# An Analysis of Network Configuration Artifacts

David Plonka and Andres Jaan Tack
*University of Wisconsin-Madison*

## Abstract

Computer networks and the Internet have become necessary tools in many daily activities; as such, they share the expectation to be "always on" and highly available. Throughout a decades-long evolution of increasing reliance, campus/enterprise networks and Wide-Area Networks (WANs) have been engineered and maintained by an increasingly large set of skilled practitioners, *i.e.*, network operators or engineers. While strikingly similar to the evolution of software and software development by programmers and software engineers, there has not been similar attention to the discipline of network operations as there has to that of software engineering.

In this work, we analyze the deployment and operation of two large networks over a period of five to ten years. Our *analogy-based* approach is to apply software source code artifact analysis techniques to network device configurations. Specifically, we analyze the repositories of router and switch configurations of both a large campus and a service-provider network; these repositories store the actions of hundreds of practitioners maintaining thousands of pieces of equipment over more than ten years time. Our results expose the evolution of these networks both longitudinally in time and by network device types and topological roles. We reverse-engineer operators' work behavior in terms of how they use version control tools, how they change network device configurations, and how long their changes last in a production network. Lastly, we evaluate our proposed analogy between software engineering and network operations, *i.e.*, that network operators are programmers, by comparing and contrasting the analysis of software development to that of modern network operations.

## 1 Introduction

The evolution of network engineering and operation has brought it to the point of being the respected profession of increasingly skilled practitioners. This evolution has brought with it tools and techniques which make the administration of large networks feasible. Networking practitioners in these large networks use integrated development environments (IDEs) to guide and control their changes and they use source code management tools to communicate with each other and record a history of their work. Networks, like software projects, have "bugs," *i.e.*, configurations that have negative effects on the system. Also like software projects, networks are subject to the culture of its governing practitioners.

An artifact is defined as "any object created by humans, especially one remaining from a particular period." The software engineering profession has coined the term, "software artifacts," to mean specifically any such object produced by human being *during the course of software development*. These artifacts include code, bug databases, communications, design documents, and revision histories by Source Code Management (SCM) and Version Control Systems (VCS). Following from this, we define *network artifacts* as anything produced by network practitioners in the course of their practice. Matching the world of software, these include device configurations (code), trouble tickets ("bug" reports), communications, design documents, and configuration change histories.

We find the similarity between the software and networking professions compelling. It suggests to us that the two professions may be closely related. However, whereas software has received a great deal of attention from the research community with respect to artifacts and practitioner workflow, the artifacts of network practitioners have gone woefully unstudied. We hypothesize that, just as the analysis of software artifacts has made an impact in the software domain, a similar analysis would be prudent in the networking domain.

We herein propose an analogy-based approach to the analysis of network artifacts, concentrating specifically on the VCS repositories of two long-standing networks as case studies. Our examination makes use of existing

tools designed for software version histories as well as our own longitudinal static analysis of device configurations. While we test our hypothesis, we point out that our approach is unprecedented in the networking community. Therefore, while we might expect some natural similarities, we must be prepared to witness patterns in network practice which do not have obvious counterparts in software development. It is discovering the extent of their similarity that is our motivation.

In this paper, we use the following set of terms to refer to elements of network configuration management repositories and network configurations (similarly to source code management and software source code):

**practitioner** regardless of domain, the actor or author that is responsible for a configuration change. In the network domain, the practitioner is a network operator or engineer; in the software development domain, this is the programmer or software engineer.

**revision** a file revision expressing a change to a *single* device configuration. This is the smallest representable change in the systems under study and typically is the work of one authoring practitioner.

**commit** a set of one or more supposedly related revisions, submitted for storage in a repository by a practitioner. In some prior work, the commit is known as a transaction; we use the CVS command name, `commit`, instead. (In this work we used a window of six hours to coalesce related revisions with `cvs2cl`.)

**module** a component of the system under study. In the networks we study, the modules are either collections of devices by similar topological role (*e.g.*, core, distribution, access) or by device type (*e.g.*, router, switch, firewall, uninterruptable power supply). In software development a module is typically is a sub-directory containing a subsystem or a class of components, such as header files or library functions.

**stanza** a line, set of adjacent related lines, or a paragraph of configuration with a common purpose. For instance, a single `interface` or `access-list` definition in Cisco's Internet Operating System (IOS) configuration language. (See Listing 1 for a sample IOS configuration fragment.)

**LOC** lines of configuration. Network devices are typically configured using a vendor-specific declarative language. This metric is roughly comparable to lines of code in more general programming languages.

The rest of this paper is organized as follows. In Section 2 we introduce the two networks that we study. We subsequently present, in Section 3, the existing tools that we applied to our task. We describe the preparation of the network configuration data in Section 4 and point out some of the similarities and differences between software development and network operations. In Section 5, we first present the results of processing this repository essentially as if it contained software source code. Following those results, we introduce two network-specific analyses and results: *(i)* revision lifetimes and *(ii)* stanza-based activity in Subsections 5.5 and 5.6, respectively. Section 6 reports on our expert interview-based validation of our analyses. Lastly, we report related work in Section 7, propose future work in Section 8, and conclude.

## 2 Networks Under Study

We studied two large networks: a *campus* network and a *service-provider* network.

Table 1 summarizes the characteristics of the two networks under study.

### 2.1 Campus Network

The campus network under study is a very large network, with approximately 90,000 ethernet access ports and pervasive wireless ethernet access in many campus buildings. In Table 1 note that the number of operators for the campus network is very high, 343 in total. This is due to the fact that the access layer of this network is partially administered by "authorized agents" employed in "end user" departments throughout the campus that use a sort of a network IDE with a web interface to perform changes, rather than a command-line interface as the super-users often use. (AANTS [16] is one example of such a network IDE.) Of the 343 campus operators, 64 of them are network "super users," *i.e.*, the most privileged operators (with similar responsibilities to the 31 operators of the service-provider network). In summary, the campus network is a large IP and ethernet network, with a 3-tiered layout: a set of core and distribution layer routers and switches providing redundant paths to a very large set of ethernet access layer switches.

### 2.2 Service-Provider Network

The service-provider network is significantly different from the campus network. It is a mostly router-based Wide-Area Network (WAN), with approximately 500 customer sites in nearly as many cities and municipalities. In Table 1, we see that it has been continually operated for more than ten years under the SCM system;

| Network | Period (years) | Operators (super users) | Files | Revisions | Lines of Code |
|---|---|---|---|---|---|
| **Campus** | 5+ | 343 (64) | 3,839 | 128,394 | 2,898,362 |
| **Service Provider** | 10+ | 31 (31) | 519 | 41,787 | 163,882 |

Table 1: Network Characteristics.

actually, the network was created in the late 1980s, and thus has been operated for nearly 20 years in total. We also see that there are many fewer operators, and devices (files) than the campus network. This is to be expected though, given that it contains almost no access layer equipment; the customers of this service-provider operate their own ethernet Local-Area Networks (LANs) and thus access devices are not part of the service-provider network under study.

## 3 Tools

As mentioned above, our goal is to utilize existing tools to form a database from our repository of RCS files for the Campus and Service-Provider networks. To this end, we surveyed and experimented with many freely-available tools, both from the research and the open source software developer communities. In general, the former seemed more applicable to our research, however the latter were more easily available and functional in that they were often still currently maintained. For instance, we initially intended to use Bloof [8] because it was feature-rich and extensible, but we found it unsatisfactory in that has not been maintained in years, would not build in our modern development environments, and was also lacking set-up documentation. Since most tools were introduced for use with the popular CVS source code management system, it was convenient that we were able to directly convert our two networks' directories of RCS files to modules within a CVS repository. (CVS actually uses RCS underneath.)

In this study we used the following existing tools to analyze both the campus and service-provider network repositories:

**StatCvs-XML** StatCvs-XML [3] is a statistics tool for CVS repositories that generates a hierarchy of HTML documents and images from CVS log files. It conveniently supplies a web presentation of both longitudinal and summary statistics.

**cvs2cl** cvs2cl [1] is a tool of singular purpose: it converts a cvs log to a more concise "`ChangeLog`" file. This is useful to us primarily because it implements the sliding-window algorithm described in German and Mockus' work [9], that coalesces individual file revisions into the author's commit transactions.

From the tool selection process, we've learned that there are *a lot* of tools available but many, while perhaps useful to practitioners, do not expose enough of the details (e.g., they only produce bit-mapped graphs rather than tabular numeric data) to facilitate new analyses.

## 4 Data Preparation and Transformation

In this work, we report on two case studies each involving the analysis of a repository of configuration files for the devices in a large network. Combined, the data comprises over four thousand files, maintained over approximately ten years, by hundreds of authors. Furthermore, the data was managed in two custom network configuration management systems written in 1997; these systems were similar, and both stored device configurations in files such as that shown in Listing 1, using the legacy file revision control system, RCS. Our analyses, however, expect the data to be in a more modern form. Consequently, perhaps it is not surprising that the raw data needed to be pre-processed, and then transformed. Here we describe the ways in which the network configuration data was prepared for our analogy-based analysis as if it were source code for large software systems.

### 4.1 Converting From RCS to CVS

Most of converting an RCS-based repository to CVS is straightforward because CVS is based on RCS. We simply created a directory structure of *modules* and move the RCS files into that structure. We chose to use modules which represented the position of each device in the hierarchical topology of a network, *e.g.,* core, distribution, or access layers.

One limitation of our conversion to CVS is that, because RCS does not record when a file has been removed, our CVS repository does not contain file deletions information, so network device removal is not exposed by our analysis. While there are some creative proposals for how this limitation might be addressed (such as using the final revision date as an approximate removal date), we chose to simply not report on any devices whose configurations were ever removed in the years studied. Overall

```
version 12.2
no service pad
service timestamps debug datetime localtime
service timestamps log datetime localtime
service password-encryption
!
hostname s-bldg-5-2-access
!
interface FastEthernet1/0/1
   description sample 100Mbps ethernet interface
   switchport access vlan 42
   switchport mode access
   ...
!
ip access-list extended nodhcpserver
   remark Id: ndhcp.acl,v 1.2 2005-05-20 11:26:03 ashley Exp
   deny   udp any eq bootps any
   permit ip any any
!
access-list 5 permit 192.2.0.1
access-list 5 remark Allow foo, bar, and baz servers
access-list 5 permit 192.2.0.10
access-list 5 permit 192.2.0.11
!
end
```

Listing 1: A representative example of IOS configuration code. Most multi-line stanzas types are separated by exclamation points.

it is relatively uncommon to remove devices completely; more often they are replaced, but keep the same device and file name, so are represented accurately.

## 4.2 Cleaning the Data

In the course of our analysis work, we discovered a few interesting features of the data itself. Some of these (including some non-printable characters) required manual attention to permit a clean analysis. Others appeared as systemic properties of the network revision control system, and deserve attention as they would have appeared as quite distracting anomalies in visualizations of the network history.

For some devices, we discovered revisions where the change committed removed every line of the configuration. These revisions, then, were immediately replaced by whole files (as they were before the removal). We identified the source of this problem as an intermittent failure of the network devices themselves; these failures were not handled sensibly by the network configuration management systems. Although there were a relatively small number of these "empty" revisions (111 in campus and 21 in service-provider), they needed to be removed so that the subsequent revisions would not have all the configuration lines erroneously attributed to a single author. We cleaned these sources with the heuristic that any revision removing 90% or more of the configuration lines, based on the most lines that had ever been observed prior, should be ignored. After manual inspectection of just that subset of candidates, we found that this heuristic yielded zero false positives and we removed all the errant revisions.

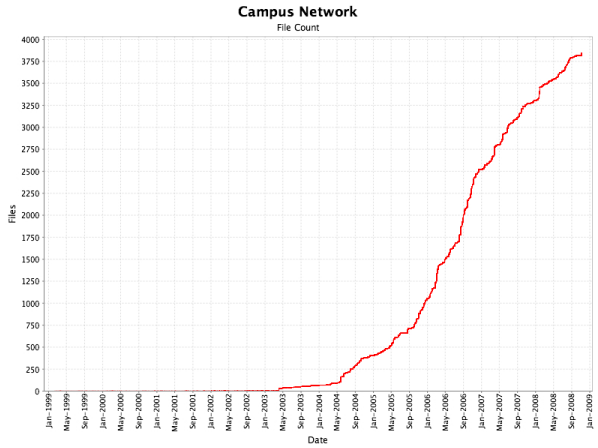
Note that the presence of these empty revisions is a side effect of one major difference between how SCM is done in network operations versus software development. In software development, especially at a large scale, there are many developers, perhaps in many remote locations, that periodically *push* their changed files back to a central repository, from which software releases are subsequently built. By contrast, in network operations, the operators typically operate the SCM system from one central server and they *pull* the configuration file content from the devices' persistent storage (such as non-volatile RAM) back to that central repository. While this *push* versus *pull* model is dramatically different, it has only limited effects on the analysis results. That said, it is worth remembering that networks typically do not have full "development" environments (as in software); the network configuration changes pulled back from devices in the network are *immediately in production*, if they weren't in production already before the revision was committed. (By contrast, software changes typically don't affect a production system until after a software release.)

## 4.3 Authors and Author Groups

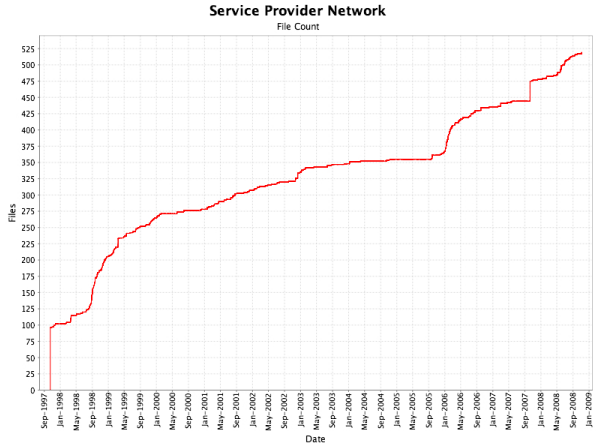
The campus network had very many active operators at 343 in total. Rather than deal with this overwhelming number of authors for visualizations, a portion of our analyses report on groups of operators rather than individuals. The task of translating the practitioner names to their corresponding group was non-trivial because, in ten years, some practitioners had left their jobs, changed to different groups, or even changed names. However, we were able to accurately assign practitioners by using a revision history of their group assignments, kept as described in [12], combined with expert knowledge of the operator employees by other employees that had remained for the duration. Manual effort was also required to combine multiple author (account) names that were really the same practitioner.

# 5 Analysis and Results

In this section we present graphical and tabular analysis results and comment on characteristics, prominent features, and anomalies that are either similar or different between the campus and service-provider networks under study. Wherever our results mention user login names or real names, these names have been anonymized.




(a)




(b)

Figure 1: Campus (a) and service-provider (b) file/device count over time. These two networks experienced very different growth rates and changes in rate.

## 5.1 Network Evolution

First, we present the entire lifetime of each network in time series, *i.e.*, each network's evolution in time. While the active portion of the campus network is approximately only five years, both networks are shown in an approximately ten year time range that allows the plots to be easily compared.

Figure 1 shows the number of devices, such as routers and switches, that existed at each point in time for both the campus and service-provider networks. In the campus graph, Figure 1a, notable elements include the growth rate, and its change over time, nearly reaching 4,000 total devices. The shape of this curve suggests that we've captured the network deployment from its inception and that it has gone through periods of differing growth rates. In the service-provider graph, Figure 1b, the adoption of the configuration management system is marked by a sudden increase in device count. There have been two other prominent increases in new devices, beginning roughly September, 1998 and January, 2006, ultimately reaching more than 500 devices in total. Our expert interview from Section 6 was able to offer an explanation for these events.

In Figure 2 we see a time series plot over that same time as Figure 1, but here we show the evolution of the portions of the topology, *i.e.*, by plotting the total LOC for all devices that serve a particular role in the network. We see in both the campus and service-provider networks, that the periphery (campus access layer and service-provider customer sites) are responsible for the most LOC, and that the peripheral topological layers most contribute to the overall growth in configuration content. This is perhaps to be expected as these devices are the most numerous, connecting approximately 90,000 ethernet ports plus wireless access points in the campus and all the service-provider's customers. Another prominent feature is the addition of management equipment after January 2007, and firewall devices after September 2007. However, It is not clear whether these devices were very quickly deployed or whether they were merely inducted into the configuration management system at this time.

## 5.2 Activity by Topological Role and Device Type

In Tables 2a and 2b we show how much of each module (collections of devices by their topological role) contributes to activity in terms of commits and LOC, for the campus and service-provider networks, respectively. One points of interest is that more than 75% of the commits are performed within each network's periphery (campus access and wireless, and service-provider's customer sites). However, the LOC per commit is quite different between campus and service-provider. This suggests that campus/enterprise access switches require much less fine-tuning than do site routers in this service-provider WAN. We also see that, in both networks, out-of-band management equipment and firewall services represent a much smaller portion of the work, in terms of commits.

## 5.3 Author Activity

Figure 3 presents the activity for *every* practitioner that authored revisions in the campus and service-provider networks. Because the number of practitioners involved in the campus network is clearly overwhelming, we present the same campus data in Figure 4 based on the group in which they are employed. Specifically, "net"

| Module | Commits | LOC | Added LOC | LOC per Commit |
|---|---|---|---|---|
| campus/access/ | 89833 (70.0%) | 1912430 (66.0%) | 2883860 (68.2%) | 21.29 |
| campus/access/wireless/ | 18164 (14.1%) | 601836 (20.8%) | 657409 (15.5%) | 33.13 |
| campus/dist/ | 7598 (5.9%) | 98921 (3.4%) | 143155 (3.4%) | 13.02 |
| campus/core/ | 6022 (4.7%) | 47272 (1.6%) | 97295 (2.3%) | 7.85 |
| campus/firewall/ | 5557 (4.3%) | 120147 (4.1%) | 319426 (7.6%) | 21.62 |
| campus/mgmt/ | 1220 (1.0%) | 117756 (4.1%) | 126903 (3.0%) | 96.52 |

(a)

| Module | Commits | LOC | Added LOC | LOC per Commit |
|---|---|---|---|---|
| isp/dist/site/ | 31931 (76.4%) | 92977 (56.7%) | 309604 (55.7%) | 2.91 |
| isp/dist/hub/ | 5203 (12.5%) | 28116 (17.2%) | 98581 (17.7%) | 5.40 |
| isp/border/ | 3373 (8.1%) | 18665 (11.4%) | 98985 (17.8%) | 5.53 |
| isp/firewall/ | 445 (1.1%) | 12516 (7.6%) | 25939 (4.7%) | 28.13 |
| isp/mgmt/ | 835 (2.0%) | 11608 (7.1%) | 22434 (4.0%) | 13.90 |

(b)

Table 2: Commits by topological role of the device for campus (a) and service-provider (b) networks.

is the network engineers, "contract" represents the contractors, "noc" is the Network Operations Center (NOC) staff, "field" the field service agents, "authorized-agents" are employees in various peripheral campus departments that are authorized to make access layer changes only, and "security" is an IT security group. From this pie chart, we see that the operators responsible for most of the LOC are network engineers proper. Also, the contractors performed a significant amount of similar work.

In Tables 3a and 3b, we show the top ten most active practitioners based on their number of commits. Note also that the LOC per commit is approximately an order of magnitude different between the campus and service-provider network operators. This suggests that the campus, with very many switches rather than routers, is in a higher state of flux and perhaps recently in a deployment mode. In contrast, the service-provider network experiences relatively small changes in terms of LOC per commit, perhaps suggesting that it is largely stable and in a maintenance mode.

## 5.4 Anomalies

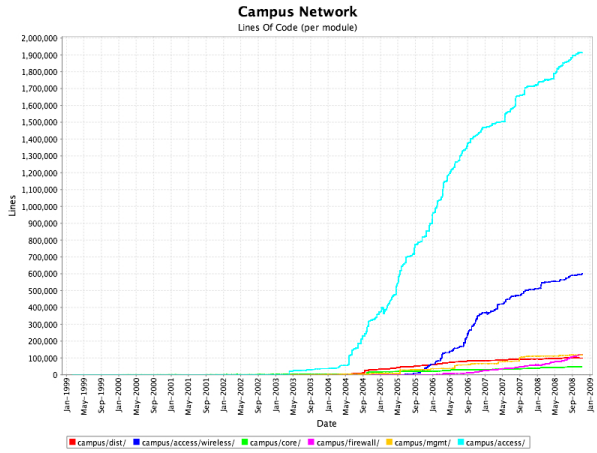
Here, we describe a number of curiosities or anomalies discovered in the networks studied, solely based upon the results presented thus far.

### 5.4.1 Activity by Campus "system" Author

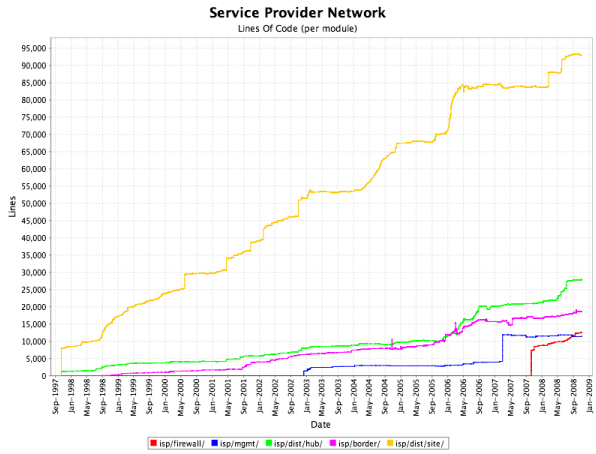
In the campus network, and shown in Table 3a, we can see that one of the "Top 10" most active authors is the software system itself (by the name `system`), rather than a real person/practitioner. This entry is additionally interesting in that overall it has removed more lines that it has added and thus is very different from the real practitioners. Further investigation identified two reasons for this unexpected significant authorship of changes by the SCM system itself: *(1)* some of the operators often do not "follow the rules," *i.e.,* they do not commit their changes in a timely fashion and thus the system sometimes has to commit their changes implicitly just prior to applying a subsequent automated change (so as not to mix unrelated changes together), and *(2)* a few operators have discovered an unintended feature of their automated change system; namely, that they can cause their earlier changes to be committed implicitly to the version repository. This avoids those changes being reported as unfinished in a nightly email report to all operators. Both of these causes demonstrate to how a VCS can produce both efficiencies and inefficiencies in the everyday work flow of network operators. This suggests that the process by which changes are merged into a network configuration version control system can be improved. It is an open question as to whether existing merging techniques from SCM systems will be similarly effective, but there are certainly both syntactic and semantic differences between the network device configuration files in a production network and the source files in software development.

### 5.4.2 Outstanding Service-Provider Author

Considering the question of which operators perform most commits, we see in Table 3b that both the most commits and most of the LOC are authored a single, seemingly "super human," outstanding author, here named "robert." This suggests that operator involvement

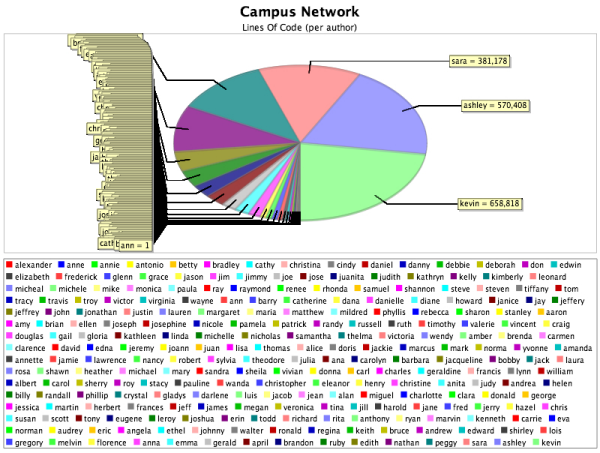



(a)

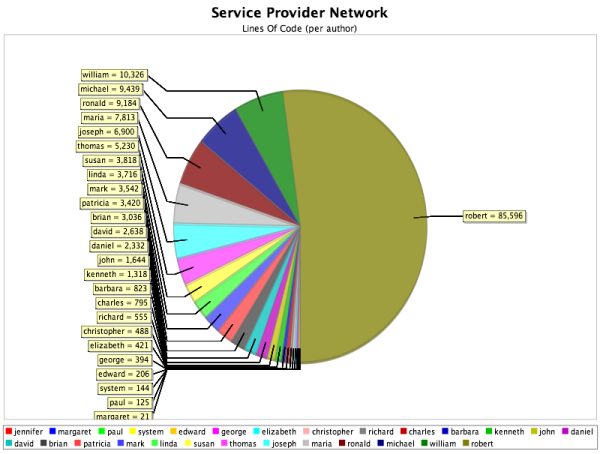



(b)

Figure 2: Campus (a) and service-provider (b) LOC by topological role over time. Most of the LOC are configuration of the periphery of each of these networks, *i.e.*, the campus access layer and service-provider's customer sites.




(a)




(b)

Figure 3: Campus (a) and service-provider (b) LOC per author. In both networks, five authors are responsible for approximately 75% of the LOC.

varies widely amongst networks and amongst individual practitioners with respect to the tasks of introducing devices (*i.e.*, introducing many LOC of their initial configuration) and subsequently managing a network's device configurations.

### 5.4.3 Common Commit Comments

Tables 4a and Tables 4b show the most common comments provided by the operators in the campus and service-provider networks, respectively. In Table 4a we see that the second most common comment is "asdf," from the home row on a qwerty keyboard, suggesting it's a cavalier refusal to supply a meaningful comment. Further investigation showed that this comment is *always* supplied by only one of the authorized agents using a web interface to perform changes. Unlike with the CLI interface, here the comment is required, and thus the practitioner is forced to supply something. Our hypothesis is that this practitioner likely sees only himself as the "audience" of the comments, and deems it unnecessary to exert effort to carefully explain the changes he commits.

In Table 4b we see that nearly 6% of all log comments are empty. Like the campus "asdf" comment, these empty comments are being supplied by only a small subset of the practitioners, again perhaps ones that don't see, or have never realized, any value from such comments. In the service-provider environment, the "?" comment was occasionally supplied by the outstanding practitioner that performs most of the commits. Further investigation suggests that he is stumbling across changes made

| Author | Commits | LOC | Added LOC | LOC per Commit |
|---|---|---|---|---|
| ashley | 16430 (12.8%) | 570408 (19.7%) | 952945 (22.5%) | 34.72 |
| kevin | 9296 (7.2%) | 658818 (22.7%) | 703006 (16.6%) | 70.87 |
| **system** | **8164 (6.4%)** | **-6595 (-0.2%)** | **49117 (1.2%)** | **-0.81** |
| nathan | 5257 (4.1%) | 279484 (9.6%) | 329512 (7.8%) | 53.16 |
| sara | 4790 (3.7%) | 381178 (13.2%) | 410738 (9.7%) | 79.58 |
| edith | 4755 (3.7%) | 122640 (4.2%) | 134277 (3.2%) | 25.79 |
| brandon | 4666 (3.6%) | 75641 (2.6%) | 91540 (2.2%) | 16.21 |
| ruby | 4626 (3.6%) | 99700 (3.4%) | 190530 (4.5%) | 21.55 |
| peggy | 3958 (3.1%) | 345232 (11.9%) | 365551 (8.6%) | 87.22 |
| emma | 3483 (2.7%) | 54658 (1.9%) | 63449 (1.5%) | 15.69 |

(a) Note that the third most active campus author, "system," is not a practitioner but records automated commit activity by the configuration management system itself.

| Author | Commits | LOC | Added LOC | LOC per Commit |
|---|---|---|---|---|
| **robert** | **30385 (72.7%)** | **85596 (52.2%)** | **396634 (71.4%)** | **2.82** |
| michael | 1489 (3.6%) | 9439 (5.8%) | 16443 (3.0%) | 6.34 |
| brian | 1444 (3.5%) | 3036 (1.9%) | 15698 (2.8%) | 2.10 |
| joseph | 1431 (3.4%) | 6900 (4.2%) | 13688 (2.5%) | 4.82 |
| linda | 1174 (2.8%) | 3716 (2.3%) | 13091 (2.4%) | 3.17 |
| william | 1058 (2.5%) | 10326 (6.3%) | 14566 (2.6%) | 9.76 |
| daniel | 673 (1.6%) | 2332 (1.4%) | 7254 (1.3%) | 3.47 |
| john | 628 (1.5%) | 1644 (1.0%) | 4952 (0.9%) | 2.62 |
| kenneth | 511 (1.2%) | 1318 (0.8%) | 5461 (1.0%) | 2.58 |
| david | 459 (1.1%) | 2638 (1.6%) | 6137 (1.1%) | 5.75 |

(b) Note that the most active service-provider author, "robert," is a single most outstanding operator that performed more than 70% of the commits and was responsible for more than half of the LOC.

Table 3: Commits by author for the (a) campus and (b) service-provider networks. The bold entries are discussed in Sections 5.4.1 and 5.4.2.

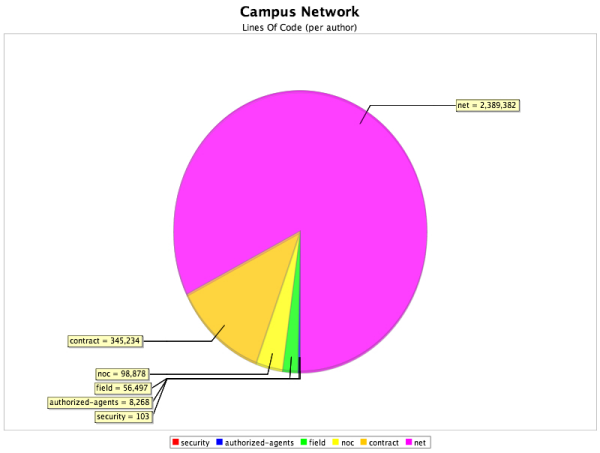



Figure 4: Campus LOC per author group. We note that the "net" (network engineering staff) group is responsible for approximately 80% of the LOC, followed distantly by contractors, field service agents, and authorized agents.

| Comment | Frequency |
|---|---|
| Initial revision | 1442 (2.8%) |
| **asdf** | **584 (1.1%)** |
| test | 437 (0.9%) |
| 'newer bulk checkin' | 411 (0.8%) |
| change vlan | 308 (0.6%) |

(a)

| Comment | Frequency |
|---|---|
| ***** empty log message ***** | **768 (5.9%)** |
| Initial revision | 350 (2.7%) |
| router swap | 117 (0.9%) |
| config cleanup | 107 (0.8%) |
| ? | 75 (0.6%) |

(b)

Table 4: Top five commit comments for (a) the campus network and (b) the service-provider network. In each of these results, garbage comments indicate operator non conformance and other habits. The bold entries are particularly unexpected and are discussed in Section 5.4.3.

by others, and is essentially using the "?" to say that he's checking in changes performed by someone else, for which he does not readily have an explanation. He thus commits that change, and carries on with his tasks without having to wait for such an explanation.

These anomalous results of system authorship of commits and common log comments both speak to the issue of operator conformance with the system used in these networks. In large part, practitioners appear to use the tools as intended, and with a high degree of compliance. However a subset of the operators seem to find it cumbersome and sometimes find workarounds that make their tasks easier. Such discoveries can effectively guide new tools and features.

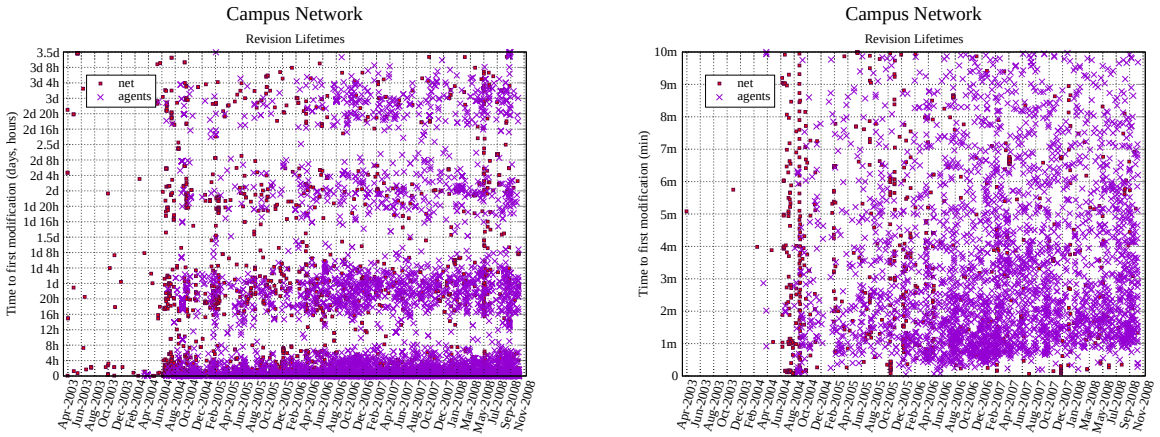
## 5.5 Revision Lifetimes

In Figure 5, we see a pair of plots demonstrating *revision lifetimes*, or the time from a revision's appearance within a file to the first subsequent revision which affects any of the same lines of configuration. Both plots are for the campus network (the service-provider network does not change often enough for this plot to be valuable). We are particularly interested in short-lived changes, here clustered to the bottom of the graph. Note that this version history is unique in that it always reflects a production environment.

In Figure 5a, we are surprised to see that such short revisions as to occur within a day or two of each other (suggesting a network "bug") are treated only during business days, and very infrequently require overnight attention from network operators even though these revisions are ostensibly part of the production network.

Figure 5b, essentially the same data on a finer time scale, tells its own story about change lifetimes from different contributor groups. The *net* group (squares) represents super-users on the network, whose access is completely unrestricted. This group makes relatively few changes in the ten-minute window shown here. The other group, authorized agents working at all levels of the network infrastructure, composes the vast majority of the plotted points (crosses). These agents make their changes through a web interface (essentially an IDE for the network) which automatically checks in the change as soon as it is applied to the router.

Based on this last observation, we see that we have two different data sets available to us in the revision history for the campus network. For network engineers (the *net* group), we see a traditional software-like history of commits, where the user commits his changes most often after he has observed their effect and deemed them a valuable contribution. From the commits made by *agents*, since they are not privileged to interact with the devices directly, we actually have a richer version history. Their history not only includes those changes which survive in the long term, but also the changes that they make as part of their efforts from one minute to the next. It is, one may consider, an extrapolation of revisions to a perfectly fine granularity of change. Thus, in the recorded history of this network, we find an artifact which is entirely unavailable from any known software project.




(a) (b)

Figure 5: Campus revisions, time to next modification: 3.5 days (a) and 10 minutes (b).

## 5.6 Activity by Stanza Type

The relatively simple structure of IOS configurations allows some static analyses which consider *stanzas*, rather than lines, as the basic units of change from one revision to the next. Tables 5 describe the results of this analysis. These results can guide the creation of tools to manage the network under inspection: In both cases described here, we confirm that any service built for the configuration of these network devices would be well-advised to cater specifically to the management of `interface` and *global* stanzas.

| Stanza Type | Total Revisions | Revisions per Instance |
|---|---|---|
| `interface` | 471,238 | 4 |
| `vlan` | 25,591 | 1 |
| *global* | 12,534 | 4 |
| `logging` | 12,390 | 9 |
| `ip` | 12,006 | 1 |
| `bridge` | 4,353 | 1 |

(a) Campus network: the ratio of `interface` stanza revisions to *global* stanza revisions is 19:1.

| Stanza Type | Total Revisions | Revisions per Instance |
|---|---|---|
| `interface` | 25,288 | 4 |
| *global* | 11,737 | 26 |
| `ip` | 8,207 | 4 |
| `line` | 6,146 | 14 |
| `router` | 3,974 | 4 |
| `policy-map` | 2,783 | 4 |

(b) Service provider network: the ratio of `interface` stanza revisions to *global* stanza revisions is roughly 2:1.

Table 5: Number of revisions made per each IOS stanza type, for the campus (a) and service provider (b) networks. The *global* meta-stanza included all unindented lines at the top of a file, preceding the appearance of any others in this list.

## 5.7 Discussion

We close this section with our observation about LOC as a metric for networks rather than software. While we have not yet done analysis of code complexity, early indications suggest that there are a number of reasons that numbers of lines of configuration (LOC) is a poor candidate as a measure of complexity or work. First, the initial versions of our configuration files (source code) contain very many "boiler-plate" lines produced by the network device itself; attributing these lines of code to the operator that introduced the device to the network dramatically exaggerates the volume of the work done by that operator. Secondly, the configuration files are rigidly formatted by the device rather than the operator (programmer), *i.e.*, it is not a free format language. Thus, the vendor-specific network device configuration language, itself, dictates the numbers of lines more so than modern general software programming languages dictate the number of lines of program source code.

## 6 Validation

This being an initial study of it's kind, to the best of our knowledge, we were left to interview domain experts in network operations to validate our approach. For the campus network, we interviewed the manager to whom most of the super-user operators have reported. For the service-provider network, we interviewed the director.

### 6.1 Campus Expert Feedback

Here are highlights of the feedback offered by our campus network expert:

- The top authors by LOC agrees with managements knowledge of their respective performance, *i.e.,* these are outstanding practitioners in that they indeed have the most responsibility for network equipment deployment.
- The data points, *e.g.,* commit volume and common comments, would be useful to demonstrate to customer departments that we know how authorized agents use the tools provided.
- The visualizations are useful to show the evolution of the network's architecture over time, *e.g.,* the wireless access deployment and the use of contract labor to do so.
- The author-specific visualizations, such as activity by days of week and times of day would be an interesting addition to existing tools, such as the network IDE provided to the practitioners themselves.

### 6.2 Service-Provider Expert Feedback

Here are highlights of the feedback offered by our expert on the service-provider network:

- The file count evolution over time clearly shows inflections due to two significant events: *(i)* a $200M influx of funding resulting in membership growth by more than 100 sites and *(ii)* the merging of the service-provider network with a similarly scoped network, resulting in many devices being replaced (to switch from T1 circuits to 10Mbps ethernet).
- One practitioner, a temporary employee, was responsible for an unexpectedly large number proportion of the code. However, this coincides with the person's role, which was to deploy replacement equipment. (Consequently, they were responsible for much of the initial device configuration, thus a large number of lines of configuration.)
- The similarity between network operations [when viewed this way] and software development is striking.
- Common commit comments suggest the need for a new standard operating procedure that would encourage practitioner's to supply meaningful comments; this would also aid analysis.
- Such linear trends over time were not expected. There are some events that had significant costs (such as router replacements by alternate brands) that do show prominently in the time series graphs. (This is akin to, perhaps, changing programming languages in a portion of a software system.)

While clearly a subjective assessment, the feedback from both experts showed the utility of our results, and consequently the value of the analogy-based application of these analyses.

## 7 Related Work

We are aware of one study in the literature, the recent work of Sung, *et al.* [15], that longitudinally examined network configuration repositories of network devices such as routers and switches. Similarly, our work also examines and reports on the configuration changes in multiple real-world networks over time, examines stanzas by type, and evaluates results by expert interview. However, our work differs in that we apply software development analysis techniques to expose practitioner behaviors and network evolution over time, whereas they apply different data mining techniques to identify correlated configuration changes. More generally, our work is informed by related work in three areas: programming languages, network management, and systems administration.

The Revision Control System (RCS [17]) is the version control sub-system with which the versions of configurations we consider are stored. In [4], Ball, *et al.,* demonstrate some of the uses of the information stored in such VCSs for software source code. Our work applies analysis and visualization techniques to expose characteristics of network management in a similar fashion to that early examination of software development via VCS. In [7], Draheim and Pekack introduced a freely-available tool, Bloof [8]. Tools such as Bloof and cvsanaly2 [2], introduced in work [14] by Robles, et al., could potentially be used similarly to the one we used (StatCvs-XML).

In this work, we study repositories of network configurations maintained by the Network Configuration Management System (NetCMS [11]) and AANTS [16]. An alternative technique often used by network operators is to retrieve device configurations using RANCID [13, 10]

and subsequently store them using tools such as CVS. A very recent work [5] by Benson, *et al.,* introduces a code complexity metric for network devices configurations. Their metric uses attributes including Lines of Code and inter-stanza references (within and amongst configuration files) to arrive at a numeric measure of complexity; they subsequently validate their proposed metric by operator interview. In this work, instead, we develop a way to measure programmer effort by revision lifetimes, but have not yet used it to evaluate a complexity metric.

There is a large literature concerning the profession of system administrator and improvement to the processes involved in system configuration. System administrators sometimes similarly use VCSs for their configurations [12] and researchers seek to improve configuration management. For instance, Sun and Couch develop a state-machine model of configuration management in [6].

## 8 Future Work

While we have completed an analysis of two ostensibly different, large networks, the process and results suggest some directions for future work.

In our consideration of revision lifetimes, we have not considered the author of the subsequent revision. It may be useful to classify or characterize practitioners based upon the lifetimes of the revisions they make. Also, one might consider whether or not practitioners do a revision that modifies the configuration that they introduced in a earlier revision, or whether or not practitioners just as easily (and often) maintain each others configuration fragments.

In this work, we did not much consider how the declarative configuration can be influenced by the revising practitioners intent or style. This because the layout of the configuration is nearly completely dictated by the device operating system. However, there are a subset of stanza types that allow for more variety in the expression of their purpose. For instance, access control lists (ACLs) contain statements that can be ordered by the operator, and multiple orderings and arrangements can have the same effect; some orderings are likely more concise or understandable than others. Therefore, it may be fruitful to consider whether or not some revisions are simply refactorings, like in software development. Further, the identification of cloned configuration fragments amongst devices, as in code clone analysis of software, could identify oft used configuration idioms.

Lastly, the goal of measuring effort in terms of revisions lifetimes was to provide a measurement of complexity. For instance, one might wonder which stanza types are more complex as evidenced by their modification (presumable fixes) in rapid succession. We did not implement nor even propose a complexity metric in this work, but future work could explore this topic, and determine whether or not certain refactorings are more or less complex.

## Conclusion

In this paper we presented two techniques: *(i)* an initial application of software development analysis tools to network operations and *(ii)* the beginnings of network operations-specific approach to measuring practitioner effort to guide new tool development. We applied these techniques in case studies of the network configuration repositories of both a large campus network and a service-provider network. By analysis and visualization, we compared and contrasted the two networks, investigating the value of metrics (*e.g.,* LOC) and exposing practitioner behaviors when using SCM and IDE-like tools. Lastly, we evaluated the analogy-based application of software development mining tools to the discipline of network operations by performing expert interviews. This expert feedback suggests the promise of our approach as both a technique to visualize the operation of real networks and as an aid to management and other stakeholders in understanding where operational effort is concentrated in large computer networks.

In closing, we have provided evidence that existing software development analysis techniques are of significant value when applied in the network operations domain. These methods expose practitioner behavior and essentially show that network operators *are* programmers, at least in their use of similar tools. By analogy to software development, this suggests that the study of network operations can effectively inform and direct network management tool development. Our hope is that the resulting improved tools will liberate the network operator from mundane tasks, will reduce mistakes in configuration, and will enable skilled operators to focus their efforts more completely on the goal of continually increasing network reliability.

## References


[1] cvs2cl. `http://www.red-bean.com/cvs2cl/`.

[2] cvsanaly2. `http://forge.morfeo-project.org/projects/libresoft-tools/`.

[3] StatCvs-XML. `http://statcvs-xml.berlios.de/`.

[4] BALL, T., MIN KIM, J., PORTER, A. A., AND SIY, H. P. If Your Version Control System Could Talk. In *In ICSE '97 Workshop on Process Modelling and Empirical Studies of Software Engineering* (1997).

[5] BENSON, T., AKELLA, A., AND MALTZ, D. Unraveling the Complexity of Network Management. In *NSDI '09: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation* (2009).

[6] COUCH, A., AND SUN, Y. On Observed Reproducibility in Network Configuration Management. *Science of Computer Programming 53*, 2 (November 2004), 215–253.

[7] DRAHEIM, D., AND PEKACKI, L. Process-Centric Analytical Processing of Version Control Data. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution* (Washington, DC, USA, 2003), IEEE Computer Society, p. 131.

[8] DRAHEIM, D., AND PEKACKI, L. The Bloof Project. `http://bloof.sourceforge.net`, 2003.

[9] GERMAN, D., AND MOCKUS, A. Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering* (2003), pp. 63–67.

[10] GOULD, W. Backing up your network with RANCID. `http://www.linux.com/feature/55873`, 2006.

[11] PLONKA, D. NetCMS - Network device Configuration Management System. `http://net.doit.wisc.edu/~plonka/NetCMS/`, 1997.

[12] PLONKA, D. Sys Admin File Revision Control with RCS. *SysAdmin - the Journal for UNIX Systems Administrators* (1998), 8–24.

[13] RANCID - Really Awesome New Cisco Config Differ. `http://www.shrubbery.net/rancid/`.

[14] ROBLES, G., KOCH, S., AND GONZALEZ-BARAHONA, J. Remote Analysis and Measurement of Libre Software Systems by Means of the CVSAnalY Tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), Edinburg, Scotland, UK* (2004), pp. 51–55.

[15] SUNG, Y., RAO, S., SEN, S., AND LEGGETT, S. Extracting Network-Wide Correlated Changes from Longitudinal Configuration Data. In *Proceedings of the 10th Passive and Active Measurement Conference (PAM)* (2009), Springer, pp. 58–67.

[16] THOMAS, C., AND PLONKA, D. AANTS: Web-Based Tools for Cooperative Campus Network Administration. In *Proceedings of the Fall 2005 Internet2 Member Meeting, Philadelphia, PA, USA* (2005).

[17] TICHY, W. F. Design, implementation, and evaluation of a Revision Control System. In *ICSE '82: Proceedings of the 6th International Conference on Software Engineering* (Los Alamitos, CA, USA, 1982), IEEE Computer Society Press, pp. 58–67.