

TroubleMiner: Mining network trouble tickets

Amelie Medem, Marc-Ismael Akodjenou, and Renata Teixeira

CNRS and UPMC Paris Universit s, Univ Paris 06

LIP6 Laboratory

{amelie.medem, marc-ismael.akodjenou, renata.teixeira}@lip6.fr

Abstract—Network operators often use a trouble ticket system to track all the steps of troubleshooting and maintenance activities. The history of trouble tickets carries valuable information for network management. Unfortunately, trouble tickets are not easy to mine, because they basically contain the human description of a problem in free text. Analyzing hundreds of trouble tickets by hand is too cumbersome; it is then important to have the support of some automatic mechanism. This paper proposes TroubleMiner, a mechanism based on document clustering techniques to automatically mine network trouble tickets. We show the utility and generality of TroubleMiner by applying it to thousands of trouble tickets from two research backbone networks. Network operators can apply TroubleMiner to trouble tickets from their network to analyze general trends in network incidents and maintenance activities. Researchers can use the insights from our characterization to decide which troubleshooting or maintenance techniques are most needed.

I. INTRODUCTION

The primary function of a network trouble ticket system is to coordinate maintenance and troubleshooting activities for a network operations center (NOC). Network operators open a trouble ticket upon the request of a customer experiencing a problem or to schedule a maintenance activity. Network equipments can also open a trouble ticket automatically after an alarm. Then, the trouble ticket system documents all steps until the closure of the ticket. Clearly, trouble ticket systems are essential for everyday network operations. Moreover, the history of trouble tickets is a valuable resource to support network management decisions and to guide research on troubleshooting and maintenance techniques. Unfortunately, trouble tickets are rarely used for these purposes, mainly because the manual inspection of hundreds of trouble tickets is impractical and there are no automatic tools to assist in this task. This paper proposes an automatic mechanism to mine trouble tickets, which we call *TroubleMiner*.

Trouble tickets contain two types of fields: fixed and free form [1]. Fixed fields are often generated automatically like the trouble ticket’s identifier or the time the ticket is opened or closed. Free-form fields have no standard values. For example, the subject of the ticket is just a sentence that summarizes the problem reported in it; the problem description is a free-text field that contains details about the problem (this field can contain a copy of a complaint email sent by a customer, for instance); and the action field describes the resolution steps. Free text often contains typos or different syntaxes to describe a single type of event. Operators already use fixed fields to get statistics on NOC’s performance such as the number of tickets over time or mean-time to close a ticket. However,

fixed fields contain very limited information about the incident itself. Although some trouble ticket systems contain a type field, this field only provides a coarse classification. Network administrators could learn about unforeseen network incidents and obtain a considerably richer classification by using the free text part of trouble tickets.

This paper addresses this challenge with TroubleMiner, a mechanism that automatically organizes trouble tickets into a hierarchy labeled according to the properties of the network events. We build this hierarchy such that the most prevalent types of events appear closer to the top, whereas more detailed descriptions appear at the leaves. This structure allows network operators and architects to quickly identify the main trends in maintenance and troubleshooting. This paper makes two main contributions:

- **TroubleMiner, a mechanism to mine the history of trouble tickets.** We leverage common techniques for document clustering [2], [3] to obtain groups of related trouble tickets. Given our goal of building a hierarchy, we use a well-known hierarchical clustering algorithm [2], [3]. Our main contributions presented in Sec. III are the methodology to process and model trouble tickets; and an algorithm to process the binary tree output by the clustering algorithm into a labeled hierarchy.
- **Insights from the analysis of trouble tickets from two networks.** We apply TroubleMiner to one to three years of trouble tickets from two different networks: Abilene and Switchlan (described in Sec. II). Our characterization of trouble tickets (presented in Sec. IV) shows some disconnect between the need from operational networks and the efforts from the research community. For instance, our analysis shows that planned maintenance account for more than half of the events that require the intervention of network operators, but there are few proposals of techniques to reduce the impact of maintenance techniques.

We hope that, by making the contents of trouble tickets easy to analyze, TroubleMiner will instigate further research using these under-explored datasets. Furthermore, our analysis should inspire the design of future management systems. The management of future networks should face many of the problems of current networks. Hence, the analysis of trouble tickets from operational networks will help focus research on management of the future Internet on practically relevant problems.

II. DATASETS OF TROUBLE TICKETS

We develop TroubleMiner based on our experience with the analysis of trouble tickets from three networks. Abilene and Switchlan are research backbones that connect research institutions and universities in the US and in Switzerland, respectively. We also use trouble tickets from a large commercial VPN provider to develop TroubleMiner. However, we do not have permission to present results based on this dataset and we point interested readers to our manual analysis of this dataset [4].

We analyze Abilene’s trouble tickets from January 1, 2005 to May 28, 2007, and Switchlan’s trouble tickets from January 9, 2007 to August 4, 2008. Abilene uses an email system to record trouble tickets [5]. We process a total set of 3,450 emails and extract 1,676 unique trouble tickets. Switchlan uses a web-based trouble ticket system [6]. A limited number of tickets are publicly available. We only find 147 trouble tickets.

III. TROUBLEMINER

This section presents *TroubleMiner*, a mechanism to structure trouble tickets into a hierarchy based on the content of the problem description. TroubleMiner works in three steps. First, it analyzes the set of trouble tickets from a network to determine the best approach to model trouble tickets. Second, it uses a well-known hierarchical clustering algorithm [2] to find groups of related trouble tickets. Last, it introduces an algorithm that uses these groups to build a hierarchy of trouble tickets. This hierarchy is organized so that problems that happen more often are close to the top of the hierarchy. Sec. IV illustrates the utility and generality of this structure with two case studies.

A. Model of trouble tickets

We need to represent each trouble ticket with a simple model that can be processed automatically. Let \mathcal{T} be the set of trouble tickets of a given network and $N = |\mathcal{T}|$ its cardinality. First, we search for the set of features that best describes \mathcal{T} . We call each selected feature a *keyword*. Then, we represent each trouble ticket, $T \in \mathcal{T}$, as a vector, where each element corresponds to a keyword.

1) *Keyword selection*: TroubleMiner uses the set of words in the subject and problem description of trouble tickets. The VPN provider’s trouble tickets contain a succinct problem description, but Abilene and Switchlan can sometimes have a long description with details about the impact that are not relevant to characterize the problem. Our manual analysis of trouble tickets of Abilene and Switchlan suggests that operators usually describe the problem with one or two sentences and then give further details. So, to focus on the essence of the problem, we only consider words in the first two sentences of the problem description. The set of features is then the combination of each word that appears in the subject and the first two sentences of the problem description of all trouble tickets, $T \in \mathcal{T}$. Given this initial set of features, we select the set of keywords as follows.

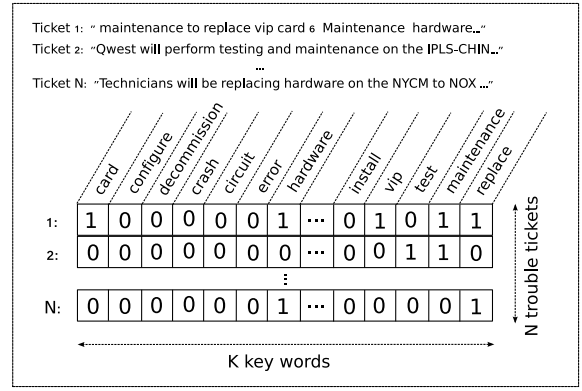


Fig. 1. Vector representation of trouble tickets

Stemming. Stemming is the process for reducing words to their stem or “root” form. For example, “migrating” and “migrated” have “migrat” in common. This process helps to reduce the number of words and to drop additional details that are not necessary in the analysis of trouble tickets. In our study, we stem words with the Porter’s Stemmer adapted to English [7].

Removal of low frequency words. Our analysis shows that more than half of the words of both networks appear in only one trouble ticket. Although such words might capture some unique detail about the problem, they cannot be used to summarize or classify the problem reported in a group of trouble tickets. Hence, we remove all words that appear only once from the set of potential keywords.

Removal of irrelevant words. Not all words in a trouble ticket express the essence of the problem. For example, in the description “we will perform a software upgrade on the router Swice2”, only “router”, “software” and “upgrade” are enough to fully describe the problem. Therefore, it is also important to filter words that do not carry information about the problem. Our analysis of trouble tickets shows that the following groups of words can be safely removed: stop words (such as “a” or “and”), general words (like “engineer” or “vendor”), some verbs, and the names of routers and neighbor networks.

We run these steps twice. The first pass outputs a list of potential keywords. We manually verify this list to correct any spelling mistakes and to find synonyms. The second pass gets the final list of keywords. Switchlan has 72 keywords, and Abilene 164.

2) *A vector per trouble ticket*: Given a set of keywords, we can model a trouble ticket as a vector, T , where each element, $T(k)$, represents the importance or the *weight* of a keyword, k , to the trouble ticket. The main challenge is to select the weighting scheme that best describes trouble tickets.

The simplest representation is to use a binary vector, where $T(k) = 1$, if keyword k appears in the trouble ticket; and $T(k) = 0$, otherwise. Fig. 1 illustrates a set of trouble tickets represented with a binary vector. The main drawback of the binary representation is that all keywords are given the same weight. This representation could work well for flat clustering,

but our goal is to build a hierarchy where keywords that appear at the top represent a broader class of network events.

We evaluate four weighting schemes proposed for document clustering [8]:

- *Term frequency* (TF) sets $T(k)$ to the number of times k occurs inside T . Consequently, the weight of a keyword k may be different between trouble tickets, i.e., $T(k) \neq T'(k)$, for two trouble tickets T and $T' \in \mathcal{T}$.
- *Document frequency* (DF) uses the number of trouble tickets in \mathcal{T} that contain k as the weight for k . If T and T' contain the keyword k , then $T(k) = T'(k)$.
- *Inverse document frequency* (IDF) is $1/DF$. The intuition behind IDF is that rare keywords are better for capturing what is unique about a document.
- $TF \times IDF$ combines the frequency of a keyword inside a trouble ticket with the number of trouble tickets in which it appears. A keyword is assigned a high weight if it appears both in few trouble tickets and many times inside a trouble ticket.

The choice of the weighting scheme is tightly related to the goal of the clustering. For instance, in the retrieval of web pages, the most common weights are TF and $TF \times IDF$ [9], [10]. Often, a large number of web pages contains a given set of keywords. In this case, a weighting scheme that takes into account the frequency of a keyword inside a web page helps single out the most relevant pages to the search.

We find that DF works best for our goal of building a hierarchy that organizes trouble tickets according to the prevalence of network problems. TF is not appropriate for this case. For example, a group of trouble tickets that contains the keywords “card” and “crash” describes the same network problem irrespective of the number of times any of these keywords appears in each trouble ticket. $TF \times IDF$ is not appropriate either to get a hierarchy of trouble tickets. $TF \times IDF$ gives high weight to words that occur many times within a small number of trouble tickets, which does not match with our goal of building a hierarchy. For example, the keyword “maintenance”, relevant to separate planned and unplanned trouble tickets at a high level of the hierarchy, will get a very low weight. On the other hand, DF gives higher weights to keywords that appear in more trouble tickets. As a result, the clustering algorithm will build a hierarchy with the more frequent keywords closer to the top, thereby emphasizing the most prevalent network incidents.

Therefore, we represent each trouble ticket T as a vector using the DF weighting scheme. Each element $T(k)$ equals the number of tickets in \mathcal{T} that contain the keyword k .

B. Correlating trouble tickets

Given a set of N trouble tickets represented as vectors, $\mathcal{T} = \{T_1, \dots, T_N\}$, TroubleMiner correlates them using hierarchical clustering [2]. Hierarchical clustering is perfect for our needs, because it already outputs a binary tree, which is easy to post-process into a hierarchy of trouble tickets. It is also simpler to configure than other clustering mechanisms, because it does not require a predefined number of clusters.

TABLE I
COMPARISON OF DISTANCES AND CLUSTER SIMILARITIES

	Abilene			Switchlan		
	avg.	comp.	single	avg.	comp.	single
Euclidian	0.96	0.96	0.96	0.93	0.92	0.92
cosine	0.94	0.93	0.85	0.88	0.85	0.76

We use the hierarchical agglomerative clustering algorithm [3], [2]. The following steps summarize this algorithm.

- 1) Create an $N \times N$ similarity matrix, where an element (i, j) in this matrix equals to the distance between trouble tickets T_i and T_j , $d(T_i, T_j)$;
- 2) Assign each trouble ticket, T , to a cluster;
- 3) Merge the two closest clusters;
- 4) Compute the similarity between the new cluster and all existing clusters;
- 5) Repeat Steps 3 and 4 until all trouble tickets fall into a single cluster of size N .

This algorithm relies on two similarity metrics: Step 1 computes the distance between pairs of trouble tickets, $d(T_i, T_j)$; and Step 4 computes the similarity between pairs of clusters. These metrics are essential to determine the structure of the resulting tree, and consequently the quality of the hierarchy.

Distance between trouble tickets. We compute the distance between two tickets, $d(T_i, T_j)$, using two commonly-used metrics: Euclidean and cosine [3].

Similarity between clusters. We evaluate three approaches to determine the similarity between two clusters C and C' : The *single-*, *complete-* and *average-link* approaches [2], [3]. The single-link approach sets the similarity of C and C' to the distance between their most similar trouble tickets (i.e., $\min_{T \in C, T' \in C'} d(T, T')$). The complete-link approach uses the distance between their most different trouble tickets (i.e., $\max_{T \in C, T' \in C'} d(T, T')$). The average-link approach computes the similarity between two clusters as the average distance between all pairs of trouble tickets.

We run the hierarchical clustering algorithm with each of the combinations of distances (Euclidean and cosine) and cluster similarities (single, complete and average link). The resulting clustering is a binary tree, which we denote as \mathcal{BT} . Then, we measure the quality of \mathcal{BT} using the cophenetic correlation coefficient [11]. The *cophenetic correlation coefficient* measures how close the distances between trouble tickets in \mathcal{BT} match the pair-wise distances of trouble tickets in the original dataset, \mathcal{T} . The closer this coefficient is to 1, the better the binary tree reflects the original structure of the trouble tickets.

Table I presents the cophenetic coefficient of the clustering of trouble tickets of Abilene and Switchlan using different distance and similarity metrics. We choose Euclidean distance, because the clustering with Euclidean distance is consistently better than with cosine (the cophenetic coefficient with Euclidean is always above 0.9). The choice of cluster similarity metric is not as trivial. Although the average-link approach is better in most of the cases, it is not significantly better

and, in the clustering of Abilene trouble tickets with Euclidean distance, all approaches perform the same. Still, we choose the average-link approach, because it does perform better in our datasets and it has been shown to be the most effective cluster similarity metric for other datasets as well [12], [9], [10].

C. Hierarchy of trouble ticket

This section introduces TroubleMiner’s algorithm to obtain a hierarchy in which groups of trouble tickets are labeled according to the network problem they report. The result of the hierarchical clustering is a binary tree (\mathcal{BT}) with unlabeled nodes. This tree captures the correlation between trouble tickets, but does not explicitly output the set of keywords that each pair of tickets shares. In addition, the binary structure of the tree is too restrictive. For example, say that there are three trouble tickets with keywords (“maintenance”, “router”), (“maintenance”, “link”), and (“maintenance”, “power”). It is clear that this dataset is better represented by a tree with one root, labeled “maintenance”, with three children (“router”, “link”, and “power”) than by a binary tree. Therefore, we transform \mathcal{BT} into a hierarchy, \mathcal{H} , which is a labeled, N -ary tree. First, we label all nodes in \mathcal{BT} . Then, we combine nodes of \mathcal{BT} that have the same label relaxing the binary structure and obtaining \mathcal{H} .

Labeling \mathcal{BT} . We label \mathcal{BT} in depth-first order. Remember that each leaf contains exactly one trouble ticket, T . So, the label of each leaf is just the set of keywords present in T . Once all nodes in the subtree of \mathcal{BT} rooted at u are labeled, we can label u with all keywords that belong to the intersection of the keywords in the labels of all of u ’s children. If this intersection is empty, then we label u as “unknown”.

From labeled \mathcal{BT} to \mathcal{H} . We can now use Alg. 1 to transform the labeled \mathcal{BT} into a concise hierarchy with meaningful labels, which we denote by \mathcal{H} . Basically, the algorithm merges a sequence of parent-child nodes that have the same label. It traverses \mathcal{BT} in breadth-first order, and iteratively compares the label of each node to that of its children. The algorithm only adds a node to \mathcal{H} if its label is different than that of its parent.

Take the example in Fig. 2, where nodes are identified with their labels and cardinality (i.e., the number of trouble tickets contained in their subtree). Fig. 2 presents the binary tree and the resulting hierarchy. The right subtree of the labeled \mathcal{BT} in Fig. 2 contains some examples of sequences of labeled nodes with the same label (for instance, “maintenance”(17), “maintenance”(5)). Similarly, the root of the tree and two nodes in the left subtree all have label “unknown”. The goal of the algorithm is to find the head of a sequence of nodes that contains a given label, ℓ . The *head* of a sequence with label ℓ is the node closest to the root of \mathcal{BT} labeled with ℓ . In our example, the node “maintenance”(17) is the head of the sequence with label “maintenance” and the root of the tree is the head of the sequence with label “unknown”.

Alg. 1 starts by visiting \mathcal{BT} ’s root and adding the root to the new tree \mathcal{H} . Every time the algorithm encounters a node u with a new label ($u.head$ is empty), it adds u to \mathcal{H} . In

Algorithm 1: Transform binary into N -ary tree

```

input :  $\mathcal{BT}$ , binary tree
output:  $\mathcal{H}$ ,  $N$ -ary tree
1 begin
2   for each node  $u \in \mathcal{BT}$  in breath-first order do
3     if  $u.head$  is empty then
4       add  $u$  to  $\mathcal{H}$ ;
5        $u.head \leftarrow u.id$ ;
6     for each child  $v$  of  $u$  do
7       if  $v.label \neq u.label$  then
8          $v.parent \leftarrow u.head$ ;
9       else if  $v.label = u.label$  then
10         $v.parent \leftarrow u.parent$ ;
11         $v.head \leftarrow u.head$ ;
12 end

```

addition, it saves u ’s identity as the head of the sequence for u by updating $u.head$ (lines 3-5). Next, Alg. 1 visits all u ’s children. If a child v has the same label as u , then we set $v.head$ with $u.head$. This step guarantees that v is not added to \mathcal{H} , because when Alg. 1 visits v , $v.head$ will not be empty and the check on line 3 will fail. On the other hand, if $u.label$ is different than $v.label$, then $v.head$ is not updated (consequently it will be empty when the algorithm visits v). Only v ’s parent information is updated to point to the head of u ’s sequence. This update is important when u is not the head of its sequence.

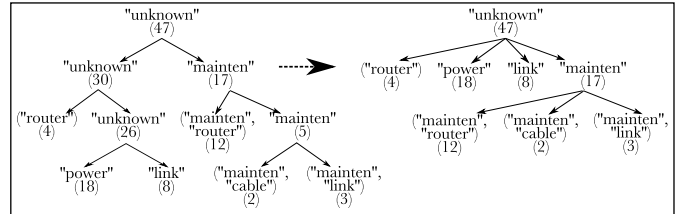


Fig. 2. Example of transformation of \mathcal{BT} into \mathcal{H} . Each node is represented by its label and its cardinality. “mainten” is equivalent to “maintenance”

IV. CASE STUDIES

We now show that TroubleMiner outputs meaningful hierarchies that are useful to characterize network incidents from trouble tickets.

Fig. 3 and 4 present TroubleMiner’s output when applied to Abilene’s and Switchlan’s trouble tickets. In these figures, we annotate each cluster with two values: the total number of trouble tickets contained in the cluster; and the percentage of tickets in the cluster. We compute this percentage over the total number of trouble tickets (or N), for each network. For clarity of presentation, we only present clusters that contain at least 0.4% of Abilene’s trouble tickets and 1% of Switchlan’s. Note that nodes inherit the labels of all of their ancestors. For example, in Fig. 3, at the third level, the label of the node “cable” should be “maintenance, link, cable”.

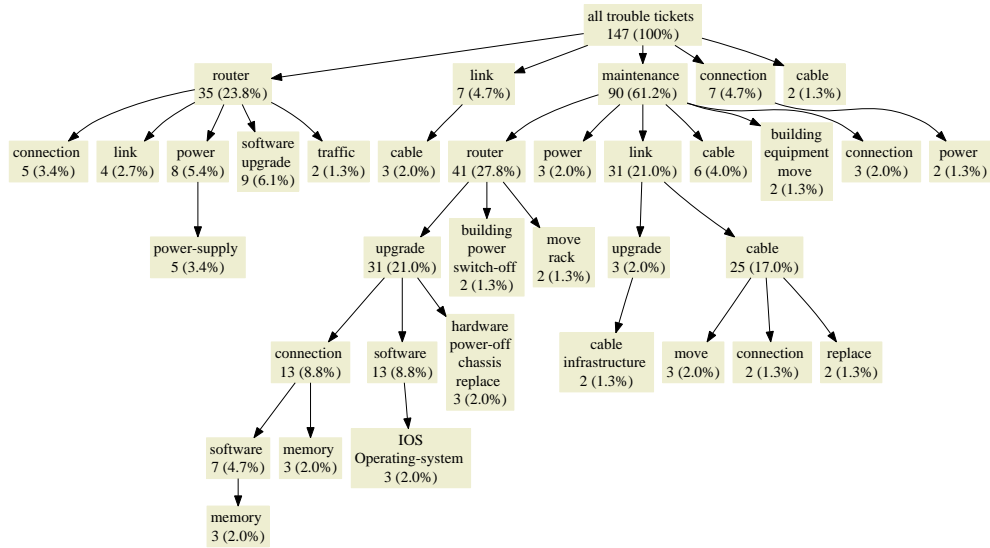


Fig. 3. Hierarchy of Switchlan’s trouble tickets (only for clusters with more 1% of trickets).

We highlight the main lessons from the characterization of Abilene’s and Switchlan’s trouble tickets.

More than half of the trouble tickets correspond to maintenance activities. Fig. 4 shows that 52% of Abilene’s trouble tickets refer to maintenance activities, this percentage is even higher for Switchlan (Fig. 3 has 61.2% of maintenance activities). Although some previous studies [13], [14], [15] have mentioned the importance of maintenance for network operations, we are the first to present a quantitative analysis of trouble tickets to corroborate this intuition.

Such a large fraction of maintenance activities may even seem to contradict previous results that show that only 20% of link failures are related to maintenance [14]. However, this previous study measures link failures that appear in IS-IS messages and it labels as maintenance only link failures that happened during the maintenance window. This implies that: (i) it cannot observe maintenance activities that do not impact IS-IS (our preliminary analysis indicates that some events in trouble tickets do not impact IS-IS [4]); (ii) it misses all maintenance activities that happen outside of the maintenance window; and (iii) the fraction of maintenance activities may seem less significant when comparing to all IS-IS link failures, because most of these failures are short [14] and consequently do not appear in trouble tickets [4].

Despite the importance of planned maintenance in network operations, there have only been a handful of proposals to limit the impact of maintenance on customer’s traffic [16], [13], [17]. Our analysis suggests that researchers should put more effort in developing techniques to reduce the impact of maintenance on customers and to automate maintenance activities, so that they require less effort from network operations’ teams.

The most common types of maintenance are related to router software upgrades and cable. Most of the router maintenance relates to software. Adding the different boxes

labeled with software under maintenance in the hierarchy of Switchlan we have 13.5% of all trouble tickets related to software upgrade; this percentage represents almost half of all router maintenance. We obtain similar results for Abilene: 6.3% of trouble tickets relate to router software maintenance (this represents more than half of all router maintenance). Router software maintenance mainly consists of operating system upgrade or installation. This indicates that router vendors should put greater effort in seamless software upgrade. One promising approach is to execute many operating system instances in parallel [18] so that while one instance is being upgraded, the others can take over.

Both Abilene and Switchlan have a considerable fraction of trouble tickets that reports planned maintenance on links. Abilene has 23.9% of trouble tickets that refer to circuit or cable maintenance, whereas Switchlan has 25% of trouble tickets on maintenance of cable or link. Link maintenance activities happen often to relocate, rollback, and test optical fibers; or to replace fiber components (such as card and optical amplifier). Network operators frequently inspect cable for potential rodent damage. They can also relocate fiber after road construction, train derailment or natural disasters. Link and cable maintenance are inevitable; it is hence essential to build networks with enough physical redundancy and provide operators with easy ways to reconfigure their network in preparation for taking out a link.

The prevalence of unexpected events depend on the network topology and connectivity. We consider any trouble ticket without the keyword “maintenance” as an unexpected incident. With this definition, we count 48% of unexpected events in Fig. 4 and 38.8% in Fig. 3. The balance between router and cable incidents is different depending on the network. Switchlan has more failures that affect routers, whereas Abilene has more failures on circuit or connection. This difference reflects the topology and connectivity of each network.

- [9] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining'00*, 2000.
- [10] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets," in *ACM Press*, 2002.
- [11] R. R. Sokal and F. J. Rohlf, "The comparison of dendrograms by objective methods," in *Taxon*, 1962.
- [12] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning; data mining, inference and prediction," in *Springer Verlag, New York*, 2001.
- [13] A. Shaikh, R. Dube, and A. Varma, "Avoiding instability during graceful shutdown of ospf," tech. rep., In Proc. IEEE INFOCOM, 2002.
- [14] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of Failures in an IP Backbone," in *Proc. IEEE INFOCOM*, 2004.
- [15] R. Teixeira and J. Rexford, "Managing routing disruptions in internet service provider networks," *IEEE Communication Magazine*, 2006.
- [16] J. Moy, P. Pillay-Esnault, and A. Lindem, "Graceful OSPF Restart." RFC 3623, Nov 2003.
- [17] N. Dubois, B. Decraene, B. Fondeviole, and Z. Ahmad, "Requirements for planned maintenance of bgp sessions." Expired Internet Draft, draft-dubois-bgp-pm-reqs-02.txt, July 2005.
- [18] M. Caesar and J. Rexford, "Building bug-tolerant routers with virtualization," in to appear at the *ACM SIGCOMM Workshop on Programmable Routers for the Extensible Services of Tomorrow (PRESTO)*., 2008.
- [19] G. Labovitz, A. Ahuja, and F. Jahanian, "Experimental study of the Internet Stability and Backbone failures," in *Proc. International Symposium on Fault-Tolerant Computing*, Jun 1999.
- [20] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP Misconfiguration," in *Proc. ACM SIGCOMM*, Oct 2002.
- [21] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, May 2005.
- [22] M. Steinle, K. Aberer, S. Girdzijauskas, and C. Lovis, "Mapping moving landscapes by mining mountains of logs: novel techniques for dependency model generation," in *Proc. of the 32nd international conference on Very large data bases*, 2006.
- [23] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proc. of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005.
- [24] C. Lim, N. Singh, and S. Yajnik, "A log mining approach to failure analysis of enterprise telephony systems," 2008.
- [25] Z. Pandi, "Analysis of public trouble ticket data," tech. rep., Budapest University of Technology and Economics, Department of Telecommunications, 2005.
- [26] J. Tanaka, "Analysis of trouble tickets Issued by APAN JP NOC." www2.jp.apan.net/meetings/busan03/noc/tanaka.ppt.
- [27] Y. Huang, N. Feamster, A. Lakhina, and J. Xu, "Diagnosing network disruptionss with network-wide analysis," in *Proc. ACM SIGMETRICS*, Jun 2007.
- [28] M. Roughan, T. Griffin, Z. Morley, M. Albert, and G. B. Freeman, "Ip forwarding anomalies and improving their detection using multiple data sources," in *ACM SIGCOMM Workshop on Network Troubleshooting*, 2004.
- [29] L. Lewis and G. Dreo, "Extending trouble ticket systems to fault diagnostics," *Network, IEEE*, 1993.
- [30] C. Melchioris and L. Tarouco, "Troubleshooting network faults using past experience," in *Network Operations and Management Symposium, 2000. NOMS 2000. IEEE/IFIP*, 2000.