Root cause localization in large scale systems

Emre Kıcıman
Stanford University

Lakshminarayanan Subramanian
University of California, Berkeley

Abstract

Root cause localization, the process of identifying the source of problems in a system using purely external observations, is a significant challenge in many large-scale systems. In this paper, we propose an abstract model that captures the common issues underlying root cause localization and hence provides the ability to leverage solutions across different systems.

1 Introduction

Many large-scale systems, as diverse as Internet service clusters, inter-domain routing in the Internet and software systems, suffer from a common problem: when the system fails to function properly, it is often difficult to determine which part of the system is the source of the problem. The fundamental challenge is that, often times, the symptoms of a failure manifest as *end-to-end failures* in the operation of the system as a whole, without causing obvious failures in the system's pieces; simply noticing that something has gone wrong is not enough to tell us where to look to fix it. We illustrate this problem using three diverse examples. First, detecting the source of a large-scale outage in Internet routing can be a nightmare—network operators often call other operators and exchange large volumes of emails on the operator mailing lists [17]. Second, a significant amount of time in managing Internet service clusters, like web farms, is spent detecting and localizing service failures [19, 7]. Finally, it is well-known that diagnosing bugs in large-scale software systems using even the best debugging technologies is a laborious task involving several human hours or more [15].

We refer to *root cause localiztion* as the process of determining the source of problems in a system purely by externally observing the behavior of the system. The communities that build these large-scale systems have historically taken different approaches to solving this problem—alternatively referred to as fault diagnosis, alarm correlation, root cause analysis, and bug isolation in the context of a wide variety of systems [4, 5, 3, 23, 11, 8, 21, 6]. Despite this, we take the position that many of the challenges are common across a surprisingly diverse set of these systems.

In this paper, we capture the commonality across different systems by defining an abstract system model and formalizing the root cause locatization problem for this model. Our model explicitly represents the nature of end-to-end failures, captures the common theoretical and computational challenges, and separates system-specific challenges into the process of mapping a system representation into the abstraction. While the generality of the model will definitely not capture several intrinsic details of a system, it does provide the ability to re-use techniques from other systems and tune them for system-specific needs.

The primary motivation of this abstract modeling is to set up a clear bridge that enables researchers in different communities to share knowledge in a common language. In particular, we hope to enable and attract theory and machine learning researchers to attack this general problem. To highlight this promise, we show how one can leverage existing techniques to solve specific aspects of the general problem and briefly illustrate how these solutions have been applied in the three application domains mentioned above.

2 Example applications

We ground our approach to root cause localization by considering the problem in the context of three very different systems: root cause analysis of Internet routing dynamics, failures in clustered Internet services, and the bug isolation problem in software systems. In this section, we present a brief introduction to these applications and the root cause localization problem in each.

2.1 Root cause analysis of BGP dynamics

Understanding the dynamics of Internet routing and pinpointing the source of routing problems is critical to address many of the shortcomings of the Border Gateway Protocol (BGP), the *de facto* interdomain routing protocol.

Observing a route update is a clear symptom that some event has occurred. A BGP health inferencing system [4, 5, 9] that performs root cause analysis of BGP dynamics uses data collection centers like Routeviews [24] and RIPE [22], which continuously receive streams of route updates from multiple vantage points. Each route update is associated with path-vector information describing the entire path at the granularity of Autonomous systems (AS's). The underlying root cause localization problem in BGP can be stated as follows: *Given route updates observed at multiple vantage points, determine the potential set of locations of events (at the granularity of AS's) that could have triggered each route update.*

Clearly characterizing the exact cause of an event is fundamentally hard, given that the potential list of causes of routing

events in BGP are innumerable (due to the volume of possible BGP policies). Hence, the root cause localization problem is restricted to determining the location of an event as opposed to identifying the exact cause.

2.2 Failures in Internet service clusters

Today's Internet services (e-commerce, search engines, enterprise applications and others) commonly suffer from *brownouts*, where part of the functionality of a site goes down or is unavailable, resulting in the failure of user requests. It is critical to quickly determine the source of such problems to reduce the overall downtime of the system. While certain types of failures such as a process crash are easy to detect, the challenging aspect arises when the only detectable symptom of a failure is an end-to-end failure, *e.g.*, a front-end web server observes users' failed HTTP requests.

Large Internet services are usually built using clusters of machines (from 100s to over 50000 machines [16]) divided into multiple tiers (a front-end tier of web servers, multiple tiers of application logic, and a back-end tier of persistent storage) and a user's HTTP request usually traverses most of the tiers in the system. To aid in the diagnosis of such a large system, it is becoming common practice to dynamically record and log the *path* of a request (the machines and services used to fulfill the request) [7, 2, 1].

The root cause localization problem in such a system can be formulated as: *Given the paths of both successful and failed requests, determine the set of components most likely to have caused the failures.*

2.3 Bug isolation

It has long been recognized that the existence of bugs is practically guaranteed in large software projects. Bugs that are deterministic and easily reproduced are relatively easy to track down and fix. Other bugs, humorously named *heisenbugs* [10], are non-deterministic and quite difficult to track down, even with the latest debugging tools.

Recently, Liblit *et al.* have proposed an approach they call *statistical debugging* [15], where they advocate constant sampling of the code-level behaviors of end-user software during normal execution. These behaviors include the results of conditional tests, the return values of functions, etc. By discovering which of these behaviors are most correlated with symptoms of a heisenbug, statistical debugging helps programmers understand and discover a bug's true cause.

Statistical debugging is a direct counterpart to the root cause localization problem in other systems, and can be re-stated as: *Given the code-level behaviors associated with both correct and buggy executions of a program, determine what parts of the code are most likely to have caused the bug.*

3 Root cause localization problem

In this section, we define a basic form of the root cause localization problem and describe how this problem abstractly captures the three example applications described above. Later in Section 5, we provide two refinements to this problem which can capture additional aspects of more complex failures.

3.1 System model

We abstractly model a large scale system simply as a collection of a specified set of components interacting with each other. While a clear characterization of the set of components is dependent on the system/application under consideration, the definition of a component should satisfy three properties: (a) each component should be *disjoint* from other components; (b) all components when considered together should completely represent the system; (c) a component, as a whole, should be *visible* to an instrumentation box diagnosing the system.

Given that failures are externally visible only as end-to-end failures, we assume that the behavior of the individual components is not diagnosable in isolation. The only observations that are visible to external instrumentation are what we define as *quarks* - the smallest end-to-end observable unit of a failure or success. The *health result* of a quark signifies whether a quark is a success or a failure. Each quark essentially represents a tuple consisting of: (a) the subset of components used by the quark; (b) a health result.

While this model encapsulates the essential computational elements of root cause localization, it explicitly abstracts several system-specific concerns. The possible locations of a failure within the system are represented in the abstract model as components, and determining what these possible locations are in the context of a specific system must be addressed on a case-by-case basis. Anticipating how the symptoms of a failure may manifest will guide the definition of a quark. Finally, for the abstract model to be an aid in root cause localization, the association between a quark and its set of used components must represent how faults propagate from their source to their symptoms. Without a reasonable approximation of this fault propagation, the failed quarks in the abstract model will not be able to lead us to the cause of a failure. All these system-specific concerns must be answered in the process of transforming a physical system model into the abstract model.

We now revisit the three examples and define the components and quarks in them. In BGP, we associate two different types of components: AS's and inter-AS links. This is primarily to distinguish between events triggered across inter-AS boundaries (*e.g.*, peering link failure) and internal routing events within an AS. Given a path-vector route (A, B, C) traversing three AS's, the components are A, B, C, (A, B) and (B, C). Every route observed at a vantage point represents a quark. Routes that are stable represent healthy quarks and any prefix that is updated is an unhealthy quark signifying the occurrence of a routing event.

In the case of Internet service clusters, the components of our model are the machines and software services running on them, across all tiers of the system. The work done to satisfy a user's HTTP request is the quark of our model. All the machines and services that process some part of the HTTP request make up the quark's associated components. Separate failure detectors, such as HTTP error monitors or anomaly detectors, can be used to determine whether an HTTP request succeeded, and mark the health of a quark accordingly.

In software programs, a code module that performs a specific functionality is a component and the code-level behavior corresponding to every execution of the program represents a quark. A correct program execution is a successful quark and a buggy execution is an unsuccessful quark. Note that every execution of the code traverses a different set of components depending on the system environment and input parameters.

When applying our model to a real system, we note that the model fundamentally assumes a componentized system—a requirement trivially met by most distributed systems. Less trivially, to practically apply this model to a real system we must be able to observe at least some of the quarks in the system along with their associated components.[1]

3.2 Modeling partial failures

Components across different types of systems have widely varying granularities. Some components, like AS's, are by themselves large distributed systems while blocks of code in a software program may be quite simple. In general, we can associate a component with one or more functionalities, where the number of functionalities is dependent on the type of system and scale of the component.

We define a *partial failure* of a component to be the case when one or more functionalities of a component fail (or are modified) while the others are unaffected. In this model, we implicitly assume that partial failures within one component are *independent* and do not influence partial failures in other components. While this does not account for more complex failures, in Section 5 we describe two refinements to relax this limitation.

The failure of a functionality within a component manifests itself externally by causing any quark that uses that functionality to fail. To account for this, we use a simple probabilistic model for a partial failure of a component: Given a component, C_i, let probability p_i represent the failure probability of a quark that utilizes component C_i. This probabilistic model makes no assumptions about the specific functionalities that are associated with a component. While in certain applications, one may be able to specify all the functionalities associated with a component, here, we assume that this information is not available. It is important to note that the value p_i is dependent on the distribution of quarks using the failed and successful functionalities of a component C_i ($p_i = 0$, if a completely unused functionality in a component has failed).

[1]For example, if we are unable to determine the path of a route in BGP or path of a user request in service clusters, one cannot apply this model.

3.3 Basic problem definition

Consider a large scale system with a set of components, $S = \{C_1, C_2, \ldots C_n\}$. An instrumentation box monitors some of the quarks of the system where each quark $Q = (Q_s, Q_h)$ where Q_s is a subset of components and Q_h is a binary health result represented as 0 or 1.

Based on this probabilistic model of partial failures and the assumption that partial failures are independent (a refinement in Section 5.2 handles dependent failures), we define two versions of the root cause localization problem:

1. **Deterministic version:** Given several quarks in the system of which some failed (*i.e.,* $Q_h = 0$), determine the potential set of components that have experienced a partial failure *i.e.*, list of components C_i with $p_i > 0$.
2. **Statistical version:** Given several quarks in the system of which some failed (*i.e.,* $Q_h = 0$), estimate the partial failure probability p_i for each component C_i.

We refer to the deterministic version as the *partial failure identification* problem. Identifying the components that may have failed in general is easier than estimating p_i and also requires a much smaller statistical sample set of quarks. However, when the potential set of faulty components is very large, the estimates of p_i can help pinpoint the mostly likely faulty components.

4 Potential solutions

In this section, we describe four approaches to handling root cause localization in different application domains. While none of these approaches completely solve the problem, they each work effectively under their own set of assumptions. Showing that these approaches may apply to the abstract model of root cause localization indicates that they may also be applied to a broader set of systems outside their original domain.

Of these solutions, the link-rank and minimum set-cover algorithms address specific aspects of the deterministic version of the problem and decision tree learning and logistic regression address the statistical version of the problem.

4.1 Generic challenges

There are two core challenges in root cause localization, both related to the quantity and quality of our observations of the system. In the end, how well we address these challenges in the context of a specific system determines how effective a solution is to the root cause localization problem.

Component Visibility and System Structure: The accuracy with which we can address the root cause localization problem is dependent on the *coverage* of the quarks (how well our

observed quarks cover the components in the system) and the *variety* of the quarks (how much the quarks' associated component sets differ from one another). In practice, the set of components that a quark covers is largely dictated by the system's structure.[2] In particular, given the limitations of end-to-end failures, we cannot localize a fault in a component which is not used by any quarks; nor can we diagnose a failure in a system where all quarks are identical, such as a parallel computing system where every calculation depends on every component in the system. In less extreme cases, we may able to localize a problem to a subset of components but not pinpoint the specific component whose failure triggered a failed quark.

Time granularity: It is not enough to state that our observed quarks must provide good coverage and variety; they must provide good coverage and variety within a relevant period of time. To accurately pinpoint component failures, we require a significant sample set of quarks during the period of the failure. While failures may tend to persist for long periods in certain applications (*e.g.,* bugs in software programs), several applications like Internet routing use inbuilt mechanisms to adapt and recover from failures. To perform diagnosis in self-adapting systems, we must have a significant sample set of quarks during short failure periods.

4.2 Link-rank algorithm

Link-rank [14] has been proposed as an algorithm to pinpoint the sources of large routing events in BGP, *i.e.,* events that simultaneously affect the dynamics of many routes. Given route updates from multiple vantage points, the basic algorithm computes a *link-rank* for every inter-AS link as the number of routes using the link. The *rank-change* associated with a link represents the change in link-rank over a short period of observation and a link is reported as a suspect candidate choice of a routing event if the rank-change is above a prescribed threshold (in absolute terms). Several recent works on BGP root cause analysis [4, 5, 9], have further developed this basic technique to improve the accuracy of the results.

The link-rank algorithm can be viewed as a simple approach for the partial failure identification problem, where the rank-change metric is a measure of the number of failed quarks that utilize a component. A component is reported as a candidate if this count is above a system-specific threshold.

We make two additional observations. First, the link-rank algorithm does not identify all components with partial failures but only pinpoints those components which clearly stand-out as candidates[3]. Second, the algorithm only uses information from failed quarks to determine failed components.

[2]Most systems do not provide the flexibility to define quarks with an arbitrary set of components but rather constrain this set based on the system structure.

[3]The threshold determines the confidence level in the correctness of the output of the algorithm.

4.3 Minimum set cover

A complete failure model represents a specific case of partial failures where $p = 0$ or $p = 1$. Under this assumption, one can view the root cause localization problem as an optimization problem to identify the *minimum set of failed components* that can explain all the failed quarks. This optimization implicitly assumes that the number of failures in the system is small at any given instant and hence attempts to minimize this number.

One can transform this optimization problem to the classical *set cover* problem [12] using two steps. First, any component that is part of a successful quark will not be the cause of a failure and is removed from consideration. Second, given the set of quarks associated with each remaining component, computing the *minimum set cover* that covers all failed quarks is equivalent to determining the minimum set of failed components that can explain the root cause of all failed quarks. While the set cover problem is NP-complete, one can leverage good approximation algorithms with an approximation ratio of $\log N$ (N = number of failed quarks) for this problem [12].

The minimum set cover method works only under certain assumptions. First, it assumes a complete failure model where the failure of a component completely lasts during the period of observation *i.e.,* no component recovers from a failure during the observation period. Second, the solution to minimum set cover is not unique. For example, if two components occur in all failed quarks, then the algorithm should report both as suspect as opposed to just one of them.

4.4 Decision tree learning

In previous work, we have used decision tree learning to localize failures in the context of Internet service clusters [7, 13]. It seems like that this technique can also be useful in our abstract model.

A *decision tree* is a data structure that represents a classification function, where each branch of the tree is a test on some attribute of the input, and where the leaves of the tree hold the result of the function. *Decision tree learning* is the process of building a decision tree to most accurately classify a set of training data [20].

To solve the root cause localization problem, we learn a decision tree to classify (predict) whether a user's HTTP request is a success or a failure based on its associated components. Of course, we already know the health of the request—what interests us is the structure of the learned decision tree; looking at which components are used as tests within the decision tree function tells us which components are correlated with request failures. Similarly, by applying decision trees to classify quarks based on their associated components, we can hope to localize faults in our abstract model.

Unlike the link-rank algorithm, the decision tree uses the information from both the healthy and unhealthy quarks to decide which components in the system are most likely to be

faulty. Its statistical nature means it gracefully tolerates inconsistencies in the training data. Also, because decision trees can naturally represent disjunctive hypotheses, they are robust to multiple simultaneous independent faults.

4.5 Logistic regression

In statistical debugging, Liblit *et al.* use *logistic regression* to discover which low-level behaviors in a program's code are most correlated with buggy program runs [15].

Logistic regression fits a linear model to a set of training data, trying to learn a linear function of the low-level code behaviors that will correctly classify a program run as either correct or buggy. Under the assumption that most of the code behaviors will not be relevant to a failure, Liblit *et al.* regularize the input parameters to force the linear model to use only the few behaviors that correctly characterize the failure.

As described in Section 2.3, in our abstract model, each program run is a quark and the low-level behaviors of the code are the "components" associated with each quark. In this context, logistic regression may be able to find the components likely causing quark failures.

Like decision trees, logistic regression takes into account both healthy and unhealthy quarks, and gracefully degrades in the face of inconsistencies. While logistic regression is computationally more efficient than decision trees, it does not handle multiple independent faults very well.

4.6 Solutions recap

In this section, we have discussed solutions to the computational and theoretical aspect of the root cause localization problem. While none of the these solutions completely address the abstract problem, they do highlight the potential for sharing ideas and solutions among the various research communities. It also enables theory and statistical experts to provide improved solutions for the general problem.

Of course, the system-specific parts of root cause localization, such as determining exactly what constitutes a failure in a domain, are also important. Additionally, there is a significant opportunity for system-specific techniques to help mitigate the challenges of component visibility, system structure and time granularity. Adding new observation points within the system may increase the visibility of quarks and perhaps increase their coverage and variety. In some systems, it is possible to artificially inject new quarks to probe the system and control the set of components in this quark. This opens up a new class of solutions to root cause localization, not detailed here, based on methodical exploration of possible fault propagation paths. To improve our time granularity in systems where collecting observations exacts some cost, we might be willing to pay that cost to observe more quarks once we notice a fault in the system. Coercing transient failures into longer faults can also lengthen the amount of time we have to collect a significant set of quarks.

5 Refining the Basic Model

The model of a partial failure, as we defined in Section 3.2 is simplistic, in that it assumes failures across components to be independent of each other. In this section, we relax this assumption and refine our model to support two types of complex failures: (a) a common problem simultaneously causes several components to fail; (b) the interaction between a set of components triggers a failure. While by no means are these refinements complete enough to capture various forms of failures, we describe them primarily to show how one can extend our problem to model different types of complex failures.

5.1 Modeling simultaneous correlated failures

There exists many types of systems where several components may simultaneously fail due to an underlying common cause.[4] Such types of failures are typically hard to model and localize without additional knowledge about possible causes of simultaneous failures. For this purpose, we define *attributes* of a component as additional descriptions specified by the underlying system about the potential causes of component failures. *E.g.,* in Internet services, one useful set of attributes might include the operating systems, middleware and versions of each component. In BGP, Caesar *et al.* [4] classify causes of routing events into *disjoint equivalence classes* where each class can be viewed as an attribute. In general, we expect several attributes to be common across different components to capture commonality in failures.

With this refinement, the root cause localization problem boils down to determining the set of components and attributes which appear to be triggering failures in quarks. Many of the statistical learning theory techniques, including decision trees and logistic regression, can be extended to model attributes in conjunction with components. Of particular interest is the case of large-scale homogeneous systems where the functionalities across several components are alike (*e.g.,* nodes in a structured peer-to-peer network) and hence the attributes of many components are alike. Here, when applying statistical techniques, one can model attributes as being equivalent to the components in the system to determine common problems across components.

5.2 Failures caused by component interactions

Some failures are caused not by faults in a single component, but by multiple components interacting together. For example, latent faults in two components and subtle incompatibilities due to version differences can cause otherwise perfectly functioning components to fail when used together. Formally, we define a set of components to have an *interaction-failure* if any quark that uses all these components always fails while any quark that uses only a subset of these components is always successful. This set of components represents the smallest set of components that have an interaction failure. Finding

[4] For example, the spread of the SQL Slammer worm triggered several routers to simultaneously reset.

this smallest set of components defines the root cause localization problem in the context of interaction-failures.

While the potential number of interaction failures is exponentially large, two specific constraints in the context of many real-world systems make this problem relatively tractable. First, the number of components involved in an interaction fault is relatively small. Secondly, the system structure and observed interactions between components limits the possible interaction failures we have to consider.

Theoretically, this problem turns out to be similar to a problem in bio-informatics where the interaction between different genetic abnormalities[5] can potentially be a source of cancerous tumor cells. In [18], Michael Newton proposes a set of statistical techniques to identify these genetic abnormalities and these techniques are potentially applicable in the context of interaction failures.

6 Conclusions

The primary goal of this paper has been to point out the commonalities among the root cause localization problem in different systems. We are able to capture this commonality through our abstract definition of components and quarks as representing in a system what can fail, how failures manifest, and how faults propagate. Part of the hope behind this abstraction is to stimulate discussion among the communities that build these systems and elicit the help of theory and machine learning experts to propose solutions for the underlying problem. For example, we were able to determine the correlation between interaction-failures and the cancer detection problem only through this abstract modeling. Two noteworthy limitations of our work are:

Limitations of our model: Not all systems fit the description of our system model. Some systems may inherently not be separable into components, due to one of following constraints: (a) the system structure inherently is not modular and does not permit division into components; (b) a failure may manifest and corrupt the entire system as opposed to specific components, such that splitting the system into components does not provide any benefit; (c) the system may be opaque and end-to-end failures convey little information about components or system behavior *i.e.,* the quark has no visible association with components.

Appropriateness of our solutions: Though a system may fit the model, statistical approaches to root cause localization may not always be appropriate. For example, [21] discusses the problem of diagnosing configuration problems on a single computer using a database of known symptoms and solutions. Though this problem statement maps to our abstract model, in a single computer environment, we will not be able to capture enough variety of quarks to apply statistical techniques.

Overall, we hope that our efforts encourage future collaboration in this area across both the various large-scale systems communities, and the theoretical and statistical research communities.

[5] regions of DNA which have be deleted, amplifi ed or modifi ed

References

[1] AppAssure, Alignment Software, 2002. `http://www.alignmentsoftware.com/`.

[2] Tealeaf Technology, Integritea, 2002. `http://www.tealeaf.com/`.

[3] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[4] M. Caesar, L. Subramanian, and R. H. Katz. Root cause analysis of Internet routing dynamics. Technical report, U.C. Berkeley UCB/CSD-04-1302, 2003.

[5] D.-F. Chang, R. Govindan, and J. Heidemann. The temporal and toplogical characteristics of BGP path changes. In *ICNP*, 2003.

[6] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. A statistical learning approach to failure diagnosis. In *International Conference on Autonomic Computing*, May 2004.

[7] M. Y. Chen, A. Accardi, E. Kı cı man, D. Patterson, A. Fox, and E. Brewer. Path-Based Failure and Evolution Management. In *USENIX/ACM NSDI*, 2004.

[8] R. Chillarege. Self-testing software probe system for failure detection and diagnosis. In *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 10. IBM Press, 1994.

[9] A. Feldmann, O.Maennel, Z. Mao, A. Berger, and B.Maggs. Locating internet routing instabilities. *ACM SIGCOMM*, 2004.

[10] J. Gray. Why do computers stop and what can be done about it? In *In Proceedings of 5th Symposium on Reliability in Distributed Software and Database Systems*, Los Angeles, CA, 1986.

[11] B. Gruschke. Integrated event management: Event correlation using dependency graphs.

[12] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 1974.

[13] E. Kı cı man and A. Fox. Detecting Application-Level Failures in Component-based Internet Services. 2004. In preparation.

[14] M. Lad, D. Massey, and L. Zhang. Link-rank: A graphical tool for capturing BGP routing dynamics. *IEEE/IFIP NOMS*, 2004.

[15] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug Isolation via Remote Program Sampling. In *ACM PLDI*, 2003.

[16] J. Markoff and G. P. Zachary. In Searching the Web, Google Finds Riches. New York Times, August 13, 2003.

[17] NANOG: The North American Network Operators Group. http://www.nanog.org/.

[18] M. A. Newton. Discovering combinations of genomic alterations associated with cancer. *Journal of the American Statistical Association*, 2002.

[19] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do internet services fail, and what can be done about it? In *USITS*, 2003.

[20] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[21] R. Redstone, M. M. Swift, and B. N. Bershad. Using Computers to Diagnose Computer Problems. In *9th Workshop on Hot Topics in Operating Systems*, Kauai, HI, 2002.

[22] RIPE's Routing Information Service Raw Data Page. http://data.ris.ripe.net/.

[23] M. Steinder and A. Sethi. Increasing robustness of fault localization through analysis of lost, spurious and positive symptoms. In *Proceedings of IEEE INFOCOM*, 2002.

[24] University of Oregon RouteViews project. http://www.routeviews.org/.