UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

ROOT CAUSE ANALYSIS IN LARGE AND COMPLEX NETWORKS

Tiago Filipe Rodrigues de Carvalho

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2008

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

ROOT CAUSE ANALYSIS
IN LARGE AND COMPLEX NETWORKS

Tiago Filipe Rodrigues de Carvalho

Orientador
Hyong S. Kim

Co-Orientador
Nuno Fuentecilla Maia Ferreira Neves

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2008

Resumo

Uma grande parte do sucesso de uma empresa depende do desempenho da função de Tecnologias de Informação. Em redes de grandes dimensões, devido à evolução do número de clientes e às constantes mudanças nas necessidades das empresas, as dependências entre sistemas e elementos de rede têm vindo a tornar-se cada vez mais complexas. Consequentemente, a localização das causas originais de problemas de desempenho de sistemas é uma tarefa complexa. A rede tem de ser analizada como um todo porque, mesmo durante a ocorrência de uma falha, todos os sistemas podem parecer estar correctos quando analizados separada e instantâneamente. O objectivo deste projecto é o estudo de uma solução automática de análise de causas originais de falhas em redes complexas e de grandes dimensões. Neste trabalho, é apresentado o Etymon, uma ferramenta que identifica os componentes e métricas mais relevantes para explicar os problemas que afectam o trabalho diário dos utilizadores finais.

O presente trabalho propõe uma arquitectura modular para executar as acções necessárias para encontrar uma explicação para um problema de desempenho. A análise começa por processar registos de falhas (*trouble-tickets*) de forma a identificar os principais períodos de desempenho degradado. O tráfego de rede é analizado continuamente para identificar as dependências entre componentes e mantê-las actualizadas. Usando a informação sobre dependências, é criado um modelo da rede que representa o ambiente para uma aplicação específica. De seguida, é avaliado o estado de cada componente do modelo durante o período do problema com base em desvios do seu comportamento habitual. Finalmente, é feita a pesquisa no modelo por caminhos causais em que o primeiro componente corresponde à causa original do problema.

Para testar a aplicação desenvolvida foi utilizada a rede empresarial de um operador de telecomunicações Europeu. Assim, foram enfrentados todos os desafios inerentes a uma rede de produção, como por exemplo, possível insuficiência de informação sobre algumas aplicações, interações complexas entre aplicações, e um grande número de fluxos de dados. A aplicação Etymon introduz conceitos como caminhos causais, modelo de rede específico para um ambiente de uma aplicação, informação sobre dependências condicionada a um contexto específico, correlação temporal de anomalias, e classificações de causas.

Palavras-chave: análise de causa-raiz, problemas de desempenho, redes complexas e de grandes dimensões, QoS, Etymon.

Abstract

A huge share of a company's success relies on the performance of its IT infrastructure. In large networks, due to the evolution of the number of clients and changes in the company requirements, the dependencies among systems and network elements tend to become increasingly complex. Consequently, the localization of root-causes of performance problems is a very challenging task. The network must be analyzed as a whole because, despite the failure, all systems may seem to work fine when analyzed separately. The purpose of this project is to study an automatic root-cause analysis of failures in large and complex networks. We present Etymon, a tool that identifies the most relevant network components and metrics to explain performance problems affecting the daily work of end-users.

We propose a modular architecture to perform the tasks necessary to find explanation root-cause of a problem. The analysis starts by processing trouble tickets in order to identify the major performance issues. Traffic monitoring and analysis are continuously performed on the network to identify the dependencies among components. Using the dependency information, we create a network model that represents the environment for a specific application. We then evaluate the state of each component of the model during the time when the trouble ticket is issued, based on deviations from observed normal behavior. Finally, we search the model for causal paths that start on a root-cause component and provide an explanation for the failure.

The testbed for our application is the enterprise IT network of a large European Telecom operator. Therefore, we face challenges of applying such tools to a production network. For example, the challenges are possible lack of information about applications, complex interactions, and high number of workflows. Etymon introduces concepts such as environment-specific network model, context-conditioned dependency information, temporal correlation of the anomalies and rankings of root-cause components and metrics.

Keywords: root cause analysis, performance problems, large and complex networks, Etymon.

Acknowledgments

This thesis represents a lot of hard work. But my effort would be useless if not for the priceless support from some people who helped me overcome all obstacles.

I would like to thank Professor Hyong Kim for his guidance and suggestions. His valuable experience was vital to keep this project on the right track.

To Luís Costa, Ricardo Marques, Ricardo Oliveira for all the companionship and interesting discussions throughout the course.

To Sihyung Lee, for making my stay in Pittsburgh so pleasant. Thanks to you, I was able to take full advantage of my experience there. To Andrew Turner, many of the ideas expressed in this work stem from our interesting brainstorms.

To José Alegria for his support, incentive and trust in my capabilities. To Adriana Luz, Ricardo Ramalho, Tiago Mendo, Nuno Almeida, Paulo Serrão, Fernando Carvalho, Pedro Simões and Rui Martins who spared no effort to provide all the information I needed.

To my family and friends, who suffered the most during this project. I promise to make up for my absence in the last 16 months.

A special thanks to my wife Cláudia for her support, comprehension and patience. My success will always be yours too.

To my Grandmother, Albertina.

Table of Contents

List of Figures

List of Tables

A distributed system is one in which the failure of a machine

I’ve never heard of can prevent me from doing my work.

Leslie Lamport

1 Introduction

Nowadays, the pressure for companies towards the short-term profit is overwhelming. A company is expected to operate continuously in any environment and sales are the top priority for the daily management. In large companies, new products are constantly being created and new promotions and campaigns are launched in a weekly or monthly basis. These demands tend to increase faster than the support IT infrastructure.

For the Telecom companies, for instance, the number of applications in daily operation is very large. From the moment a client requests a new phone or DSL line, until the moment it is installed in his house, a high number of interactions must occur within the IT infrastructure of the company. It starts with the application that registers the request for a new DSL line. Then, a new record must be placed in the application that manages the teams that will physically install the line, and configure the circuits. This application will interact with other system that identifies the place where the client lives, and retrieves the information about the relevant paths of the network. After the installation, another application is responsible for triggering physical tests to the phone line to check for any possible problems. In addition to these applications, there are other applications that manage calls in the call centers, handle billing, or simply process the monthly invoices. Failures may happen not only in the application but also in the network. The network covers almost the entire country and comprises a large number of routers, switches, firewalls and links with different capacities. There are many possible points of failure than can affect the company's daily work and may have a significant impact on the company's image.

Another source of problems is the frequent release of new products and promotions. These changes usually require adjustments to the applications. Sometimes, due to lack of time or negligence, these new developments are not adequately tested and the impact on the current infrastructure can be disastrous. Some examples are common: a new SQL query, if not adequately tested can take a long time to run and consequently block many other users and applications; a new request can demand transferring a large file over the network to a remote store with a low bandwidth connection. Sometimes, the quick fixes applied to solve these problems are themselves a source of performance issues.

Finally, another important aspect is the natural evolution of the company's services and of the IT infrastructure. Throughout the years the number of employees and clients has grown and therefore the applications add to cope with more demanding usage profiles. The usual solution is to keep adding more resources (i.e. more and better servers, increase memory, upgrade links etc.) and deploy improved and more complex applications. As the network and systems became more complex, more intricate solutions are necessary to allow new applications to communicate with older and legacy applications. The middleware solutions adapted to make this interaction possible constitute new points of failure. The personnel that use and manage the system as changed also. Some undocumented changes were lost, and valuable know-how about the network and the applications was lost.

All these changes and evolutions can make managing network and systems a chaotic task. It is vital to the company to have adequate control over the network and its systems. For instance, there must be detailed records of all elements on the network as hosts, servers, links, routers and switches. Real-time monitoring is another fundamental function to be used to generate alarms, trend analysis or technical risk assessment. These tasks should be performed automatically and several monitoring solutions are available (either commercial or open source). But, in large and complex networks, these applications generate a large amount of data that is very hard to filter and analyze manually.

The main purpose of this project is to study a root-cause analysis of failures in large and complex networks. The final solution must be as automatic as possible. For the reasons stated earlier, the task of identifying the root cause for a performance problem experienced by the end-user a large enterprise IT network is very difficult. Sometimes the problem has its origin in a dependency that the network and systems operators did not even had knowledge of and also, despite the failure, all systems may seem to work fine when checked individually.

The first step of the analysis is acquiring as much information as possible from the network. Due to the constant changes on the network and systems, one cannot trust completely the existing network systems architectures diagrams, so one of the first components of a root cause analysis tool must be a network discovery mechanism. This component should also identify dependencies among systems. A second important component is fault diagnosis, i.e. one should identify faulty behavior by one node of the network or by some component of the system. Having identified faults in several nodes, they have to be correlated in order to understand which fault is responsible for the failure perceived by the user.

One of the main goals of this project is to create a useful tool to apply in the company whose network is used as testbed. Thus, focus has been given in creating small applicable modules that could not only be used as a component of root-cause analysis system, but also be applied independently and provide interesting information to system managers. Another purpose is to identify what are the requirements to improve the accuracy of this root-cause analysis tool. For this reason, a real and very complex testbed is used rather than a scenario created from scratch. Using a real and complex network for such a short term project introduced some obstacles: lack of information about the systems, high number of workflows for which there is no documentation available, difficulty to implement some important metrics in production systems, huge amounts of data to process, and naturally, the existence of numerous complex interactions between the same systems which may add some noise in the service dependency discovery process.

1.1. Related Work

One of the most complete projects in the field of root-cause analysis found in the literature today is probably Sherlock [**1**]. Its main advantage is that the network is considered in its entirety instead of analyzing only a specific application. The authors describe techniques for building a dependency model that allows the identification of root-cause candidates. The model also simulates the effect of fail-over and load balancing mechanisms. The inference graph model consists of nodes and edges. The node represents an associated probability of having performance problems and the edge represents a dependency in the system. Detection of the root-cause is limited to a server or service and does not address finer resolution. The authors argue that the model can be applied to any granularity but the methods used to identify the dependencies could be limiting as they are based on simple traffic analysis. The analysis uses the response time to detect and understand problems.

The first component of this project infers causality relations among TCP flows that establish between the servers that compose the entire ecosystem. The correlation of these flows follows a similar approach to two convolution algorithms in [**2**]. The first one involves recognizing RPC-like communication and combining the calls and responses into path sequences. The second one organizes messages exchanged between each pair of nodes in a time series, and then correlates them by calculating the convolution of each pair of functions. Applying this procedure recursively, the authors obtain possible paths used in the application. The latter approach uses any kind of messages, thus it is more generic but less accurate than the one based on remote procedure calls. The method used for RPC communications can be extended to other protocols.

In Pinpoint [**3**], a unique ID is assigned to every HTTP request that enters the system. This tool is intrusive, as it must intercept all requests in order to assign them the unique IDs. This is easy to do on a basic J2EE platform, but almost impossible to implement pervasively in a large network with diverse software and hardware systems. Pinpoint is evaluated using simple J2EE web applications without intricate relations among a large number of servers. Besides Pinpoint, other approaches [**4**] [**5**] require the programmers to instrument applications to reveal relevant events. Instrumentation of large and sometimes legacy applications in a compatible fashion could be a very expensive task.

Another approach with a higher level of control is Magpie [**6**]. Magpie makes an extensive use of event monitoring in all the components in the system. By capturing all the relevant events and guaranteeing their causal order, Magpie is able to identify the workload of the application. In order to organize these events, an event schema must be generated and is as complex as the number of different events monitored. Although it is difficult to apply this approach on the entire network supporting several interacting applications and to every possible event, it could be effective if it is applied in specific points of the IT infrastructure.

1.2. Main Challenges

The following list presents challenges of this project:

- **Complexity of networks and dependencies:** To find the root cause of a problem in a large distributed network, using only a manual analysis is often very difficult or even impossible. Using an automatic tool, although we are able to do more processing in less time, we lose some of the benefits of a manual analysis done by experts: we lose intuition! Due to the lack of this human characteristic it is very hard to choose, among a large set of elements, those that should be analyzed most deeply. It is also hard to configure all the possible dependencies in such a system. The overabundance of elements in such a network and the complexity of the dependencies among them is one of the main challenges to address in this project;
- **Number of metrics:** The number of metrics available is always an issue, no matter if we have too many or too few available. In the most frequent case, we lack some metrics which are important to explain a problem. In this case, the application should try to identify some

component or group of components where the root cause may be located, i.e. the application should reduce its granularity. In contrast, when the number of metrics available is too large, we have complexity and performance issues. Too many metrics is equivalent to too much processing and it is harder to distinguish the most important metrics.

- **Automation of procedures:** An application that overcomes the problem of manual analysis should be, by definition, as automatic as possible. One must minimize the amount of expert opinion needed to construct models, configure them, etc. This field of research is addressed by Expert Systems [**7**], where all components of the system are correlated automatically using their attributes. The construction of an automatic system raises many issues:
    - How to find which metrics are behaving badly? One approach is to focus on modeling failure behavior. This is an approach to avoid because different failures tend to have different manifestations. Therefore, such method is prone to false positives. Another approach is to identify deviations from normal behavior. In any case, how should we process these anomalies to obtain a node state?
    - How should metrics influence the state of a component? This is one of the most difficult questions to answer without introducing some expert opinion. Different failures may be identified by different metrics. Therefore, the state of the component should use as much past history as possible to understand how to use metrics to evaluate the state of a specific component.

1.3. Contributions

The main goal of this project is to create a functional root cause analysis tool that could be helpful for problem diagnosis in a large enterprise networks. The system is called Etymon[1]. As it is shown in the related work, most of the current solutions make extensive use of instrumentation of applications. The company whose testbed is chosen has an ongoing project that monitors part of IT infrastructure. The project is named Pulso[2] [**8**] and has been extending its scope throughout the past few years. Pulso plays a major role in this project, as it constitutes the main data source used to feed Etymon.

This section describes three components of an automated root cause analysis system: network discovery, failure diagnosis, and network model.

The major contribution is the study of the deployment of such a system in a large and complex production network, which has been adapted to the company's needs over several years. This implies major changes to standard applications, many times without sufficient documentation, monitoring, logging capabilities and with "unexplainable" collateral effects on the performance of the application.

In such a large system, it is extremely hard to characterize its use and what constitutes a performance error. The company has a trouble ticket system used to identify performance issues.

[1] Etymon is a Greek word for "true meaning of a word".

[2] The Portuguese word "pulso" means both "pulse" (for representing the act of monitoring and sensing the network) and "wrist" (for representing the need of having control over the way network and systems behave).

This application stores end-users complains, which represent the best sign that an abnormal behavior is affecting the application's performance. This method can later be replaced by automatic methods of failure diagnosis, possibly based on the knowledge acquired from the analysis of the issues raised by users.

We provide our system with five main properties, so that it can fit well in such large scale network. The application should be:

- **Usable** – the application should be immediately useful to network operators, despite its accuracy in identifying root-causes of failures. Therefore, a set of views are carefully designed and implemented to analyze data. Using this view, network and system operators can easily identify the most relevant elements to analyze from a large dataset. Therefore, they must have access to features like ticket analysis, traffic analysis and correlator, network model, components and metrics relevant to study a performance problem;
- **Automatic** – the developed tool must search for causes using automatic mechanisms whenever possible, minimizing any intentional human intervention or opinion. Therefore, the methods are mainly based on the recent relations among application components and on the detection of deviations with respect to a baseline of the recent behavior. The use of these applications in production systems is locally stable, i.e. the usage of each application is more or less the same when seen in a time window of a few weeks, which allows the application of anomaly-based methods;
- **Adaptable** – the application fits and adjusts well to the discovery of new components and to the inclusion of new metrics on the underlying monitoring systems. The network model is dynamic and contains a different view for each problem, including only the most relevant elements. The model uses information available in the company's underlying applications that monitor the network and keep records of each network element. Whenever a new element is added to these applications, it is reflected on our application in the relevant models;
- **Granular** – the model is easily extendable to increase the resolution of root-cause identification. Whenever we have more information about a specific component, we may be able to pinpoint a more detailed root-cause. Consequently, all mechanisms used must be generic and should be applied to any type of component added to the model. This property will facilitate the implementation of extensions to the application by adding new model components that will be processed as any other component, but that will add a new level of detail to the application;
- **Accurate** – the application improves the accuracy of a normal intuitive analysis. Although it is hard for an application to surpass the years of user experience, it should be able to pinpoint components as being problematic in a more accurate way than users would do in a manual analysis. The advantage of automatic applications is that they can scan the complete set of dependencies of an application, without being biased by any frequent anomalies of a system.
- **Scalability –** the application must be able to process data for all nodes represented in the network model, in an efficient. Analyzing past issues involves access to large databases with historical data, which slows down the entire process. Therefore, the statistics needed to determine the network state are updated in real time, whenever a new measurement is

available. Then, if a relevant event is detected, the network state can be saved for later analysis.

This project is strongly related to the company whose network is used as testbed. Therefore, the feedback to be given to the company constitutes an extra requirement. When we tried to correlate components to create the network model, we noticed some inconsistencies that could thwart any automatic analysis. The network and the information available constitute the main advantage and, at the same time, a challenging task. There are numerous metrics available covering many of the areas of the analysis but, even so, root-cause analysis demands some more metrics to be able to find a coherent causal path, i.e. an explanation of how components have affected each other until they provoked the complain of a user. Therefore, the feedback about what needs to be reorganized, what extra metrics should be obtained and what parts of a system should be documented, is very important.

2 Background

Root cause analysis projects require a huge amount of information. To be able to identify problems in the network, systems or interactions among them, we need information about the network architecture, mechanisms to trigger the collection of metrics and traffic samples and to constantly monitor the state of all elements that may be involved in this kind of analysis. When one starts an analysis, it is hard to choose a subset of elements where we are sure to find the original cause, thus every single element may have its own importance.

In order to choose the testbed, several criteria have to be fulfilled. The main requirements that influence the choice of the company and testbed to use during this project are:

- **Existence of widely deployed monitoring tools which are easy to interact with:** the enterprise IT network chosen has developed and implemented during the past four years a tool that monitors and stores information about systems, network and application. The diversity of metrics is very large and is growing continuously. This not only allows some rich insights for the current analysis, but also opens hopeful perspectives for evolution in the near future for this tool;
- **Existence of information about the network and systems and how they are related:** another tool available in the enterprise IT network monitors relations among systems and maintains records about the hosts identified. This tool is frequently updated with information about the systems, including their characteristics and functions. This allows the recognition of the IP addresses in our traffic analysis, making it possible to include its statistics in the network model used for root cause analysis;
- **Existence of complex and diverse relations among systems:** many of the papers about root cause analysis present solutions to approach the problem. But one of the main drawbacks found in most of them, is that they usually test their results using simplistic application architectures. Normally, these results are obtained for systems involving nothing fancier than a web server with a database and a DNS for name resolution. Actually, although they are important scenarios due to its pervasiveness on most companies, they are not challenging because the relations between systems are very simple and similar throughout the network. Also, the chosen testbed should be challenging in terms of the network architecture. The testbed used in this project includes end-users spread throughout the country and more than one data center where the main systems are localized.

2.1. The Network

The enterprise IT network possesses several important applications. The employees, and potential end users of the applications, are spread all over the country and are connected by a private internal network. The density of end users is not homogeneous as most users are naturally located near the larger cities. Therefore the network capacity is much higher in the larger cities than in some remote locations. All this asymmetry must be accounted for in an analysis of the complaints made by users.

The testbed application is one of the most critical and used within the enterprise IT network. Therefore, the set of users of this application is a proportional sample of the overall set of employees. Another aspect of this application is that it deals with several different types of information and therefore it must interact with a high number of different systems. This constitutes an ideal environment to test the solutions developed in this project.

The application has been developed inside the company and its functionalities have been augmented throughout its development. During its evolution, the application had to be changed in order to add new features, to add new interactions with other applications often resorting to middleware systems, to cope with new system requirements due to the increasing number of users and clients and so on. This fact increased the complexity of the system and the difficulty to explain performance problems detected by the users. We should add to this, the disparity of technology used in the several systems. For instance the operating system can vary from Linux to Windows while the databases can go from Oracle to MSSQL servers.

To have a sense of how complex the used system is consider Figure 1.

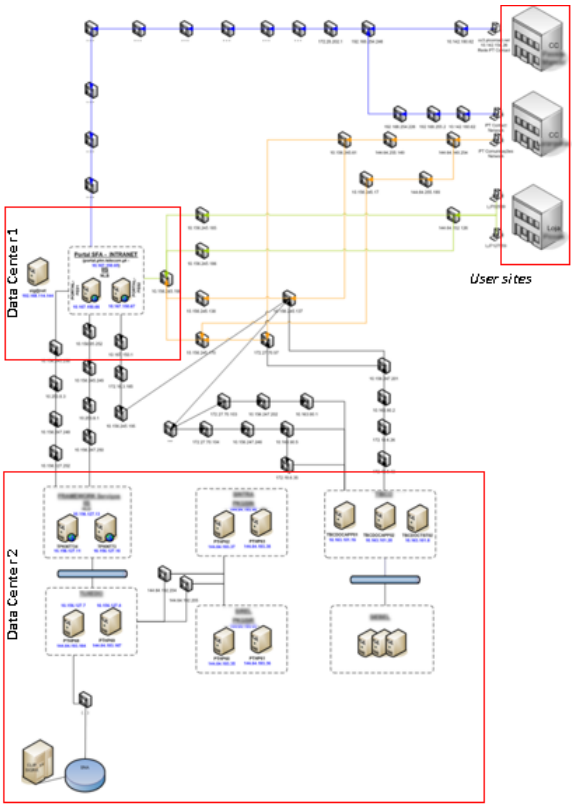


Figure 1 – Sample diagram of an application with multiple dependencies

As we can see the end users (user sites) communicate with a single application. Many times they are unaware of the other applications, with which the central application must communicate to provide some service. Some of the servers are in different data centers, thus introducing other possible points of failure. The smaller boxes represent routers. Thus, we can verify that between a user and the application or between data centers, the paths are long and traverse many routers. This gives an idea of how complex the system is, and of the quantity of elements that may fail when a user makes a request.

We may easily conclude that the tool to be developed must consider each application as being part of a large ecosystem. This ecosystem includes not only the target application, but also every single system that communicates with it (even in second or third order, i.e. applications that communicate with the target application through one or more other systems). It should also include the network that supports the communication between all systems, and between each system and its end-users.

2.2. The Monitoring Application

The company which supplied the testbed application for this project has implemented in past few years a project for event processing and monitoring (Pulso), covering a large fraction of the IT infrastructure. In this section we provide a description of the system's architecture and of the metrics available. This information is important because this monitoring project is the major source of information used in the root cause analysis tool.

As we referred before, new metrics are difficult to implement in production systems and a lot of time is necessary to obtain the authorizations, thus the main option is to use only the metrics available in the company's monitoring tool at the start time of this research project. The idea behind this is that a huge and complex model, with many middle components without any information, tends to be less useful than a simpler model where all elements contain some information about its state. Naturally, this approach should only be followed if the model is able to adapt to new knowledge being acquired and inserted into the application.

Figure 2 represents the architecture of the system. As we can see the philosophy of the Pulso system follows the ideas developed under the subject of Complex Event Processing [**9**]. In this type of systems, all measurements are treated like events that should be stored using a canonical format. Each event can trigger some processing that result on different events with a higher degree of abstraction. In this system, the events are metrics applied over servers, network links, applications, etc.

The low level events are used in this system to evaluate the state of the components. The model is created using the information available about the network. The network records contain the existent servers, applications, and links and provide some attributes which we use to construct our model. Naturally the model is only as complete as the network records.

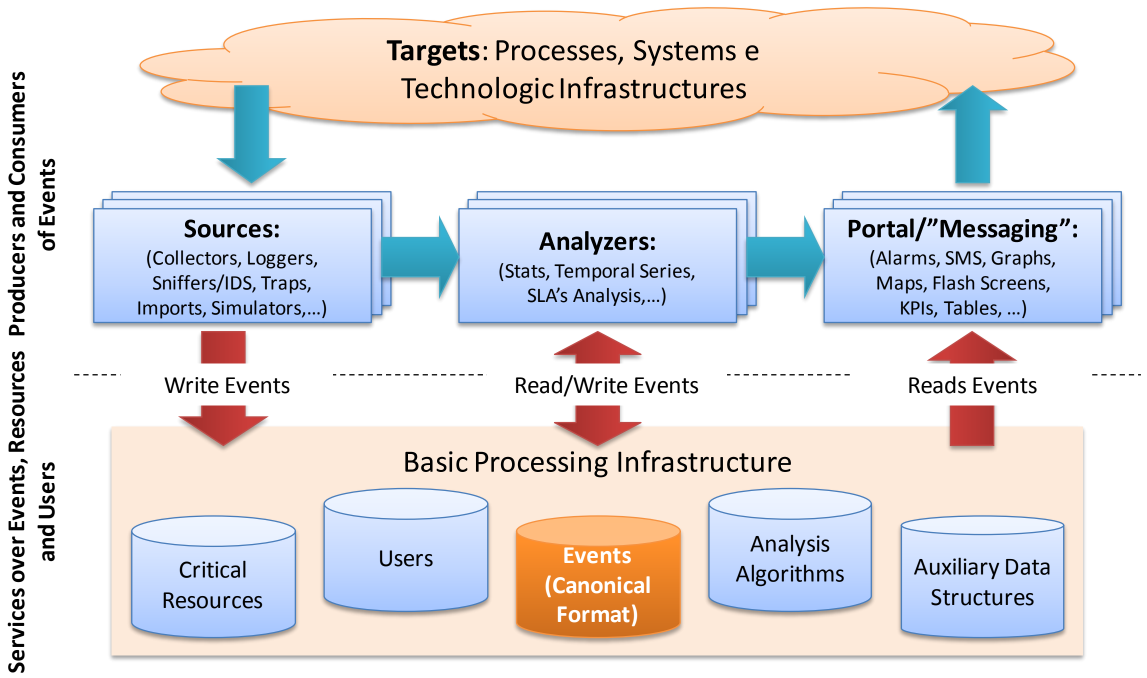

Targets: Processes, Systems e Technologic Infrastructures
Producers and Consumers of Events
Sources:
(Collectors, Loggers, Sniffers/IDS, Traps, Imports, Simulators,...)
Analyzers:
(Stats, Temporal Series, SLA's Analysis,...)
Portal/"Messaging":
(Alarms, SMS, Graphs, Maps, Flash Screens, KPIs, Tables, ...)
Write Events
Read/Write Events
Reads Events
Services over Events, Resources and Users
Basic Processing Infrastructure
Critical Resources
Users
Events (Canonical Format)
Analysis Algorithms
Auxiliary Data Structures

Figure 2 – Generic architecture of the monitoring application

The system metrics are collected from a large number of servers which use different technologies. These servers are in production thus the retrieval of metrics should not require making major changes to the system and should have a minimal impact on it. This makes more difficult the obtainment of some metrics in specific operation systems and/or applications. Examples of system metrics obtained at the machine level are CPU usage, load average, memory, etc.

Some other metrics depend on the specific purpose of the system. For instance, in a database we can collect several metrics which can also vary with the database type (Oracle, MSSQL, MySQL, etc.). Other types of servers, such as mail or web servers, may also have specific metrics. Examples of application-specific metrics are server-side transactions response time, number of application/server errors, database wait time, etc.

Within the Pulso system, the network is seen as a black box. The measurements are made to end-to-end links between critical points of the network. These points correspond normally to two types of locations: data centers where the main servers are located and the end user sites. The links are defined for a pair <data center, user site>. For each of the links the system provides measurements of bandwidth and latency.

Other type of links is considered when the analysis is made in the transport and application layers. Here, the application defines application links, which connect some user site to some application. The metrics collected refer to aspects of the communication to or from a specific application. Some of the metrics available are number of timeouts, resets, end-to-end duration of a transaction, etc.

2.3. The Network Record Application

There is also an application that keeps track of the network hosts. This application monitors network traffic and stores information about the flows seen on the network. It can be fed with more information manually, by introducing a list of the network hosts and of their purpose, or automatically, by using tools like NMAP to retrieve information about a host. The automatic analysis is rather limited, as the servers often do not allow this type of scans or provide insufficient information.

The information contained in this application is useful to correlate the data gathered in our traffic captures with the components defined in the monitoring application. For instance, whenever we want to correlate two servers, we can identify their IPs using this application, and then lookup the frequency of their relation in a specific traffic capture.

2.4. Limitations

There are several limitations to this work due to the time available for the project and to some operational constraints. The time available, together with the fact that we are working with a production environment, does not allow the retrieval of some metrics considered relevant and that are not being collected by the monitoring system. The probe to retrieve some metrics in a production server must be extensively tested and specific authorizations are required. Therefore, the approach for this project, given its scope, is to develop an end-to-end application using whatever information is available.

Another limitation is that we only have probes to capture traffic from one server to another, i.e. we have mirrors for the traffic generated and received by the most critical servers. Naturally, this prevents us from obtaining information from the control plane. The traffic captures are used and analyzed to detect the network and transport flows between servers. Although it would be interesting to verify and analyze some of the lower level protocols (for instance, routing protocols), this is considered to be out of the scope of this project due to this limitation.

3 Etymon Overview

Etymon deals with finding a root cause of problems identified by internal end-users[3] of a large enterprise IT network. The application starts by filtering and grouping failure information reported by the end-users. These tickets are processed to identify periods of unavailability or performance degradation. For each period of degraded performance, Etymon constructs a model of the IT infrastructure using the information available on Pulso and obtained in traffic analysis. Afterwards, a time series analysis is computed for all metrics identified for each node of the model. This analysis identifies patterns of behavior of the recent past for each metric. Subsequently, the state of each node can be computed by analyzing deviations from the pattern or identifying abnormal events.

The application runs either in online or offline mode. In online mode the failure reports are received continuously and patterns are updated on the fly using the recent past values. This mode allows quick identification of root cause candidates as processing is being performed continuously. The offline mode is the most used to test the application. Using this mode we can calculate the network model and patterns on demand and identify a root cause in the past. Naturally, these computations are time and CPU consuming and therefore the application takes some time to identify the root cause candidates. Naturally, to speed up processing one can use pre-calculated patterns at the expense of a reduced accuracy.

This section describes the architecture of the application, where both modes are presented and the main components are described. Then we focus on each specific component. First, in Section 3.2 we describe the event correlator module and the process of filtering failure reports and identifying performance issues. Section 3.3 describes the traffic analysis components and the main outputs possible from this component. Section 3.4 describes the network model and how it is computed using information from Pulso and information obtained through the traffic analysis. Sections 3.5 covers the time series analysis, which allowed us to compute the probability of a node being affected by a failure or being a cause of the failure. These computations are presented in Section 3.6. Finally, in Section 3.7 we describe the component that finger-points the components and metrics that may identify the root causes of the problem.

3.1. Architecture

The Etymon application has several components that can be applied individually or in sequence for ongoing events. Figure 3 depicts the several components and data flows for online and offline event processing.

[3] Internal users are company employees that need to use its applications and IT infrastructure to execute their daily tasks.

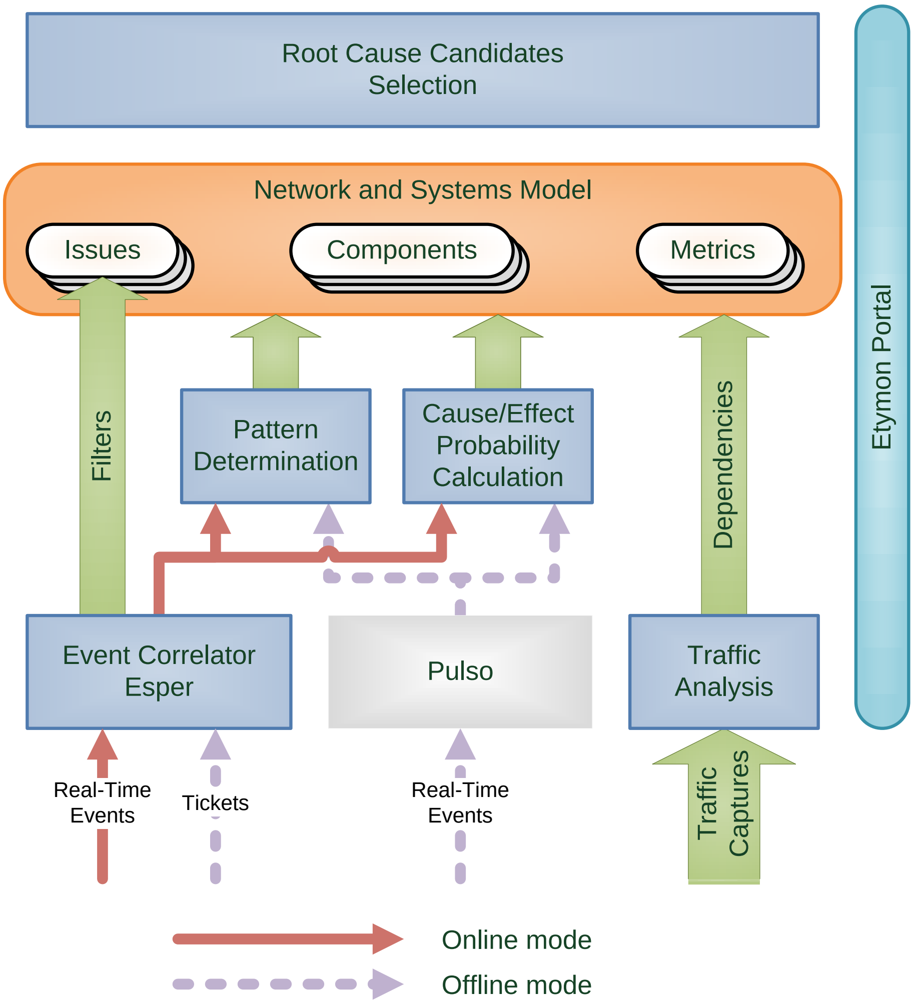


Figure 3 – Etymon architecture

The main inputs of the application are the events collected by Pulso. As seen previously, these events are collected in real time and stored on a central database. The tool has two possible modes of operation:

- **Online mode:** this mode is used to identify and analyze performance issues as they occur. The online mode is represented in Figure 3 by solid arrows.
- **Offline mode:** this mode is used to identify and analyze performance issues that have occurred in the past. The offline mode is represented in Figure 3 by dashed arrows.

The core components of the tool, which are represented in the darker boxes, are the following:

- **Event Correlator (uses Esper Java Framework)** [**10**]**:** complex event processor framework that receives events and triggers the corresponding methods for pattern calculation or state update. The Esper engine enables the implementation of Complex Event Processing (CEP) applications [**9**]. A CEP engine, like Esper, is a platform to allow easy development of applications that process and analyze real time data. Esper implements an in-memory database which is better suited to applications where a high number of events needs to be processed quickly and frequent queries are made to correlate real time data;

- **Traffic Analysis:** traffic stream analyzer responsible for identifying flows, calculate traffic statistics and output conclusions about network nodes' roles and dependencies among nodes and flows;
- **Pattern Determination:** component used for detection of abnormal events either by comparison with an usual pattern of behavior, a prediction based on the recent past or a a chosen threshold;
- **Causal and/or Effect Probability Calculation:** module that computes the state of each node given a specific period. In the online mode this period has a fixed size ending on the current time instant. In the offline mode this period corresponds to the duration of the issue plus an extra fixed period where the causal behavior may have happened;
- **Root Cause Candidates Selection:** module responsible for crawling the graph and search for the most problematic component or metrics in the graph. The path between these nodes and the root issue influences the overall probability of the node being pinpointed as a cause for the problem.

The technology used is an Apache/Tomcat web server for the graphical user interface and a database MySQL for data storage. The application is entirely developed using Java.

3.1.1. Online mode

When in online mode, the Etymon tool can receive events from several sources. Mainly, the sources used are the database and data collectors from the company's monitoring application and the ticketing application which receives the end users reports for problems. The initial component in this mode is the event correlator. This component allows the setup of filters and correlation operations over the streams of events. The stream of events corresponding to the tickets opened by the end users constitutes the first input of the application. This stream is analyzed to identify periods of degraded performance in each application. This information will then trigger the creation of a network model. A model is built based primarily on the information retrieved from the tickets about the systems and network locations affected. The ticket filtering and model creation operations are described in Section 3.2.1 and Section 3.4 respectively.

The online mode bases the state calculation on a fixed-size past period. All nodes for which information is received must be updated continuously. The patterns must be available whenever a user desires to analyze the model state or whenever an issue is detected and, consequently, the root cause candidates need to be determined.

The performance of the online operation is better than that of the offline mode because the patterns are immediately updated whenever an event enters the application. Therefore, this mode is used to pinpoint causes to issues that occur at the time of the analysis. The event correlator module is responsible for triggering these updates and for sending the events for the pattern calculation component. Also, the state of the model nodes can be updated as soon as the pattern update is performed. The methods used to calculate the patterns or other abnormal events detection are described in Section 3.5.

3.1.2. Offline mode

In the offline mode the user can request an analysis of a past issue. The application can, for instance, analyze an input file containing the tickets opened for a specified period. These files are in the CSV format and are obtained through queries to the Ticketing application of the company. The ticket filtering operation uses, as in the case of the online mode the event correlator module, to process the events (see Section 3.2.1).

When the user chooses one of the issues the application creates a model based on it. The process follows the same steps of the online mode but, in this case, all time series analysis, pattern detection and state calculation must be computed. For the state calculation, the analysis period includes the time interval of the issue plus some previous period. The size of this extra period is determined by the duration of the issue.

As a consequence, an offline analysis is considerably slower due to data retrieval constraints. The Pulso database contains large volumes of data and the patterns must be computed for a large period, which increases the time needed for data retrieval queries

The offline mode is useful to evaluate issues that are inserted in the application some time after their occurrence. For the past issues, that have been detected online by the application some optimizations can be made. For instance, the online mode can save information about the state immediately after an issue as occurred. Whenever the user desires to analyze the issue again the model can be quickly loaded from the database.

3.2. Event Correlator

The event correlator model is based on the Esper framework. The input data is aggregated in streams of events. This framework executes a call-back function whenever a new event of a specific stream arrives. The programmer of a module using the Esper framework must define the call-back function, the stream aggregation and mapping between functions and event streams. Events on Esper are tuples. The streams are defined by specifying a query that will be applied to each of the tuple. These queries are defined using the EsperSQL language that, as the name implies, is very similar to SQL. The inputs for this component can be either real time events of input files containing a list of events to be processed in batch.

3.2.1. Issue Identification through Problem Ticket Filtering

The main goal of the Etymon application is to find explanations for performance issues. A performance issue can be defined as problem that occurs in some application or network element affecting the normal behavior of the company’s information systems. Therefore, these issues affect the end-users productivity. The users are, in general, company employees that depend on the systems performance to be able to do their job efficiently. Whenever a user sense a problem on a system that prevents him from doing some operation, he may report the problem by opening a ticket using the e-mail or the telephone. These tickets are registered centrally and are the perfect input for an application like Etymon.

The input of this module can be live events sent by the ticketing application or a CSV file containing a list of tickets opened during a specific period that the user wants to analyze. We apply a text-filtering function to all the ticket events received, aggregate them by their timestamp, and generate an issue event for each group.

The ticket opened by end-users can have several subjects: blocked accounts, software errors, unexpected results or performance problems. Etymon identifies root-causes for performance problems like slow or unresponsive applications, or unreachable servers. In the context of this project we use the name **ticket** to denominate a user report of any type of problem and an **issue** to refer a set of one or more tickets of performance.

The issue constitutes the root of the dependency graph used to find root cause candidates. The main fields of an issue needed to create the relevant graph of dependencies are the period's start and end timestamp, the application name and the network location of the users who reported the tickets. To construct the model relation we correlate these values with the systems and network information stored by the monitoring application (see Section 3.4) and with the traffic analysis results (see Section 3.3).

When a ticket is created it includes a description made by the user (see Figure 4). As this description is formed using free text, the possible descriptions are infinite. The ticketing application has a large number of categories to characterize the problem. Nevertheless, sometimes, either by lack of an adequate category or by inexperience of the call center operator, the ticket is not correctly or accurately categorized. The solution to identify relevant issues is to search for a set of specific keywords in the description provided by the user and group the tickets with near timestamps.

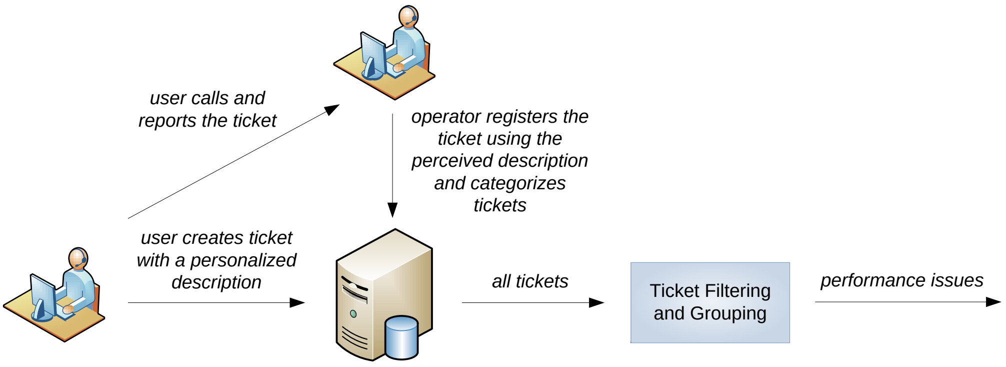

user calls and reports the ticket
operator registers the ticket using the perceived description and categorizes tickets
user creates ticket with a personalized description
all tickets
Ticket Filtering and Grouping
performance issues

Figure 4 – From ticket registration to performance issue detection

The filtering process is presented on Table 1. Here one can see some typical descriptions provided by the users[4]. The keywords used are presented in the second row and in the bottom of the table it is possible to see the final issue. This approach has proven to be very effective given the common descriptions found. Whenever a false positive is encountered, it probably is isolatedticket that leads to an issue of duration zero. Such an issue has a very low relevance, because the main target of this application are issues with long durations, i.e. at least over thirty minutes. Even if it does not

[4] The ticket description was translated and therefore the solution is slightly different due to languages' grammatical differences.

correspond to a filtering error, a single ticket may indicate that only one user is experiencing problems. In this situation, the cause of the problem is probablylocated in the user's workstation. As we are not able to monitor the users' workstations for now, this kind of causes cannot be analyzed.

Raw Tickets	App 1	*2008-10-04 14:00*	*Lisboa*	*App 1 is unavailable*
	App 1	*2008-10-04 14:03*	*Aveiro*	*Record is not shown on the interface*
	App 1	*2008-10-04 14:10*	*Lisboa*	*The App 1 is extremely slow*
	App 1	*2008-10-04 14:13*	*Porto*	*App 1 keeps blocking*
	App 1	*2008-10-04 14:28*	*Lisboa*	*App 1 has stopped*
Keywords	*Unavailable, slow, block, stop…*			
Issue	Performance Issue: **2008-10-04 14:00-14:28**, Application: **App 1**, Locations: **Lisbon, Porto**			

Table 1 – Ticket filtering and grouping

3.3. Traffic analysis and network discovery

The first goal of the traffic analysis is to identify the application flows established between the most relevant servers. Given the scope of this traffic analysis it is feasible to apply it to any servers available. Another objective is to find correlations between flows, i.e. flows that, with some high probability, start a short time after the termination of another flow. Using this information we characterize some of the relations of the dependency graph (see Section 3.4). Using this module is also possible to confirm the application's workflows, identify clients and servers automatically or discover unknown dependencies of the application.

To obtain the capture files for the servers we use a feature already implemented on the company's network discovery tool. This system is mirroring the traffic of some of the most relevant servers to one of its machines, and thus, we use this mechanism to obtain traffic to our application. The files are created in the CAP format and then processed within Etymon in order to the transport level flows. To open and process the packet headers we use a Java library: the protocol decoder jNetStream [**11**].

To identify the TCP flows, we resort to the implementation of a state machine driven by the packets captured. The definition of the state machine is available in [**12**], and is reproduced in Figure 5. The most important instants to identify for each flow are naturally those when a flow is established and when the flows are terminated. Abnormally terminated connections can be a sign that a server is having problems. We will use the information about resets and timeouts as metrics in the modules that compute the state of each node.

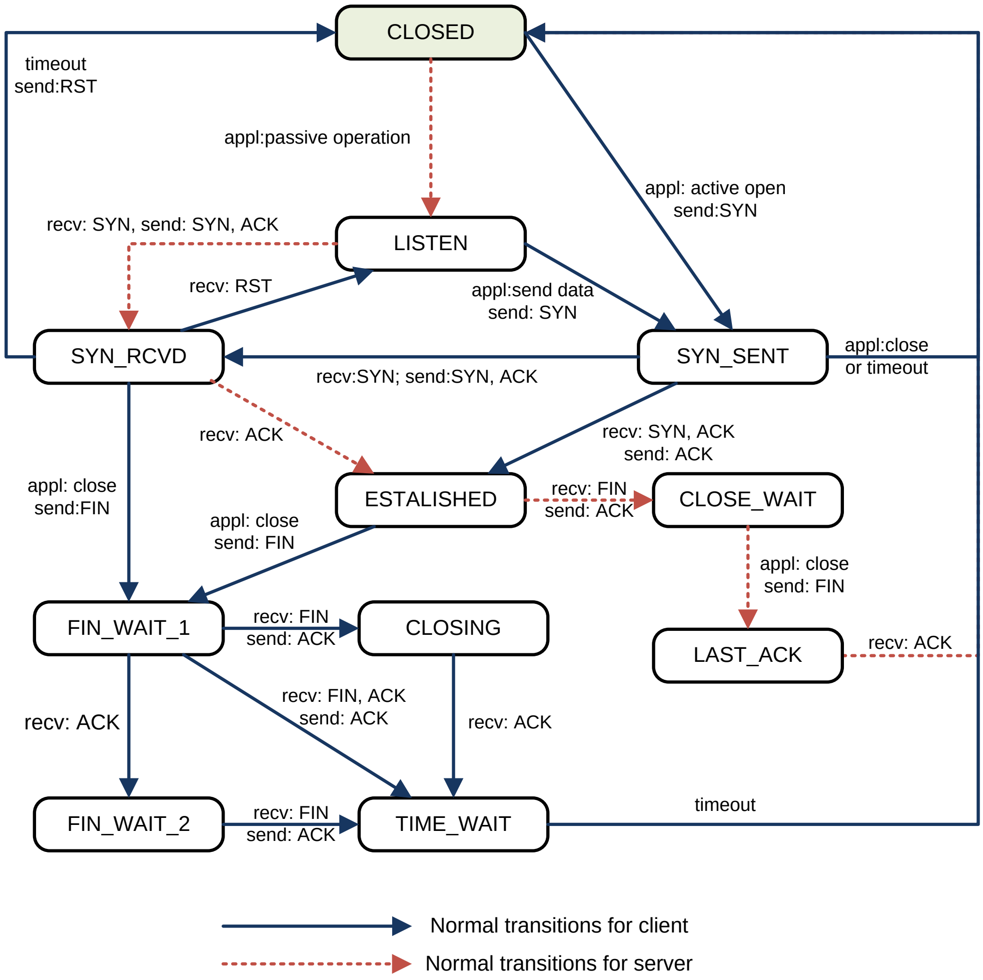


Figure 5 – TCP State Machine

3.4. Network Model

The central component of Etymon is the network model. The network model should represent, as faithfully as possible, the relationships between systems, servers, network links and application links. Naturally the process of gathering information about a system and identify each dependency in a large and complex network is a huge task. We follow an approach that optimizes the use of the already available metrics instead of building a complete and very complex model of dependencies. The construction of such model would have failed because many of the middle nodes would not have any metric associated and would compromise the detection of failures.

The model is created using an automatic approach to correlate the information available. The monitoring system which manages data collection has already information about servers, systems and network links. Hence, the correlation methods use the properties of each entity to correlate them with order type of entities. For instance, a server has some information associated with it, e.g. IP address, application, and so on. Through the company's network records, it is possible to identify a host location based on its IP address and, consequently, we can identify the network link used. Also, as the servers have a reference to the application they belong to, we are able to identify the most relevant servers in each application, using statistic of the traffic analysis.

Until now, we only have access to the quality of the end-to-end communications between key points of the network. These points are, in one side, the sites where most end-users are located and, on the other side, the data centers where the main servers are located. As we do not have control plane information, we will address low-level network issues in future work. The model can easily accommodate new information collected on the network and new dependencies between middle routers, gateways and other network elements. Nevertheless, the current setup allows us to pinpoint the network as a cause of problems although it is not possible to identify the exact cause component accurately.

3.4.1. Generic Model

The model is a directed graph (or digraph) where the nodes are the enterprise IT network components and the edges represent a possible causal relation. Each node has a state representing the probability of having some anomaly. Each edge has an associated probability the child node influencing the parent node. Naturally, the final probability of the parent node having a problem and of it being caused by an anomaly in the child node will depend both on the state of the nodes and on the probability of the edge between them.

The generic model, which is represented in Figure 6, uses five different classes of components:

- **Issue:** corresponds to the root of the causal graph, and represents an event of degraded performance identified during the ticket analysis and filtering operation;
- **Application:** represents a specific application. Although it does not correspond to a physical component per se, it is important to define an element that represents the application and to which several application level metrics are applied. For instance, metrics pertaining to higher level transactions or some parameters specific to a given application may be associated with the Application component instead of being associated with the Server which maps to the hardware level component;
- **Server:** represents a specific physical or virtual machine. Servers are the elements that compose the applications, and can correspond to DNS servers, web servers, databases or simply virtual servers. In some cases, the virtual server may also help making load balancing or active-passive replication mechanisms transparent to the model because it always represents the active replica on a cluster, i.e. the one being accessed by the users;
- **Network Link:** represents a physical connection between two end-points. The two end-points are a site where end-users are and a data center. The metrics for these network links indicate the quality of the communication (e.g. bandwidth, latency) between the two points;
- **Application Link:** represents the connection at the transport and application layer. The state of the application link corresponds to the probability of the communication between two applications having problems. For instance, it will consider metrics like the speed of the transactions performed on a specific link, timeouts of those transactions, resets sent by client or server applications, anomalies on the traffic pattern observed, etc. Thus, the Application Link component represents a relation between two applications. The possible relations are:
  - **Application to Application:** when the relation does not specify any servers but is only the intervenient applications;

- **Application to Server:** when the destination application has a specific server that relates to another application;
- **Server to Application:** when the source application has a specific server that relates to another application;
- **Server to Server:** when the relation between two applications is performed by two the servers explicitly defined;
- **Location to Application:** when an application has many accesses from a specific site and thus metrics are obtained for several connections from different host but only to one application, etc. Naturally this association typically represents accesses from end-users to the application.

Each component will have a state expressed in the form of a probability. This probability tries to represent how likely it is for the node to have problems. The calculation of this state is described in Section 3.6.

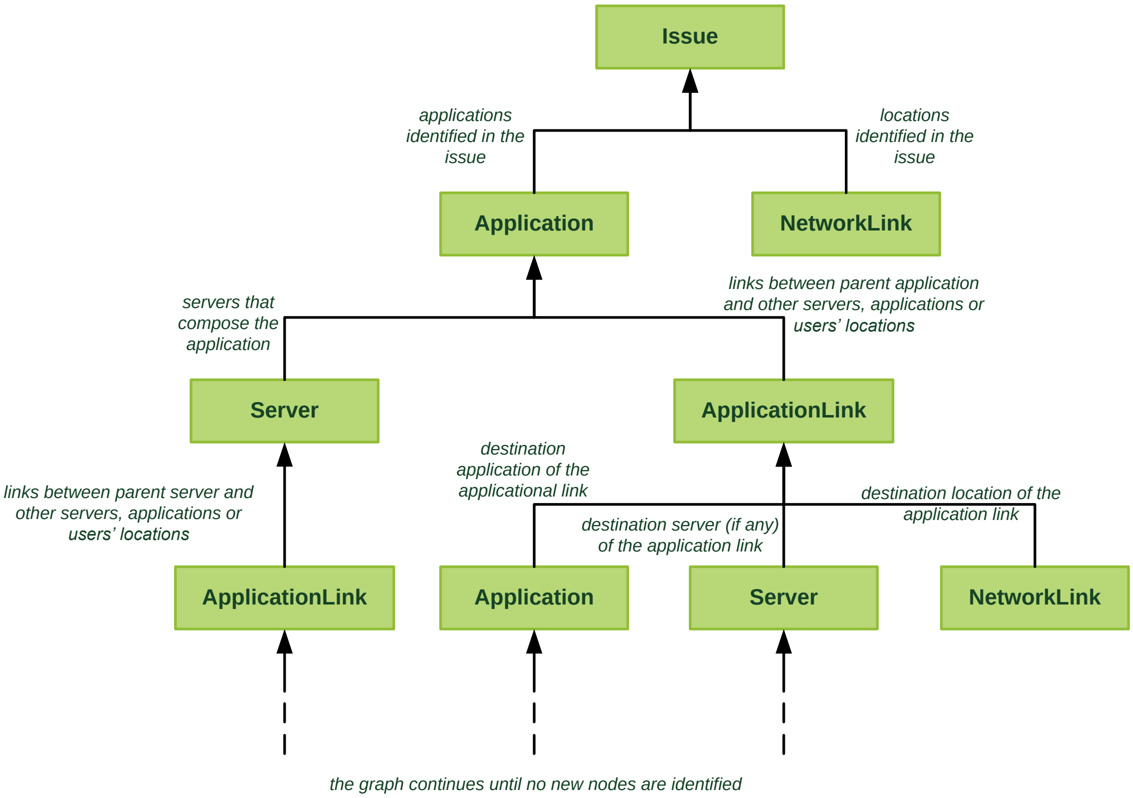

Issue
applications identified in the issue
locations identified in the issue
Application
NetworkLink
servers that compose the application
links between parent application and other servers, applications or users' locations
Server
ApplicationLink
links between parent server and other servers, applications or users' locations
destination application of the applicational link
destination server (if any) of the application link
destination location of the application link
ApplicationLink
Application
Server
NetworkLink
the graph continues until no new nodes are identified

Figure 6 – Generic Network Graph

As we can see from Figure 6 the identification of an issue triggers the creation of the model. An issue results from tickets opened by users. One of the properties of the users' tickets, used to do the second iteration of the model, is their location. The other important property is the application that has been exhibiting performance problems. Using the location and application attributes of the issue, we identify the relevant network and application links (location to application). These links are those that connect the users' sites to the application reported on the issue. Using the two layers, the network and the application, we can cover a wide range of metrics. By also adding the application as a child of the issues component we complete the first iteration of our model.

The second step is to identify the servers that compose the application. When a server is added to knowledge base of our application one of the required attributes is the application to which it belongs. Thus, we use this attribute to identify all the servers that belong to the parent application. Also, we must identify the relations that the parent application has with other applications. To solve this problem, we have two possible approaches. The first one is to use the traffic analysis results to identify all the relations from the servers of a specific application to other applications. The second one and chosen approach is to add to the model only those relations already defined by the underlying monitoring application and that already have metrics associated. The latter approach is chosen due to both time and simplicity constraints. The time constraints prevent us from building a thorough and detailed model. Thus, we opt for building a simpler model where all the nodes have some metrics associated instead of building a larger and complex model, hard to visualize and with several nodes that would not add much to the final root cause analysis due to the absence of metrics. In the future, if we define new metrics and components correctly on the monitoring application they will be automatically included in our model.

The application links are the remaining nodes to connect to the application element, namely, the application to application and the application to server links. These represent application level transactions that do not necessary involve a specific server on one or both sides of the communication. The option to relate some metrics with either one server or the overall application depends on the monitoring application. One of the goals of this project is that it must be easily attached to any infrastructure already in place and complement the existent monitoring and analysis tools. Therefore, whenever possible, we follow the decisions taken within the scope of the monitoring application.

The next iteration of the model construction is to find the application links related to the identified servers. Here the approach is similar to the one used in the application case but, this time, we search for application to server, server to application or server to server links.

The remaining iterations correspond to finding the child nodes for each application link. For these, the sons are created based on the three main attributes of an application link: the destination application which originates a child application, the destination server which originates a child server and/or the location of destination node which may refer to a network link.

After defining what to look for in each component, the model is constructed recursively until no new nodes are created. We only need to define the edges' probability for finishing the model's skeleton. The methods used to define the dependencies strengths are described in following section.

3.4.2. Model Dependencies

There are two main sources for assigning strengths to the edges of the dependency graph. The first source is the traffic analysis results that can be used to identify the most active servers and links. This way, we identify those components that will have a major impact on they parents in case of failure. The second source used is human knowledge. Despite of being an inefficient way of assigning dependencies, the human intuition and intelligence is what we are trying to add to this kind of tools. Therefore we use some human intuition when classifying the dependencies on the generic model (Figure 6) that is used to construct the final model. Nevertheless, we should not neglect the possibility of replacing any manual parameterization by any automatically determined parameters, whenever enough information is available.

The strength of each edge depends on two values. The first value corresponds to the type of child and the predicted impact on a parent node. The second value corresponds to a statistical analysis of how relevant the child is to understand the behavior of the parent. Normally, this is determined from how frequently it relates to the parent. Let us take as an example the relations between the issue nodes and their child nodes (see Figure 7). The issue component has three sets of child nodes. The first set is composed by a single element and corresponds to the application referred on the issue. The two other sets correspond to the communication with the application. Thus it is assigned an equal probability of the problem being originated on the application or on the communication[5]. This probability impacts on the parent if the child nodes have a degraded state (we explain the node state calculation in Section 3.6). The probability assigned to the communication between the users and the application is further split in two as the problem can either be due to application or network level. Then for each set, a statistical analysis is performed in order to decide the second value for the probability of an edge. The sum of the probabilities of all sets (the first value) is equal to 1. And the sum of the probabilities within a set is also equal to 1. The final probability of an edge results of the multiplication of the two probabilities. Therefore the total sum of the probabilities of all edges departing from one node is equal to 1.

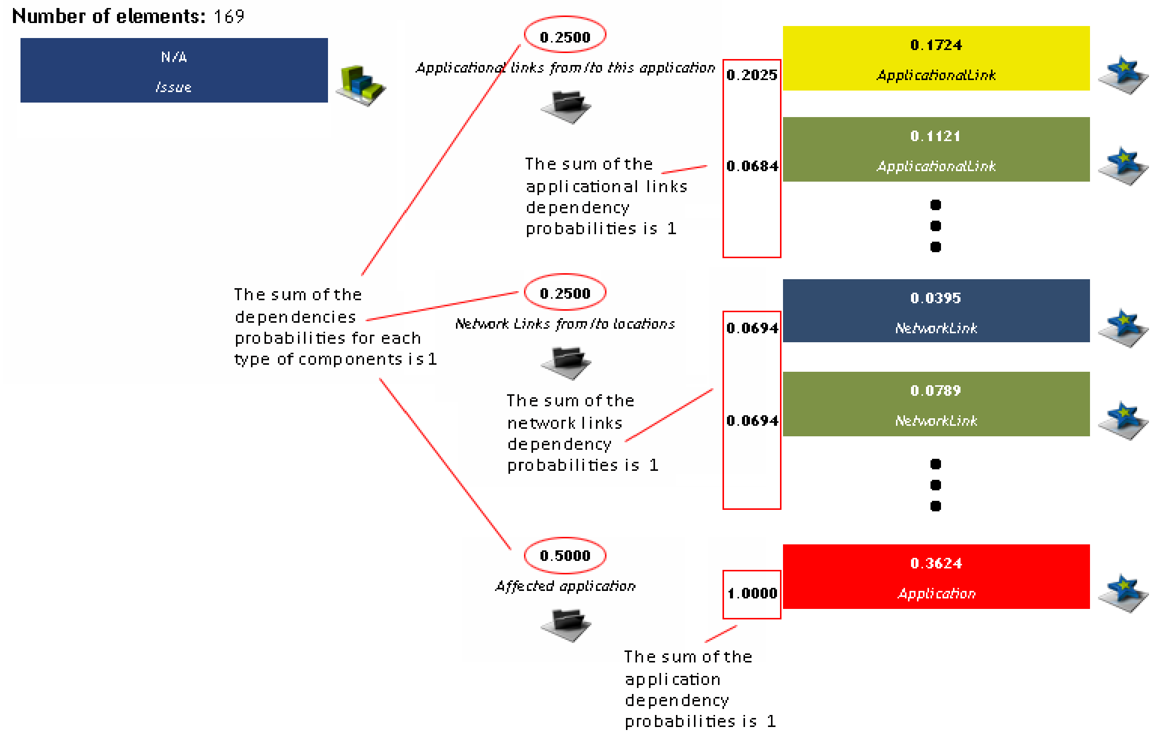

Number of elements: 169
N/A
Issue
0.2500
Applicational links from/to this application
0.2025
0.1724
ApplicationalLink
0.0684
0.1121
ApplicationalLink
The sum of the applicational links dependency probabilities is 1
The sum of the dependencies probabilities for each type of components is 1
0.2500
Network Links from/to locations
0.0694
0.0395
NetworkLink
0.0694
0.0789
NetworkLink
The sum of the network links dependency probabilities is 1
0.5000
Affected application
1.0000
0.3624
Application
The sum of the application dependency probabilities is 1

Figure 7 – Example of one level of the model

[5] Given the automatic nature of this tool, all manual parameterization, even if it corresponds to an interpretation of the reality, should be replaced by automatic inference in the future (see Future Work in Section 5). The manual solutions are justified by time constraints.

The values assigned for the first component of the strength of a relationship have been defined together with the tool's generic network model. The second component of the relationship strength is determined by the traffic analysis module described above. Using the statistics obtained from the traffic analysis, we are able to assign relative probabilities to many of the relationships present in the model. Basically, the dependency probabilities are based on how frequently a relationship is seen in the network. Two machines that communicate often are more likely to influence the behavior of each other and to propagate failures. Nevertheless, as we will see in the future work section, many other properties may be used to correlate entities. In order, to be able to relate the several entity types to the traffic observed in the network, we must identify the location, server or user site to which an IP address corresponds. The following dependencies are present in the model:

- An **Issue** depends on:
  - **Network Links –**the issue contains references to several user locations and one application, hence the dependency is as strong as the number of requests made from each user site to the application. The probability associated with the dependency is given by the ratio of the number of requests per location to the total number of requests;
  - **Application Links (Client Sites to Application) –**the same attributes are used to identify the application links, and therefore the probability of relation will also be given by the fraction of the number of requests from the specified site to the total number of requests;
  - **Application –** each issue has a reference to only one application, consequently the dependency strength will correspond to the value one, as there are not any other child nodes.
- An **Application** depends on:
  - **Servers –** an application is composed by several servers. The most used servers are considered to have more impact to the overall state of the application. As we can easily conclude this is true for the generality of the applications. If the transactions are relevant for the immediate perception that end users have of the application performance, they are executed online and considered on the statistics. Otherwise, they are executed offline using batch processing at non-work hours, will overload the application and with high probability impact the performance of the application. The probability for the application to server relation will thus be given by the number of flows to and from each server with respect to the total number of the server's flows;
  - **Application Links (Application to Application) –** the application links that are loaded to an application correspond to interactions between different applications, where no servers are specified. Thus, the probability associated with each dependency will be directly proportional to the number of flows between the servers of each of the two applications.
- An **Application Link** may depend on:
  - **Application –** each application link only has references to a single application, to a single server and/or to a single location. Thus, the dependency between an application and the parent application link will correspond to 1;
  - **Server–** as in the previous case the dependency will also correspond to the value 1.
  - **Location –** as in the previous situations the dependency will have a strength of 1.

- A **Server** depends on:
    - **Application Links –** the only dependencies considered for servers are their interactions with other servers. The probability associated with each application link will correspond to the number of flows between the parent server and the peer server identified in the application link.

3.4.3. Nodes and Metrics

Each node has several metrics defined for it. The metrics used are all those available on the monitoring applications for each node. As more focus is given to the development of the model and node state evaluation, the option is to optimize the use of the information already available. Naturally, the absence of certain metrics made it difficult to develop some reasoning about what is happening inside of the application. Thus, in this first version of the tool, the metrics associated with a node are used to evaluate how abnormally a node is behaving. This knowledge, together with the fact that we are analyzing a time period that corresponds to some performance degradation perceived by end users, will allow the identification of the nodes (servers, links, applications, etc.) that are most probable to have originated the problem. Table 2 presents a summary of the metrics or type of metrics that might be available for each type of node.

Component	Metrics	Description
Application	Process Execution	Indicates if a specific process of the application is running (1 metric per process)
	Transaction Response Time (Server side)	Measures the response time of specific transaction of the application measured on the server side (no network involved)
Server	Active Processes	Number of processes active on server
	CPU Usage	Percentage of CPU in use
	Disk Space	Percentage of free space on disk (1 metric per disk partition)
	Latency	Response time of the server
	Load Average	Mean load of the server
	Memory	Percentage of used memory
	Oracle Database Metrics	These are several oracle-specific metrics that represent the state of the database (if it is running, how long a login takes, average wait time for queries, etc.)
	Process Execution	Indicates if a specific process of the server is running (1 metric per process)
	Swap	Percentage of swap memory used
	Time-drift	Deviation of the system clock
	Traffic	Volume of Network Traffic
	Users	Number of users logged in the system
Application Link	Resets	Number of TCP resets observed
	Timeout	Number of TCP timeouts observed
	Traffic	Volume of Network Traffic
	Transaction Response Time (Client-Side)	Measures the response time of specific transaction of the application measured on the client side (includes the time spent on the network)
Network Link	Bandwidth	Available bandwidth
	Latency	Latency of the link

Table 2 – List of possible metrics

3.5. Time series analysis

One of the components with a major impact on the final results is the time series analysis module. This module is responsible for identifying the most relevant events that are observed in a metric, during a time period considered relevant for the issue analysis. These events may be deviations from the normal behavior observed on a metric at a specific time of day, or simply the violation of pre-defined thresholds which represent widely known situations that can cause problems on servers or links. For instance, for some metrics, as disk space or occurrence of timeouts, we only need to verify if some conditions are met. If we have no space on one partition or if we note the occurrence of timeouts we know that these situations can eventually cause or indicate problems.

The first step of the model is to choose the time period during which we need to analyze the behavior of the metrics. The analysis time period includes the time during which tickets were opened by end users (corresponds to the start and end timestamps of the issue) and some previous period during which the causes may have been reflected on the collected metrics. The user reaction is not immediate. Figure 8 represents the two main periods we must consider. The causal behavior has to start earlier than the effect that may be noticed past the end of the cause's period. The effect period includes (but may be larger than) the period referred in the issue.

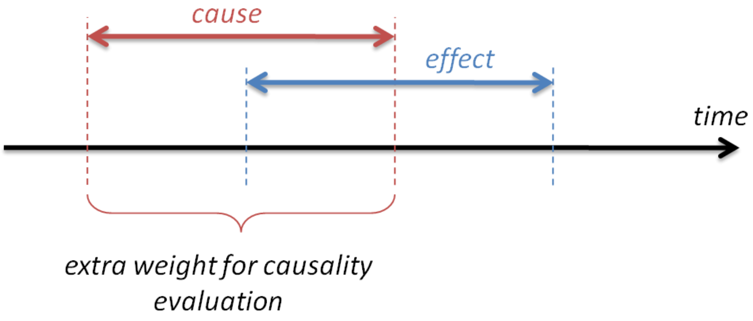


Figure 8 – Cause and effect on the evaluation time period

Therefore we must choose how long we should extend our analysis period in order to include the cause interval. The choice may be based on the period identified by the issue, but this option may bring some problems. First, the issue is created using the tickets opened by users, which may conduce to an issue duration that may not represent the reality faithfully. In that case, we will propagate the error to our analysis. On the other hand, if the issue has a very short duration, any arbitrary deviation in some metrics will have a huge influence in the final results. Thus, our option is to make an extension of the same size of the issue's duration, but using a minimum period size for smaller issues.

The next step would be to identify deviations from the normal behavior for each of the metrics. To achieve this goal we must use training data from the last few weeks or months. At this point, one must use some knowledge about the usage pattern of the. In large companies, the use of the applications follows a seasonal pattern. Although some variations are observed between months or weeks the main unit of pattern is one day. In weekdays the usage pattern of almost all applications is very similar and even on Fridays the pattern does not change much. The same can be concluded concerning weekends and holidays. Another advantage of using the day as the basic unit for pattern definition is the number of data points we have per each pattern point. If we were to consider weekly or monthly patterns we would have to use much larger training periods, and we would end up using too old data. The earlier periods would be of little or no relevance to assess the present behavior of an application.

The pattern determination uses statistical information about the historical data of a metric. The granularity of the pattern will be equal to the cycle attribute of the metric. The cycle, a value defined by the company's monitoring application, represents the period of data collection. The first step of the pattern creation consists on dividing the day in intervals of size equal to the metric's cycle value. Then, for each point of the training data, we must identify the interval of day to which it belongs and update the statistics for the interval by adding the point's value. In the end we will obtain the standard deviation and mean for each interval in a day. These and other statistics (e.g. kurtosis, variance, etc.) are all stored and will constitute the raw materials for setting up a pattern. The final pattern will be defined by two time series that correspond to the lower and upper bounds of the expected values for the data. Several statistics are stored at the time the historical data is analyzed. Therefore, the pattern can be built in different ways for each metric. The generic procedure to obtain the patterns is based on the characteristics of the standard deviations for several distributions. Table 3 represents the percentage of values that are within several units of standard deviation for the normal distribution and for any distribution.

	Interval			
Distribution	*within 2σ of the average*	*within 3σ of the average*	*within 4σ of the average*	*within 5σ of the average*
Normal/Gaussian	95.44%	99.72%	99.994%	99.99994%
All	75%	89%	94%	96%

Table 3 – Distribution of values for a distribution regarding the standard deviation

The last row represents the worst-case situation, i.e. the percentage of values that are within the several units of standard deviation despite the actual distribution of the metric values. One should remember that the distribution is applied to each of the day’s intervals. For most cases the normal distribution would fit well to the analyzed data.

After analyzing the data contained on Table 3 it we decided to use an interval of three standard deviations around the mean value for each point. For each interval in a day we will have different statistics and thus different predictions based on the behavior of the metric in that interval in the past. The state determination for a metric will be based on how much the values are deviating from the pattern (see Section 3.6).

Other approach to detect relevant events is to compare the metric values with a pre-defined threshold. In this case, no training data is necessary but, on the other hand, we need to provide the threshold value. In most situations, this is an undesirable approach because is prone to errors of human reasoning and is non-adaptive, i.e. it does not change with time and behavior of the metrics. But, in some cases it is intrinsically the most adequate approach. If we are measuring the free disk space for a disk partition, the behavior of the metric is irrelevant. What is really important is to detect if the free space reaches zero at any point. This is a typical case where we rather use threshold violation analysis instead of pattern detection. The procedure for applying the threshold is trivial, as it consists on comparing the values with the threshold and verifying if there is a violation. One should note that even for applying thresholds, the day is also split into fixed size intervals. The reason for this is to make it possible to apply the state calculation algorithms uniformly to all metrics.

Several other analyses are possible and will be explored in the future. For instance, we can try to find other relevant events that are not necessarily synonym of a problem but can be relevant to find the root cause. Examples of these events are a metric reaching an historical maximum or minimum, reaching a maximum or minimum on a specific time window, presenting a higher variation than normally without crossing the pattern boundaries, and so on. Other possibility to explore is to learn from the analysis made and from the evaluation of the final results using a machine learning algorithm. The most common behaviors under failure may be saved and utilized in future analyses by the tool.

3.6. Cause and Effect Probability

The next step is to compute the state of each metric for the period in analysis and the overall state of a component given the metrics associated with it. The state aims to represent the probability of a given node being affected in some manner by the occurrence of problems. During a performance degradation period, a node that presents some deviations of the usual behavior may be a:

- **Causal Node** – if the node is in the origin of the problem, which means that its behavioral discrepancies are causing other nodes to behave strangely. Normally these nodes are affected early in the considered time period;
- **Affected Node** – if the node is affected in response to abnormal or problematic behaviors from other nodes. These nodes may be affected for the whole period of analysis just like the causal nodes.

To accurately identify the root cause for an issue, one should be able to distinguish between causal nodes and nodes that are only exhibiting the effects of the problems. The main problem is that isolating the two cases is neither always possible nor easy. Both nodes may present anomalies almost at the same time.

Several slightly different approaches were tried in order to determine the state of a metric. The approaches taken for this module are all based on how much a metric is deviating from its normal behavior or how much it violates a pre-defined threshold (as explained earlier this approach is only used with metrics that do not exhibit a pattern on their behavior and have values that are intrinsically a synonym of problems). As was described in Section 3.5 a day is divided into intervals of size defined by the metrics cycle, i.e. the metric sampling period. Thus, we are able to define an issue vector for each metric, which is a set of values representing the behavior of the metric during the relevant period.

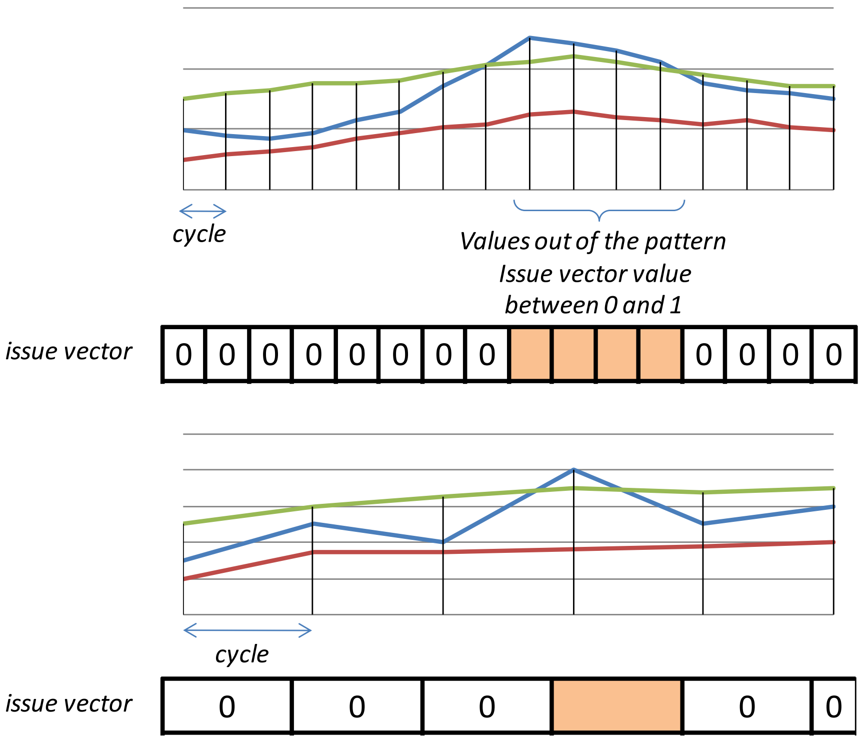


Figure 9 – Relation between issue vectors for metrics with different cycles

Figure 9 shows the notion of the issue vector and how it is filled based on the data obtained from the pattern analysis. The size of the issue vector is determined by the metric's cycle. Hence, each element of this vector has a scope inversely proportional to the size of the vector. On other words, for issue vectors with fewer elements each element represents the metric behavior during a larger period of time. While the metric's value is inside the values predicted, the issue vector elements have a value of zero. Whenever the metric's value goes beyond one of the patterns limits, the value of the issue vector is a number between zero and one. These abnormal values are represented on the figure using a light shadow. The exact value placed in the issue vector is determined by measuring the deviation from the predicted pattern. The value is assigned according to the following formula:

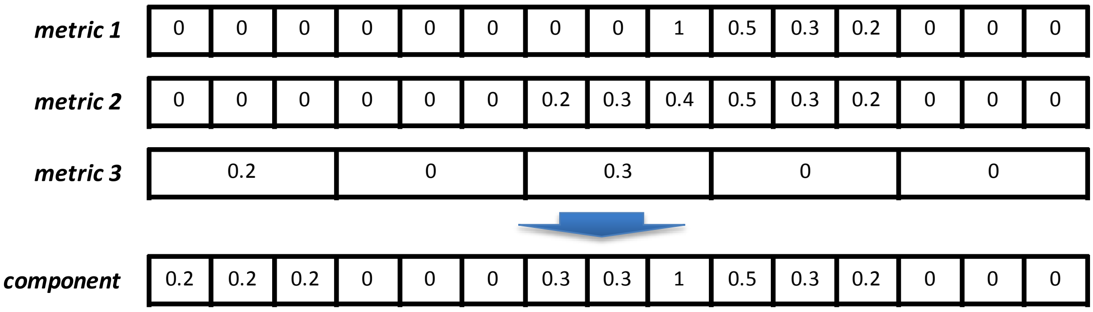
$$issuevector[k] = \begin{cases} \dfrac{|timeseries[k] - pattern[k]|}{\sigma} & \Leftarrow \quad |timeseries[k] - pattern[k]| < \sigma \\ 1 \quad \Leftarrow \quad |timeseries[k] - pattern[k]| \geq \sigma \end{cases}$$

The idea of the previous equation is to evaluate the *amount* of deviation from the predicted pattern. If it surpasses one standard deviation it is considered a large deviation from the normal behavior. Otherwise it is classified according to the difference from the pattern boundary.

After determining the issue vector for all metrics associated with a component, the state of the metrics must be transposed for the component. For determining the issue vector of the component we apply the addition rule of probabilities:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

To apply the previous formula, the issue vector must have a size equal to the maximum size among the issue vectors of the associated metrics. Then, for each position of the issue vector of the component, the higher probability on that position for all metrics will be assigned.



metric 1	0	0	0	0	0	0	0	0	1	0.5	0.3	0.2	0	0	0
metric 2	0	0	0	0	0	0	0.2	0.3	0.4	0.5	0.3	0.2	0	0	0
metric 3	0.2			0			0.3			0			0		
component	0.2	0.2	0.2	0	0	0	0.3	0.3	1	0.5	0.3	0.2	0	0	0

Figure 10 – Calculation of the component state

From each issue vector we will compute the probability ofa metric or component being a cause of the issue ($p_{metric}\ p_{node}$). Several methods were tried to obtain the value. The following four methods were tested:

- **State based on the number of relevant events** – in this method each element of the vector is either zero or one, depending on the existence or not of a relevant event for the corresponding time interval. The final state value will be the simple ratio of relevant elements to the total elements of the vector:

$$p_{metric} = \frac{\sum_k [issue\ vector[k]]}{\#\ issue\ vector}$$

- **State based on the measure of deviation** – as we have seen in the description of the issue vector, a value between zero and one is assigned to each element. Using this method the final state will be determined dividing the sum of all elements by the total number of elements of the vector:

$$p_{metric} = \frac{\sum_k issue\ vector[k]}{\#\ issue\ vector}$$

- **State based on the causal period**–earlier we have explained the idea behind of dissociating the causal and the effect period. This method is based on only taking into account the period until the impact is felt by the user, i.e. the start instant of the issue. Therefore, the state is determined by applying the previous method, but only to the period before the start of the issue:

$$p_{metric} = \frac{\sum_k issue\ vector[k]}{\#\ issue\ vector} \quad for\ \ k < k_{issue\ start}$$

- **State based on weight for the causal period** –this method involves applying a weighted function to the entire issue vector. We assign a higher weight for the values under the "causal" period. The value of the weight will be decreasing linearly until the end of the issue.

The parameters for determining the weighting function must be determined for every different issue, because it depends on the duration of the issue and on the total period of analysis:

$$f_{weighted} = \begin{cases} a \;\Leftarrow\; k < k_{issue\ start} \\ a - a \times \dfrac{k - k_{issue\ start}}{k_{issue\ end} - k_{issue\ start}} \;\Leftarrow\; k \geq k_{issue\ start} \end{cases}$$

$$where \quad a = \frac{2}{k_{issue\ start} + k_{issue\ end}}$$

3.7. Root Cause Candidates Selection

After creating the network model all nodes have an associated state that corresponds to the probability that the node is experiencing anomalies. All relations between the components of the model have also been assigned a strength representing the probability that the child node is affecting its parent node.

This module is responsible for the identification of the most relevant components and metrics to explain the origin of the issue. The choice of root cause candidates should consider not only the state of the nodes, but also how likely it is that they have caused or influenced the identified issue (i.e. the dependency strengths).

To identify the most relevant components and metrics, two main approaches were tested:

- **Independent analysis of the components**, where each component is assessed considering its state and its dependencies until it reaches the root issue;
- **Causal path lookups**, where each node in the graph may “inherit” the state of its children. In this case, the state of the parent is determined by computing the weighted sum of the children’s states based on the strength of the dependencies and by comparing it with the parent’s internal state.

The first approach has one major drawback: the search is biased towards the components that are nearer to the root issue.

To apply any of these approaches, one must travel through the graph recursively until a childless or repeated node is found. Each node returns its internal state and triggers the execution of the chosen protocol on its sons. The calculation must stop when a repeated node is reached because the graph may contain loops.

3.7.1. Independent Analysis of Components

As we described earlier, this approach consists in evaluating each component individually. The state of each component is multiplied by all the dependencies observed in the path from it to the root issue. Thus, as farther away the component is from the root issues, less likely it is to be considered relevant to contribute to the performance problem.

Figure 11 shows an example of how this type of component selection is applied. The larger boxes represent the network model components, to which generic names are assigned. Near each edge representing a dependency, there is a value, which represents the corresponding strength. The smaller rectangles contain the final value assigned to the component that is obtained by multiplying their internal states by their upward dependencies. The components that have a state equal to zero are those for which no anomaly was detected during the issue's period and during the period that preceded it.

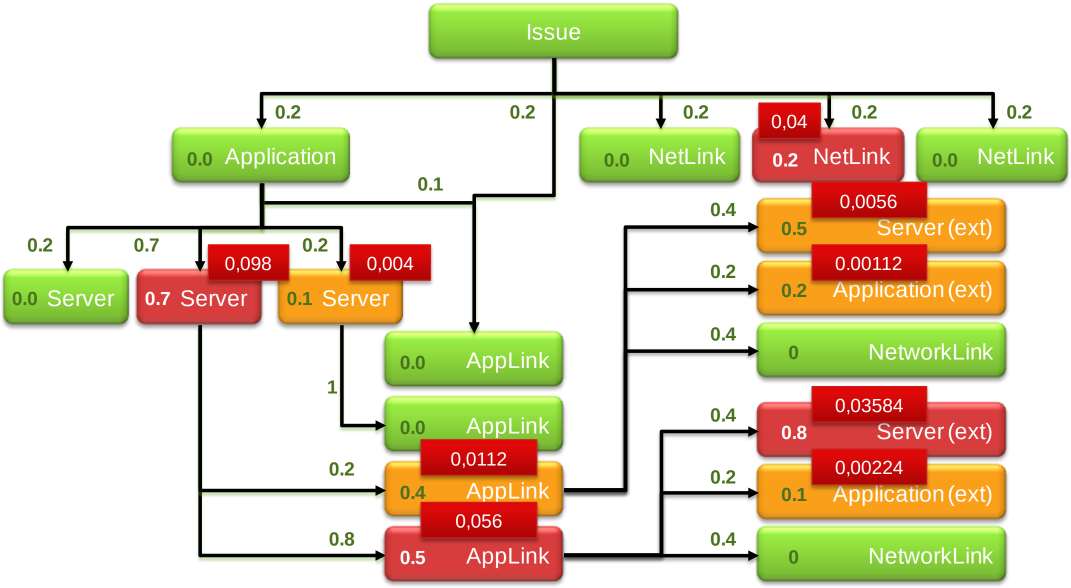


Figure 11 – Example of identification of relevant components using independent analysis

As we can see the first level nodes, i.e. those immediately below the root issue, are easily considered as extremely relevant. When compared to nodes several layers down the graph, these components may have only a small number of deviations from the pattern detected. On the other hand the nodes have a direct impact on the end-user experience, and that immediate influence should be valued.

3.7.2. Causal Path Lookup

This second approach tries to make a balanced approach to dependencies and state evaluation. This strategy overcomes the limitation of the previous method concerning overrating the nodes closer to the root node. In each round, the most probable causal path is identified.

A causal path ($causal\_path_{root_j}$) of a root component (c_{root}) is identified by the probability of the path (p_{CP_j}) and the set of components (S_{CP_j}) included on the causal path. Each component is represented by a tuple containing its identification (c_i), the dependency strength for the parent node (dep_{c_i}) and its state (p_{c_i}).

$$causal\_path_{root_j} = \left\{p_{CP_j}, c_{root}, \left\{dep_{c_i}, p_{c_i}, c_i \middle| c_i \in S_{CP_j}\right\}\right\}$$

In order to obtain an ordered list of the most probable causal paths, the state of a node that is identified as a causal node at the end of a round, should be set to zero in the following one. This allows the algorithm to remove the effect that the node is having in the overall model, enabling the identification of alternative causal paths. The algorithm recursively travels through the graph, to calculate the node state. Therefore, the following procedure is executed for each node:

1. Compute or retrieve the internal state of the node;
2. For each son trigger the execution of this procedure;
3. Compute the weighted sum of the child nodes states, where the weights are given by their dependency probability to the parent;
4. If the internal state of the node is higher than the weighted sum of its sons, then the node is assigned as the end-node of a new causal path which is returned to its parent;
   4.1. The final state of the node corresponds to its internal state.
5. Otherwise, the node is added to the causal path returned by the node with the highest value.
   5.1. The final state of the node corresponds to the weighted sum of its sons' states.

In the end of each round, the final node of the causal path is made null, i.e. its state will be considered zero, for the next iteration. The output of this algorithm will be a rank of causal paths ordered by their probability.

3.8. The graphical user interface

The graphical user interface assumes an important role in this project. As one of the main goals is to create an application that can ease the work of network and system operators, the interaction with the application must be as easy and as intuitive as possible. In order to achieve the goal of compatibility with existing software, the project's interface is web-based. This characteristic allows a simple integration with other tools already deployed in the company and can be made available through their intranet.

The graphical user interface has two main sections:

- **Traffic Analysis:** where one can upload traffic analysis requests and see the results down to the level of a single packet (naturally this level of information should not be kept indefinitely, as it requires a huge amount of storage space). This section contains some graphical analysis of the results obtained enabling the acquisition or confirmation of information about the network and infrastructure[6];
- **Root Cause Analysis:** where one can visualize the root issues and trigger a root cause analysis for each one of them. This section includes the network model visualization and the list of relevant metrics, components and identified causal paths.

Some of the main features of the traffic interface are represented in Figure 12. The interface allows the user to visualize the information both in tabular and graphic form. The tables and graphs represent a first level of simple traffic statistics. Furthermore, Etymon has an interface that presents

[6] During this project we did not took full advantage of this extra information about the network in the construction of the network model. Some ideas about the future work in this area will be provided in Section 5.

results of traffic flows correlation. The results may be seen in a table, but the most intuitive interface is a graph representation. This view allows a quick perception by the user of the composite flows identified.

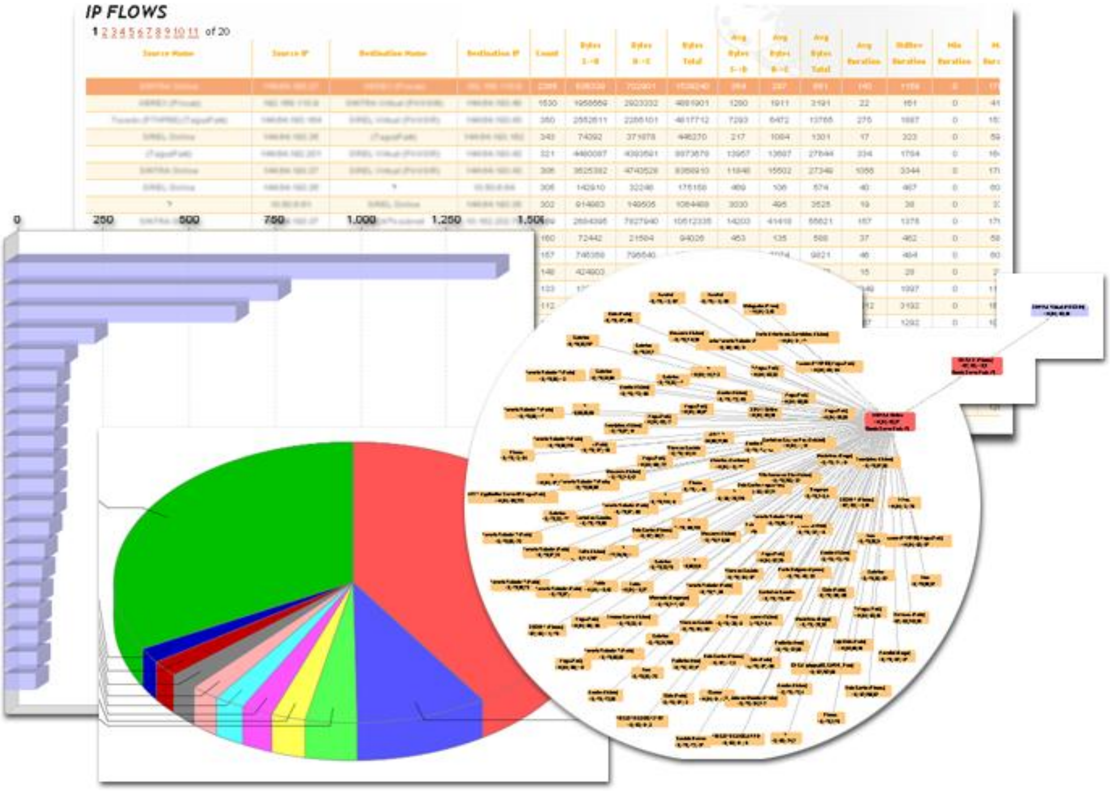


Figure 12 – Traffic Analysis Interface

Figure 13 presents the interface developed for visualizing the network model. As we can see, the model is a directed graph which is represented two levels at a time (one parent and the respective child nodes). Each child node has a specific color representing the probability of influencing the parent's state. The root causes and components can later be listed in a normal table view.

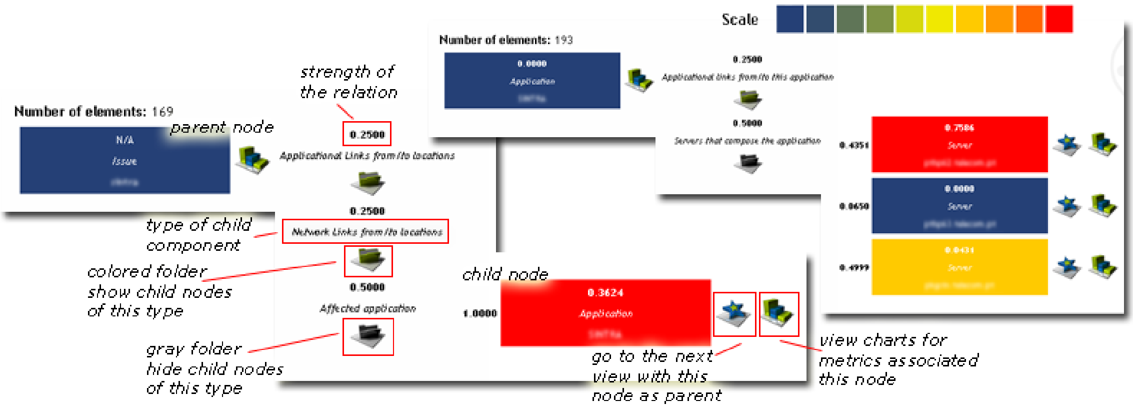

Scale
Number of elements: 193
0.0000
Application
0.2500
Applicational links from/to this application
0.5000
Servers that compose the application
0.4351
0.7506
Server
0.0650
0.0000
Server
0.4999
0.0431
Server
strength of the relation
Number of elements: 169
parent node
N/A
Issue
0.2500
Applicational Links from/to locations
0.2500
Network Links from/to locations
type of child component
colored folder show child nodes of this type
0.5000
Affected application
gray folder hide child nodes of this type
child node
1.0000
0.3624
Application
go to the next view with this node as parent
view charts for metrics associated this node

Figure 13 – Network model interface

4 Results

In the previous section we described the framework developed to choose the most relevant components and metrics when performing root cause analysis. In this section, we describe our findings and provide some insight of how these results can be used in order to gather information on the network and to analyze the performance problems that frequently occur in such complex networks.

The network used was chosen due to its large size and high complexity. As we have described in Section 2.1, we focus only in one application. But this application has many dependencies, and for that reason the network model ended up including many applications and network locations. Therefore as it was explained earlier the problem may be located in a remote system which is, due to the nature of the applications and the relations among them, influencing the performance perceived by the end user.

Section 4.1 comprises a sample evaluation of the results obtained during traffic analysis and they can be used to increase the information about the network. Section 4.2 includes a study about the issues identified and their characteristics. The network model obtained for the analyzed issues is described in Section 4.3. As the target application was the same for all issues the network model is similar. Finally, Section 4.4 presents the major final results obtained and they should be evaluated in this phase of the project.

4.1. Traffic Analysis Results

The traffic analysis has two main purposes. The most important goal is to gather information about which systems are used more frequently and whose servers recurrently interact with the servers of the main system. A secondary objective is to gather some knowledge about the network, to identify the most important or frequent flows and to discover correlations among flows.

Most of the information obtained is specific to the network of this company and cannot be described here without disclosing its proprietary information. The approach taken on this section is to display some of the results, without revealing the real physical identities (application names, server names or addresses, etc.).

The traffic analysis can be filtered by the IP addresses of a specific application. In Figure 14 we represent the most frequent flows identified for the application chosen as testbed. As we can see there are flows that are very important in the normal functioning of one application. The three major flows represent the three different communication streams, in which one of the participants is a server from the chosen application. These results lead us to conclude that the other applications involved in these flows must be considered important when analyzing the issues for the central application. This will be done automatically by including these results in the network model. The relative frequency of each flow will influence the dependency relations between the applications involved.

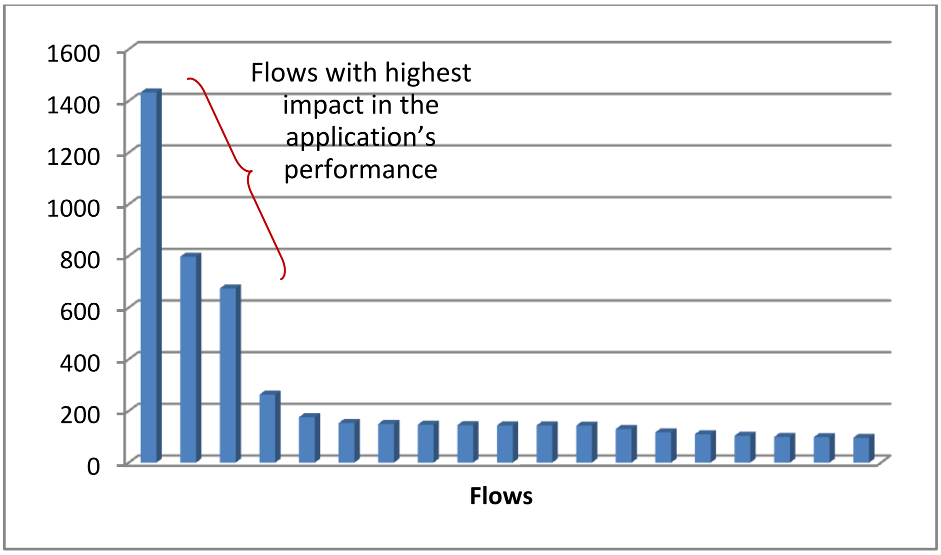


Figure 14 – Frequency of the detected flows

While these main flows normally remain in the top places of the list of most frequent flows, connections involving other servers may have a momentary importance. It was observed during some other period of observation that one flow, confirmed as being unusual, was responsible for a large volume of data during the normal working hours. The traffic transferred had been significant and had caused some network congestion. Therefore, at the time of its occurrence that abnormal flow would be relevant to explain any perturbations felt by the users. The observations depend on the period of observation and recent events may be extremely important to explain root causes.

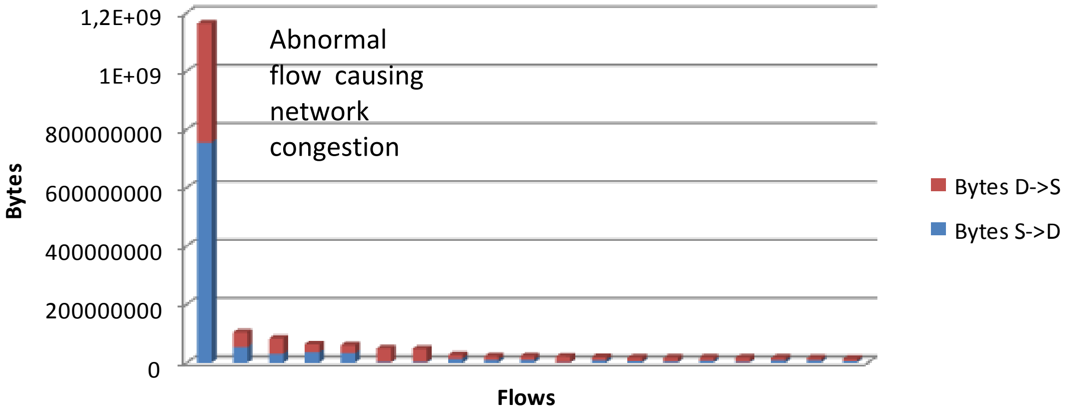


Figure 15 – Abnormal flow identified on a limited period

Other results were obtained by analyzing composite graphs of traffic flows. In this case, the traffic analysis is not filtered, in order to identify workflows involving other applications. The sample graph shown in Figure 16 represents an example of one interesting observation. The graph is intentionally small in order to make server names and addresses illegible. The graph is obtained by finding sequences of flows, i.e. flows that start from a node that has received a flow immediately before. The large cloud of nodes on the left side, are clients making a request to the node on the center of the image. We can see that, immediately after these requests, this node has communicated with other application. Although not seen in image we could verify, through the ports used that it corresponded to a communication between two databases. The final application responded to the first application, but now using a virtual address. This kind of dependency is interesting because not only it informs us of the direct relations between machines but can provide information about more complex workflows.

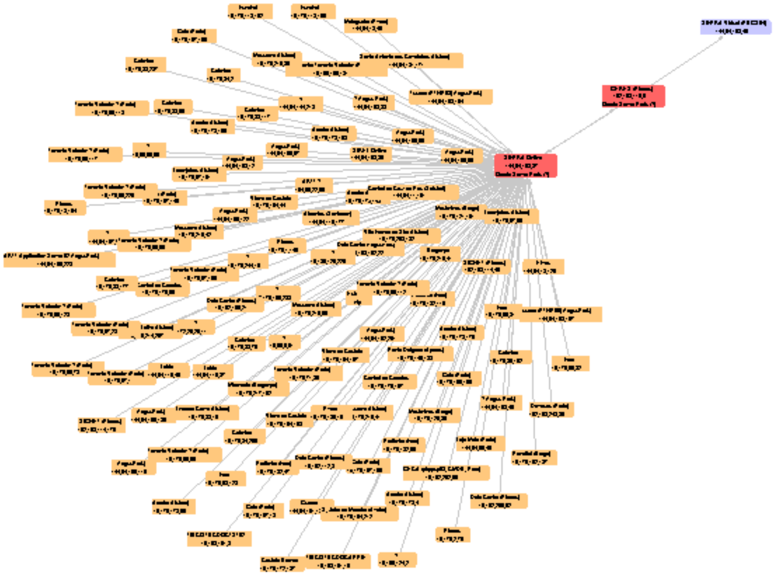


Figure 16 – Graph of related traffic flows

From this point on we can start characterizing workflows. This information can be complemented using more information in packets and by identifying some actions. The communication with the database uses a proprietary protocol that provides some indication about the type of action that is being performed. For instance, if a user is logging in, there is a specific header field value that indicates that action. Updating this analysis with this type of information, allows us to create a profile of usage of the applications. We plan to extend these results in order to draw some conclusions concerning the role of each identified host. This intention is based on several logical observations. For instance, we observe that most frequent flows are between server machines, that clients are normally only initiators of a flow, that the protocols and services in use provide clear information if the destination (or even source) host is a database, web server or simply an middle-tier application server. We can also clearly identify groups of hosts that are mainly initiators of connections in order to characterize each user site in terms of volume of data and number of users.

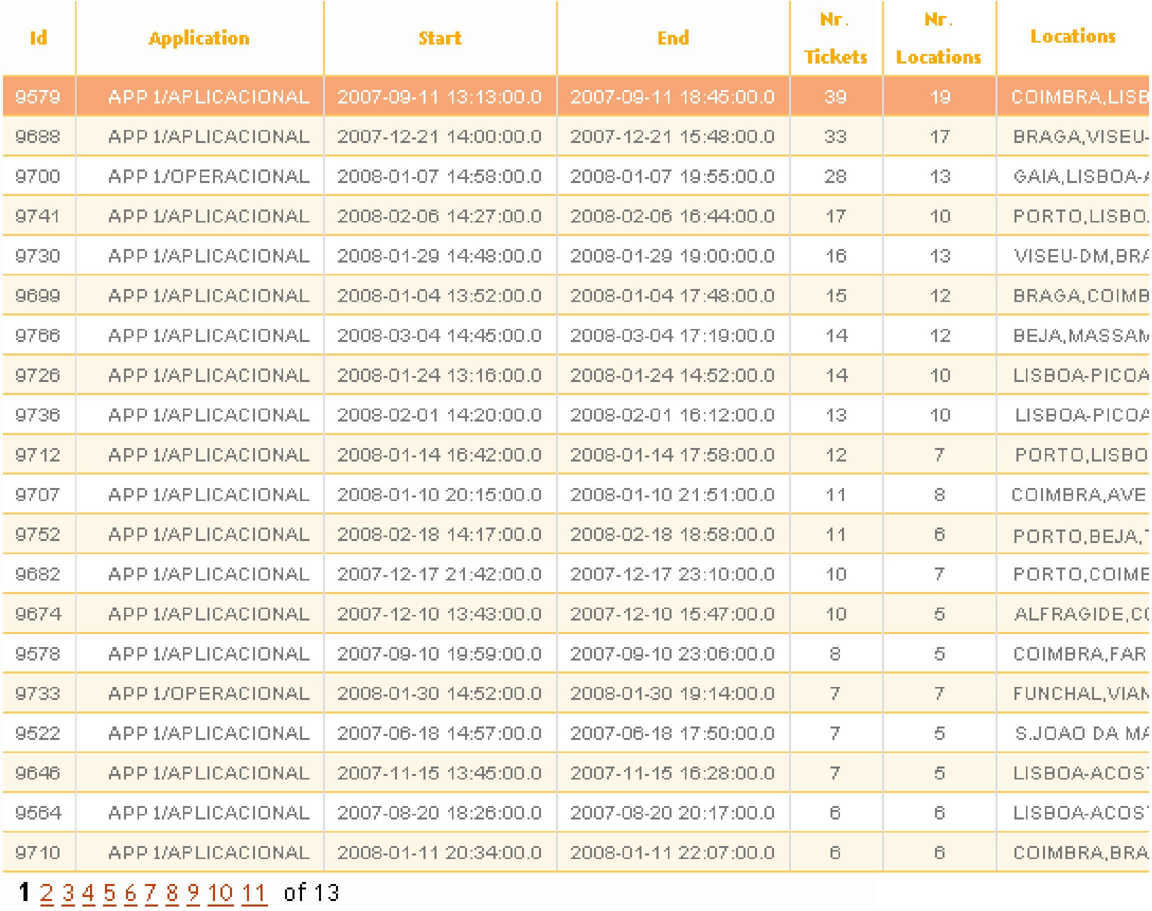
4.2. Issue Identification

This section presents some of the results obtained after aggregating the reports of problems sent by users into an entity with more significance: the issue. An issue represents the global problem occurring in the network and may involve one or more tickets and/or locations. The method used to determine the issues consisted, as explained earlier, on filtering and grouping tickets by application and by their proximity in time.

In Figure 17 we present the top results obtained for a period of nine months. In this period we can observe that a diversity of issues have been raised for the testbed application, here represented by “App 1”. The most relevant columns are the start and end columns, from which we can derive the duration of the issue, the number of tickets which represents how many users reported the problem and the number of different locations which represents how spread was the problem.

As we can see, the number of tickets tends to grow with the number of locations. This happens because, normally, one site of users does not open many tickets for what they perceive as being the same problem. Normally, these sites are call centers having one or two coordinators, to whom the operators complain. Therefore the number of tickets does not necessarily represent the number of users affected by the problem.

In this application the objective is to analyze the root causes for the most relevant issues. To judge which issues should be given a higher priority one should measure the number of end-users affected or, depending on the application, the number of clients too whom the operators are unable to provide some service. As an indirect measure, we can use both the number of tickets and the number of locations to be able to understand which tickets have more impact in the network. The duration of the issue may be taken into consideration but a large duration does not necessary mean that the issue is more severe. For instance, some issues may include some less loaded periods as the lunch time, thus reducing its overall impact.



Id	Application	Start	End	Nr. Tickets	Nr. Locations	Locations
9579	APP 1/APLICACIONAL	2007-09-11 13:13:00.0	2007-09-11 18:45:00.0	39	19	COIMBRA,LISB
9688	APP 1/APLICACIONAL	2007-12-21 14:00:00.0	2007-12-21 15:48:00.0	33	17	BRAGA,VISEU-
9700	APP 1/OPERACIONAL	2008-01-07 14:58:00.0	2008-01-07 19:55:00.0	28	13	GAIA,LISBOA-
9741	APP 1/APLICACIONAL	2008-02-06 14:27:00.0	2008-02-06 16:44:00.0	17	10	PORTO,LISBO
9730	APP 1/APLICACIONAL	2008-01-29 14:48:00.0	2008-01-29 19:00:00.0	16	13	VISEU-DM,BR
9699	APP 1/APLICACIONAL	2008-01-04 13:52:00.0	2008-01-04 17:48:00.0	15	12	BRAGA,COIMB
9766	APP 1/APLICACIONAL	2008-03-04 14:45:00.0	2008-03-04 17:19:00.0	14	12	BEJA,MASSAM
9726	APP 1/APLICACIONAL	2008-01-24 13:16:00.0	2008-01-24 14:52:00.0	14	10	LISBOA-PICOA
9736	APP 1/APLICACIONAL	2008-02-01 14:20:00.0	2008-02-01 16:12:00.0	13	10	LISBOA-PICO
9712	APP 1/APLICACIONAL	2008-01-14 16:42:00.0	2008-01-14 17:58:00.0	12	7	PORTO,LISBO
9707	APP 1/APLICACIONAL	2008-01-10 20:15:00.0	2008-01-10 21:51:00.0	11	8	COIMBRA,AVE
9752	APP 1/APLICACIONAL	2008-02-18 14:17:00.0	2008-02-18 18:58:00.0	11	6	PORTO,BEJA,
9682	APP 1/APLICACIONAL	2007-12-17 21:42:00.0	2007-12-17 23:10:00.0	10	7	PORTO,COIM
9674	APP 1/APLICACIONAL	2007-12-10 13:43:00.0	2007-12-10 15:47:00.0	10	5	ALFRAGIDE,C
9578	APP 1/APLICACIONAL	2007-09-10 19:59:00.0	2007-09-10 23:06:00.0	8	5	COIMBRA,FAR
9733	APP 1/OPERACIONAL	2008-01-30 14:52:00.0	2008-01-30 19:14:00.0	7	7	FUNCHAL,VIA
9522	APP 1/APLICACIONAL	2007-06-18 14:57:00.0	2007-06-18 17:50:00.0	7	5	S.JOAO DA M
9646	APP 1/APLICACIONAL	2007-11-15 13:45:00.0	2007-11-15 16:28:00.0	7	5	LISBOA-ACOS
9564	APP 1/APLICACIONAL	2007-08-20 18:26:00.0	2007-08-20 20:17:00.0	6	6	LISBOA-ACOS
9710	APP 1/APLICACIONAL	2008-01-11 20:34:00.0	2008-01-11 22:07:00.0	6	6	COIMBRA,BRA

1 2 3 4 5 6 7 8 9 10 11 of 13

Figure 17 – List of top issues ordered by number of tickets

We can therefore consider the issues containing more user reports (tickets) as being more reliable, i.e. there is a higher probability that these entries really correspond to a performance issue. We can assume this because these issues depend on the user perceived performance and, logically, users have different opinions of qualitative notions of the performance of an application. The characteristics of a problem in an application or the notion of what is a slow application are different for each user because they are influenced by their past experience. A transaction that, for some user, accustomed to slow network connections, may be considered normal, may be taking too much time from the perspective of users used to have more bandwidth available. Therefore, as many users report the problem, the idea of the existence of an issue becomes more credible.

The following analysis includes some statistics used to characterize the performance problems detected on the testbed network. Figure 18 represents the duration of the issues. The issues with duration zero are composed by only a single ticket instance. These results result of a single user reporting a problem and are normally the result of heavy load, misconfigurations or other problems on the user workstations. Since the users' workstations are not monitored at the time of this project, these issues are not appropriate to test this approach to root cause analysis.

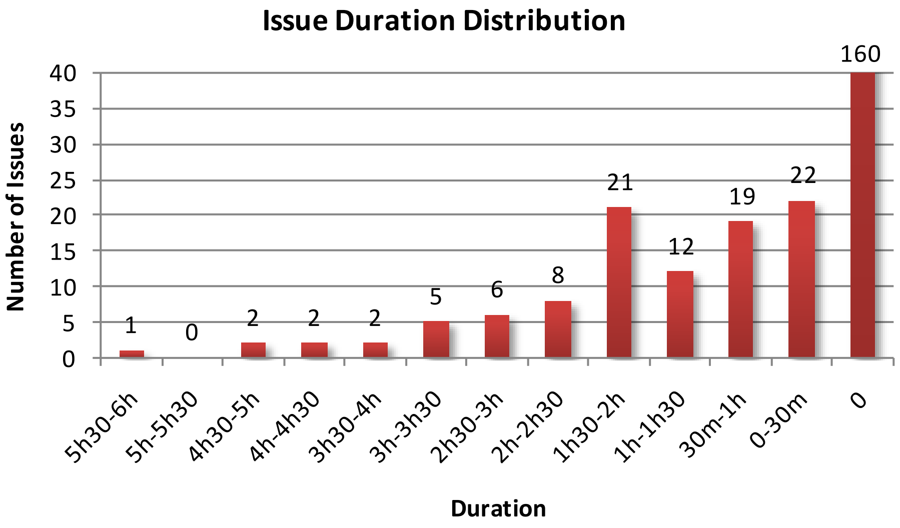


Figure 18 – Distribution of the Issues Duration

We observe that most of the remaining tickets may take from less of thirty minutes up to two hours. These are the most common issues. Finally, we have issues that last for four and even six hours. These are the issues that may affect the company's normal functioning and therefore indirectly affect the income and the image of the company. Also these are problems that must be solved quicker and to which a root cause analysis tool would be most helpful.

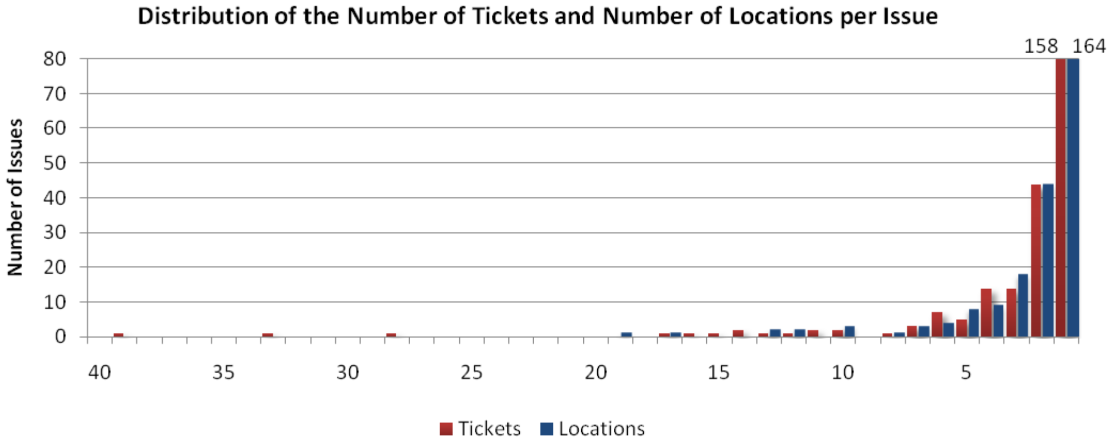


Figure 19 – Distribution of the number of tickets and locations per issue

Figure 19 presents the distribution of the number of tickets per issue. Once again the majority of issues have only one ticket that could not be correlated with any other ticket. The main observation is that the majority of issues affect only up to eight locations. But normally, this is enough to cover a large part of the network. Except in most rare situations where the entire network is affected, this corresponds to problems in the backend, i.e. in the main application or in some other one on which the main application depends. That is why it affects several distant users with different network conditions simultaneously. Therefore, we identify both generalized issues (high number of different locations) and localized issues (reduced number of locations).

Finally, we present in Figure 20 the hourly distribution of tickets. We can visualize the two typical humps that correspond to the peak work-hours. The highest peak happens between 9 and 10 o'clock. This is the hour when most people arrive at the call centers or distribution centers. These are the main sites containing end-users of the application used as testbed. Therefore, any issues already happening in the network, are only noticed when these users start trying to use the application.

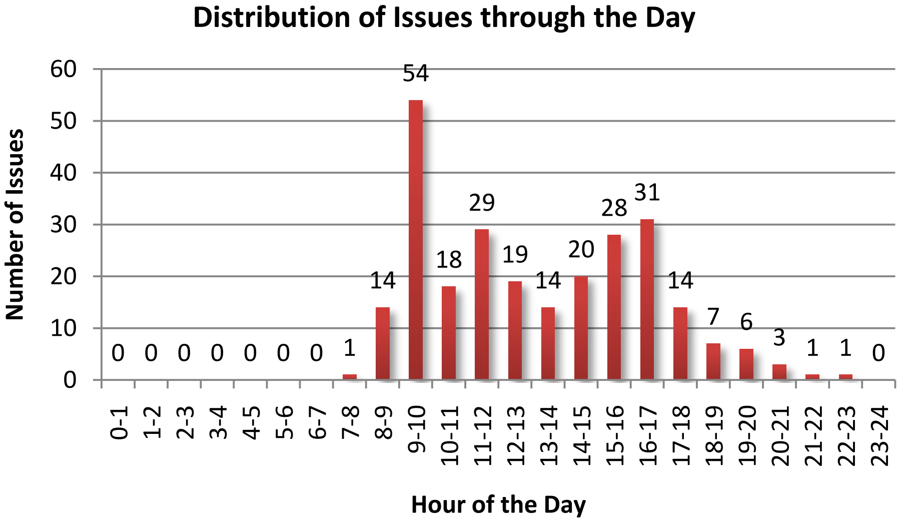


Figure 20 – Distribution of the issues start time through the hours of a day

This characterization of the issues identified on the network can probably give us some hints concerning the problems we may find in a root cause analysis. Most relevant problems (with more than one tickets opened) affect a large and diversified number of users. Issues affecting two distant locations (Lisbon and Porto) in a small country can immediately mean that a great part of the network is being used by the affected users. The hourly distribution of issues has a distribution similar to the number of users of the application. This indicates that many issues may be created due to the increase of the load in the system, which would naturally explain why most issues happen in the peak work hours.

4.3. Model Statistics

The network model is a graph whose root node is the issue chosen by the user. When the user clicks on an issue, the model is created as explained in Section 3.4 by using the attributes of the issue to find applications and links, and then by using the application attributes to find servers, and so on. All the issues identified during this project generate similar models, because all of them are related with the same application. Nevertheless, the model differs on the number of locations (user sites) affected. This has an impact on the number of application and network links. As an example, we show on Table 4 the statistics of the network model for the issue that affected most locations. As we can see we have only 1 issue represented in the model, but we are able to correlate it with 18 applications and 28 servers. With these statistics one can have an idea of how complex is the server-side application and of the numerous components with potential impact on the application's performance. The application links represent communications between user sites and servers and between servers. For that reason each of these application links represents a dependency on the network model. As we can see from the table, the number of application links is one 107, and therefore we can have an idea of the number of dependencies involved for a specific issue.

Component	Count
Issue	1
Applications	18
Servers	28
Application Links	107
Network Links	39
Total	**193**

Table 4 – Statistics of the network model for the first issue (id=9579) which affects 19 sites

4.4. Root Cause Listings

In this section we will provide an example based on the results obtained for the most relevant issue, i.e. the issue that affects the largest number of user sites. We will provide a sample explanation that illustrates how the results of the tool should be analyzed. Besides pointing the main components to uncover the cause of the problem, the application suggests the metrics that most probably help identifying what happened.

In the results description we will use generic names for the applications (App-1, App-2, App-3, etc.), for the servers (Server-i-j, for the server of the server j of the application i) and for the sites (Site-1, Site-2, Site-3, etc.). On Table 5 we show the top five root-cause components, identified by the Entity and Type columns. The approach taken to obtain these results was the independent analysis of components described in Section 3.7.1. The column Base Value represents the state internal state of the component while the column Value represents the impact of the component over the root issue, i.e. the internal state multiplied by the upward dependencies. The column causal start indicates the first element of the issue vector containing anomalies. This value can indicate if the node is a manifestation of the problem or a cause.

Value	Entity	Type	Base Value	Causal Start
0.0957	Server-1-1 (App-1)	Server	0.8793	1
0.0290	From: App-1, To: App-2	Application Link	0.5776	1
0.0156	From: App-1, Server-1-1, To: App-2, Server-2-1	Application Link	0.2414	4
0.0126	From: App-1, Server-1-1 , To: Site-1	Application Link	0.1466	57
0.0118	From: App-1, Server-1-1, To: Site-2	Application Link	0.1121	51

Table 5 – Results of the independent analysis of components for the first issue (id = 9579)

Analyzing these top five components we can take some interesting conclusions. The central server, normally the most obvious element to consider seems to be the prime suspect, and the first position on this rank indicates it. But the tool alerts to a relevant relation between the central application and a second one. The problem is probably caused inside the central server and due to some aspect on the processing related to the second application. The fourth and fifth entries are links to user sites where, as we can see in the causal start index, the problems have started some time later. Therefore, they reflect the impact on the communication between the user's client application and the application's servers.

Table 6 represents the values obtained for the causal path lookup approach (see Section 3.7.2). The main difference for the results obtained using the first approach (see Table 5) is that the application links from user sites to applications are no longer on the top positions of the causes' ranking. In this approach the communication between the central application (App-1) and two other applications (here represented by App-2 and App-3) are now present. These components have been underestimated on the previous approach due to their distance to the root node.

Causal Path Probability	Path
0,1965	[0.1965] *Issue*: Issue-app1-9579
	0,5
	[0.2968] *Application*:App-1
	0,2176
	[0.8793] *Server*: Server-1-1
0,1246	[0.1246] *Issue*: Issue-app1-9579
	0,5
	[0.1547] *Application*: App-1
	0,1005
	[0.5776] *Application Link*: From: App-1, To: App-2
0,0953	[0.0953] *Issue*: Issue-app1-9579
	0,5
	[0.0967] *Application*: App-1
	0,2176
	[0.2261] *Server*: Server-1-1
	0,5955
	[0.2414] *Application Link*: From: App-1, Server-1-1, To: App-2, Server-2-1
0,0794	[0.0794] *Issue*: Issue-app1-9579
	0,5
	[0.0654] *Application*: App-1
	0,2176
	[0.0824] *Server*: Server-1-1
	0,2999
	[0.2328] *Application Link*: From: App-1, Server-1-1, To: App-3, Server-3-1
0,0732	[0.0732] *Issue*: Issue-app1-9579
	0,5
	[0.0530] *Application*: App-1
	0,0506
	[0.2328] *Application Link*: From: App-1, To: App-3

Table 6 – Results of the causal path lookup for the first issue (id = 9579)

Also, we should notice that the values of the causal path probability are much closer than the values obtain for the first approach. Once again, the first approach overestimates the values obtained for the main server of the central application (App-1). As this application's servers tend to be clearly affected whenever a problem occurs and is near the root issue, the analysis would be biased towards considering these servers as root causes. The presence of the application links between the central application and two other applications lead us to conclude, that these applications may have been affected by the issue.

Let us now take a look at the list of metrics most relevant to explain the performance issue. Table 7 presents the top ranked.

Value	Entity	Type	Metric	Base Value	Causal Start
0.0706	Server-1-1 (App1)	Server	oracle_wait_event	0.6486	7
0.0647	Server-1-1 (App1)	Server	oracle_statistic	0.5946	1
0.0633	Server-1-1 (App1)	Server	swap	0.5818	1
0.0554	Server-1-1 (App1)	Server	active_processes	0.5091	1
0.0549	Server-1-1 (App1)	Server	cpu	0.5046	25

Table 7 – Top five of the most relevant metrics for the first issue (id=9579)

From the results obtained for the metrics we may conclude that the problem had its origin in the database server of App-1. The cause was most probably some slow query that triggered a high usage of the system resources. The database average wait time is the metric presenting the most anomalous behavior and leads us to conclude that all other queries were affected by the problem. The swap metric has also presented some anomalous values indicating a high consumption of memory. The CPU usage metric appears on the fifth position of this rank and, together with the swap value, suggests that the machine workload started to increase probably due to swap memory I/O operations.

Given this scenario, the first action to be performed by an administrator, would be to identify the query causing the problem for later analysis. In a future work version of the tool, this action can be automated by logging the list of queries that are consuming more resources. The query analysis could reveal some more information about why the query has had such an impact on the database. Some typical problems are incorrect use of condition clauses given the available table indexes, large amount of data returned by the query or poor query design (for instance, containing too many joins). In the future, the analysis to some of these problems could also be included in the tool by triggering an EXPLAIN SQL statement. This command provides some insight about the use of the table indexes and how many rows have to be scanned to execute the query. These problematic SQL queries are normally the result of insufficient testing, automatic query generation by applications, or poor programming skills.

The main action to recover the database on a short time could be to kill the query. But the problem may lead to a state where such action is not enough. For instance, if the server presents a high load it may take some time to recover, users may have tried to shutdown their client processes leaving “zombie” connections and queries on the server or they may have simply retried the same actions which would multiply the number of queries to be processed. Therefore, in some cases, an effective recovery can only be achieved by restarting the server.

Etymon mixes dependency between network elements with analysis of the behavior intra-host. Therefore, as happens in some of the related work, the analysis does not end when we identify a possible root host or server on the network. The relevant metrics are identified and graded given their deviations from the normal behavior in the recent past. Besides pointing out the most probable root cause, Etymon provides complete rankings of components and metrics, allowing a network operator to understand not only which component is failing, but also what other applications, servers, or network segments may be affected by the failure. This is important because a single metric or component normally is insufficient to explain what is happening in the network.

The temporal information also helps identifying some components as being affected by the problem. If we had developed the network model as a snapshot of the state of the network at a given instant, these components could be wrongly mistaken by possible root causes. This would introduce some noise in our analysis.

5 Future Work

The scope of a project about root cause analysis in such a large and complex network as the one used as a testbed is an open subject. The main goal of building a first iteration of a root-cause analysis tool was accomplished. The architecture presented in Section 3.1 was implemented using a simple version of each module. For each model we tried to implement some ideas, but several improvements were left behind due to the lack of time to experiment them. Therefore, this section has some relevance in this report because much has been left to try in the future.

Besides the improvements on each module, also the data gathered from the testbed network must be re-evaluated. With more time to interact with network and system administrators, several more metrics and data sources can be deployed. This will contribute to create a richer model for the network, not only in terms of diversity and number of components, but also in terms of what can be concluded from the metrics.

Also the network-level, i.e. routers, network protocols, are not analyzed. This reduces the possibilities of accurately identifying problems in the network. Therefore when we detect any anomaly in the network link components, we cannot confirm if any problem is detected underneath.

Some of the ideas to explore in future versions of the tool are:

- **Parameterization Removal**

  Some modules need human parameterization. For instance, when the network model is built some of the dependencies are conditioned from the start. The dependencies between a parent and its children are conditioned not only by the traffic analysis but also by some intuitive parameters that depend on the type of each component. These parameters have some influence in the final result and should be removed from the application. The study of how to organize the network model without these parameters was left to future work. Also, it was described before that some metrics could not be analyzed using the patterns described. By that reason, threshold violation was used. Naturally, this type of analysis is very dependent of who configures the data and, although the thresholds used were very simplistic and logical (e.g. 0% disk space, number of timeouts equal to or greater than 1, etc.) they should be replaced by some automatic method of evaluation.

- **Pattern Detection and Deviation Analysis Improvements**

  Another module than can be improved is the pattern detection. Several techniques are described on the literature to design patterns. One of such approaches is the ARIMA (Auto-Regressive Integrated Moving Average) model that can be used to forecast the behavior of a time series. This forecast can then be compared to the actual behavior to detect unexpected behavior. Furthermore, a small improvement can be done by applying some mechanism to automatically sanitize input data used in the patterns. For instance, one can make data smoother by removing some sporadic peaks that may have strong influence in the final prediction.

Other approach that can be used to complement pattern detection is an analysis of the distribution of a metric. One can use several tests for finding the best-fitting distribution for a specific dataset. Some of the available tests are Kolmogorov-Smirnov test, the Anderson-Darling test, the Shapiro-Wilk test and the Chi-Square goodness of fit test. After determining the best-fitting distribution for a specific metric some improvements can be made. For instance, it will be possible to optimize the values used in the algorithm to define the pattern limits, i.e. knowing the distribution we can choose a better value for multiplying by the standard deviation to obtain the upper and lower limits.

Also, to perform a generic analysis all metrics to which a pattern was applied were treated in the same way. The metrics were considered to have a seasonal pattern and therefore the pattern would depend on the time of day. For some metrics that seasonal approach is much conservative because the metric's behavior is similar throughout the entire day. Therefore, one could apply similar methods with different granularities.

- **Feedback to the Company**

One of the most important results of this project is the acquisition of know-how in what root-cause analysis is concerned. Throughout the development of Etymon, several obstacles were encountered that could not be overcome in the time available. Some of these obstacles arise from the fact that the testbed network used is very complex, with lack of information about some aspects of its architecture and that corresponds to a production environment, where every change can take some time to effectuate.

Therefore, one of the future steps will be to provide some feedback about the results obtained and difficulties encountered. The idea is to provide feedback about the monitoring and documentation process, in order to facilitate the deployment of this kind of applications (e.g. additional metrics to collect, important network locations to probe, metrics reorganization to facilitate correlation mechanisms).

One other important aspect to improve in future analysis is to close the gap between obtaining results for the network and confirming them using manual analysis. Therefore, each incident should be assessed and a root cause whenever possible identified manually. For this purpose, one can monitor for some period all the incidents that are occurring in real time using this tool and try to confirm the root causes manually with the help of network and system operators and/or administrators immediately after their occurrence.

- **Increase Model's Granularity**

The model was built using only elements already defined on the monitoring application, and there was no time left to extend the model to use other sources (besides the traffic analysis to compute the dependencies' strengths). With the increase of the material available the model can consider more specific components. For instance, instead of using a component named server, we can add more information about its functionality, i.e. specify web servers, application servers, database servers, etc. An initial idea of can be defined for a generic host is depicted in Figure 21. This figure is part from a more detailed ongoing study about the elements that a generic network model should contain. This diagram provides a small idea of the numerous elements to consider in such analysis and of some of the elements to include in the model in the near to medium term. It also displays possible root causes that can occur in each element.

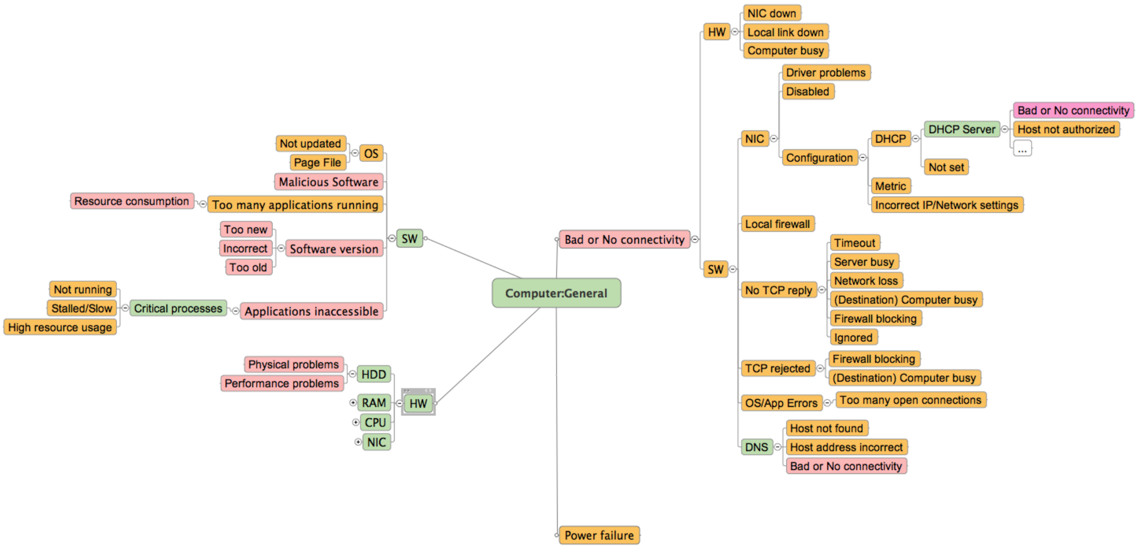


Figure 21 – Sketch of a detailed model of the network

With the increased complexity we also plan to develop templates to analyze each specific component. At the moment, metrics are used to identify anomalies. In the future, we should be able to value metrics differently and establish some correlations between them. Then for the behavior of each metric or set of metrics we want to associate root causes descriptions (i.e. a human understandable tag). The figure contains some of the root causes descriptions and metrics we can look for.

- **State Machines Module**

One interesting way of establishing causal relations is to define state machines for several aspects of the system and networks. For instance, one can try to identify workflows in an application and search ways to measure each state and/or transition. From this point on, we can easily detect failures by checking where some execution has failed or in which state most them are stopping. Also, a state machine was used to analyze the TCP protocol and can be applied to any other protocol in use in the network. The inclusion of a generic module for following and detecting errors in protocols and workflows is also one future step.

- **Network Simulation**

Another possible approach is to build a model of the network where all inputs can be controlled. The idea can be to simulate controlled failures with well known root causes, in order to see how the model reacts to them, instead of using unpredictable and noisy operation failures. The observations made in a controlled model can probably help us to understand the reactions of the model to the operation failures in the real network.

- **Add More Knowledge to Dependencies**

A future step will be to add some extra functionality to the module that calculates the dependencies strength, i.e. some knowledge about the function of each node. For instance the protocol or the port in use for communication allow us to identify the service in use and draw some conclusions about the node, and therefore to extrapolate something about the importance of the relations. Also some correlation may be made between end-users requests and communication between applications in order to understand which flows have more direct impact in the users' daily work. We could, for instance, value more online transactions than batch processing.

Also, we can make dependencies as a function of time. For instance, if a connection is only used periodically and if we are able to detect it we can increase the dependency only near the instant where the connection is suppose to occur.

Another idea to explore is the inclusion of not only direct dependencies (1st order dependencies) but also dependencies that depend on the recent past, i.e. 2nd and higher order dependencies. If we identify that some workflows (or sequence of transactions) are more common than others, than we can influence the dependencies strengths based on a sequence of transactions observed before.

- **Automatically Trigger Deeper Analyses**

As we have seen in Section 4.4, given a specific failure scenario, we can immediately trigger some analysis. In the future, with the inclusion of new metrics we will be able to achieve a finer granularity, which will enable some customized analysis for specific situations. Upon a detection of a failure, a network operator could also define the necessary actions to perform in a future similar situation. The actions could be saved and automatically executed once the same root-causes were identified.

- **Performance of the Tool**

The main focus during the development phase was functionality. We tried to implement all the functions necessary to conduct a root cause analysis end-to-end. Therefore, one issue that has been neglected was the performance of the tool. The application has been used mostly in the offline mode. In this mode, all data is retrieved on demand not only for the period of analysis but also for the pattern processing which includes much more data. This makes analyzing an issue a time consuming task, despite of being much faster than a manual analysis.

The performance of the tool in the offline mode is mostly influenced by the format of the database. Therefore, one of the solutions would be too reorganize the database so that the tables become smaller. In the present case, on data table contains information about an entire month and thus may contain up to 6 or 7 GB of data. Any query that tries to retrieve some weeks or even a few hours from such a table will be very slow, regardless of any optimization make to the table indexes.

Also, the application works in a most intuitive manner, i.e. as it goes through the model it retrieves the information for each metric separately. Although not as intuitive as this approach, retrieving all the information from the database in one step (and few larger queries) is much faster than doing several small queries. Thus, the code can be reorganized to cope with this change, and a significant performance improvement is expected.

These are some of the ideas already considered to implement in the near future. This area of research depends on being able to test the applications in real environments and on the information available about those environments. Therefore, depending on the effort placed in this research by interested parties (as large companies), one can explore even more demanding solutions.

6 Conclusion

In this project our proposal was to build a root-cause analysis tool end to end. Root cause analysis involve, as we have seen throughout this report, failure diagnosis (in this project was based on users' reports but could be done also by automatic metric analysis), creation of a network model (the components should be integrated as automatically as possible), determination of node state (by analyzing the metrics available for each component) and given the results of the other three main modules the determination of the root cause components and the indicative metrics.

Instead of focusing on one single component of such a system, we decided to build a tool end-to-end, i.e. to develop a module for each of the tasks described in the last chapter. This objective was accomplished. In the process, we acquired a deeper knowledge of the requirements of such a system and gathered some ideas of what should be done in the future to improve the results obtained. Another important decision was to focus on performance problems that are usually the most complex to solve and the most abstract.

For the identification of performance failures we use the reports of problems made by end-users. For companies today, the evaluation of this kind of problems is normally made by comparing the reports of the IT management teams with the number of problems identified by users. In the end, what really matters is if the users were able to perform their daily work. Therefore, these are the main issues we want to solve. Naturally we can also use this information about problems and correlate those with metrics' behavior and focus, in the future, on automatically detection of performance failures.

For creating the model we decided to use any information available on the deployed company tools to correlate metrics. The idea is that any element that is introduced on those applications is reflected in the model. To keep using automatic methods, we use traffic statistics to determine the dependencies among the components of the model.

The determination of the node state has followed the most practical approach for detecting events that are very difficult (if not impossible) to model. We focus on detecting anomalies given a baseline computed over the recent past of the issue. This approach fits well on these applications that keep a usual behavior during several weeks. The work load is normally very similar from one week to the other. The final results are obtained by checking how much the metrics deviate from the considered "normal" behavior.

The final results of the tool are obtained either by crawling the graph and search for the components with highest relation between their influence over the root node and their state, or by following the strongest path where sons influence their fathers. In our list of relevant components and metrics to decide on the root cause, we can see several different applications that have considerable impact on the central application.

Given the implemented features we focus on four of them that differ from the approaches taken in the related projects.

- **Creation of a model for a specific environment** – the issues identified immediately set a target application which is directly accessed by the users. The relations dependencies identified in this application define an environment for the network model. This prevents us from analyzing components that are not relevant to the behavior of the main application.
- **Differentiating correlation from causality** – we introduce a temporal analysis in order to check how a node can be considered as cause instead of being just a reflection of the performance problem. Thus we analyze the recent past of the issue, to see for how long has the component experiencing anomalies.
- **Use of dependencies over a recent period** – we also use dynamic dependencies computed by analyzing traffic behavior over a recent period at the time of the issue. This allows us to introduce in the model, on run time, the relevant flows that have occurred only on the recent past. This is done while maintaining a period of analysis long enough to calculate reliable statistics.
- **Identification of more than one root-cause element** – the final results of the tool identify not only the most relevant components but different metrics that can contribute to clarify what happened. As we have seen in Section 4.4, we can take some interesting conclusions not only from the top list of causal paths as from the top list of the metrics presenting major manifestations. Therefore, this solution presents something more than just the finger-pointing of the component (host or link) responsible for the problem. Sometimes the problem is complex and cannot be explained by the behavior of a sole element. The explanation may have to include the interactions from different components.

In the introduction we presented five main properties to characterize the requirements of a root-cause analysis application. Etymon is:

- **Usable** – the application includes a complete graphical user interface that allows the users to perform the most important actions of the tool. The user can trigger traffic analyses and see the results through the interface. The results are shown using detailed tables, easy to visualize charts and intuitive graphs that can help operators to become aware of features that, at a first glance, could go unnoticed. We added also an interface for visualizing the network model that allows navigation from node to node. The user can also see charts of the metrics that influence a node state, for the period of the issues that have been analyzed;
- **Automatic** – the methods used for the construction of the model are mostly automatic. The calculation of the nodes' state, the dependencies among components and the determination of the most affected elements are all made with minimal manual parameterization. The only exception was the definition of the generic model, where generic components are related to each other, but the mapping of this initial (and very simple) model to the production network is made automatically;
- **Adaptable** – the application loads all the components defined on the monitoring application and all the low level metrics that can be used to characterize the component behavior. Consequently, all new components introduced in the monitoring application are reflected on the model;
- **Granular** – the model is generic and is as generic as the definition of the initial model using abstract components. The definition of new components and new relations on this model will allow to easily increase the granularity of the model;

- **Accurate** – the application clear identifies the most affected components. Besides the central server the application focus the attention of its users on other server that show signs of being affecting the overall performance. Given the absence of explanations for the issues analyzed in this tool we were not able to confirm how close the results obtained were from the real root cause. This is, perhaps, the most important task to perform in the future.
- **Scalability** - the online mode provides the ability to process node states in real time. This mode uses on-going descriptive statistics objects that can be updated once an event arrives. Therefore, it allows the quick identification of root causes for recent issues. These network model states in the moment the issue is closed can be saved for latter analysis in the offline mode. This reduces the time necessary to analyze past issues. Also, the network model is created for a specific environment, i.e. includes only the elements that directly or indirectly interact with the main application. This reduces the number of elements to be analyzed and consequently increases the efficiency of the tool.

Etymon looks at the network as one should be looking to any distributed system. All elements should be considered as any of them can influence the behavior major servers. In the end of day, the companies do not even care why and how a failure occurred. Their only desire is to avoid failures and, if they occur, to solve them fast. Whoever analyses the problem, will need tools like Etymon to be able to quickly identify the root-cause of the failure.

7 Bibliography

[1] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Kyoto, Japan, 2007, pp. 13-24.

[2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," in *Proceedings of the19th ACM Symposium on Operating Systems Principles*, Boston Landing, NY, USA, 2003, pp. 74-89.

[3] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewe, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," in *Proceedings of the International Conference on Dependable Systems and Networks*, Florence, Italy, 2002, p. 595–604.

[4] J. L. Hellerstein, M. Maccabee, W. N. Mills, and J. J. Turek, "ETE: A Customizable Approach to Measuring End-to-End Response Times and Their Components in Distributed Systems," in *19nth IEEE International Conference on Distributed Computing Systems*, Austin, TX, USA, 1999, pp. 152-162.

[5] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, "The NetLogger methodology for high performance distributed systems performance analysis," in *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*, Chicago, IL, USA, 1998, pp. 260-267.

[6] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for Request extraction and Workload Modelling," in *6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, USA, 2004, pp. 259-272.

[7] P. Jackson, *Introduction to Expert Systems*, 3rd ed. Addison-Wesley, 1998.

[8] J. A. Alegria, T. Carvalho, and R. Ramalho, "Uma Experiência Open Source para "Tomar o Pulso" e "Ter Pulso" sobre a Função Sistemas e Tecnologias de Informação," in *5th CAPSI*, Lisboa, 2004.

[9] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.

[10] EsperTech. Esper. [Online]. http://esper.codehaus.org

[11] M. Bednarczyk. jNetStream OpenSource, Protocol Analyzer and Decoder SDK. [Online]. http://jnetstream.com

[12] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Professional Computing Series, 1994.