Gold Standard Auditing for Router Configurations

Donald Caldwell, Seungjoon Lee, Shubho Sen, Jennifer Yates
{dfwc, slee, sen, jyates}@research.att.com
AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932, USA

***Abstract*—Network providers face a huge challenge of running their network without service disruption in the presence of constant network change. Such change often involves router configuration update. The goal of gold standard auditing is to ensure all *field* configs are equivalent to a certified *gold* config. To handle constant change involving many features configured for numerous routers at the network edge, we need a scalable system that can perform gold standard auditing in a timely fashion. We present the design of GSAT (Gold Standard Audit Tool) that utilizes the syntactic structure of configs. When there is difference between gold and field configs, GSAT can provide users with enough structural hints to understand the context of the difference. Being technology-independent, GSAT is highly scalable and extensible. We have been using GSAT to audit customer-facing routers in a large-scale operational IP network. We present the implementation details and audit performance of our system.**

***Index Terms*—Configuration management, structured diff, gold standard auditing**

I. Introduction

Constant change is simply a fact of life for large ISP networks. Change is introduced for numerous reasons—new customers are regularly added, new services or features are introduced, and network hardware is augmented or upgraded. Realizing such changes often involves making significant configuration modifications to existing routers. Correctness of these updates is essential for the network to operate as intended; misconfigurations have the potential to create catastrophic service disruptions and other adverse fallouts for the customer and provider. However, for a large network consisting of 100s or 1000s of routers each with tens of thousands of configuration lines, it is a huge challenge to realize the intended change correctly [9, 11].

Automated generation and update of router configuration can reduce these risks significantly, if such capabilities are available. However, automated configuration creation is in itself an immensely challenging area of active research [9], and is far from ubiquitous. Even where automated configuration generation does exist, it often focuses only on new deployments as opposed to the more complex task of making configuration changes. Systems that automatically generate router configurations are also subject to errors – the inputs to these systems may be wrong, or there may be failures in writing the configs to the routers. Thus, in both scenarios (with or without automated configuration generation), we need to carefully audit the deployed base of router configurations to ensure that they match what is intended.

Obviously, manual auditing of individual router configs is infeasible for large networks—it simply does not scale.

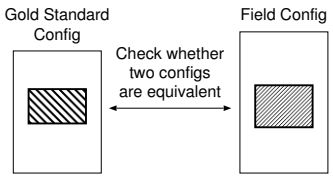

Fig. 1. The goal of gold standard auditing is to ensure all field configs are equivalent to a gold standard config. In some cases, we are interested in auditing some portion of configs (marked as boxes inside). If there is difference, then operators often want to find out exactly what is different, so that they can promptly repair it.

Caldwell et al. [3] propose creating detailed models of router configurations for use in configuration auditing. While hugely useful, such modeling necessarily requires tremendous effort to model all of the features in use within the network. As new features are introduced, the model must be updated. Thus, a developer creating such an auditing system must maintain an extremely detailed and up-to-date knowledge of the usage and semantics of the field configs. If we want to audit a new network that uses a different router technology (e.g., new vendor), this approach necessarily incurs a massive start-up cost. The goal of this paper is to design a configuration auditing system that can readily embrace new configuration features and be rapidly applied to different configuration languages and vendor technologies in a scalable manner. We achieve this by avoiding the detailed modeling required in existing approaches.

Let us consider an ideal scenario where we have a per-router "gold standard configuration" (or *gold config* for short), a working configuration file for each *individual* router that completely captures the design intent. Then, gold standard auditing compares a router's field configuration (called *field config*) against its gold config (see Figure 1). However, in a network with a large number of routers with complex configuration, it is a tremendous challenge to generate an accurate gold standard router config file that precisely captures configuration intent and tracks constant network changes (e.g., to provision edge routers for thousands of new customers every day). However, our analysis of router config files from a large broadband service network has demonstrated that a large portion (e.g., $\sim$50%) of configs for customer facing routers is highly similar across the network, although there exist slight differences in a small number of features within the similar portion. These configurations correspond to important global and regional policies such as security filters and QoS settings. In this work, we obtain gold standard configlets (e.g., Figure 2)

for these features and focus on auditing whether (portions of) the field configs match them.

As the goal of gold standard auditing appears conceptually simple, one might think that simple text-based tools such as `diff` or `grep` may suffice. While such tools are helpful, their utility quickly diminishes as the number of routers and their configuration lines increases. In a typical scenario, to check or edit the config of a router, network operators use the command line interface on the router to enter a specific segment of the config, without the notion of line number in a text file. Thus, a config audit tool needs to report the logical segment of the config that is erroneous, rather than reporting the specific line number(s) of interest. Similarly if a segment of the config (e.g., a rather lengthy packet filter) is added or deleted, an operator should be informed of such, rather than specific lines of config that are added/deleted. However, `diff` displays only the textual change and line number (Figures 3 and 4), and the network operator needs to go through additional steps before he can get the desired context information. We further discuss this in more detail in Section II.

In this paper, we present our gold standard auditing system, GSAT (Gold Standard Audit Tool). GSAT recognizes meaningful configuration entities based on the syntactic structure. GSAT can use these structural entities to compare the gold config and a field config. When a difference between the gold config and a field config is identified, GSAT reports a sufficient structural context regarding the related config. When a single syntactic entity that is large in size (e.g., many config lines) is missing or added, GSAT can provide the summary information about the change (e.g., by displaying the top level information of the entity). Since it does not require any detailed modeling in the semantics level, GSAT is highly flexible and can be readily extended to different technologies (e.g., new vendors) without having to pay the large start-up cost of the traditional model-based auditing approaches [3].

Our system also provides users with the ability to fine-tune various aspects of the comparison operation without involving tool developers. For example, after a network upgrade, a network operator can himself update the set of gold configs that GSAT uses for auditing, replacing the pre-upgrade gold configs with the new ones. GSAT also provides a simple command set by which a user can customize the comparison operation for certain portions of the configs (e.g., ignoring changes in interface description fields). We design this feature to be technology-agnostic, so that the entire system is scalable and extensible. We present our GSAT design in more detail in Section III. In Section IV, we present insights and experience gleaned from the use of GSAT by network operators to audit edge routers in a large operational network. We review related work in Section V and conclude in Section VI. Before presenting our system design, we first discuss desired properties of a gold standard audit system in the following section.

II. Design Goals

In this paper, we focus on the scenario where network operators have a set of fixed gold configs and audit field configs against them. Although the task might sound simple, achieving the goal in a scalable way calls for a careful system design, especially to meet a number of user requirements for handling differences. We next describe some key desired properties of an auditing system. To provide concrete examples and also illustrate the limitation of simple tools such as `diff`, we use an example gold config shown in Figure 2 that configures multiple packet filters (also known as access control lists (ACLs)). Note that many lines have been omitted in the interest of space and clarity of exposition. In this example, line ranges $3904-4594$ and $4595-6480$ respectively correspond to two different filters, 9 and 10. A single packet filter consists of a series of rules called "entries". The first rule in filter 10 is `entry 5` (lines $4597-4603$), which drops any packet with destination IP matching 10.0.0/8 (lines $4599-4601$).

A. Provide Sufficient Context

In addition to showing the actual differences between the gold and field configs, it is important for an audit system to provide sufficient information about those differences such that the operators can quickly grasp the context of the differences. For example, suppose in line 4600 in Figure 2, a field config has `dst-ip 192.168.0.0/16` instead of `dst-ip 10.0.0.0/8` as in the gold config. Figure 3 shows the output by `diff` for this scenario. To fix the destination IP address value, an operator would need to know that the change belongs to the matching rule in entry 5 of filter 10. `diff` just shows that there is a difference in line 4600, but does not provide sufficient context information. As a result, the operator would need to look up the line in question in the original config file before he can comprehend the relevant context.

B. Suppress Unnecessary Details

Network operators typically have to deal with many routers each with many lines of configurations, and displaying all the details about a change is often more confusing then helpful. For instance, consider a case where entire packet filter 10 in Figure 2 is missing in a field config. In this case, `diff` would display all thousands of configuration lines for that filter (Figure 4). In this case, a higher level summary that just displays the filter number for each missing packet filter would be much more useful to the operator.

C. Identify Functional Entities

Ideally, an audit tool should be able to identify logically related lines of configuration (maybe related to a given feature) and handle them together as a single functional unit for auditing. Also, there can be multiple different configs that are functionally equivalent. The audit tool should preferably be able to recognize these equivalences. For example, in Figure 2, packet filters are *numerically* sorted using their filter number (e.g., 1, 2, ..., 9, 10, 11, ...). Suppose that a field config has differently sorted packet filters, e.g., alpha-numerically (e.g., 1, 10, 11, ..., 2, ...). In this case the gold config and the field config are functionally equivalent. Ideally, an audit tool should report that. Figure 5 shows that if we compare the two variants

```
    1   configure
    2     filter
  ...       (earlier filters omitted)
 3904       ip-filter 9 create
 3905         description "filter 9"
  ...         (entries omitted)
 4594       exit
 4595       ip-filter 10 create
 4596         description "filter 10"
 4597         entry 5 create
 4598           description "private IPs"
 4599           match
 4600             dst-ip 10.0.0.0/8
 4601           exit
 4602           action drop
 4603         exit
  ...         (other entries omitted)
 6480       exit
  ...     (other filters omitted)
20937     exit
20938   exit
```

Fig. 2. Example gold config with multiple packet filters. Line numbers are shown on the left. The syntax used here is for Alcatel-Lucent routers [1].

```
4600c4600
<      dst-ip 10.0.0.0/8
---
>      dst-ip 192.168.0.0/16
```

Fig. 3. `diff` output when content of an entry has changed

```
4595,6480d4594
<   ip-filter 10 create
<      description "filter 10"
<      entry 5 create
<         description "private IPs"
<         match
<            dst-ip 10.0.0.0/8
<         exit
<         action drop
<      exit
...
```

Fig. 4. `diff` output when filter 10 is missing. Part of 1887 lines are shown.

```
3,4c3,4
<    ip-filter 9 create
<       description "filter 9"
---
>    ip-filter 10 create
>       description "filter 10"
132,135c132,136
<       entry 5000 create
...
693,697c848,849
<    exit
<    ip-filter 10 create
...
1772,1773c1888,1899
...
>    ip-filter 9 create
...
```

Fig. 5. `diff` output when two files are sorted differently. Only part of 3592 lines are shown.

using `diff`, we get a complicated output where differences are identified in many line ranges. From this output, it is hard to gauge that the two configs are in fact equivalent, and all the differences are due to the configuration statements appearing in different orders.

D. Enable user customization

We observe two cases where it is beneficial for operators to have flexibility in guiding the audit process. First, after performing network upgrades, network operators want to specify what gold standard each router should be audited against. For instance, an operator may want to audit routers in Texas against new gold standard G2, while using old gold standard G1 for routers in California that are not yet upgraded. Second, network operators often want to specify the exact comparison and exception rules, which may be different for different operators. Suppose filter 10 in our example has an entry with a city-specific value (e.g., different IP address for different cities). Here an operator may desire that the audit ignore any differences in the city-specific value between the field and gold configs. We could certainly build a tool where tool developers address the different customization requests. However, a better alternative is a flexible audit tool that can support a range of different user-initiated customizations without the overhead of additional development lead times.

E. Scalability and Extensibility

Our primary focus is to build an audit tool that can handle very large numbers of routers in a network, can accommodate differences in audit requirements across the network (e.g., rule for customer facing routers in Texas might be different from the one for core facing routers in California), and work for different configuration languages.

In the following section, we present the design of our system that achieves these goals.

III. Gold Standard Audit System

In this section, we present the details of our system GSAT and describe how it satisfies the design goals listed above.

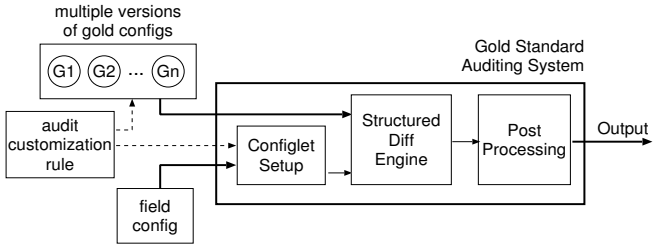

Fig. 6. System overview

A. Overview

In Figure 6, we show the main components of GSAT and inputs to the system. GSAT heavily leverages the syntactic structure of configs, and its key component is the structured diff engine, which compares two configs and outputs the structural difference. Then, actual users, who are config experts (e.g., network operators), can look at the difference and interpret what the change means. In that sense, our system is semantics-agnostic; it just provides structural hints about change and relies on human experts to read and understand them with proper semantics. We designed our system to work with XML (Extensible Markup Language) configs (e.g., Figures 7 and 8) which are supported by many modern routers and which offer the key benefit of having a well-structured syntax. For some networks, configs are available only in CLI (Command Line Interface) format (e.g., early Cisco IOS), or a user may prefer CLI format (e.g., due to readability). In such cases, we convert the CLI configs into XML (see Section III-B). In addition to being able to utilize existing XML tools [2, 6, 7, 8], our approach of working with generic syntactic structure based on XML and thus being semantics-agnostic is key to making GSAT scalable (e.g., new vendor's devices or different configuration syntax).

GSAT also allows a user to specify detailed *audit customization rules*. Audit customization rules, field configs, and gold configs constitute the input to our tool. We next describe each component of GSAT in detail.

```
<configure>
  <filter>
    <ip-filter id="10">
     <desc>description &quot;filter 10&quot;</desc>
     <entry id="5">
      <action>action drop</action>
      <desc>description &quot;Private IPs&quot;</desc>
      <line>entry 5 create</line>
      <match>
       <dst-ip>dst-ip 10.0.0/8</dst-ip>
       <line>match</line>
      </match>
     </entry>
     <line>ip-filter 10 create</line>
    </ip-filter>
  </filter>
</configure>
```

Fig. 7. Example gold config converted to XML. Only entry 5 in filter 10 is shown for brevity.

```
<configure>
  <filter>
    <ip-filter id="10">
     <desc>description &quot;ip-filter 10&quot;</desc>
     <entry id="5">
      <action>action drop</action>
      <desc>description &quot;Private IPs&quot;</desc>
      <line>entry 5 create</line>
      <match>
       <dst-ip>dst-ip 192.168.0.0/16</dst-ip>
       <line>match</line>
      </match>
     </entry>
     <line>ip-filter 10 create</line>
    </ip-filter>
  </filter>
</configure>
```

Fig. 8. Example field config. The difference from the gold config is marked in bold.

B. System Inputs

1) Field and Gold Configs: The structured diff engine in GSAT uses configs in XML format. When configs are available only CLI format, our system uses a config parser that translates CLI configs into XML. Such a parser should also be semantics-agnostic in order for our entire system to be scalable and extensible and we discuss this further in Section IV.

2) Audit Customization Rule: GSAT allows users to specify rules to customize the audit process. Note that there can be multiple versions of gold configs (e.g., different versions of packet filters). Also, since we are focusing on gold configs for the common features across the network, a user must be able to specify what portion of a field config to audit. First, using a *meta-file* provided by GSAT, a user can specify what version of gold config GSAT should use for a particular field config. Specifically, the format of the meta-file is as follows:

```
filter,DLLSTX,/gold/G2,/gold/rule.txt
```

The above line is comma-separated and has four fields: audit name, region, gold config location, and file name for customization rules. Based on this specification, our system will audit all Dallas routers against the gold config stored in the location `/gold/G2`. In the case of network upgrades, a user (e.g., network operator) can use this feature to ensure the prompt auditing of Dallas routers against an updated gold config, without involving tool developer.

Through interactions with network operations teams, we have found that they often want the capability to fine-tune the audit process. For instance, a user may want to ignore the text change in the description field shown in Figure 8. To enable such customization, we defined four user commands, which together satisfy the majority of requests from our users: ROOT, EXCLUDE, IGNORE-VALUE, and IGNORE-ADD. User-specified rules are stored in a file, which appears in the last field of the audit customization rule (e.g., `/gold/rule.txt`). To further leverage the user's domain knowledge and achieve system scalability, we define an XPath-like syntax for command parameter [7] that is semantics-agnostic, and yet provides enough control for config experts to express what they want to achieve. We show an example in Figure 9. ROOT and EXCLUDE specify the portion of field config that is audited against gold config. In Figure 9, the user wants to audit the `filter` section (ROOT command), but wants to exclude (EXCLUDE) a certain filter entry (e.g., a location-specific entry not in gold config). The user also wants to ignore value change in the filter description (IGNORE-VALUE). In addition, some routers have location specific entries in `ip-filter Z`, and the user wants to ignore those entries in the output (IGNORE-ADD). We next describe how GSAT handles these inputs.

```
ROOT,/configure/filter
EXCLUDE,/configure/filter/ip-filter[@id='Y']/entry[@id='X']
IGNORE-VALUE,/configure/filter/ip-filter/desc
IGNORE-ADD,/configure/filter/ip-filter[@id='Z']/entry
```

Fig. 9. Example `/gold/rule.txt`

C. Configlet Setup

In this component, we apply the audit customization rule to each field config to obtain a configlet, which in turn is fed into the diff engine for auditing. First, GSAT uses ROOT and EXCLUDE parameters to select the particular config portion of interest. Second, we annotate the configlet based on IGNORE-VALUE and IGNORE-ADD command parameters. We define special attributes (not in the XML schema) and appropriately embed them into the configlet. For example, to implement the IGNORE-VALUE command in Figure 9, we annotate the field config in Figure 8 as follows:

```
<desc ignore-val='1'>description &quot;ip-filter
                10&quot;</desc>
```

In GSAT, we add annotations only to field configs and maintain the gold configs unchanged. We use a similar annotation approach for IGNORE-ADD commands, which we do not show here for brevity.

D. Structured Diff Engine

GSAT is based on a key observation that we do not need to fully understand detailed meanings of configs to perform gold standard auditing and meet our listed goals. GSAT *compares the syntactic structure* of the gold and field XML configs and reports any structural difference between the two. There are a number of existing XML diff tools to perform the diff operation [8, 10, 13]. Figure 10 shows an example diff output

```
<delta>
  <desc ignore-val='1'>
    <path>
      <configure><filter><ip-filter id="10"/>
      </filter></configure>
    </path>
    <d>description &quot;filter 10&quot;</d>
    <i>description &quot;ip-filter 10&quot;</i>
    <ai a='ignore-val' v='1'/>
  </desc>
  <dst-ip>
    <path>
      <configure><filter><ip-filter id="10"><match/>
      </ip-filter></filter></configure>
    </path>
    <d>dst-ip 10.0.0.0/8</d>
    <i>dst-ip 192.168.0.0/16</i>
  </dst-ip>
</delta>
```

Fig. 10. Example of XML delta.

between the two XML configs in Figures 7 and 8 (using XyDiff-like syntax [8]). The block for the `<desc>` change contains the path, the deleted gold config value (denoted by `<d>`), and the inserted field config value (denoted by `<i>`). The block also has an element for a new attribute (denoted by `<ai>`), which is due to the annotation attribute for audit customization. We next describe how to process the diff output.

E. Post-Processing

In this step, we first suppress displaying entries with annotation attributes. In Figure 10, the deletion and insertion tags of the `desc` block have simple text alone, which indicates a value change for the `desc` element. Since the block has the `ignore-val` attribute, the post-processing step does not include this block for display. Although not shown here, if a block has `ignore-add` attribute due to IGNORE-ADD and the insertion tag contains an XML subtree, then we also ignore the change due to a subtree addition. For the `<dst-ip>` block, we have all the structural information to provide enough context about the change:

```
/configure/filter/ip-filter 10/entry 5/match/dst-ip:
     modified from 10.0.0.0/8 to 192.168.0.0/16
```

Although not shown here, an insertion or deletion tag (`<i>` or `<d>`) can contain an XML subtree. In such a case, the default behavior is to show only the root of the subblock and suppress the details of the block, although GSAT allows a user to choose to display the entire configuration subtree.

IV. System Implementation and Evaluation

We have implemented the GSAT system and been using it for the gold standard auditing of an operational network that provides high speed Internet and video service to more than 2 million customers. In this section, we describe the implementation details and present evaluation results.

A. Current System Implementation

In our current system, we are given CLI configs. To convert them to XML format, we can use any parsing technique such as [4]. Our current implementation employs a simple parsing scheme where we identify configuration blocks and recursively construct subtrees based on the indentation level. We use the first token as the element name. Also, if an element contains a block, then the second token is used as its ID attribute value, which we use to distinguish multiple children with the same element name. For instance, we get `<ip-filter id="10">` from CLI line `ip-filter 10 create`.

For the configlet setup phase, we use a simple script to read audit customization rules to construct an XQuery that implements the rules [2]. Then we use Galax open-source XQuery engine (available at http://galax.sourceforge.net) to run the XQuery on field configs and extract the configlet of interest. Another possible option to extract configlets is to use an XML database (e.g., Berkeley DB XML). Such a tool also supports performance improvement features such as indexing, which we plan to explore in the future.

For the actual comparison of XML configs, we use a perl module called SemanticDiff from CPAN (http://search.cpan.org/dist/XML-SemanticDiff/). We also considered other XML tools and found them to be unsuitable for our needs, which we further explain later in this section. The SemanticDiff perl module satisfies all our requirements after minimal changes. Specifically, we modified it to handle ID attribute values differently, such that we can properly handle multiple children with the same name (e.g., Figure 5). We also modified its post-processing engine such that it recognizes new attributes for audit customization rules as described in Section III.

B. Evaluation Results

In the rest of this section, we present evaluation results for our implementation. We use around 2000 configs for customer-facing routers from an operational ISP network. We focus on auditing packet filter and QoS sections, which constitute a large part of the configs and have various configuration constructs. Specifically, the QoS gold config includes configurations for rate-limiting queues, scheduling policies, and parameters for different traffic classes. We run our experiments on a Sun server running SunOS 5.10.

We first measure the running time of the configlet setup phase. For the customer facing routers we use, a CLI config is 1.6 MB on average, and an XML config becomes larger (around 2.0 MB), which is typical due to extra tags. On average, a config is around 49 thousand lines in CLI and 61 thousand in XML. The filter section is around 40% of a config ($\sim$24 thousand lines in XML), and extracting that part using XQuery takes around 3.4 seconds. In comparison, the QoS section is much smaller (8–9% of entire config), but the processing time (2.5 seconds) is comparable to that of filter. We think that it is mainly because either processing needs to take an entire pass on each config. We also report that our simple parser takes around 6 seconds on average to convert a CLI config to XML.

We next measure the running time of structured diff operation and compare it against GNU diff. In this experiment, compared to the field config, the actual gold config for filter is sorted differently, which we call "unsorted" filter. Since we are using XML, we can use an XQuery to sort the gold config in

		diff	GSAT verbose	GSAT concise
Running Time (in sec)	unsorted filter	0.144	64.0	68.4
	sorted filter	0.135	60.7	61.8
	QoS	0.070	5.8	8.3
Relative Verbosity	unsorted filter	48187.5	7.6	1
	sorted filter	10.5	7.6	1
	QoS	4912.5	17.1	1

TABLE I
COMPARISON BETWEEN GNU DIFF AND GSAT.

the same order as the field config, which we call "sorted" filter. While GNU diff can use CLI format, we use XML configs for the ease of comparison. For GSAT, we consider two variants. In case of a subtree addition or deletion, GSAT-verbose lists the whole subtree, and GSAT-concise displays only the root. In our experiment, we do not use post-processing commands such as IGNORE-VALUE for fair comparison.

In Table I, we report the average running time to perform textual and structured diff between a gold config and a field config. For the filter section, GNU diff takes a fraction of a second to compare the gold config and an average field config. We observe that the structured diff engine currently used in GSAT is slower than diff, especially when the config becomes larger (e.g., filter). However, GSAT is typically used for off-line analysis, which does not require immediate diff operation. Another performance aspect for efficient auditing is accurate and intuitive output, which we describe next.

GSAT can provide highly concise output. In Table I, we also report the amount of output by each scheme. We define *relative verbosity* as the output line count divided by the output line count by GSAT-concise. When the gold config for filter is sorted in a different order from field configs, the output of `diff` is orders of magnitude larger than that of GSAT. Such an output by `diff` is useless for users to understand what the difference is. Even when the filters sorted in the same order, the output for `diff` is 10+ times larger than GSAT-concise and comparable to GSAT-verbose. Although we do not show in the table, we also experimented with XyDiff [8]. In our experiment, XyDiff was not able to provide concise output for simple change, which was the main reason we do not use it in our system. For instance, when a simple block was added to a QoS config, XyDiff reported a large list of small blocks that are moved and added, which resulted in an order of magnitude more lines than GSAT-verbose. By contrast, GSAT provides accurate high-level information about actual changes. After more than a year of usage, GSAT has become an essential tool for network operators to identify and fix unintended configuration changes in the large ISP network.

V. RELATED WORK

NCGuard [12] uses XML configs and XML schema to express high-level objectives for router config auditing. Decor [5] abstracts a network as a distributed database and applies a declarative language framework to network management. Both NCGuard and Decor require technology-dependent low-level implementation to realize their high-level abstraction. By utilizing the syntactic structure of configs, GSAT is technology-agnostic and requires minimal semantics knowledge.

A number of XML diff tools have been proposed [8, 10, 13]. XyDiff [8] is one of the best known XML diff tools. Instead of finding the minimum diff tree, it uses a heuristic to find a reasonably good diff tree efficiently. We plan to investigate those tools to speed up the structured diff phase in our GSAT system.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our gold standard audit system called GSAT. GSAT leverages on the syntactic structure of router configs to identify functional entities. When a gold config and a field config are different, GSAT can provide concise yet sufficient context for config experts to locate and repair the difference promptly. It also allows users to fine-tune the comparison operation. Being semantics-agnostic, GSAT is highly extensible to multiple languages and technologies. GSAT has been an essential tool to audit configs for a large operational network for more than a year.

GSAT provides users with a limited capability for fine-tuning the audit process. One possible generalization of this would be to allow users to specify their own audit logics in an interactive manner. We also plan to explore how to further leverage the syntactic structure of configs for different types of configuration management tasks.

Acknowledgments The authors thank Steven Perks, Ezell Smith, Gary Flack, Eric Sung, Joe Benhabib, and Andrew Gauld for the valuable discussions and feedback.

REFERENCES

[1] Alcatel-Lucent. 7750 SR OS Basic System Configuration Guide, 2007. http://www.alcatel-lucent.com.
[2] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C Recommendation, 2007. http://www.w3.org/TR/xquery/.
[3] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The cutting EDGE of IP router configuration. In *Proceedings of ACM HotNets Workshop*, 2003.
[4] D. Caldwell, S. Lee, and Y. Mandelbaum. Adaptive parsing of router configuration languages. In *Proceedings of Internet Network Management (INM) Workshop*, 2008.
[5] X. Chen, Y. Mao, Z. M. Mao, and J. Van der Merwe. Decor: Declaritive network management and operation. In *ACM PRESOT Workshop*, 2009.
[6] J. Clark. XSL transformations (XSLT). W3C Recommendation, 1999. http://www.w3.org/TR/xslt/.
[7] J. Clark and S. DeRose. XML path language (XPath). W3C Recommendation, 1999. http://www.w3.org/TR/xpath/.
[8] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2002.
[9] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello. Configuration Management at Massive Scale: System Design and Experience. In *USENIX Annual Technical Conference*, Santa Clara, CA, June 2007.
[10] T. Lindholm, J. Kangasharju, and S. Tarkoma. Fast and simple XML tree differencing by sequence alignment. In *Proceedings of ACM Symposium on Document Engineering*. ACM, 2006.
[11] D. A. Maltz, J. Zhan, G. Xie, H. Zhang, G. Hjálmtýsson, A. Greenberg, and J. Rexford. Structure preserving anonymization of router configuration data. In *Proceedings of IMC*. ACM, 2004.
[12] L. Vanbever, G. Pardoen, and O. Bonaventure. Towards validated network configurations with NCGuard. In *Proc. of INM Workshop*, 2008.
[13] Y. Wang, D. DeWitt, and J.-Y. Cai. X-diff: An effective change detection algorithm for XML documents. In *Proceedings of IEEE ICDE*, 2003.