resulting individual and fragmented configurations lead to many inconsistency errors since updates are deployed individually to these configurations (Section 5.4). | Allow global reuse of common configurations across a network and also allow hierarchical description of configurations similar to class hierarchies in object-oriented programming. |

**Figure 4. Classification of Main Causes of Misconfigurations, and Design Guidelines for Better Routing Configuration Management**

Given the misconfigurations, we go over them to identify a set of possible reasons. In particular, we analyze daily snapshots of the configurations over the eight-month period in order to obtain more information about why the errors are made. We then ask the network operators to confirm the set of reasons that we identify. We also ask these operators for feedback by asking questions, such as: Do the identified reasons also increase the time needed to configure a network? What needs to be improved on the configuration management? In Sections 5 and 6, we discuss interpretations of the reasons that cause misconfigurations and then present suggestions for improving current routing policy configuration management according to these interpretations.

## 5. Design Guidelines

We present (i) problems of routing policy configuration management and (ii) design guidelines that address these problems by reducing misconfigurations and reducing the time taken to configure routing policies. Although we do see some problems coming from the decentralized and low-level nature of configuration management, we also see a number of other aspects that complicate configuration management. We present these other aspects in this section. The guidelines are not targeted to a particular vendor's configuration languages and user interfaces. Instead, the guidelines are aimed at improving any existing management as well as the design of any new routing policy configuration management.

We summarize our findings from the misconfiguration analysis in Figure 4. The network operators pointed out that the problems include the large number of irrelevant and ineffective configurations, the subtle interactions among a number of different technologies, the complex set of configuration commands, and the heterogeneity of configurations. In addition, the operators mentioned that these factors do not only cause misconfigurations but they also increase the time needed to understand and modify routing policy configurations.

### 5.1 Ineffective Configurations

As a network evolves, its configurations become cluttered with ineffective configuration commands. A configuration command is ineffective if its removal does not change network behavior (e.g., a reference to a component that does not exist, a command that is never executed, or a predicate whose condition always evaluates to either true or false). Ineffective configurations are reported to comprise a significant portion, up to 30%, of routing policy configurations [17]. From the misconfiguration analysis, we observed strong evidence that ineffective configurations have a negative impact on network operations. One negative aspect of ineffective configurations is that they increase the time taken to understand routing policy configurations by misdirecting users to read irrelevant configurations. This misdirection can lead to misconceptions of routing policies and incorrect design of changes. Another negative aspect is that certain types of ineffective configurations are due to operator errors, leading to unexpected network behavior.

Most of the ineffective configurations can be classified into four categories, according to the reasons that the configurations become ineffective: *outdated*, *prospective*, *repetitive*, and *erroneous*. According to this categorization, we can determine the actions to perform on an ineffective configuration in order to reduce the complexity of routing policy configurations. Here, we present the definitions of the four categories. More work needs to be done in order to automate the process of classification. *Outdated* configurations play no role in a network's functionality and therefore can be removed to simplify configurations. *Prospective* and *repetitive* configurations are not currently functional but can be used later. These types of configurations can be either marked differently or can be shown only when a user requests to see them, for an improved level of readability. The *erroneous* configurations are errors and have to be repaired in order to restore network operators' intent.

*Outdated configurations* include those that are no longer used but are not removed in time. For example, patches are sometimes put into configurations to temporarily deal with a problem, and then they are forgotten and left in place after the problem is handled. Old configurations often remain to ensure the network operation will work

until a transition is complete. These configurations can also be forgotten and remain after the transition. A few errors happened when an operator mistakenly reused an outdated configuration instead of its newer replacement. This error could have been avoided if the outdated configuration had been cleared up in time.

*Prospective configurations* are not currently used nor are they referred to by any other configurations, but they are defined for future use. For instance, an ineffective routing policy can be defined to deal with a possible modification in peer relationships with neighbor ASes or unforeseen problems in the future. The network operators mentioned that these configurations make it even more time-consuming and confusing to understand routing policies. They suggested that *prospective* configurations be hidden from the user interface when they were not being used and that these configurations be shown when they were called for by users.

*Repetitive configurations* restate important configurations to assure that the network continues to work as intended, even in the case where the original configurations fail to work. For example, one network repeats the same policies in two connected routers so that when the upstream router becomes faulty, the downstream router can still enforce the same policy. In another network, we observed that the *if-then-else* chain in router 3, as shown in Figure 3(a), is implemented with two *if-then* clauses,

"**if** (route $\subset$ route_from_AS222), **deny**", and
"**else if** (route $\not\subset$ route_from_AS222) and (route $\subset$ route_from_AS111), **permit**",

where "(route $\not\subset$ route_from_AS222)" in the second *if-then* clause is *repetitive*. This repetitive configuration is added to ensure the continuing function of the network despite an accidental removal of the first *if-then* clause. This type of design for robustness is called *defensive programming* [0] in software engineering. Defensive programming adds code that reduces the negative impact of possible errors and misuse of a code. The use of *repetitive* configurations depends on a network operator's style of configuration, so we leave the decision to network operators as to whether to show, hide, or remove these configurations.

*Erroneous configurations* represent misconfigurations. Due to these errors, the behavior of the network differs from the operators' original intention. We observe nearly a hundred of this type of ineffective configurations. Some of these errors require prompt corrections of the configurations (e.g., filters that remove intended routes so that no backup route exists, or filters that leak private addresses outside a network). We present an instance of *erroneous* configuration in an AS-path list. Initially, this AS-path was used in an *if-then* clause to match a set of routes, which exclude routes that are received from AS 555. Later, this policy was modified to include a subset of the routes from AS 555, the routes that originate from AS 555. However, this new command was placed in an incorrect order, as shown below in line 02, causing it to become ineffective.

```
01  ip as-path access-list 5 deny ^555
…
02  ip as-path access-list 5 permit ^555$
```

The command lines belong to the same AS-path list, and they are compared in the order they appear in the configuration. The comparison returns when the first match is found. The return value is `yes` if the regular expression of an AS-path follows `permit`,

and the return value is `no` if the regular expression follows `deny`. The regular expression '`^555`' (i.e. any routes from AS 555) in line 01 matches a superset of the routes that '`^555$`' (i.e. routes originated and received from AS 555) in line 02 matches. Line 02 will therefore not match any routes and becomes ineffective. The impact of this misconfiguration is to not advertise a set of routes to a neighbor, causing a number of destinations in AS 555 to become unreachable from the neighbor. To restore the operator's intention, line 02 has to precede line 01.

A few previous works identify different types of ineffective configurations and take actions based on the types. Although these works can efficiently detect certain types of ineffective configurations, more work needs to be done to fully discover all of the ineffective configurations and also to correctly categorize ineffective configurations into the four types. NetPiler [11] identifies *erroneous* ineffective configurations, which are then corrected by network operators. NetPiler defines particular types of ineffective configurations that are highly likely to be erroneous (e.g., an *if-then* clause that is always set to false), and detects these types of configurations so that other types of ineffective configurations are not incorrectly identified as being erroneous. [17] identifies *outdated* and *prospective* configurations, which are then presented to network operators who determine proper actions to take.

### 5.2 Complex Interactions among Relevant Technologies

In addition to ineffective configurations, correctly configuring a network is even more complicated due to a combination of relevant technologies and their interactions. In particular, conflicts are often found in technologies related to the main function of networks – forwarding of packets. These technologies include routing, firewalls, VPN, MPLS tunnels, QoS, VLAN, and NAT. Because of the interactions, a configuration change in one technology may surface as an error in another technology. Manual detection of this type of conflict requires an understanding of different technologies and therefore costs time and effort.

Figure 5 illustrates an instance of conflicts between routing policy configurations and firewall configurations. The two route filters in Network 3, *F1* and *F2*, allow routes toward a particular destination *d* to be advertised to Network 4 so that Network 4 can reach *d* via multiple paths in Network 3. When there is no failure in the networks, Network 4 forwards packets toward *d* via the primary path at the top, (401, 311, 312). These packets go through the firewall *FW1*. However, when the link between routers 311 and 401 goes down, the routing may forward the same flow of packets via the bottom path (403, 321, 322), rather than using the other available path (402, 311, 312). This rerouting has undesirable effects, which might be inconsistent with the intended policies – the packets go through a different set of two firewalls (*FW3, FW2*), which may prevent the packets from reaching their destinations, due to the lack of proper firewall rules. This problem can be solved either by (i) reconfiguring firewalls *FW3* and *FW2* such that these two firewalls are consistent with *FW1* (i.e., allow/disallow the same sets of packets) or (ii) reconfiguring route filter *F2* such that the routes toward *d* are no longer advertised to router 403. In case (ii), if the link between routers 311 and 401 goes down, the path (402, 311, 312) will be used as an alternative, and the packets will continue to go through the same firewall *FW1*.

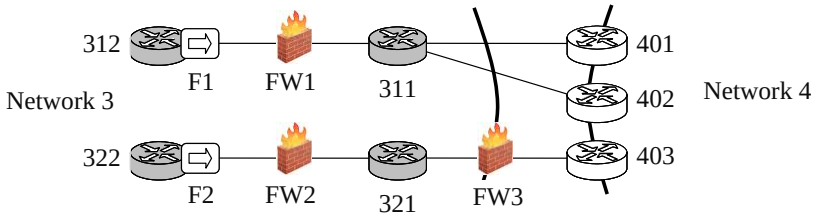



**Figure 5. Example of a Network of Route Filters and Firewalls**

A short-term solution to this problem is to develop systems that detect conflicts among multiple technologies by analyzing their interactions. Prometheus [19] focuses on two technologies, routing policies and firewall policies, but more work needs to be done to incorporate additional technologies. A long-term solution to the problem is to propose a coherent model that unifies several relevant technologies. We presume that such a unification is possible since the different technologies are all related to the forwarding of packets – (i) where to route a particular set of packets, and (ii) how (e.g., disallow, secure, distribute over multiple paths, and give priority when forwarding). The model will expose a single unified interface to a network operator so that the different technologies can be configured without needing to consider their subtle interactions.

### 5.3 Large and Complex Vocabulary

One major problem with current routing policy configuration management is the variety of configuration options that the configuration languages provide, which number more than thousands. This variety of options often confuses users as to the correct meanings of the options. The number of options is analogous to a software complexity measure, the vocabulary size, which is the number of distinct operations and operands. As the vocabulary increases, it takes more time and effort to correctly understand and configure routing policies. This large vocabulary is partially the result of the Internet's evolution: continuous additions of new technologies (e.g., VPN, MPLS, VLAN, and NAT) and incremental patches to existing functionalities. The vocabulary has also been expanded to increase the flexibility of routing policy configurations.

As a result of the growth in the vocabulary, too many features exist for network operators to correctly learn and remember their functions. For example, a combination of BGP attributes determines the preference of different routes. A shorter AS-path prepending, a higher local-preference, and a lower Multiple Exit Discriminator (MED) represent higher preference. Their default values are 1, 100, and 0. These multiple attributes conform to ordering rules such that one attribute preempts another; a route with a higher local-preference is more favored regardless of the route's AS-path; if two routes have the same local-preference value, the route with a shorter AS-path is more preferred regardless of the routes'MED; if two routes have the same local-preference value and the same length of the AS-path attribute, the route with a lower MED is more preferred.

In addition to the large number of different options, the flexibility of the configuration languages allows the same routing policy to be implemented in different ways. This flexibility leads to inconsistent and diverse styles of routing policy configurations, often lengthening the time taken to understand configurations and also confusing the users as to the correct intentions of the policy. For example, the `as-path` command can be applied either within the `route-map` command or directly to the `neighbor` command. In the latter case, a different keyword `filter-list` has to be used instead of `as-path`. Hence, the routing policy

configured with `route-map`, as shown in Figure 3(b), can be implemented differently with `filter-list`, as shown in Figure 6(a). Since the two different implementations in Fig. 3(b) and Figure 6(a) appear to be different, it takes time to discover that these two implementations are semantically and functionally equivalent. Note that incorrect understanding of existing configurations lead to incorrect plan for changes, and thus configuration errors.

```
…
01 neighbor 10.10.10.3 remote-as 333
02 neighbor 10.10.10.3 description ProviderInAS333
03 neighbor 10.10.10.3 filter-list 3 out
…
04 ip as-path access-list 3 deny ^111
05 ip as-path access-list 3 permit ^222
…
```

(a) Implementation that applies the AS-path list directly to the `neighbor` command

```
…
01 neighbor 10.10.10.3 remote-as 333
02 neighbor 10.10.10.3 description ProviderInAS333
03 neighbor 10.10.10.3 route-map TO_AS333 out
…
04 route-map TO_AS333 deny 10
05  match ip next-hop 22
06 !
07 route-map TO_AS333 permit 20
08  match ip next-hop 11
09 !
…
10 access-list 11 permit 10.10.10.1
11 access-list 22 permit 10.10.10.2
…
```

(b) Implementation that uses the next-hop attribute instead of the AS-path attribute. The IP addresses `10.10.10.1` and `10.10.10.2` in `access-lists 11` and `22` are those of the neighbors in ASes 111 and 222, respectively.

**Figure 6. Two Functionally-equivalent Implementations of the Routing Policy Shown in Figure 3**

As another example of the flexibility, an *if-then* clause can match the same set of routes based on a combination of one or more BGP attributes, such as AS-path, destination prefix, next-hop, and BGP community. In Figure 6(b), we present an equivalent implementation of the routing policy to the one shown in Figure 3(b). The route filter in Figure 6(b) matches the set of routes advertised from AS 111 and AS 222 by using the next-hop attribute, as opposed to that in Figure 3(b) where the AS-path attribute is used to match the same set of routes. The next-hops refer to the IP addresses of the neighbors in AS 111 and AS 222, `10.10.10.1` and `10.10.10.2`, respectively. More effort is required to understand the functional equivalence between the two implementations since another step is needed to map the IP addresses in the `access-lists 11` and `22` to those of the neighbors.

Although the vocabulary size can be reduced by extreme measures, such as a clean-slate redesign of network architectures and routing protocols, we suggest two more viable solutions that can be applied to both new network architectures and existing networks. One solution is to define a set of simple and usable configuration options that unify overlapping functions but still provide the flexibility that network operators need. For example, we can expose users to one unified value to represent route preference and conditionally show the multiple attributes (e.g., AS-path, local-preference, MED) if users request to see these attributes. The other

solution is to enforce a consistent configuration style standard for a network and to provide this standard as configuration templates. We present ways to choose better options in the next paragraph.

For both of the solutions, we need to reduce ambiguities in the vocabulary by carefully choosing the options that are (i) more intuitive to users and (ii) less likely to cause mistakes. As an example of (i), the prefix length of an IP address range can be represented in two different formats, the prefix format (e.g., `/27`) and the mask format (e.g., `255.255.255.234`). The prefix format is more intuitive since using the mask format will require the users to spend additional time to map the unfamiliar mask format to the familiar prefix format. As an example of (ii), the same keyword `deny` can be used both in a command that drops a route (as shown in line04, Figure 3(b)) as well as in a command that negates a predicate. We found many cases where network operators mistakenly use one to mean the other (i.e., use `deny` in a predicate to drop a route). The consequence of this misconfiguration is to implement "**if** route $\not\subset$ route_set, **permit**" rather than "**if** route $\subset$ route_set, **deny**", thereby leaking unauthorized routes into neighbor ASes. To prevent this type of error, we can either use a different keyword for negation, or we can disable the dangerous use of `deny` in a predicate and propose a safer alternative when a route needs to be negated (e.g., add an additional *if-then* clause that matches and drops the route). Note that a safer alternative is not necessarily longer than its dangerous counterpart. For example,

```
ip prefix-list LESS24 deny 0.0.0.0/0 le 24
ip prefix-list LESS24 permit 0.0.0.0/0 le 32,
```

which means "if (a route's prefix length $\not\leq$ 24) and (a route's prefix length $\leq$ 32)", can be replaced with the safer alternative that does not use negation,

```
ip prefix-list LESS24 permit 0.0.0.0/0 ge 25,
```

which means "if (a route's prefix length $\geq$ 25)".

Finally, we argue that the flexibility and expressibility of configurations be increased with caution. Before a new feature is added, its usability and possible misconfigurations should be thoroughly investigated. For example, making arbitrary changes to the strict ordering of attributes in the BGP decision process (i.e., local-preference preempts AS-path, which preempts MED) can significantly increase the expressibility of routing policy configurations [20]. However, this kind of change has been prohibited since it can complicate configurations and lead to unforeseen side effects [16]. The software engineering community also suggests that we should not allow unnecessary flexibility but should focus on making functions easy to use since additional flexibility makes it more difficult to find errors [18].

### 5.4 Modularity Support

A network is often configured with common routing policies that are applied to multiple places in the network. However, this process of configuring common policies is supported to a limited extent. For example, a number of networks use a `bogon` list, a list of IP addresses that are private or that are not allocated. This list is applied to most of the connections to neighbor ASes, in order to prevent those addresses from being advertised into these networks. Currently, these common policies are individually configured in each device. The same policy can be reused within the same device, and the routing policy of one neighbor can be reused for another neighbor only in its entirety, not partially (e.g., the `peer-group` command in Cisco IOS applies an entire routing policy to a neighbor). Some networks keep reference

configurations and then copy, modify, and paste the reference configurations when similar policies are configured. Large networks use home-brewed scripts when new configurations need to be deployed in many devices.

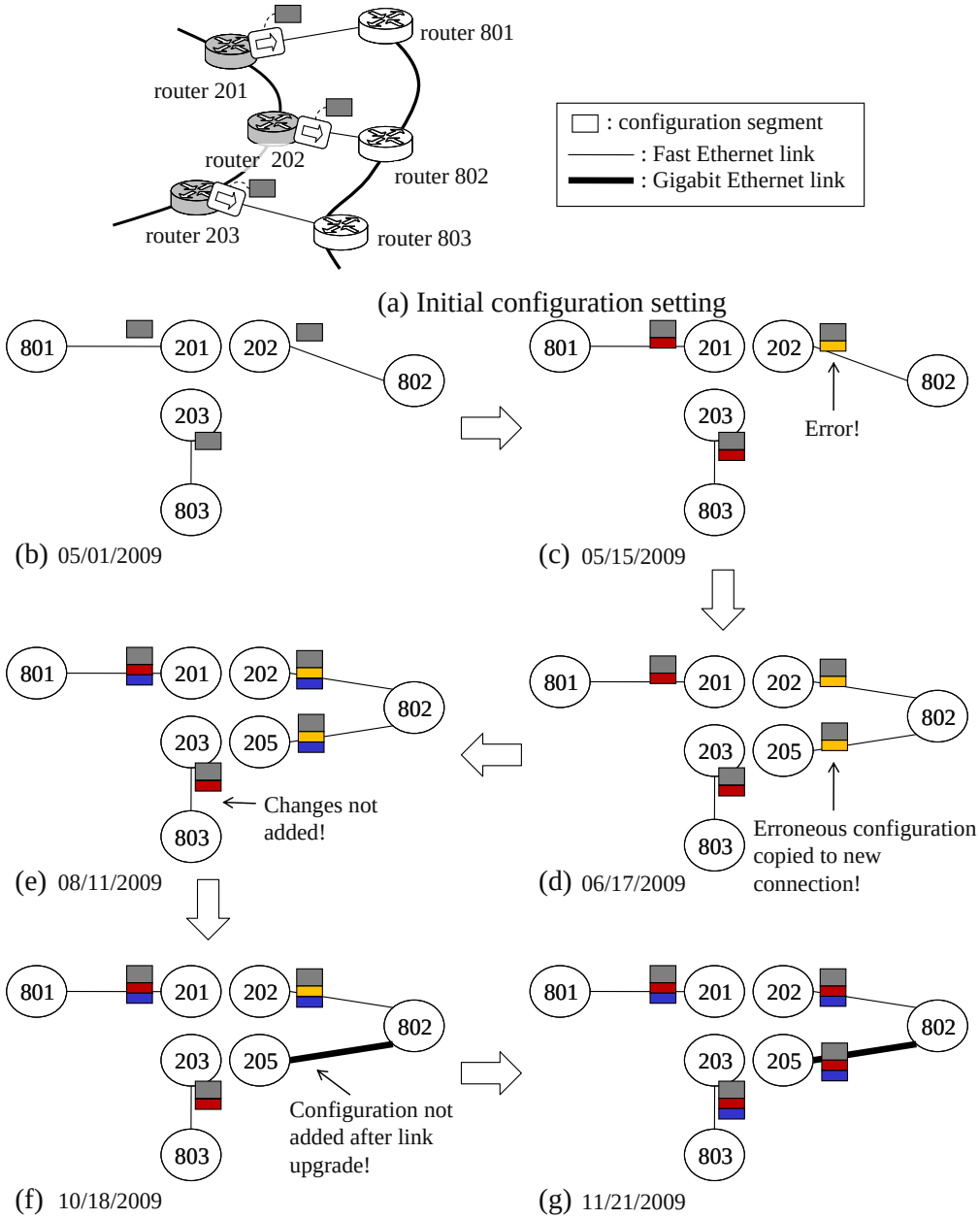



**Figure 7. Illutration of Configuration Errors that Occur Due to the Lack of Modularity Support. Throughout the Seven-month Period, the Configurations in Routers {201,202,203,205} are Supposed to be the Same.**

Although the limited support of common policy configuration delays the initial deployment of a common policy, a more serious problem occurs when the policy continues to be updated as the network evolves over time. When a network operator makes a large number of complex changes in multiple devices, the operator may either forget to update a few

devices or make errata in different places, leading to inconsistent policy configurations [3, 10, 12]. These inconsistencies can accumulate over time and often turn into negative consequences (e.g., loss of connectivity, access to networks that need to be protected). Figure 7 shows an instance of such inconsistencies that we found in one of the four networks. It illustrates how configurations become inconsistent as changes are made over seven months without proper support of configuration reuse. In the illustration (Figure 7(a)), the three shaded routers {201, 202, 203} exchange BGP routes with external routers {801, 802, 803}, respectively. Each of the routers {201, 202, 203) has an outbound route filter, and the rectangle connected to this filter represents configurations in the route filter. Rectangles of the same color represent the same configurations, and those of different colors mean the configurations are different (i.e., gray, red, yellow, and blue). Throughout the illustration, we use a simplified illustration of the same settings, as shown in Figure 7(b) through Figure 7(g).

- Figure 7(b): Initially, the three route filters have the same content. These configurations are supposed to evolve consistently throughout the seven-month period.
- Figure 7(c): It is decided that routes to destination `10.10.0.253` is not advertised to the three external routers {801, 802, 803}. While the three internal routers {201, 202, 203} are separately configured, the policy is incorrectly configured in router 202 such that routes to `10.10.0.235` is not advertised (yellow rectangle). The other two routers are correctly configured (red rectangles).
- Figure 7(d): A new router 205 is connected to router 802 in order to accommodate increased amount of data from 802. The erroneous configurations in router 202 (those with yellow rectangle) are copied and configured for this new connection.
- Figure 7(e): Another set of changes are made in order to permit the advertisement of a route (blue rectangle). By mistake, this change is not added to router 203.
- Figure 7(f): The Fast-Ethernet link between routers 205 and 802 is upgraded to a Gigabit-Ethernet link. During this transition, the existing configurations for the Fast-Ethernet link are forgotten and not configured in the new Gigabit-Ethernet link. At his point, all of the four connections have different sets of configurations, even though they are supposed to be consistent.
- Figure 7(g): The inconsistent configurations are finally identified by the network operators and then corrected. Note that the inconsistencies have existed over six months, and this results in financial implications by providing free transit services, loss of backup routes, and access to otherwise protected networks.

The process of identifying and updating common configurations becomes even more difficult if two equivalent policies are implemented in different ways by multiple operators, due to the large vocabulary size as shown in Section 5.3. This often happens when a network acquires another, which has been configured in different styles. Another problem of the limited support of common policy configuration is that slightly different policies have to be separately written with different names, increasing the time necessary for comprehension and modification of policies.

We suggest that the problems with inconsistencies be solved in two ways: (i) allow global reuse of common configurations across a network through a central database, and (ii) allow hierarchical description of configurations similar to class hierarchies in object-oriented programming. For example, we can define a routing policy for a connection to a provider AS in New York. The policy reuses two policy groups, `common-provider-policy` and

`common-NewYork-policy`, with an additional configuration of some attributes specific to the connection. With this kind of improved modularity support, common policies can be applied and updated consistently throughout a network. Another benefit of the modularity support is that a customized policy can be easily built on top of a set of existing policies by overriding a few attributes of the existing policies, similar to class inheritance in object-oriented programming.

## 6. Discussion- High-level Configuration Management

A few frameworks have been proposed for inter-domain routing policy configuration [4-7]. These frameworks attempt to address the need for a high-level management where network-wide goals are specified in a central place. However, a number of low-level details are still left for network operators to individually decide and configure. None of these frameworks fully address the problems discussed in this paper nor are they widely used.

After the discussion with the network operators, we found three main issues that need to be addressed in order for high-level management to become widely used. First, high-level description of a policy leads to an automatic implementation of this policy, and this thus requires a concrete low-level procedure to implement the policy. However, in many cases, identifying the right procedure is not obvious since network states change dynamically and are often unpredictable; thus, operators identify the right implementation by trial-and-error. For example, operators control incoming traffic by continuously adjusting AS-path prepending of inter-domain routes since its success depends on the policies and traffic of adjacent networks [21]. The second issue with high-level management is that certain configuration parameters cannot be determined automatically [13]. For example, new IP address blocks are assigned to networks by the IANA. This sometimes necessitates the manual re-configuration of route and packet filters. Another example is BGP communities, which are negotiated between neighboring networks and then manually configured. The last issue with high-level management is that network operators want to stay familiar with low-level configurations even when they have high-level management, as shown by user studies [8]. In this way, network operators can easily drill down to low-level details when a problem occurs. One solution to the three issues with high-level management is to implement a simple high-level interface useful for common cases and allow the access to low-level properties for the cases where highly-customized policies are required.

Although the full development of high-level management for routing policy configurations has to overcome a number of obstacles, we believe that a better management framework will eventually replace the current low-level management of network configurations. For example, a GUI-based firewall configuration tool, CheckPoint, has largely replaced the respective CLI (Command Line Interface) in many production networks. No matter whether a framework is based on a GUI, CLI or a table, the framework can eventually prevail if it properly and consistently supports all of the functionalities that a user would need.

## 7. Conclusions

Although routing policies control the primary functions of the Internet by providing global connectivity, the process of configuring routing policies is extremely complex, rendering a large number of destinations unreachable. We believe that these problems can be significantly mitigated by improving the management that we use to configure routing policies. These improvements include (i) proper choice for treatment of non-functional types of configurations (i.e., hide, remove, or highlight for correction), (ii) unified model of multiple, relevant technologies, (iii) a robust, intuitive, and non-overlapping set of configuration commands, and (iv) support for the global reuse of configuration segments. We also argue from network operators' feedback that high-level configuration management might not be always preferred and several issues must be addressed in order for high-level management to be fully developed for the configuration of routing policies.

Note that some of the identified problems are related to the issues that the software engineering community has worked to improve programming languages. This indicates that we can find additional improvements for routing policy configuration management by considering previous experience in programming languages. For example, routing policy configuration management can be further improved with the help of configuration analysis tools, similar to the way that a program development environment is equipped with program analysis tools, such as compilers and code sanity checkers. Configuration analysis tools can highlight the use of dangerous features, potential errors, and redundancies in routing policy configurations, and then they can offer a set of corrective actions. Of course, different network operators have disparate interests and priorities, so users can always enable or disable specific warnings. The analysis tools can also present complexity models of given routing policy configurations, such as the reuse metric [22] and a dependency graph. The complexity metrics help manage the complexity of routing policy configurations since network operators can choose the design options that keep the complexity low. Finally, allowing configurations to be annotated with assertions (e.g., *assert()* in the C language and A [23] in distributed systems) will also help network operators by verifying their assumptions about network behavior.

## Acknowledgements

This work was supported by a research grant from Seoul Women's University (2012).

## References

[1] A. P. Popescu, B. J. Premore and T. Underwood, "Anatomy of a leak: AS9121", NANOG 34, Seattle, Washington, **(2005).**
[2] R. Mahajan, D. Wetherall and T. Anderson, "Understanding BGP misconfigurations", ACM SIGCOMM, Pittsburgh, Pennsylvania, **(2002)**, pp. 3-16.
[3] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis", USENIX NSDI, Boston, Massachusetts, **(2005)**, pp. 43-56**.**
[4] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg and M. Terpstra, "Routing Policy Specification Language (RPSL)", IETF RFC-2622, **(1999).**

[5] L. Vanbever, G. Pardoen and O. Bonaventure, "Towards validated network configurations with NCGuard", Integrated Network Management Workshop (INM), Orlando, Florida, **(2008).**
[6] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenburg, S. Rao and W. Aiello, "Configuration management at massive scale: system design and experience", IEEE Journal on Selected Areas in Communications (JSAC) Special Issue on Network Infrastructure Configuration, vol. 27, no. 3, **(2009)**, pp. 323-335.
[7] X. Chen, Y. Mao, Z. M. Mao and J. Van der Merwe, "Declarative configuration management for complex and dynamic networks", ACM CoNEXT, Philadelphia, Pennsylvania, **(2010).**
[8] E. Haber and J. Bailey, "Design guidelines for system administration tools developed through ethnographic field studies", ACM Computer Human Interaction for Management of IT (CHIMIT), Cambridge, Massachusetts, **(2007).**
[9] S. Lee, T. Wong and H. S. Kim, "To automate or not to automate: on the complexity of network configuration", IEEE International Conference on Communications (ICC), Beijing, China, **(2008)**, pp. 5726-5731.
[10] A. Feldman and J. Rexford, "IP network configuration for intradomain traffic engineering", IEEE Network Magazine, vol. 15, no. 5, **(2001)**, pp. 46-57.
[11] S. Lee, T. Wong and H. S. Kim, "NetPiler: Detection of ineffective router configurations", IEEE Journal on Selected Areas in Communications (JSAC) Special Issue on Network Infrastructure Configuration, vol. **27,** no. 3, **(2009)**, pp. 291-301.
[12] F. Le, S. Lee, T. Wong, H. S. Kim and D. Newcomb, "Detecting network-wide and router-specific misconfigurations through data mining", IEEE/ACM Transactions on Networking (ToN), vol. 17, no. 1, **(2009)**, pp. 66-79.
[13] T. Benson, A. Akella and D. Maltz, "Unraveling the complexity of network management", USENIX NSDI, Boston, MA, **(2009)**.
[14] B. Greene and P. Smith, "Essential IOS Features Every ISP Should Consider", Cisco Systems Press, **(2000)**.
[15] Y. Rekhter, I. Avramopoulos and J. Rexford, "A Border Gateway Protocol 4 (BGP-4)", IETF RFC-4271, **(2006)**.
[16] M. Caesar and J. Rexford, "BGP routing policies in ISP networks", IEEE Network Magazine Special Issue on Inter-domain Routing, vol. 19, no. 6, **(2005)**, pp. 5-11.
[17] S. Lee, T. Wong and H. S. Kim, "Improving dependability of network configurations through policy classification", IEEE/IFIP DSN, Anchorage, AL, **(2008)**.
[18] S. Maguire, "Writing Solid Code", Microsoft Press, Redmond, **(1995)**.
[19] R. Oliveira, S. Lee and H. S. Kim, "Automatic detection of firewall misconfigurations using firewall and network routing policies", IEEE DSN Workshop on Proactive Failure Avoidance, Recovery, and Maintenance (PFARM), Lisbon, Portugal, **(2009)**.
[20] Y. Wang, I. Avramopoulos and J. Rexford, "Design for configurability: Rethinking interdomain routing policies from the groud up", IEEE Journal on Selected Areas in Communications (JSAC) Special Issue on Network Infrastructure Configuration, vol. 27, no. 3, **(2009)**, pp. 336-348.
[21] R. K. C. Chang and M. Lo, "Inbound traffic engineering for multi-homed ASes using AS-path prepending", IEEE/IFIP NOMS, Seoul, Korea, **(2004)**.
[22] P. Devanbu, S. Karstu, W. Melo and W. Thomas, "Analytical and empirical evaluation of software reuse metrics", IEEE Int'l Conference on Software Engineering (ICSE), Berlin, Germany, **(1996)**, pp. 189-199.
[23] A. Tjang, F. Oliveira, R. P. Martin and T. D. Nguyen, "A: an assertion language for distributed systems", Workshop on Programming Languages and Operating Systems (PLOS), San Jose, CA **(2006)**.

## Authors

**Sihyung Lee** received the B.S. (with *summa cum laude*) and M.S. degrees in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2000 and 2004, respectively, and a Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University (CMU) in 2010. He then worked at IBM TJ Watson Research Center as a post-doctoral researcher. He is currently an assistant professor at Seoul Women's University in the Department of Information Security. His research interests include the management of large-scale network configurations and sentiment pattern mining from social network traffic.