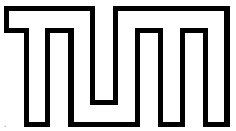


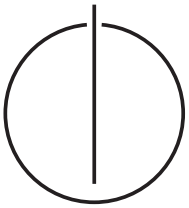
Technische Universität München

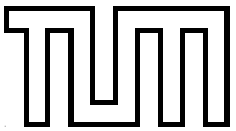
Faculty of Informatics

Diploma thesis in Informatics

Configuration Management via frequency analysis of synthetic load fluctuations

Johann Schlamp





Technische Universität München

Faculty of Informatics

Diploma thesis in Informatics

Configuration Management via frequency analysis of synthetic load fluctuations

Konfigurationsmanagement durch Frequenzanalyse künstlicher Lastfluktuationen

Originator:	Johann Schlamp
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dipl.-Inform. Marc Fouquet Dipl.-Inform. Heiko Niedermayer
Submission Date:	August 14th 2009

Statement

I assure the single handed composition of this diploma thesis only supported by declared resources.

Munich, August 14th 2009

. . . . . .
(Signature of candidate)

Abstract

The prime purpose of IT services in larger enterprise environments is to support and improve business processes. For providing a certain level of quality, terms like availability and reliability directly apply to the underlying IT infrastructure. Different kinds of problems have to be addressed quickly, which usually requires an exhaustive knowledge of the configuration of the IT landscape. Particularly in larger enterprises often no centralized IT concept exists or relevant information is lost over time due to longterm growth.

The subject of this thesis is a logical continuation of current asset management approaches, which simply deal with the inventory of networks. Current techniques inherently lack the ability of detecting links and dependencies. But for performing efficient problem handling on a large scale, and also for supporting strategic IT decisions, the knowledge of dependencies throughout the network is indispensable. The method to be introduced focuses on this dependency discovery and is mainly based on the generation of load variations on selected systems. These synthetic fluctuations should in principle be also detectable on all depending systems. Methods of frequency analysis as well as the use of various load and timing services will serve to optimize the results and to extend the scope of deployment.

Zusammenfassung

Das wesentliche Ziel von IT-Dienstleistungen in großen Unternehmen ist die Unterstützung und Verbesserung von Geschäftsprozessen. Für die Sicherstellung eines gewissen Maßes an Qualität müssen dazu Vorgaben wie Verfügbarkeit und Zuverlässigkeit direkt auf der zugrunde liegenden IT-Infrastruktur adressiert werden. Ständig können ganz unterschiedliche Arten von Problemen auftreten, für deren Behebung in der Regel ein fundiertes Wissen über die Konfiguration der IT-Landschaft notwendig ist. Gerade in größeren Unternehmen ist jedoch oft kein zentrales IT-Konzept vorhanden oder entsprechende Informationen sind durch langjähriges Wachstum verloren gegangen.

Gegenstand dieser Arbeit ist eine konsequente Fortsetzung gängiger Asset Management Ansätze, die sich nicht nur auf die Inventarisierung von Netzen beschränkt. Diese Methoden verfügen prinzipiell über keine Möglichkeit zur Erkennung von Verbindungen und Abhängigkeiten. Dabei ist gerade für effizientes Problemlösen, aber auch zur Unterstützung strategischer IT-Entscheidungen das Wissen über Abhängigkeiten im gesamten Netz unerlässlich. Die hier vorgestellte Verfahrensweise konzentriert sich auf die Erkennung dieser Abhängigkeiten und beruht maßgeblich auf der Erzeugung von Laständerungen an ausgewählten Systemen. Derartige künstliche Fluktuationen sollten grundsätzlich bei allen in Abhängigkeit stehenden Systemen ebenso nachweisbar sein. Methoden der Frequenzanalyse sowie die Verwendung verschiedener Last- und Messdienste dienen zur Optimierung der Ergebnisse und zur Erweiterung des Einsatzgebietes.

Contents

Contents

Contents

Chapter 1

1 Introduction

"Information is the resolution of uncertainty."

Claude E. Shannon

Communication as described in classic information theory terms is typically based on a sender recipient relationship. Already in 1948, in his famous paper *"A Mathematical Theory of Communication"*, Claude E. Shannon basically described a communication system by a transmitter, channel and receiver [1, p. 2]. While this paradigm is successful for standard telecommunication engineering, which means the transportation of a specific message from one point to another, it lacks a vital part of modern communication: dynamics. In general, information today is a compound of different pieces of information gathered from a variety of sources. This compilation of information fragments is often supported by particular services, which may again rely on each other on a higher level.

In large scale computer networks, vast amounts of information correlate in surprisingly complex ways. A single query can disperse to a dozen systems, and there again recursively spread out, resulting in an exponential increase of complexity. Such compounds and dependencies can originate from technical implementations like the domain name system or the use of proxy servers, but also deliberately separated data is common practice in respect of physical security, privacy, confidentiality or efficiency. While the aggregation of different information sources is a challenging task for itself, the detection of dependencies in computer networks is still to be considered in its infancy.

This work focuses on the automated discovery of dependencies in networked systems, introducing an innovative approach, providing a ready-to-use prototype and evaluating it through a distinct series of experiments.

1.1 Goal and approach

Dependency discovery is hardly an end in itself. Although it can directly support and improve operational processes by providing additional information on particularly important systems, there is a much higher business value to it when combined with the implementation of certain guidelines for IT service management. The most popular advance in this area is the IT Infrastructure Library© (ITIL©), a collection of best practices released by the UK Office of Government Commerce [2]. While providing a variety of processes for decision making in IT environments, the process of configuration management is the vital part virtually connected to all other disciplines. Figure 1.1 illustrates its central position.

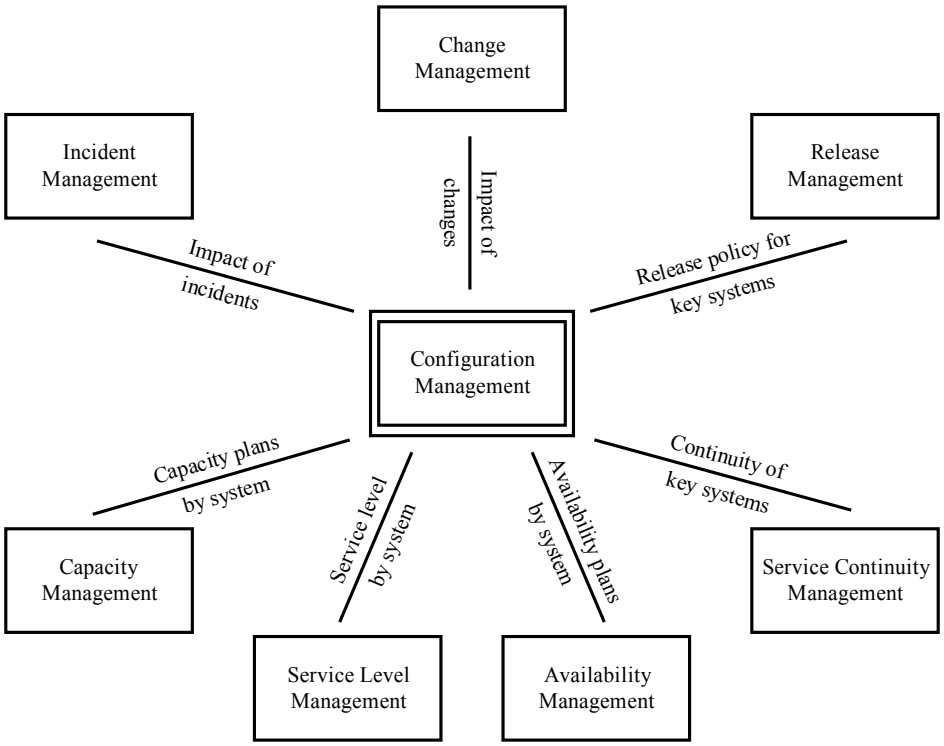


Figure 1.1: Configuration management in ITIL© [3, p. 7]

Configuration management according to ITIL© [4, p. 194] can be broken down into three domains. First, an extensive asset management serves as the information base for all configuration knowledge. The second area is about discovering dependencies between configuration items themselves whereas the third part addresses the

storage and handling of this fairly huge amount of information. This is done in a so-called Configuration Management Database (CMDB).

The intention of this thesis is not the implementation of configuration management as a whole, but the delivery of a new approach for the hardest part of it. While there are suitable solutions for handling assets, and also for storing this knowledge along with additional information in a CMDB, the process of discovering dependencies is still a challenging task. Before giving an overview of related work and different advances in the next chapter, this thesis's approach is being explained.

In principle, previous knowledge of available network services and corresponding systems is assumed, potentially obtained from already implemented asset management approaches. The basic idea for the detection of dependencies between these assets is thereby provocation of load. For that, additional workload is generated by utilizing synthetic service queries. Geared towards a specific service, the load will spread out to all dependent systems and then be externally verified on-site (see Fig. 1.2). A particular interesting point of this approach is that no assumptions are made to the dependency's underlying hard- or software, and especially not to the network topology. It is (by theory) autonomously deployable as it simply follows and amplifies already established and authorized ways of communication.

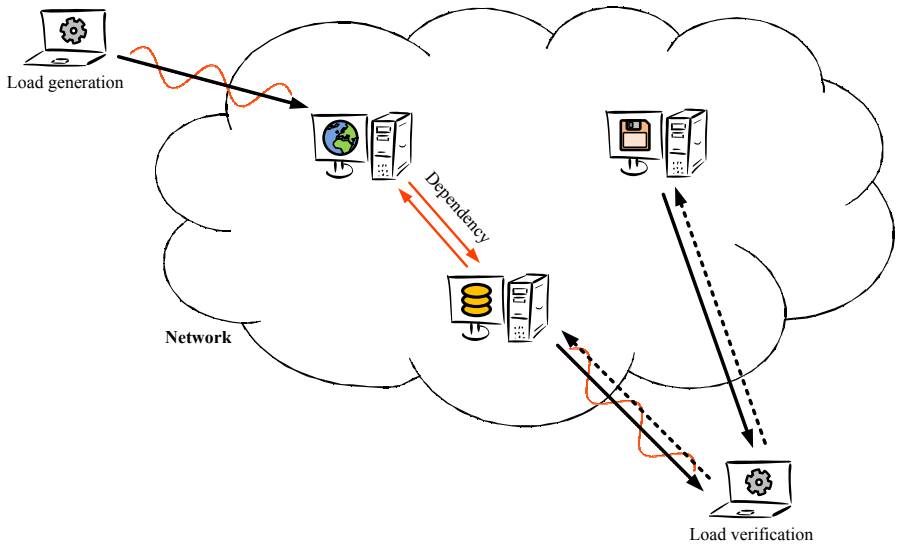


Figure 1.2: Schematic approach

Furthermore, to reduce local and network workloads and more important to improve the significance of results, a more sophisticated approach will be pursued. The generated load doesn't have to be static, in fact a fluctuating pattern is much more promising. With methods of signal processing, specifically through filtering and frequency analysis, the number of false positives and false negatives will decrease dramatically. As a matter of fact, this approach is de facto independent of any randomness inherently present on distributed system. In addition this method will satisfy the following constraints, which are most important for a suitable and reliable dependency discovery technique for large scale networks.

1. Peer-less design. A suitable method for mapping huge parts of networks (presumably spread out to different sites) cannot be agent-based or requiring direct access to specific systems. The administrative and financial overhead would render it useless for most scenarios.

2. Selective usage. While it is certainly useful to have an exhaustive company-wide dependency map, there are situations where a quick and immediate scan of particular systems will be crucial. Such a selective targeting is clearly an important (but also rare) quality of dependency discovery approaches.

3. Automatable processing. Keeping track of local peculiarities at division scale or below usually relies on the exploitation of human expert knowledge. Deploying a technique which is independent from such specialists, but on the other hand requires expert opinion to interprete the results simply relocates the problem. A reliable automatic detection of dependencies or at least a qualitative rating of results has to be considered.

4. Deterministic base. In principle, distributed applications as well as the underlying hardware and network infrastructure involve a certain level of randomness. Furthermore, dependencies occur randomly (as user demands are random) and usually are not persistent or predictable in any regard. Provoking the dependency directly at examination time subsidiarily with a specific pattern will eliminate the randomness by orders of magnitude.

Actively interfering with productive environments is a sensitive task and needs careful examination of possible side-effects. Section 6.4.1 addresses this concern. In addition, initiating a certain dependency requires particular knowledge of the systems to be scanned. Sections 3.1 and 4.1 focus on rather generic deployment strategies.

1.2 Outline

Following to this brief introduction, other approaches are discussed in Chp. 2. This includes common practices and commercial products for configuration management as well as active research on the topic of dependency discovery. Chp. 3 will afterwards provide the basics for this work. First of all, an overview of different dependency relations is made, followed by some mathmatical fundamentals on signal processing concluded with considerations on measuring the quality of anticipated results. Chapter 4 characterizes a working prototype, including use cases and requirements as well as the architectural design with essential components. The corresponding implementation and its main features, plus technical details like the use of external libraries, execution procedures and not less important deployment strategies are explained in Chp. 5. Eventually, a set of distinct experiments gradually advancing in complexity along with corresponding conclusions is given in Chp. 6, which culminates in large series of analyses for further investigation of underlying interrelations. Finally, Chp. 7 recapitulates the outcome of this work and provides an outlook for future developments.

1 Introduction

Chapter 2

2 Related work

"Don't worry about people stealing an idea. If it's original, you will have to ram it down their throats."

Howard H. Aiken

The inspiration for this work arose from a diploma thesis on signal-based dependency discovery for networked services by Christoph Probst [5]. That thesis focused on the detection of changes in response times on manually stressed services and their dependencies. In contrast to the objective of this work, the main goal was basically to examine constant load propagation verified in the timings of simple ping messages. A more practical approach to dependency discovery, including studies on common technical constellations and different use cases with regard to configuration management is developed within this thesis. Furthermore, an unambiguous and automatable detection method is described by the utilization of wavelike load patterns and the full exploitation of means of frequency analysis.

In the following, other ideas approaching the subject from a different angle are presented. At first, common practices followed by proprietary approaches to configuration management including an overview of their advantages and peculiarities are considered. Afterwards, recent advances in academic research on the matter of dependency discovery are discussed and compared to the work at hand.

2.1 Common practice

Managing system configurations and keeping track of interrelations today is most often organized by the administrator in charge. In the simplest case this is achieved with the help of ordinary work sheets, requiring elaborate coordination efforts in large environments where several administrators may be appointed as well. Administering the list of configuration items and especially identifying dependencies among them thereby relys on the examination of relevant system configuration files. This complex process can be supported by automatically mining configuration data, as described in [6]. For that, specific application information already contained in the operating system (e.g. package manager) are examined, where for example installed client software implies a dependency to the corresponding server counterpart. In addition, particular configuration files are analyzed to obtain further information (e.g. NFS clients know from which server a file system needs to be mounted). Although this approach succeeds in detecting technical dependencies with proper configuration data available, it lacks the possibility of identifying dynamic dependencies like a webserver requesting information from a distant database. Besides, it assumes unlimited access to the systems in question which is in many cases not available.

Another improvement for the situation of manual configuration management is accomplished with [7]. This solution offers a standardized and centralized interface for administrating dependency information. Although it does not deal with the discovery itself but only with its handling, it is mentioned here as it could be very well augmented with this thesis's technique. More precisely, the application holds a database of known dependencies and allows – by making use of different queries and filters – the output of consolidated dependency graphs. The input of information on dependencies is thereby arranged by an open XML format [8] and thus can originate from arbitrary sources. As an extension to this thesis, the developed reporting mechanism could be easily adapted to implement this protocol.

2.2 Commercial tools

With configuration management being fully recognized as an essential and valuable process for enterprises, there is also a growing market for commercial solutions in the area of IT management software. Proprietary tools are available from different well-known vendors like *HP OpenView* [9], *Microsoft Operations Manager* [10], *Novell ZENworks* [11], *Red Hat Network* [12] or *IBM Tivoli* [13]. Although these tools offer a great capacity for management and support and do a good job on the whole, they

reveal serious weaknesses in the details. The modules for asset and configuration management most often rely on homogenous networks (built up from vendor-own products), depend on several (also vendor-specific) third-party tools or simply implement this functionality only rudimentarily. Moreover they are agent-based in the majority of cases, clumsy with regard to operational use and last but not least high in price. Nevertheless these commercial solutions do serve a purpose, and could most certainly be augmented with functionality derived from this work.

2.3 Current research

The previously mentioned practices for configuration management heavily depend on user interaction or on specialized proprietary software. However, there are many approaches for automating dependency discovery in a more generic way. In the following, different techniques from current research projects are presented. As a remark, especially *Microsoft*'s research division is driving further developments.

There are a few similar approaches which cover the same topic, each of them operating in a passive manner and with particular shortcomings compared to this thesis's technique of active dependency discovery. First, there are applications based on inference using packet timings and machine learning techniques. One representative of this kind is *Constellation* [14]. This system infers the configuration of individual machines from observing the relationship between input and output traffic, and then synthesizes this information to provide a global view. Ideally every network host should capture its own traffic to a local database with a dedicated agent providing an interface for search queries. However, it is possible to deploy a single sensor per subnet working with larger packet traces as well. The centralized analysis then uses statistical methods for inspecting packet timings, where sophisticated algorithms allow a guaranteed confidence level. In contrast to that, *Sherlock* [15] gathers dependencies with the help of a network-wide inference graph (also based on local or recorded packet inspection) from another point of view. Every agent is thereby responsible for detecting dependencies between the systems a certain host is accessing. If for instance all requests to a webserver from a specific host are preceded with the communication to a DNS server (within a certain time window), the probability of the webserver being dependent on DNS is raised. Obviously this technique focuses on obtaining information about indirect dependencies maintaining the network's operational workflows rather than detecting structural dependencies as such. Both techniques rely on probabilistic methods with particular parameters defining certain thresholds and are thus more or less fragile to timing variations resulting

from server and network load. Furthermore, infrequently triggered dependencies are only detectable within large periods of scanning or even remain undetectable due to small probability rates.

Beyond that, there are tools which analyze the network traffic as a whole by identifying certain packet flows. *Orion* [16] discovers typical spikes in the time delay distribution of dependent flows while *eXpose* [17] detects cooccurrences of flow pairs with frequencies significantly greater than chance. Both of them have a peer-less and passive design which depends on huge amounts of real network activity for drawing substantive conclusions and are thus as time-consuming as the approaches mentioned above. In addition, the heavy use of probabilistic algorithms in *Orion* reveals the same disadvantages associated with inherent randomness. For *eXpose* the statistical means are reduced to a minimum, however it still depends on events occuring with a certain frequency.

All considered techniques provide a global map for dependencies, lacking the flexibility for direct operational use like quick analyses of relationships between single services. Finally, none of these approaches follows deterministic principles, which categorically leads to false positives and false negatives. However, the use of advanced statistical algorithms reducing these side-effects is still an active subject of research.

For modern *service-oriented architectures* (SOA), another approach is described in [18]. Common SOA implementations are based on a standardized way of transporting information, where the communication between services is consequently carried out by the exchange of well-defined messages. With access to the full log of sent messages (which is often part of a SOA implementation as well), cascades of communication parts with identical start and end points can be detected, infering a dependency structure of all engaged systems. Although these closed circles of message "hops" may seem to have causal origin, there is still a chance of random occurences. Reducing that uncertainty is once again achieved by statistical methods relying on specific parameters like maximum allowed time between different hops. Nevertheless, for environments implementing the SOA concept, this approach may be a reasonable choice.

The apparently only approach actively interfering with the network environment and thus being related most closely to this thesis is described in [19]. This methodology relies on active perturbation of systems to be analyzed for dependencies and therefore eliminates the disadvantages of passive techniques as described above. By applying an additional (constant) workload to a certain dependency, increasing response times for the dependent system are assumed. Although the approach proved

to be successful in a limited scale, it unfolds some major disadvantages. The basic idea virtually obeys causality, but for the afterward analysis statistical methods are still in use for determining verification success and its quality. Besides, direct access to the dependency is necessary for carrying out the perturbation. To some extent however, this thesis can be considered as further development based on the same principal idea of active dependency discovery, with introducing many advantages and also being closely verified through distinct series of experiments.

To conclude this chapter, the approach developed with this thesis is compared to the related work presented above. Basically it replaces all raised disadvantages and shortcomings in exchange for a technique actively interfering with productive environments. The application of frequency analysis thereby fully supports deterministic decisions and allows the use of qualitative metrics measuring the detection significance. This proceeding along with very short scanning periods as well as elaborate deployment strategies reduce the necessary load to a reasonable amount. With being flexible enough for a variety of use cases and a solid evaluation background for the basic functionality, the technique is well-established for further studies and future extensions. Finally, it could be combined with other approaches mentioned above. Especially methods for creating network-wide dependency maps in conjunction with active discovery shows great promise for improvements.

2 Related work

Chapter 3

3 Basics

"Basic research is what I am doing when I don't know what I am doing."

Wernher von Braun

Understanding the potential and also the limitations of this thesis requires a certain knowledge base. This involves three elementary domains, covered throughout the next sections.

At first a basic terminology will be deduced from different operational setups. These terms and definitions will serve to clarify the differences between certain dependency relations and to form an abstract concept for further analysis techniques. Second, some mathematical fundamentals on signal processing will be provided. The main focus is thereby on the filtering of raw experiment data for noise reduction and a certain flavour of *Fourier analysis*[1] for detecting single frequency parts. Nevertheless, a basic knowledge of analytical methods is assumed and mathematical proofs and deductions are abandoned. At last, approaches for measuring the quality of frequency identification will be examined. The main goal is on the one hand the evaluation of result significance through spectrum analysis and on the other hand the provision of a base for automated dependency discovery.

[1] Jean Baptiste Joseph Fourier (1768 – 1830)

3.1 Dependency relations

As *"dependency"* is a wide-ranging term a more explicit terminology aimed at the requirements of this work is introduced in the following. In general, IT services can depend on other services or provide resources themselves. Furthermore, a specific service can be directly dependent on another in a persistent way or contrarily request information on a sporadic basis. And of course there are also dependencies which then again depend on further services. These facts account for the necessity of terms reflecting the dependency's direction as well as its quality. All emerging concepts are illustrated on the basis of exemplary network setups.

3.1.1 *Association*

A specific service can have an *Association*. This term basically reflects the primal meaning of *"dependency"*, i.e. from the target's point of view a connected system which provides essential information. However, there are four different types of *Associations*, as shown in Fig. 3.1.

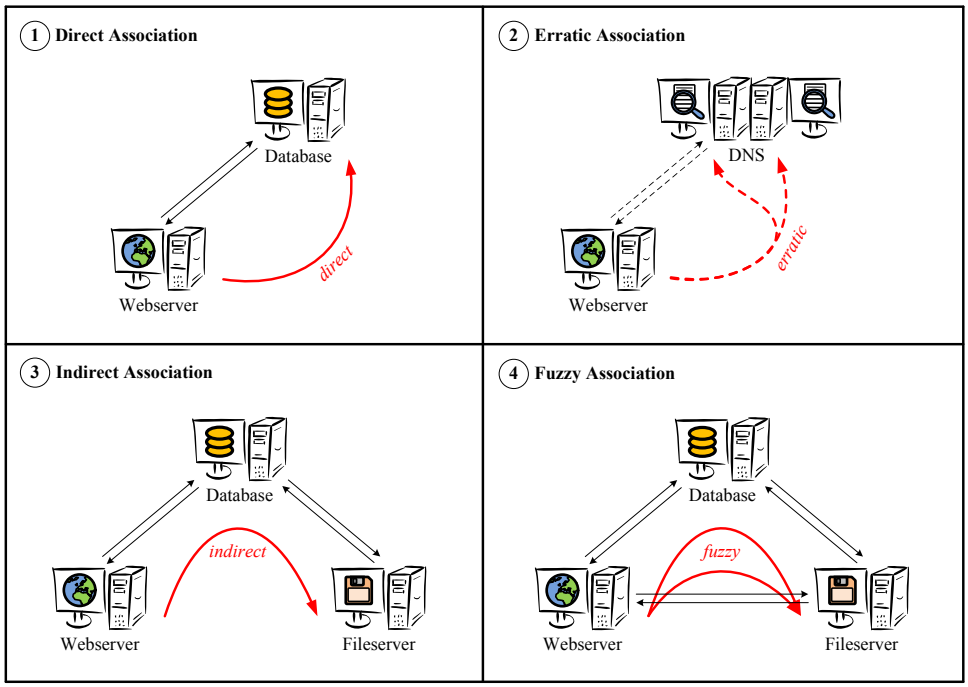


Figure 3.1: Types of *Association*

① **Direct Association** is the most general dependency. One system obtains information from another through a direct connection. This type will allow to generally proof the operability of the load based dependency discovery approach and additionally provide a basis for studies on load and timing parameters as well as on the influence of various other variables. Most of the work will be executed on this setup.

② **Erratic Association** covers the fact that some dependencies – even when actively provoked – only occur infrequently. This results for instance from caching mechanisms on the targeted machine itself and is thus hard to deal with. Variable ways of communication like redundant setups are also counted among this sort of *Associations*. Further studies on this kind of relations are ommitted, but should be considered for future work. Caching processes on dependent systems on the other hand are manageable and will be addressed in Sect. 6.2.2.

③ **Indirect Association** is in principal hard to detect. Information is gathered over multiple "hops" which is generally invisible on the targeted system. Breaking down the detection to single links may not lead to success either because the dependency is possibly engaged only on access of a specific system prior to the analyzed relation. Nevertheless, it is imaginable for the load to overleap several links in the chain and to be verifiable at the end. Section 6.3.4 gives more detail. A closed circle of successive dependencies is arguably unlikely, but further discussed in Sect. 3.1.3.

④ **Fuzzy Association** is similar to *Indirect Associations*, with the difference that the analyzed system also directly communicates to a distant machine in the dependency chain. This would probably lead to frequency overlays of two identical signals, which will be simulated in Sect. 6.3.2.

Detecting such *Associations* is straightforward. The component for load generation is directed towards a specific service and constantly querying it (with a well-known pattern). Simultaneously, the timing component (or potentially multiple instances of it) measures response times of possibly affected systems (in case of doubt of *all* systems known to deliver services). If the pattern can be verified, the *Association* is confirmed. For recognizing advanced dependency types like numbers two to four in the table above, more insight than frequency verification alone is probably necessary (e.g. disrupting a single link for recognizing *Indirect Associations*). Corresponding use cases for detecting and exploiting *Associations* are explained in Sect. 4.1.

3.1.2 *Affiliation*

In some cases the target for dependency discovery is a secondary system not being dependent on other services, but providing information for them. These parent services are then called *Affiliations*. Basically, this term describes the inverse direction of *Associations* (with the difference that the *Affiliation* isn't required for running operations on the system in question), but it educes some slightly different types. Figure 3.2 gives an overview.

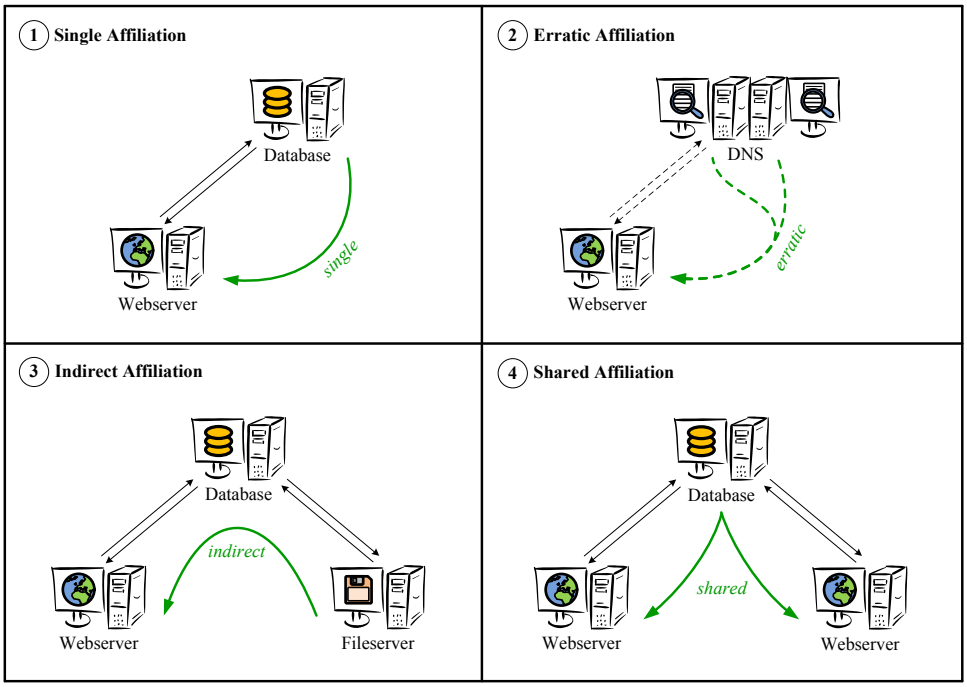


Figure 3.2: Types of *Affiliation*

(1) **Single Affiliation** on its face resembles *Direct Associations*. However, this term differs as it implies a single parent system only. This actually makes a difference here because detecting multiple superordinate systems needs special considerations whereas the detection of multiple *Associations* is straightforward. Type four accounts for this discrepancy.

(2) **Erratic Affiliation** reflects exactly the same type of *Association*. It describes a system which sporadically requests information, either because of a redundant service structure or a caching-capable service making use of the dependency from time to time only. Again, this type of relation is put aside for future work.

③ **Indirect Affiliation** defines the relation to a distant system in a dependency chain. Although it is recognized as an extra type, it is also a subset of *Shared Affiliations*.

④ **Shared Affiliation** covers the fact that a single service can provide information to a variety of other systems. This is just like *Single Affiliation* a common scenario, but compared to multiple *Direct Associations* there is an important difference. Parallel execution of different analyses would lead to (not necessarily identical) frequency overlays which has to be considered accordingly. Sections 6.3.2 and 6.3.3 provide further studies.

The identification of *Affiliations* is not as obvious as for *Associations*. In general, the detection works along regular paths of information exchange. According to that, a dependency cannot be provoked in an *Affiliation's* direction as it provides information only on request, which is the other way around. This means detecting *Affiliations* is nevertheless based on engaging the dependency on the *Affiliation* itself. However, it should theoretically be possible to gear the load pattern towards the dependency and measure the response times (likewise provoking the dependency) on the parent system. This would allow a more specific selection of the systems to be under load. Section 4.1 extends these considerations.

3.1.3 *Coalition*

Another sort of dependency relation is the so-called *Coalition*. With the evolving concept of cloud computing, which basically expresses a large pool of virtualized resources (such as hardware or services) [20], a specific service is no more bound to a dedicated system. Carrying it to extremes, all relevant services could be running in a cloud with interchanging hardware or also on a single mainframe hosting all of them concurrently. Regarding the first case, the absence of a determined communication structure could possibly lead to difficulties with the current approach. In the second case, frequency overlays and maybe also interference would probably occur if network interfaces aren't used mutually exclusive. According simulations can be found in Sect. 6.3.3. In spite of all possible problems, virtualization-based scenarios could certainly yield suitable results and should be addressed in future work.

Besides, another type of dependency relation might occur (Fig. 3.3). Usually, two services won't be interdependent as it could lead to deadlocks or infinite loops. But with multiple services running on a single system, two server machines could certainly be reciprocally exchanging information and would thus form a *Coalition*.

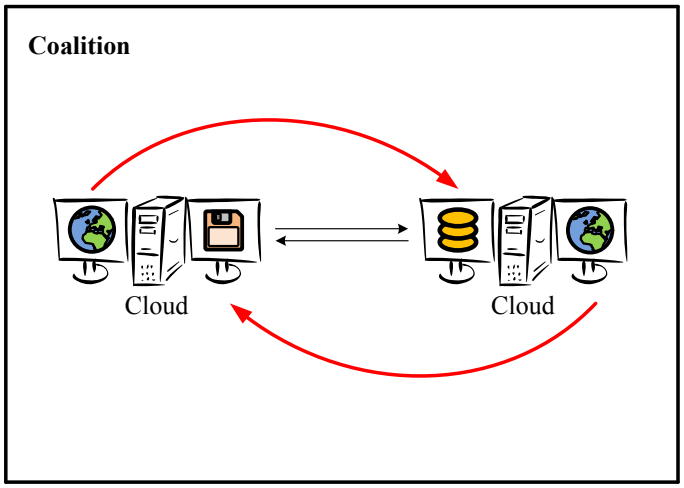


Figure 3.3: *Coalition*

Provoking one dependency doesn't generally influence another, but simultaneously analyzing multiple relations could again lead to frequency overlays, examined in Sect. 6.3.3.

3.2 Signal processing

Detecting frequency parts in a specific signal is divided into two steps. At first, a filter is applied to reduce random outliers and to clean the signal for further analysis. Afterwards it is transformed into the frequency space allowing a precise examination of all occuring frequency parts. The following subsections provide the basics.

3.2.1 Median filter

The median filter is counted among the group of rank-selection filters. This sort of nonlinear filters is useful for suppressing noise, and mainly applied in digital image processing [21, p. 271]. Nevertheless it allows the elimination of randomly occuring spikes and the smoothing of gradients in arbitrary signals. Figure 3.4 shows an exemplary curve and its corresponding filtered result. In the first graph a specific signal is hard to distinguish. By applying the median filter nearly all pervading spikes are eliminated, revealing two areas of higher signal strength. The effects on the frequency spectrum can be seen in the next subsection (Fig. 3.5).

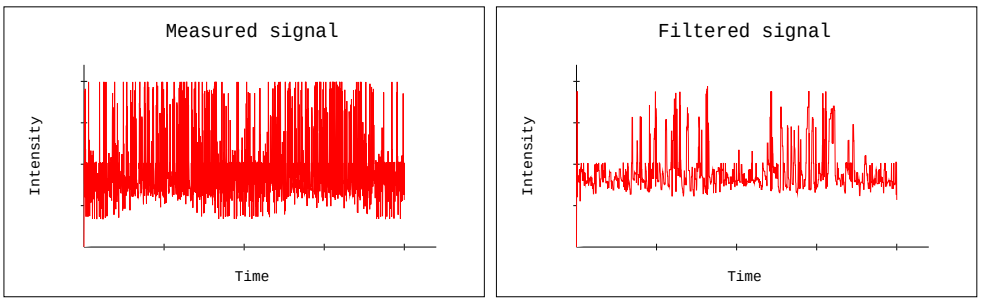


Figure 3.4: Applying a median filter

To actually apply the median filter, a given sample sequence is processed successively. All samples within a small window are first ranked by its intensity. Then the center sample is replaced by a new value – which is as the filter's name already implies the median – and the window is shifted one sample ahead. After processing all samples individually, the operation is finished.

Calculating the local median for a discrete sequence of N values (in a window of length N, for N odd) is done by finding that member for which $(N-1)/2$ elements are smaller or equal in value and $(N-1)/2$ elements are larger or equal in value. After arranging the sequence by intensity, it is then simply the new center element. The following numerical example

$$z = [2, 45, 7, 13, 29]$$

shows a window of five samples. After sorting the array,

$$z = [2, 7, \mathbf{13}, 29, 45]$$

the median can be determined as 13. The computational complexity for applying a N-median filter (i.e. with a window size of N) is $\mathcal{O}(N^2 \cdot \log N)$ per sample in the general case, however there are approaches with a complexity of $\mathcal{O}(N)$ and even $\mathcal{O}(1)$ [22].

3.2.2 Fourier analysis

In mathematics, Fourier analysis performs studies on the representation of general functions by sums of simpler trigonometric functions. The process of decomposing a

function is thereby called *Fourier transform*. Applied to signal processing, this operation delivers oscillatory functions representing the frequency domain of the original function. For finite-length signals an alternative Fourier represantation exists, referred to as the *discrete Fourier transform (DFT)*. A full theoretical examination on this matter including proofs and deductions can be found in [23, pp. 541–628].

For an integrable function $X : \mathbb{R} \to \mathbb{C}$, the Fourier transform is defined as

$$Y(\omega) = \int_{-\infty}^{\infty} X(t) e^{-2\pi i \omega t} dt.$$

In signal processing, the input function most often is a discrete set of real numbers $X_j, j \in \mathbb{Z}$. This leads to the more specific *discrete-time Fourier transform*

$$Y(\omega) = \sum_{j=-\infty}^{\infty} X_j e^{-ij\omega}$$

which is an approximation of the general continous-time Fourier transform. Its output still represents the entire continous frequency domain. If it is sufficient to only obtain the frequency components representing the finite time segment of length N that was analyzed, the discrete Fourier transform is used:

$$Y_k = \sum_{j=0}^{N-1} X_j e^{-2\pi i j k / N}.$$

These values Y_k can be interpreted as frequency domain samples of the continous spectrum provided by the discrete-time Fourier transform.

Applying the DFT to a given set of samples X_j, $j = 0, ..., N-1$ provides an according number of complex values Y_k, $k = 0, ..., N-1$. These values represent the amplitude and phase of the different sinusoidal components of the input signal X_j. By making use of *Euler's formula*[2]

$$e^{ix} = \cos x + i \sin x$$

the sinusoid amplitudes A_k can be obtained from the complex modulus:

$$A_k = \|Y_k\|_2 = \sqrt{Re(Y_k)^2 + Im(Y_k)^2}$$

followed by an according normalization (usually the factor $1/N$). For real-valued

[2] Leonhard Euler (1707 – 1783)

input data X_j, the DFT possesses the *Hermitian symmetry*[3] $Y_k = Y_{N-k}{}^*$. Therefore the frequency spectrum is symmetrical, too, so half of the frequency components are redundant. As a remark, this leads to a significant saving of storage (with half of the output being the complex conjugate of the other half).

Figure 3.5 shows the application of the DFT on the measured curves displayed in Fig. 3.4. These are composed of 1,500 samples each, so the frequency domains consist of 750 different components. Note that the use of a median filter has also some influence on the frequency domain, mainly reducing ultra-high frequency parts as well as overall noise.

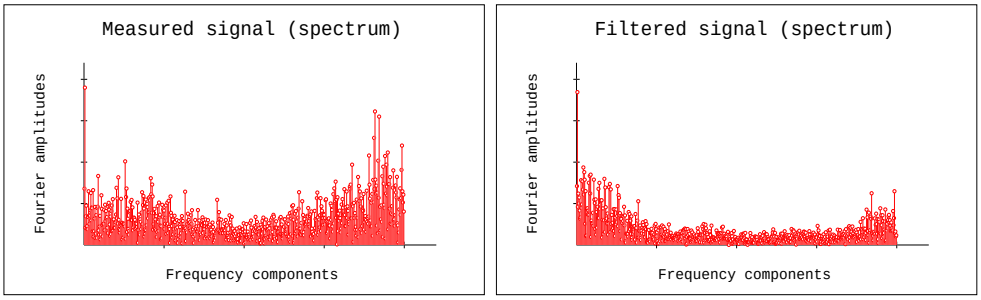


Figure 3.5: DFT frequency domains

Basically the computation of the DFT has a complexity of $\mathcal{O}(N^2)$. But the more sophisticated *Fast Fourier Transform (FFT)* [24] accomplishes a complexity of $\mathcal{O}(N \cdot \log N)$. This algorithm is the de facto standard for calculating Fourier transforms.

3.3 Significance of results

Although analyzing a signal with Fourier methods poses a significant advantage compared to the naked eye, it is still a challenging task to determine if a certain frequency part is simply random noise or definitely synthetic generated. Applying a median filter as discussed in Sect. 3.2.1 in fact reduces the overall noise and filters out some higher frequencies, but verifying a specific frequency can nevertheless be a shot in the dark. To compensate, an analysis relative to the whole spectrum is necessary. Two basic methods are presented in the following.

[3] Charles Hermite (1822 – 1901)

3.3.1 Median rating

A good indication for a synthetic frequency to be included in a signal is for it to produce the greatest amplitude in value. However, this absolute value doesn't include information on the differential to other amplitudes, which could be a small percentage above the mean value of all amplitudes occuring in the spectrum, or contrarily a significant multiple of it. These differences in the quality of verification account for the median rating concept.

At first, the median of the whole spectrum is computed. This operation will find that particular frequency part which amplitude divides the spectrum in two even sized subsets, one that hosts all members smaller or equal in value and another one for which all members are larger or equal in value. Assuming the target frequency to be greater in value than the median, further investigation would be worthwile. The actual rating R_{f_k} for that frequency f_k is calculated as

$$R_{f_k} = A_{f_k} / M_k,$$

where A_{f_k} is the corresponding amplitude and M_k the median over k samples. This value R_{f_k} provides a reasonable base for measuring the quality of detecting a specific frequency; the higher its absolute value, the better the result. It also allows the well-defined comparison of different measurements.

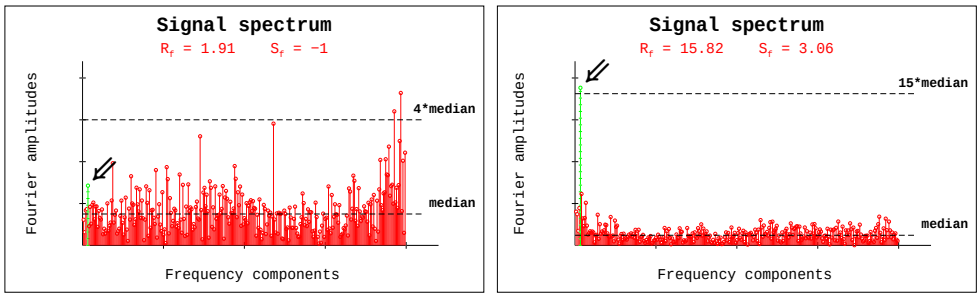


Figure 3.6: Median rating

Figure 3.6 presents a specific example. The plotted `median` line shows the median of the whole spectrum, whereas the second line indicates the ratio of the highest frequency part and the median. For the target frequency (which is well-known and tinted in green as well as marked by an arrow), its rating R_{f_k} is also drawn. S_{f_k} represents the value for the inharmonic rating, discussed in the next subsection. The left diagram illustrates a weak detection with low significance while the right diagram exemplifies an outstanding verification success.

3.3.2 Inharmonic rating

With the concept of median rating, an absolute value has been introduced for directly measuring the quality for a specific frequency verification. In spite of its evident usefulness, it still lacks closer examination for the rest of the spectrum. Assuming a target frequency rated with ten times the median, there could still occur numerous other frequency parts with ratings nearly as high as its own. Obviously another rating concept operating relative to the rest of the spectrum could help to improve this situation.

First of all, the frequency component with the second greatest amplitude in value is determined. With calculating the fraction

$$S_{f_k} = A_{f_k} / A_{f_m},$$

where A_{f_k} is the amplitude of the target frequency part f_k and f_m that one with the second greatest amplitude in value, a simple spectrum-relative rating S_{f_k} is determined. For values of this ratio greater than one, it describes the relative protrusion of the target frequency, whereas the result for targeted frequencies not producing the highest amplitude in value (and thus not being clearly detected) is defined as `-1`.

But for all that another fact has to be considered. As described in Sect. 3.2.2, the Fourier transform decomposes a certain signal into sums of trigonometric functions. For exact signals (like a sinusoid) only one sum representing its specific frequency part evaluates as greater than zero. However, for signals encountering distortion and noise, further components become relevant. These *harmonic* frequencies inherently occur in such environments and can only be reduced to a small extent (likewise some particular spikes in Fig. 3.5 survive the median filter or even get amplified whereas others don't).

As a result, the spectrum-relative rating described above has a fundamental limit in its relevance due to higher-frequency harmonics inherently present in the freqeuncy spectrum. Therefore, the concept is extended to the inharmonic rating approach. That implies the harmonic frequencies are left out, and the second greatest amplitude in value not to be a multiple of the target frequency is drawn into the calculation. This improved value S_{f_k} could – combined with the median rating – serve as a base for automatically computed decisions on the detection of frequency components. However, these values are currently only drawn into the frequency analysis diagram (e.g. Fig. 3.6). Advanced concepts like *signal-to-noise ratio* [25, pp. 21–35] should be considered in future work, too.

3 Basics

Chapter 4

4 System design

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies."

C.A.R. Hoare

Designing a software system is all about requirements. These are the descriptions of the services provided by the system and its operational constraints. However, requirements can be divided into two areas, mainly by their level of abstraction. *Functional requirements* define what exact services or functions the system is expected to provide, from a user's or developer's point of view. In contrast, *non-functional requirements* specify global system properties or particular criteria for measuring various quality aspects [26, pp. 120–125].

Eventually the system design results in an architectural plan for the software system as a whole. Technical details may still be left out, but major components and its interfaces should have been specified. After implementation, the system should pass processes of verification (*"Was the system built right?"*) as well as validation (*"Was the right system built?"*) [27], at which the fundamental design is the linchpin of success.

This chapter deduces functional requirements from specific use cases auxiliary to the development of an architectural design. Non-functional requirements like documentation, extensibility, reliability or performance are assumed to be self-evident and therefore not further mentioned.

4.1 Use cases

In software engineering, a use case is a description of a continous work cycle. It is initiated by an *actor* respectively a *trigger* and results in an achievement noticeable by the actors. Therefore it determines a system's external behaviour from a user's point of view. Use cases describe *what* a system should supply, and not *how* it is achieved [28].

For operational use, in many cases a single dependency discovery will suffice certain demands like verifying a particular connection between two systems. Performing that basic analysis resolves into a set of prime procedures shown in Fig. 4.1.

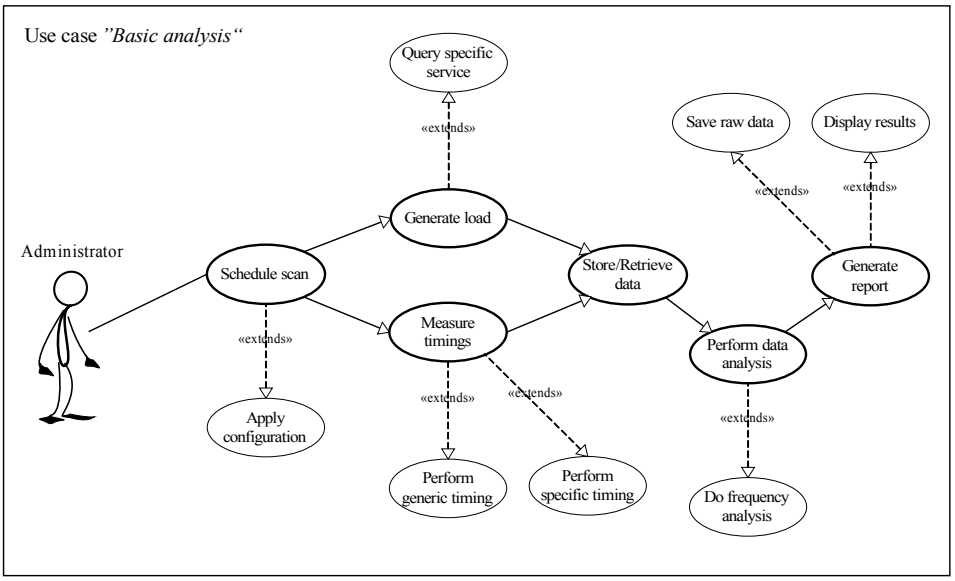


Figure 4.1: Use case *"Basic analysis"*

The administrator first schedules a scan, which implies a dedicated configuration mechanism. Subsequently, based on that configuration the load generation and timing procedures are initiated. Consequently the obtained data is returned and examined by means of frequency analysis. Finally, along with the preservation of raw experiment data a report showing the scan result is generated. Section 6.1.2 practically evaluates this elemantary kind of work flow.

For detecting *Associations*, no further knowledge on their nature is necessary and thus a generic timing method for eligible systems is sufficient. The search for *Affiliations* by contrast would rely on multiple load patterns generated on selected

systems to be verified on the dependency afterwards. A more appropriate procedure should draw upon generating the load directly on the dependency and measure the timings on possible *Affiliations*. While requiring additional knowledge for querying the dependency's services instead of performing generic timing measurements, the amount of disturbance invoked on the analyzed systems is significantly reduced. Both alternatives are examined in Sects. 6.1.2 and 6.3.1.

Furthermore, this basic use case also fits into more advanced scenarios supporting higher levels of configuration management. In the following, three essential procedures focusing on strategic applications are explained.

4.1.1 *"Fail safe expansion"* – top-down scan

Large IT environments indispensably need a profound concept for backup and problem handling. Procedures for restoring single systems thereby have to be combined with measures of rebuilding dependencies as well. In addition, for resolving errors on particular services the *root cause* of the problem has to be identified, for which the knowledge of dependencies is essential. According to that, a full list of all *Associations* attached to a particular service is the vital knowledge base for developing a solid backup concept or being able to quickly respond to failures. A corresponding use case providing that *Association* map is shown in Fig. 4.2.

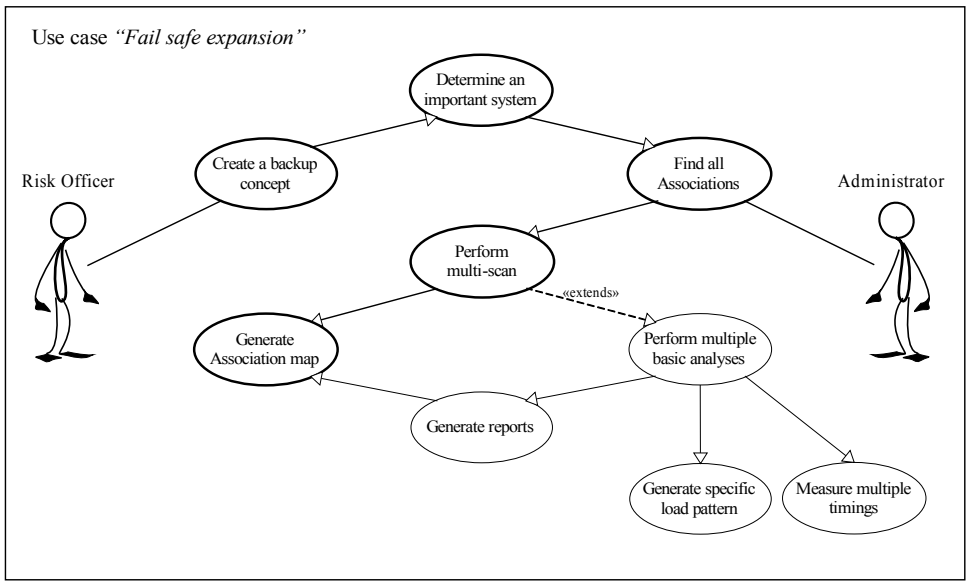


Figure 4.2: Use case *"Fail safe expansion"*

Commonly, a risk officer as the person in charge is assigned to create a certain backup concept. As an important part of it, he determines essential systems for which a fail safe procedure is required. Afterwards, he delegates the task of finding all *Associations* to a commissioned system administrator who first performs a multi-scan and then is responsible for generating the dependency map. That multi-scan should eventually provide the complete *Association* map fully automatical, presumably by exploiting given knowledge on available assets. Nevertheless, the scope of this work confines that process to multiple basic timing measurements interactively put together by the administrator.

4.1.2 *"Impact analysis"* – bottom-up scan

Daily operational affairs frequently involve maintenance work on key services. In doing so, inconsiderate or ill-prepared cutoffs can lead to failures or downtime of dependent services and result in customer complaints and financial loss. Hence in case of an anticipated or unexpected service shutdown information on other systems to be affected is required for deploying adequate prearrangements. This is also known as *impact analysis*. Figure 4.3 illustrates this use case.

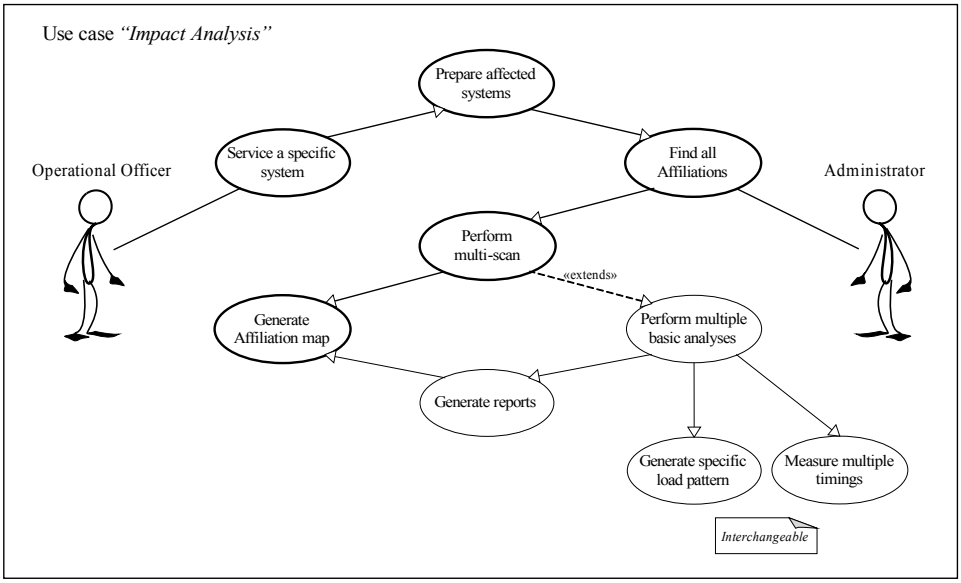


Figure 4.3: Use case *"Impact analysis"*

The basic work flow is similar to that expressed in Fig. 4.2. But in difference, an operational officer requests a map of all *Affiliations* of a certain system to be main-

tained in order to prepare affected systems or inform their administrators and/or users. This procedure again relies on a multi-scan, which can be accordingly replaced by a sequence of basic analyses. Regardless of the effectively implemented approach, it has to be decided if the load is directed towards the questioned system (which assumes particular knowledge of the running services) or engaged on possible *Affiliations* and verified by a generic timing measurement on the system to be maintained.

4.1.3 *"ITIL© CMDB"* – full scan

The ultimate use case for discovering dependencies is the generation of a full network-wide configuration map respectively the support for an implementation of a configuration managemant database according to ITIL© (IT Infrastructure Library©, [2]). Considered superficially, this requires finding all *Associations* and *Affiliations* for all network assets known to deliver services (Fig. 4.4).

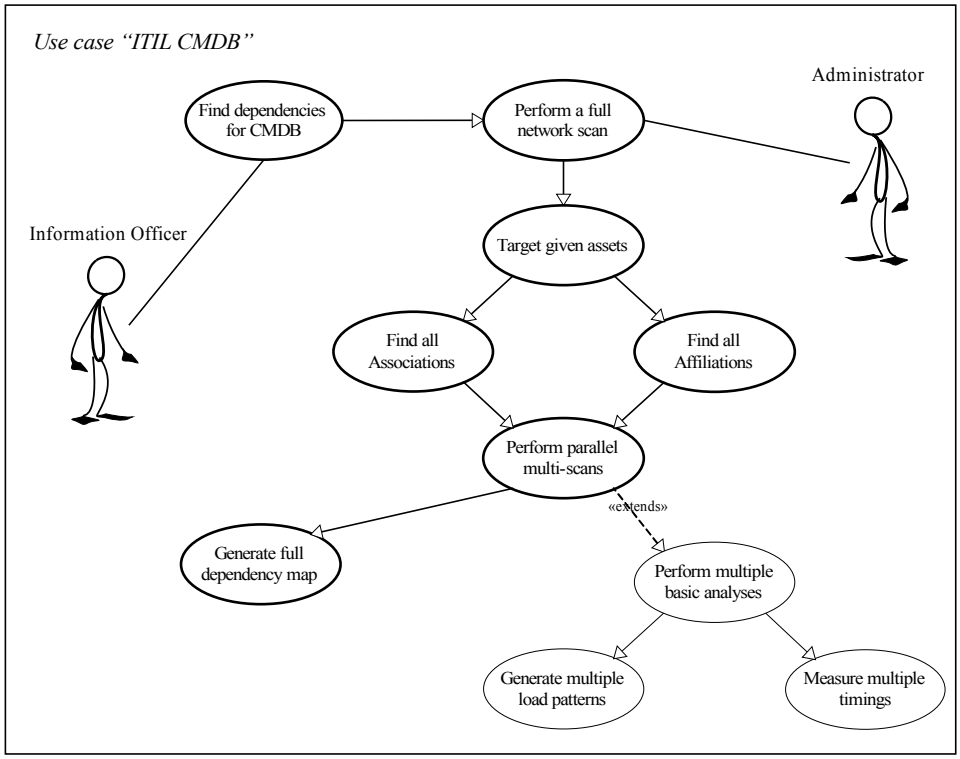


Figure 4.4: Use case *"ITIL© CMDB"*

However, the use of parallel scanning techniques taking advantage of different load patterns could significantly reduce time expenses in return for a negligible increase of controlling complexity. Section 6.3.3 provides initial studies on that matter. Moreover, bringing into play additional concepts like graph theory [29] could probably decrease overall efforts, too. This research is no constituent of this work but recommended for future studies though.

4.2 Functional requirements

The basic analysis design acquired in the previous section already inferes three major components. At first, a dedicated control mechanism serving as an interface for user input/output operations is required. Moreover, further components for generating load as well as for performing timing measurements are necessary. For a better structuring, the functional requirements are divided accordingly.

4.2.1 Control system

The control system has to provide several services for performing basic experiments. It is the only interface to the user and encapsulates a variety of functions. Table 4.1 gives a compilation.

Requirement	**Description**
`A1` Configuration mechanism	A basic experiment has to run non-interactively. The user preliminarily schedules the experiment through a designated configuration file. This includes parameters for targeting services (including addresses and load and timing specifications) as well as flags for disabling certain components and enabling debug outputs. The design has to be generic enough to allow further extensions like parallel or multi-scans.

continued on next page

continued from previous page

Requirement	Description
A2 Distributed controlling	A fully distributed deployment has to be considered right from the beginning. Thereby the user must not have to communicate directly with the associated components. A controlling mechanism allowing to operate distributed parts also across network bounds or regional barriers fully independent from user inputs has to be implemented. This includes procedures for starting and stopping as well as well-defined processes for data exchange.
A3 Frequency analysis	As described earlier in Sect. 3.2, all gathered timing data has to be analyzed for the verification of the configured load signal. By means of signal processing, significant results have to be obtainable, fully autonomously of the user's amount of background knowledge. The so prepared data has to permit the plotting of easily understandable graphical output.
A4 Rating estimation	In addition to the mathematical analysis, a user-friendly rating as explained in Sect. 3.3 has to be implemented allowing a quality measurement for specific results. Furthermore, the rating has to be integrable into output plots and should be designed with future improvements in mind.
A5 Result plotting	An experiment's result must not be returned as a huge amount of raw data, but be refined to a meaningful graph. This has to include information on the dedicated signal, the actually achieved load as well as the corresponding timing measurements in locus and frequency domain interpretation.

continued on next page

continued from previous page

Requirement	Description
A6 Reporting	The output visible for the user has to be a single report allowing the determination of the overall result at a glance as well as the extraction of detailed information if necessary. In addition, it has to provide a standardized basis for comparing different experiments. Besides, all raw data has to be stored for manual investigation.

Table 4.1: Requirements for control system

This component will be called `fansyCOM`, an acronym for *"frequency analysis of synthetic load for configuration management"*. Besides, this designation also names the project as a whole.

4.2.2 Load generation

The load generation component is the core of this project. It has to be carefully designed as generating an impure signal would lead to distortion of results or even render the whole approach useless. Table 4.2 defines essential requirements.

Requirement	Description
B1 Distribution-oriented interface	The distributed approach (already described in Requirement A2) implies the provision of an interface for communication with the control system to transfer configuration details, invoke procedures and return experiment data. Furthermore, the component has to offer a so-called *daemon mode* for listening on procedure calls as a background process to allow the simultaneous start of multiple components without further user interaction.

continued on next page

continued from previous page

Requirement	Description
B2 Sophisticated signal generation	Getting a clear load signal heavily relies on exactly timed queries in the magnitude of microseconds. This assumes a high performance implementation on the one hand, but on the other hand also an elaborate design of the load signal itself, which should be independent from the utilized load module, easily configurable and immune to scaling effects of any kind.
B3 Modular design	Specific services need particular procedures for querying its information and thus generating the load. This requires a modular design providing various subcomponents for targeting different services (like for example a webserver or database), which are selected through the transfered configuration. Although it is sufficient for this work to cover the examples mentioned above, the design must permit further extensions. In addition, each load module has to parallelize every single query for not being dependent on direct response times.

Table 4.2: Requirements for load generation

This component is called `loadCTRL`.

4.2.3 Timing measurement

Although the process of sampling a service is relatively straightforward, some important requirements still have to be considered. These are expressed in Table 4.3. As the design of the timing measurement part is very similar to that of the load generation, some requirements are redundant to list and thus refer to the previous subsection.

Requirement	Description
C1 Distribution-oriented interface	Analog to Requirement B1.
C2 Reliable timing process	Although the timing measurement is based on a simple equidistant sampling, it has to be carefully implemented for every particular service. A suitable method for compensating too weak response times not allowing the verification of the target signal has to be considered. Furthermore, parameters for adjusting the sampling have to be made available through the configuration mechanism.
C3 Modular design	Analog to Requirement B3.

Table 4.3: Requirements for timing measurement

The designated name for this component is `timeCTRL`.

4.3 Architectural design

The following design for architectural key points results from the requirements analysis carried out in the previous sections. Although details are set aside for Chp. 5, all fundamental services to be provided are included. Figure 4.5 shows the graphical representation.

Basically, the architecture consists of three subsystems each divided into several subcomponents with preassigned communication interfaces. The usage of `loadCTRL` and `timeCTRL` is thereby optional for achieving better flexibility in performing sophisticated experiments (Sect. 6.3). The next chapter focuses on more concrete implementation details regarding every subcomponent and the system as a whole.

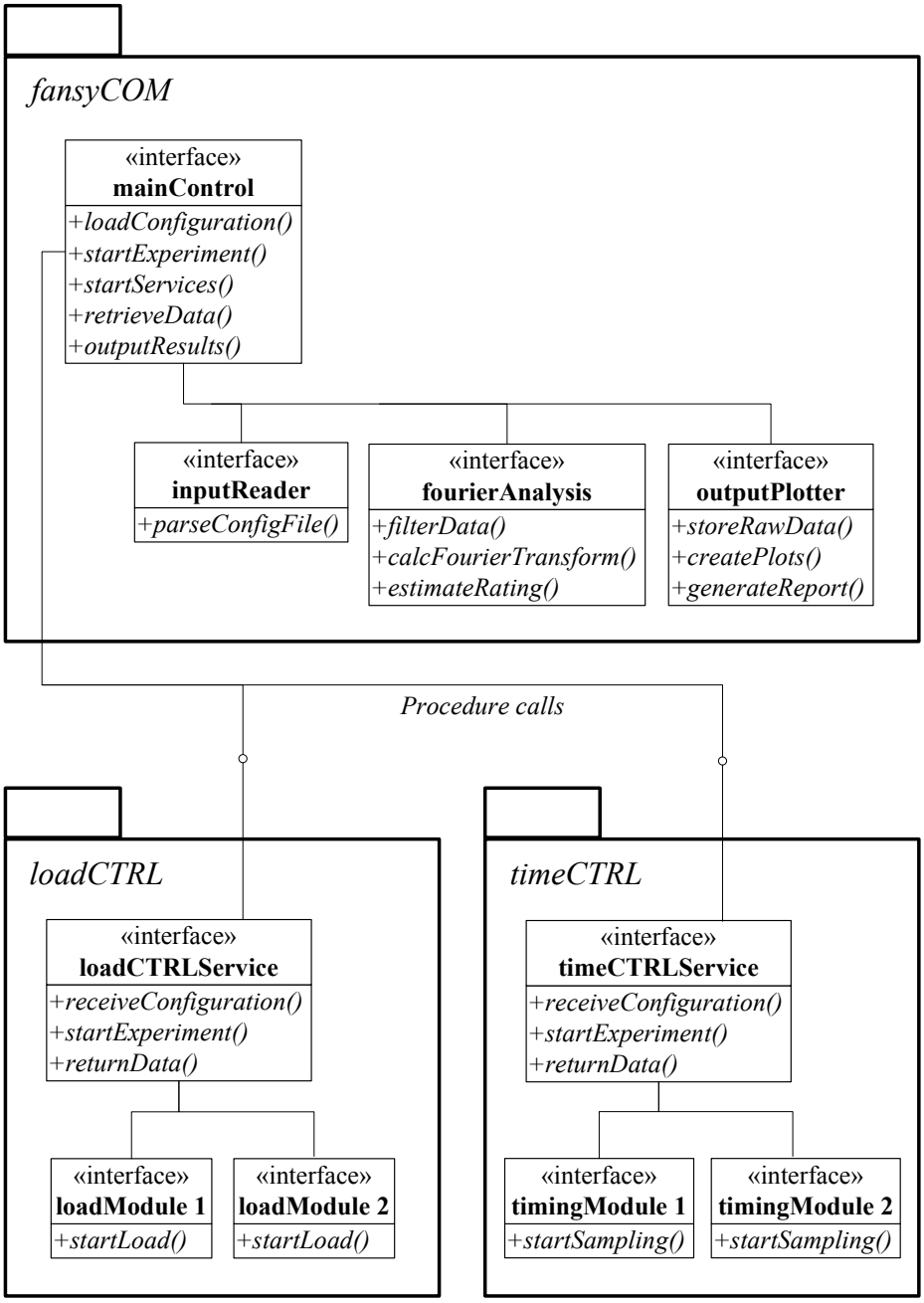


Figure 4.5: Architectural design

4 System design

5 Chapter 5 Implementation

"The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency."

Bill Gates

The goal of this thesis is to develop a solid prototype for active dependency discovery. On the one hand it has to be able to generally verify the basic idea behind the introduced approach. Succeeding in that presumes the feasibility of performing thousands of experiments in a row to obtain a substantial amount of data, for proving certain functionalities to be applicable. On the other hand this prototype should be easily extendable and therefor be well-programmed according to common software engineering paradigms. Further extensions could be aimed at scientifically disclosing more facts on dependency disocovery, adding more practical features or even developing a working product (Chp. 7). For all of that, a systematic and efficient system has to be implemented, which reliably allows the execution of basic analyses for the moment while being sound enough for future use.

The following sections provide details on particular implementation concerns as well as recommended deployment strategies. Specific descriptions of concrete programming can be found in the documentation attached to the source code delivered in Table A.4.

5.1 General

Before developing the actual application, a proof of concept had been implemented. This small set of scripts written in PERL [30] served to discover principal problems early on and to verify the approach to be basically realizable. Considerations like concepts of multithreading or implementing a suitable load generation signal already surfaced during the development of this pre-prototype. Nevertheless this chapter solely focuses on the resulting operational prototype, but the corresponding source code is included in the software package (Table A.4) as well.

The final application is completely written in C [31] while making use of different external open source libraries. It has been designed for UNIX-like operating systems, but should be portable to other operating systems with little effort. Some general issues are discussed in the next subsections, followed by specific implementation details for every component itemized in Sect. 4.3. Minor issues like the designated logging and debugging mechanism as well as more practical usage information like execution procedures or the use of the `scheduler` for performing large series of experiments are documented along with the source code.

5.1.1 Makefiles

For fast development a sophisticated *Makefile* hierarchy was built. `make` is a tool which controls compiling procedures and thus simplifies the generation of executables. In large software projects, dependencies among different classes and libraries have to be considered when compiling and linking a certain source file, which often leads to complicated commands in the magnitude of several hundred characters. With the use of `make` the effort is reduced to writing a corresponding *Makefile* only once while additionally obtaining dependency checks as well as pre- and postprocessing operations. `make` is available for various common operating systems, e.g. *GNU make* [32] for Linux.

Furthermore, `make` also supports the deployment process. It is possible to generate specific components or the source code documentation (Sect. 5.1.4) separately. Additional features like differentiated compiling (e.g. into *daemon mode* specified in Requirements B1 and C1) and clean-up operations provide even more functionality. A detailed description of how to make use of these features can be found in the documentation attached to the source code (Table A.4).

5.1.2 Remote procedure calls

The standard instrument for inter-process communication is called *remote procedure calls* (RPCs). This technology first described in RFC 707 [33] in 1976 allows a program to execute subroutines in another address space also by crossing over physical limits (Requirement A2). For the programmer, this functionality is almost transparent and can be used just like local procedure execution. Nevertheless, the communication interface and data structures to be exchanged have to be specified precedingly [34]. According information to a more advanced level can be found in the source code documentation (Table A.4).

To create code that supports remote communication, the tool `rpcgen` included in common Linux distributions was used. It takes a protocol specification file with C-like syntax as input and generates four files of C code (header and source files for interface and data structures) to be compiled with the main application. Additional parameters were used to create a service indefinitely listening on requests until terminated by the user and also for optionally running a component in *daemon mode* (Requirements B1 and C1). The corresponding *Makefile* handles that generation process independently, however a full list of possible parameters can be obtained via the associated manpage.

5.1.3 Multithreading

Controlling distributed components and above all sending thousands of requests in a particularly narrow timetable are highly parallelized operations. First, the control application `fansyCOM` has to manage two different subcomponents which heavily rely on simultaneous execution, so all corresponding RPC communication events have to take place synchronously. Furthermore, to be able to fulfill a predefined timing schedule, every single query to be sent out must not be dependent on previous execution times or delayed because of alternating service response times. Here, a so-called *"fire-and-forget"* strategy is essential which has to allow the creation of single requests on a specific predefined point in time without influencing the creation of other requests to follow.

For all of that an extensive use of the *multithreading* paradigm has been applied. This concept effectively allows a parallel execution of code on multi-core CPUs, but it is similarly working on single-core CPUs as well. A popular implementation of this feature is based on the utilization of *POSIX threads*, which can be easily accessed in the C programming language [35]. For this work, the common library `pthread` has

been adopted. In doing so, some major issues had to be addressed. At first, the operating system's scheduler had to be adapted to allow the priorization of threads for preventing system process to be executed in favor. This is achieved by manually setting the scheduler's attributes (`PTHREAD_EXPLICIT_SCHED`) to a *round robin* scheme (`SCHED_RR`) and assigning all threads a high priority (`pthread_setschedparam()`). Besides, every thread reserves by default a private stack of 8 MB in main memory, leading to a maximum number of simultaneous threads in the range of a few hundreds on modern hardware. By setting the stack size to its minimum of 16 kB (`PTHREAD_STACK_MIN`), this restriction was resolved.

5.1.4 Documentation

Detailed explanations of the programming concepts raised in the previous subsections along with references to the source code are provided with a separate technical documentation. With additional control statements placed inside the source code, the tool `doxygen` [36] generates an interactive HTML documentation as well as a comprehensive PDF memorandum. This material is supplied with the software package, but can also be created manually with a corresponding *Makefile* procedure.

5.2 `fansyCOM`

By making use of the design pattern *Facade* [37], `fansyCOM` consists of four elementary parts as shown in Fig. 4.5: the main control class plus additional packages for parsing XML [8] configuration files, calculating Fourier transforms (Sect. 3.2.2) and generating reports. This pattern reduces complexity by hiding subsystems and providing an easy-to-use interface to technical features. However, performing a basic analysis can be broken down to six internal operations:

1. **Parsing the configuration.** At first, an XML configuration file is parsed to obtain the user-provided configuration settings (Requirement A1). The exact location of this file is passed through the standard UNIX input mechanism. Sect. 5.2.1 goes into detail.
2. **Setting up the RPC services.** Before start, the `loadCTRL` and `timeCTRL` services have to be informed that an experiment is imminent. At that time, the configuration is forwarded and applied to these components through RPCs and memory is allocated for both `fansyCOM` and the services (Requirement A2).

3. **Starting the experiment.** After all subcomponents report readiness, the starting signal is given and `fansyCOM` waits for the completion of the experiment. Subsequently the experiment data is returned and prepared for analysis.

4. **Analyzing the experiment data.** All means of signal processing and result valuation as described in Sects. 3.2 and 3.3 are performed directly after the end of the experiment (Requirements A3 and A4).

5. **Plotting results.** Finally the obtained data is visualized through various charts and put together to a comprehensive PDF report (Requirements A5 and A6). Section 5.2.2 gives a more detailed description of technical considerations.

6. **Finishing the experiment.** To finish the experiment and exit the execution, another RPC event is sent out for signaling success to the subcomponents and invoking procedures for memory freeing.

In the following, a more advanced view on some topics mentioned above is provided. However, a comprehensive documentation of the whole source code goes beyond the scope of this thesis and has to be limited to interesting details useful for understanding the basic functionalities of the prototype. For further reading reference is made to the technical documentation attached to the source code (Table A.4).

5.2.1 XML configuration

As postulated in Requirement A1 a sophisticated configuration mechanism had to be implemented. For that, the user creates a well-defined configuration file which is then parsed by `fansyCOM`. It is based on an XML [8] structure as shown in Table 5.1.

```
<configuration>
    <fansyCOM> </fansyCOM>
    <loadCTRL> </loadCTRL>
    <timeCTRL> </timeCTRL>
</configuration>
```

Table 5.1: Basic XML configuration structure

Every subtag specifies all necessary settings for each subcomponent, while global configuration options are located in `<fansyCOM>`. All parameters are defined through attributes of the corresponding subtag, as demonstrated in Table 5.2.

```
<fansyCOM
    host            = [IP address]
    writeToFile     = [0|1]
    filter          = [0|1]
    debug           = [0|1] >
</fansyCOM>
```

Table 5.2: Exemplary `fansyCOM` configuration

The `host` attribute is common for each component and specifies the system it is running on. This can be a single machine or multiple servers in a distributed setup as well. Furthermore, the tag `<fansyCOM>` provides switches for turning on and off the output (`writeToFile`) or the median filtering mechanism (`filter`) described in Sect. 3.2.1 and also for advanced debug output (`debug`). The syntax for the `loadCTRL` and `timeCTRL` subcomponents is identical, exemplarily shown in Table 5.3.

```
<xxxxCTRL
    inUse           = [0|1]
    debug           = [0|1]
    host            = [IP address]
    target          = [IP address]
    ... >
    <SPECIFIC_MODULE> </SPECIFIC_MODULE>

</xxxxCTRL>
```

Table 5.3: Exemplary subcomponent configuration

It is possible to disable a component through the parameter `inUse`, which is useful for performing scans with multiple signals or timing measurements. In addition,

the target to be queried has to be specified via the `target` attribute. Besides, the `host` and `debug` parameters provide similar functionality as for `fansyCOM`. Further modules for specific timing or load generation features can be integrated with corresponding subtags. These modules as well as secondary parameters for setting up component-specific details are discussed in the respective subsections later on in this chapter, a more concrete configuration file for productive use can be found in the Appendix (Table A.1).

Parsing the XML file is achieved with the help of the open source library `libxml2` [38]. This library offers advanced features for handling XML files, including input reading with extraction of single pieces of information as well as building up a whole tree, different output operations and sophisticated parsing mechanisms. Nevertheless, only a few basic functions were used to obtain the configuration data and parse it into internal representation. More details on that are given in the source code documentation.

5.2.2 Generating results

Since key issues for the generation of results were already discussed in Sects. 3.2 and 3.3 as well as Requirements A3–A6, a more technical view is presented in the following.

For meeting Requirement A3, filtering out random outliers in the target response times and the timing samples for the dependency is recommended. A 7-median filter as described in Sect. 3.2.1 proved useful for that matter. To apply it, it is necessary to loop over all given samples replacing every value with the median of the interval surrounding it. In doing so, every interval has to be sorted by value. Although there are many high performance sorting algorithms, the standard C built-in function `qsort` has been utilized since runtime considerations are irrelevant for the experiment analysis. This function implements the *quicksort* algorithm developed in 1962 by Hoare[1] [39]. Rating the results (Sect. 3.3) also makes use of this algorithm, but contrarily to local filtering the median is calculated for the whole set of samples.

Calculating the discrete Fourier transform (Sect. 3.2.2, Requirement A3) is based on the open source library `libfftw3` [40]. This collection of C subroutines allows the easy computation of various DFTs, but for this work only one case has to be considered. For real-valued and arbitrary-sized input data, the function `r2c_1d()`

[1] Sir Charles Antony Richard Hoare ($*$ 1934)

computes the one-dimensional discrete Fourier transform, resulting in a complex-valued output vector of half of the length, respecting the *Hermitian symmetry* as already described in Sect. 3.2.2. Subsequently, the complex conjugate is calculated to obtain the frequency parts of the sampled signal, which are then returned to the main `fansyCOM` operating function.

The final step for handling results is the generation of single plots and a compound report (Requirements A4–A6). For that, the free `libplot` library provided with the GNU `plotutils` package [41] was utilized. This library allows the drawing of two-dimensional vector graphics on different devices (including animations for the X Window System), but only the export functions to the graphics format EPS were used. However, future extensions like an interactive GUI are already considered. For drawing, `libplot` basically offers functions for simple geometric objects like points, lines and circles as well as for producing text. These functions were combined to create advanced charts with a detailed coordinate system and a clear representation of signals in its locus and frequency domain (including a cut-out of the targeted frequency). After the generation, the external standard Linux tool `epstopdf` is used (if available) for converting all EPS files to PDF in order to permit the creation of advanced PDF reports. This is done with a hard-coded LaTeX template translated into PDF with `pdflatex`. Some examples for the plots can be found in Sect. 3.2, whereas whole reports of concrete experiments are provided with the Appendix.

Last but not least, the raw experiment data is stored in ASCII format for future examinations. This includes information on experiment parameters as well as specific timing data for `loadCTRL` and `timeCTRL` and all results from the corresponding frequency analyses.

Every analysis results in a set of files which is uniformly named and structured. An exemplary output of a single experiment is shown in Table 5.4.

```
▶ experiment__YYYY-MM-DD__HH.MM.SS/
    ─ experiment__YYYY-MM-DD__HH.MM.SS.dat
    ─ loadCTRL_analytic_signalCurve.pdf
    ─ loadCTRL_analytic_spectrum.pdf
    ─ loadCTRL_analytic_spectrumDetail.pdf
    ─ loadCTRL_signalCurve.pdf
```

continued on next page

continued from previous page

```
    – loadCTRL_spectrum.pdf
    – loadCTRL_spectrumDetail.pdf
    – timeCTRL_signalCurve.pdf
    – timeCTRL_spectrum.pdf
    – timeCTRL_spectrumDetail.pdf
▶ experiment__YYYY-MM-DD__HH.MM.SS.pdf
```

Table 5.4: File structure of experiment results

The last PDF file is the overall report, the DAT file hosts the raw data. Both files as well as the directory are tagged with a timestamp for easily finding specific results. If filtering is enabeld, two corresponding sets of files are created.

5.3 loadCTRL

The `loadCTRL` component acts as a *Proxy* [37] for the load generation. It is designed to host multiple different modules being able to create load on various services, determined through the user-provided configuration. However, the mathematical function behind the signals is always the same and was carefully chosen. It will be described along with corresponding parameters for modifying the signal's characteristics in the next subsection. After that, two modules for generating load on webservers and databases are discussed.

5.3.1 Load generation

Generating the load signal requires a precomputed table of timings for the queries to be sent. Each single request has to be made at a specific point in time, whereas smaller displacements result in higher load on the targeted machine. This timing information is calculated before the start of the experiment depending on a variety of additional `loadCTRL` parameters. Table 5.5 gives a comprehensive summary of possible settings.

```
<loadCTRL
    ...
    loadstyle       = [HTTP-GET|SQL]
    wavestyle       = [cosine]
    timeframe       = [sec]
    resolution      = [steps/sec]
    frequency       = [Hz]
    amplitude       = [samples/sec] >
</loadCTRL>
```

Table 5.5: Special `loadCTRL` parameters

The parameter `loadstyle` determines the specific service request for the target to be put under load. Currently `HTTP-GET` and `SQL` queries are supported, further details on these modules can be found in Sect. 5.3.2. `wavestyle` sets the mathematical function behind the load signal, which is for the moment a modified cosine. Defining the amplitude of the signal is done by specifying a peak rate of samples per second. This value is afterwards scaled to the time period of a single discretization step (controlled by `timeframe` and `resolution`) and taken as the absolute peak amplitude. The signal's frequency is regulated by the `frequency` parameter.

The actual function used for the generation of the load signal is a modified cosine given by the term

$$f(t) = \frac{1}{2} \cdot (-\cos t + 1),$$

where $t \in [0, ..., 2\pi]$ and $f(t) \in [0, ..., 1]$. This resembles one wave form in the given interval with a maximum value of 1. By applying the configuration parameters as described above, the function $f(t)$ can be expanded to

$$f(t) = \texttt{amplitude}_{abs} \cdot \frac{1}{2} \cdot \left[-\cos\left(2\pi \cdot \texttt{timeframe} \cdot \texttt{frequency} \cdot \frac{t}{\texttt{resolution}_{abs}} \right) + 1 \right].$$

Now $t \in [0, ..., \texttt{resolution}_{abs}]$ and $f(t) \in [0, ..., \texttt{amplitude}_{abs}]$ apply, where the newly introduced parameter $\texttt{resolution}_{abs}$ equals the absolute number of discretiza-

tion steps over the whole `timeframe`, and $\texttt{amplitude}_{abs}$ equals the given parameter `amplitude` converted to the length of one time step.

Sending out $f(t)$ number of samples per time step t results in a cosine-like distribution of requests and therefore in a cosine-shaped load pattern (Fig. 5.1, left plot). However, this calculation doesn't provide a constant number of samples for increasing resolutions because of side-effects incorporated with the discretization.

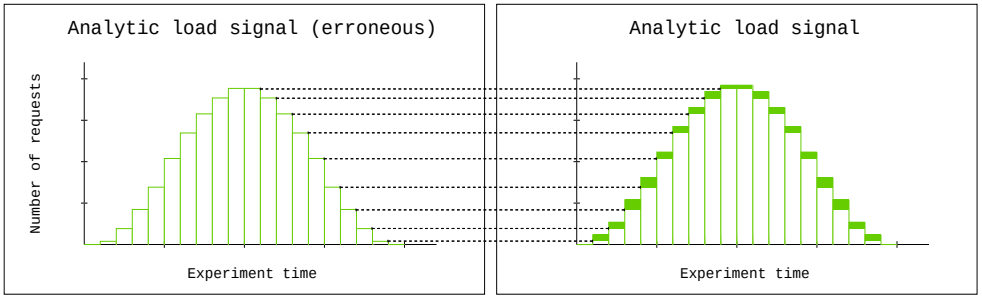


Figure 5.1: Basic load signal

An improved calculation of the number of samples for each time step is achieved by considering the integral under the cosine. For every discretization interval, its local part of the integral is computed and scaled according to the maximum number of samples specified via $\texttt{amplitude}_{abs}$. The corresponding sample distribution function $\widehat{f}(t_k)$ thereby results in

$$\widehat{f}(t_k) = \frac{\texttt{amplitude}_{abs}}{I_{max}} \cdot \int_{t_k}^{t_{k+1}} \frac{1}{2} \cdot (-\cos u) + 1 \, du,$$

where I_{max} equals that part of the integral with the highest value, which assures $\widehat{f}(t_{max}) = amplitude_{abs}$. In addition, the substitution $t \to t_k$ with

$$t_k = 2\pi \cdot \texttt{timeframe} \cdot \texttt{frequency} \cdot \frac{k}{\texttt{resolution}_{abs}}$$

for $k \in [0, ..., \texttt{resolution}_{abs}]$ is applied. Determining the maximum integral value I_{max} assumes the preceding calculations of all local integral parts. To do so, the inner function was manually integrated, which leads together with the newly introduced variable t_k to the term

$$\widehat{f}(t_k) = \frac{\texttt{amplitude}_{abs}}{I_{max}} \cdot \frac{1}{2} \cdot \Big[(t_{k+1} - \sin t_{k+1}) - (t_k - \sin t_k) \Big].$$

This is the conclusive method for calculating the cosine-based load signal. Figure 5.1 compares the resulting distribution to the flawed one described above, revealing some slight differences (marked by thick strokes). Nevertheless, these differences accumulate to severe irregularities when performing large series of experiments. It has to be noted however that this generation of the load signal does not provide a perfectly constant sample distribution for increasing resolutions either. Building the ratio of I_{max} and the corresponding integral parts still leads to slight inconsistencies. It can be shown that the effect is negligible though. For that, the factor I_{max} is more closely investigated:

$$\begin{aligned} I_{max} &= (t_{max+1} - \sin t_{max+1}) - (t_{max} - \sin t_{max}) = \\ &= (t_{max+1} - t_{max}) - (\sin t_{max+1} - \sin t_{max}) = \\ &= \Delta t - \underbrace{(\sin t_{max+1} - \sin t_{max})}_{\to 0 \ (*)} \end{aligned}$$

For $\texttt{resolution}_{abs} \gg \texttt{timeframe}$, the approximative transition at $^{(*)}$ can be applied due to the fact that $\sin t_{max+1} \approx \sin t_{max} = 1$. Accordingly, for increasing resolutions I_{max} converges on Δt, which scales reciprocally with the resolution. The same accounts for $\texttt{amplitude}_{abs}$, so the whole prefactor is in approximative terms independent from the resolution. Regarding the integral, it is obviously also independent from the resolution as it represents the area under the cosine, which has always the same value (apart from discretization approximations for small resolutions). Alltogether it can be stated that if fine enough, the parameter `resolution` doesn't have any significant influence on the sample distribution.

The reason for this modified calculation scaled by I_{max} is mostly a more practical approach for controlling the actual load curve. Therefore the load generation is considered to take effect only locally and mostly independent from other timing steps, so this proceeding gives a more direct control of the load peak as it is exactly specified with the `amplitude` parameter, i.e. $\widehat{f}(t_{max}) = amplitude_{abs}$ as already mentioned earlier.

Although the cosine-based load signal fulfills Requirement B2 and proved to provide outstanding verfication success (Chp. 6), further improvements are imaginable. Future work could focus on testing other mathematical functions less susceptible to harmonics (Sect. 3.3.2) or even follow a completely different approach. By directly specifying the load signal's frequency spectrum (preferredly with a specifically unique pattern) and recreating its time domain via *Fourier synthesis* (i.e. via the *inverse* discrete Fourier transform [23, p. 567], Sect. 3.2.2), an even more explicit and accurate verification could potentially be achieved.

5.3.2 Load modules

Additional to the precomputation of the load signal explained in the previous subsection, specific service queries had to be implemented (Requirement B3). The modular design thereby allows the replacement of the request type or the underlying load function independently. At the moment there are two different types of queries available, for targeting webservers and databases.

The standard service to be scanned for dependencies (more precisely for *Associations*, Sect. 3.1) used in most of the experiments listed in Chp. 6 is a common webserver. Table 5.6 shows the configuration for the corresponding load module.

```
<loadCTRL
    ... >
    <httpGET_LOADMODULE
        url           = ... >
    </httpGET_LOADMODULE>

</loadCTRL>
```

Table 5.6: Special load module configuration (HTTP-GET)

The load samples are generated in an extra thread each and send a HTTP-conform request [42] to the specified URL. Further configuration is not necessary. Briefly, this action requires the creation of a so-called *socket* [43], which is a pair of an IP address and port. A valid GET request as depicted in [42] is then written to

that socket, and the passed time until the response is measured. Further details on timing measurements are provided in the next section.

Note: every single request could possibly generate a corresponding entry in the access logs of the webserver. Although important webservers normally answer tens of thousands of requests every day, this issue may have to be considered for weaker systems.

Another type of load request is geared towards secondary systems providing information for main services. As described in Sect. 4.1.2, this operation may be useful for detecting *Associations*. For general verification of this approach, a load module for SQL databases was implemented. Its XML configuration slightly differs from that for HTTP, as visible in Table 5.7.

```
<loadCTRL
    ... >
    <SQL_LOADMODULE
        port          = ...
        user          = ...
        password      = ...
        database      = ...
        burst-length  = ... >
    </SQL_LOADMODULE>

</loadCTRL>
```

Table 5.7: Special load module configuration (SQL)

The major disadvantage in using the SQL load module is that detailed connection properties have to be provided. Where this knowledge is not available, the load can still be directed towards the *Affiliations* (Sect. 3.1) and verified at the backend with generic timings. Nevertheless, it is possible to ommit the parameters `user`, `password` and `database` for generating load with simple (denied) connection tries. For that, the additional parameter `burst-length` can be specified to open more than one connection per sample, resulting in many burst-like requests. This proceeding increases the overall load and transposes the cosine load curve to higher absolute values. Nevertheless, another issue has to be considered. Often databases restrict

the access to specific IP addresses, and discard all other incoming requests. This would render the generation of load to be impossible, so the `loadCTRL` application must run on a privileged system in that case.

For implementing SQL queries, the library `mysqlclient` delivered with the MySQL database [44] was used. It offers easy-to-use functions for connecting to databases and querying information. Generating load is thereby achieved (if proper authentication credentials are provided) by querying the standard database `mysql` for information on version and users or the user-provided one directly on its content (more precisely on general information like size and description).

5.4 timeCTRL

Similar to `loadCTRL`, the `timeCTRL` component acts as a *Proxy* [37] for the timing measurement. For that, a uniform sampling mechanism is augmented with specific queries for different services (Requirement C3). However, the strength of this timing module is to provide a generic method for measuring the load by sending indifferent *telnet* messages, i.e. simple messages to a certain socket. A detailed description of that operation follows after some considerations on the underlying sampling process.

5.4.1 Sampling

Detecting load fluctuations is based on measuring response times for a set of queries and thereby verifying substantial differences which should reflect the original load signal. These queries are equally distributed in time, determined through the configuration parameters `timeframe` and `samplerate` (Table 5.8, Requirement C2). Similar to `loadCTRL`, every sample is created in an extra thread each allowing precise timing measurements.

```
<timeCTRL
    ...
    timingstyle      = [telnet|HTTP-GET]
```

continued on next page

continued from previous page

```
        timeframe        = [sec]
        samplerate       = [Hz] >
    </timeCTRL>
```

Table 5.8: Special `timeCTRL` parameters

Special attention has to be paid to the implementation of the timing mechanism. C offers the standard function `gettimeofday()`, which allows timings with a resolution of one microsecond. Nevertheless, its accuracy depends on the kernel and could be as bad as one hundred microseconds on older systems. As `timeCTRL` is designed to measure response times in the magnitude of ten microseconds and above, this issue has to be considered. However, with a current kernel (> 2.6.21) a satisfactory accuracy higher than the actual resolution is achieved and therefore used for the prototype.

There are other possibilities for timing measurements though. `clock_gettime()` is a high-resolution timer and generally able to resolve timings down to one nanosecond. Furthermore, it offers a variety of different clocks like `CLOCK_REALTIME` or `CLOCK_THREAD_CPUTIME`, which both practically achieved an accuracy of about 300 nanoseconds. But for all that, this function proved to be highly unreliable. The reason for that is its utilization of the CPU's timestamp counter (TSC) in contrast to the system clock used in `gettimeofday()`. This 64-bit register counts the number of clock ticks since resetted by the user. But on modern hardware power-saving measures like *downclocking* lead to anomalies in the counting. In addition, multi-core architectures falsify the timing as well since it is not determinable on which core certain threads are running and thus which TSC gets increased. Using `CLOCK_MONOTONIC` falls back to the system clock (with a resolution and accuracy of one microsecond), so for the prototype `clock_gettime()` was discarded and replaced by `gettimeofday()`.

If for some reason the current accuracy of one microsecond won't be sufficient in the future, other low-level approaches like utilizing the soundcard (i.e. sending an instruction which takes a well-defined amount of time) for obtaining stable units of time may be considered.

5.4.2 Timing modules

To perform timing measurements, the sampling technique explained in the previous subsection has to be extended with concrete service requests. For generating these specific samples, two timing modules were implemented (Requirement C3). However, in most cases the first module will be the one of choice. Table 5.9 shows the corresponding module configuration.

```
<timeCTRL
    ... >
    <telnet_TIMINGMODULE
        port          = ...
        burst-length  = ... >
    </telnet_TIMINGMODULE>

</timeCTRL>
```

Table 5.9: Special timing module configuration (telnet)

This module provides timing samples by opening and querying a specific *socket* [43] and measuring the time until the response. The process is similar to the functionality of the standard UNIX tool `telnet`, which allows sending messages to arbitrary sockets, but for perfomance reasons it was manually implemented in C. With the parameter `burst-length`, multiple messages can be sent out per sample for increasing its overall response time and so providing a more stable base for detecting actual differences.

Port 22 is of special interest. Behind that port *ssh daemons* [45] are listening on almost every system providing services. This protocol allows remote maintenance over an encrypted channel, which makes it a standard tool in the repertoire of every system administrator. Querying that service proved to provide excellent signal verification results, so it is the first choice when performing a dependency discovery.

Opening a connection to a socket without a listening service behind it isn't provoking any load at all. If there is a daemon listening, but not answering any requests (due to malformed sample messages or access restrictions), the situation is hardly improved. But in contrast, the ssh daemon inherently listens on almost all servers,

most often with marginal access restrictions and above all with a well-defined and freely accessible communication protocol. By initiating the *protocol version exchange* procedure [46], a certain amount of computation time is awarded for generating a valid answer. This time span is influenced by other processes running on the system, especially by that being under load, and precise enough for verifying the corresponding load signal.

Although the explained procedure is most satisfactory for this prototype, it could be improved by utilizing further parts of the ssh communication protocol. By subsequently invoking the *key exchange* [46] and maybe also the *authentication* procedure [47], even more computation time could be provoked. However, continuing the communication in that respect will lead to entries in the ssh daemon's access logs, which is not the case for the simpler *protocol version exchange* procedure.

Another available timing module is for sending HTTP-GET requests. It is almost identical with the corresponding load module used for the `loadCTRL` application (Sect. 5.3), and can be configured accordingly as shown in Table 5.10.

```
<timeCTRL
    ... >
    <httpGET_timingMODULE
        url          = ... >
    </httpGET_timingMODULE>

</timeCTRL>
```

Table 5.10: Special timing module configuration (HTTP-GET)

This module is designated for the detection of *Affiliations* with the dependency being under specific load as already described in Sects. 4.1.2 and 5.3.2.

5.5 Deployment

To conclude this chapter about implementational concerns, a few suggestions on possible deployment strategies are made. Although it is possible to deploy all components on a single machine, it cannot be recommended for multiple reasons. First,

if the CPU has only a few cores, threads from `loadCTRL` and `timeCTRL` will definitely handicap each other leading to time delays for the samples and to distorted results. Furthermore, if there is only one network card available, even a many-core CPU could not prevent mutual interference.

However, it is reasonable to deploy `fansyCOM` together with one of the other components on a single system, as it is sleeping during an experiment anyway. Nevertheless, there are situations demanding a dedicated machine for `fansyCOM`, e.g. when both subcomponents lie in different subnets and don't have a direct connection due to routing restrictions. It is also imaginable to deploy certain subcomponents directly on targeted systems if no dedicated hardware is available and the corresponding access rights are granted. For the scope of this work, a full deployment structure (i.e. a single machine for every component) has been arranged. Figure 5.2 goes into detail.

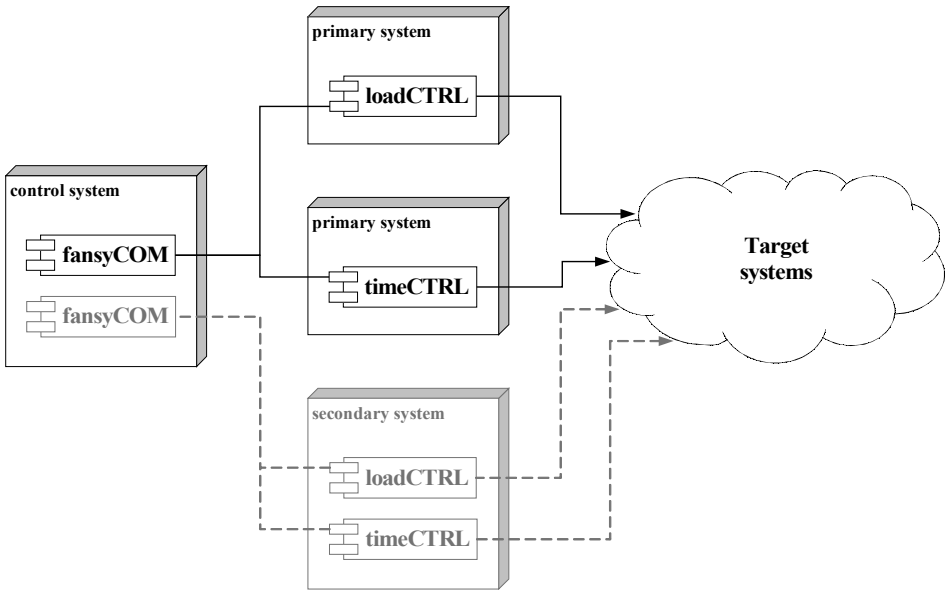


Figure 5.2: Effective deployment

The second instance of `fansyCOM` on the control system and the whole secondary system are present for advanced experiments like multi-signal interference or multi-level dependency analyses, further described in Sect. 6.3.

Targeting systems for timing measurements on the one hand is an easy task. After techniques of asset management have been applied, a full list of services associated to specific hosts should be available. Scanning such a host implies no further knowledge

on its constitution, except for a ssh daemon to be listening on incoming connections. If the SQL timing is selected however, information on connection details like port number as well as user and password credentials are required. Nevertheless this information is generally easy to gather.

A more elaborate task on the other hand could be the expedient targeting of a primary system being dependent on another. Although the corresponding services can be obtained from an asset management process as well, further considerations have to be made for explicitly generating load. Simply querying the root website of a webserver for instance doesn't necessarily provoke the dependency communication. For that, a dynamic website has to be located, which presumably initiates this relationship. Evidence for such a behaviour could be gathered from the corresponding provided data, e.g. if up-to-date information is displayed in a table or the web application itself already implies a dynamic backend. In addition, specific parameters transfered with interactive requests should be closer examined. But in the majority of cases, technical considerations already infere the dynamic structure of a service. For the webserver example, the utilized implementation technology is most significant. Programming languages for actively generating HTML content are most often solid evidence for dependency relations, which can be easily recognized by investigating the corresponding file extensions. If confronted with `.ASP`, `.CGI`, `.PHP`, or `.PY` for example, the HTML content is definitely dynamically generated at runtime and a dependency analysis is promising.

Nevertheless, further investigation of more general targeting strategies is necessary and has to be addressed in future work. Thereby the focus should be on automating this process in combination with common asset management systems along with determining more indications for potential dependency structures of other services as well.

Chapter 6

6 Evaluation

"I didn't think; I experimented."

Wilhelm Röntgen

Although there are domains in academic science where experiments are the result of any theoretical concept and its formal proof, in applied informatics the experiment itself most often serves as validation. This process – closely related to physics – not only proofs an abstract concept, but in the end could moreover deliver solutions finding a way into practical use.

With a solid prototype at hand, it is now possible to evaluate the fundamental idea behind this work. First, this assumes basic experiments to prove general operability, feasibility and stability. By gradually introducing more perturbing variables, a realistic setup is approached. Based on that, further analysis becomes feasible. As already discussed in previous chapters, there is also a variety of interesting phenomena to be examined arising from different fields of possible application. Finally, conclusive studies on potential side-effects together with a systematic and comprehensive series of analyses for getting insights on hidden interrelations complete the evaluation. A productive field test ultimately subjects the prototype to close scrutiny.

6.1 Basic experiments

Before performing advanced experiments or doing progressive research, a stable working base has to be created. For that, a most fundamental experiment for proving the concept is determined, introducing as few as possible variables and external influences. Based on its successful validation, this experiment will then be gradually extended and generalized to the point of resembling practical scenarios. Only at that stage the analysis of anticipated phenomena that potentially occur for realistic fields of application (Sect. 4.1) is possible.

6.1.1 General setting

Basically, all experiments throughout this chapter implement the setting of a webserver querying a database. This common setup is regarded as a standard dependency relation presumably involved in most networked environments of arbitrary scale and thus perfectly supports the experiment design to be of universal and fundamental extent. Nevertheless, future work should include other services as well to gain more broadly based conclusions.

The concrete testing systems are built from standard open source software with default configuration settings. For the webserver, this is the *Apache* HTTP server[1] [48]. A simple PHP script[2] [49] queries a MySQL database[3] [44] filled with random data in a constant or randomized manner. For the basic experiments, it is assumed that the database won't falsify results by utilizing internal caching mechanisms, so the PHP script is set to randomize its queries. However, possible side-effects for static requests are examined in Sect. 6.2.2.

For every deployment node (Sect. 4.3), the underlying hardware[4] is identical. The use of indistinguishable hardware eliminates possible subtleties leading to distortion of results, whereas well-proportioned hardware obviates potential performance bottlenecks which could have an effect on the outcome as well. As a remark, every system is (depending on its usage) equipped with two or three identical `3Com` 100 Mbit network cards. Integrated cards proved to be faster than additional components, so for obtaining comparable results every single link is based on such a separate PCI card. However, Sect. 6.5 performs analyses on unknown hardware as well.

[1] Apache version `2.2.9`
[2] PHP version `5.2.4`
[3] MySQL version `5.0.51`
[4] AMD Sempron 2600+, 1 GB of RAM

6.1.2 Performing basic experiments

The configuration for the experiments is kept similar, allowing direct comparison of all obtained results. For generating the load signal, the parameters stated below (Table 6.1) are applied.

`timeframe:`	`30 sec`
`resolution:`	`5 steps/sec`
`frequency:`	`0.10 Hz`
`amplitude:`	`600 samples/sec`

Table 6.1: `loadCTRL` configuration for basic experiments

As for the dependency response time sampling, the following settings (Table 6.2) are used. Stronger timing measurements by utilizing the parameter `burst-length` are needless for this basic setup.

`timeframe:`	`30 sec`
`samplerate:`	`30 Hz`

Table 6.2: `timeCTRL` configuration for basic experiments

This configuration is considered to be generating `75%` of the maximum achievable load while still providing reasonable results. Further studies on the influence of diversified parameters are shown in Sect. 6.4. With these settings, a theoretical signal as depicted in Fig. 6.1 is expected. Three wave forms are generated within the timeframe, with a maximum amplitude of `600 samples/sec`. Due to the resolution of `5 steps/sec`, the absolute number of load queries is reduced to `120` samples per discretization step in the peak. For generating the load, the `HTTP-GET` module is used, while the timing measurement is handled by an `SSH` sampling; the complete configuration file used for these experiments can be found in the Appendix (Table A.1). The stability of results is considered in Sect. 6.1.4.

The following analysis is first based on verifying the given load in the webserver's response times (*blue diagrams*) and subsequently seeking that signal in the database's sampling (*red diagrams*). Effective results for the dependency discovery are obtained by examining these signals' time domains as well as their representation in the frequency domain and comparing it to the actually acquired initial signal (*green diagrams*). Furthermore, the introduced rating concepts (Sect. 3.3) are applied for allowing direct comparison of different experiments with absolute metrics

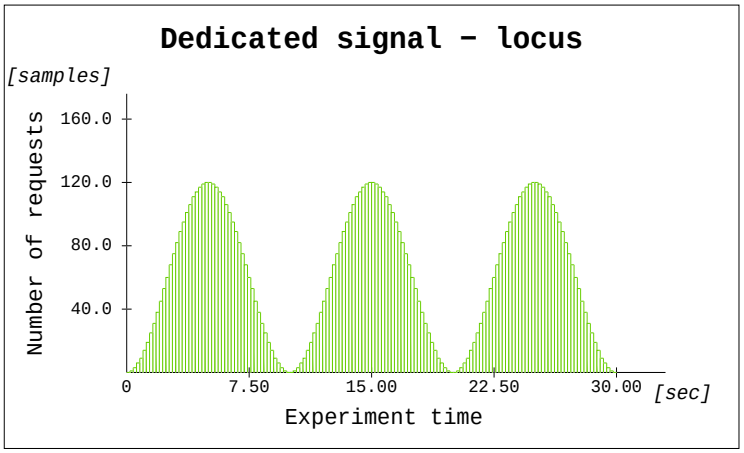


Figure 6.1: Load signal for basic experiments

for the quality of the dependency detection. Comprehensive reports for each experiment including all obtained diagrams in a well-arranged manner can be found in the Appendix along with raw data in the software package attached to this thesis (Table A.4) as well.

6.1.3 Feasibility evaluation

This subsection provides initial studies on basic feasibility. All experiments are performed with the same configuration as described above, but for slightly different setups with increasing complexity. The influence of other variables potentially arising in productive use is addressed later on in Sect. 6.2.

Start with direct cabling

The most basic setup for generally proofing the introduced approach is a direct cabling of all components. Every connection – except for the `fansyCOM` control net which is inactive during the experiment – is given a dedicated line on a separate network card for direct communication. Figure 6.2 shows the setup.

The shared use of a network bridge (i.e. the central switch) is set aside, so distortional effects of competing packet streams are not expected. This setup can be interpreted as the simulation of an arbitrary dependency relation regardless of its actual network link, but more important being analyzed by directly plugging

in the active components to the corresponding systems without any interference to their environments. However, this assumes the availability of independent interfaces on the targeted machines configured for direct access to the services in question. For scenarios where this might not be possible, the next experiments operate on more generic topologies. Nevertheless, this setup delivers practical results under ideal conditions and coming along with that a significant benchmark for further experiments.

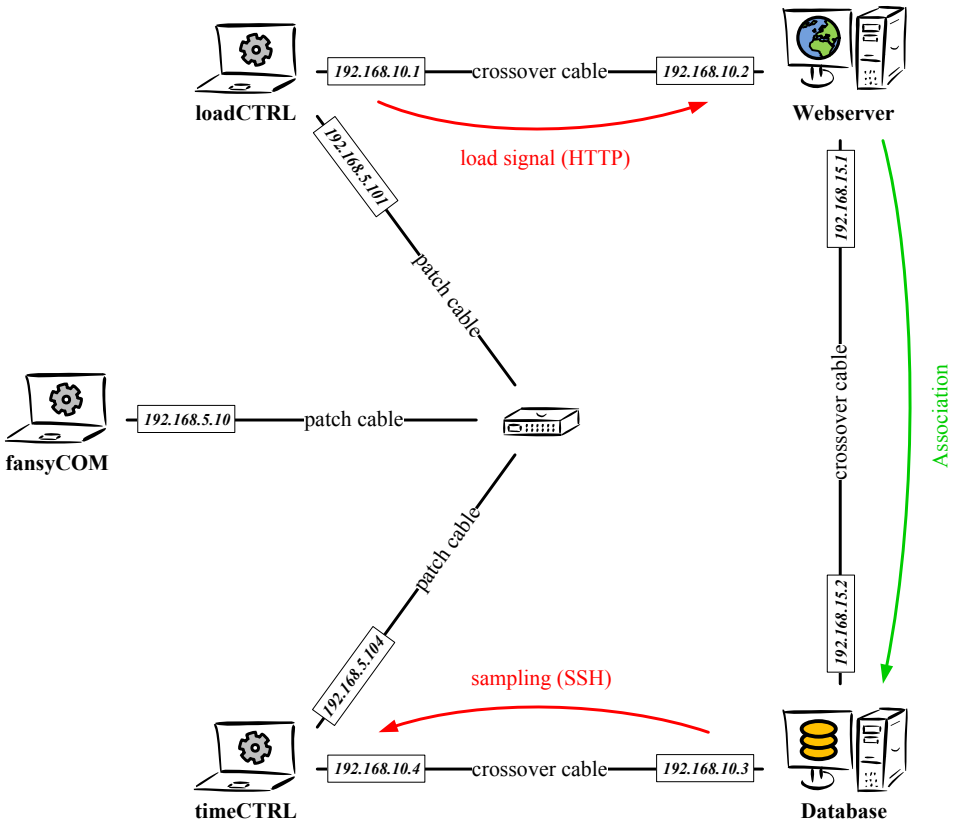


Figure 6.2: Basic experiment (direct cabling)

For exhibiting the webserver's load the response times of all load samples are measured. To prevent irregularities resulting from the uneven distribution of samples, these response times are thereby averaged over each discretization step. The corresponding load signal diagrams (*blue*) are thus basically used as a metric for the webserver's average workload. The equivalent timing signal diagrams (*red*) by contrast are obtained from a constant sampling process and therefore exactly reflect the effective load on the database on a per-sample basis. Both resulting signal plots for direct cabling are shown in Fig. 6.3.

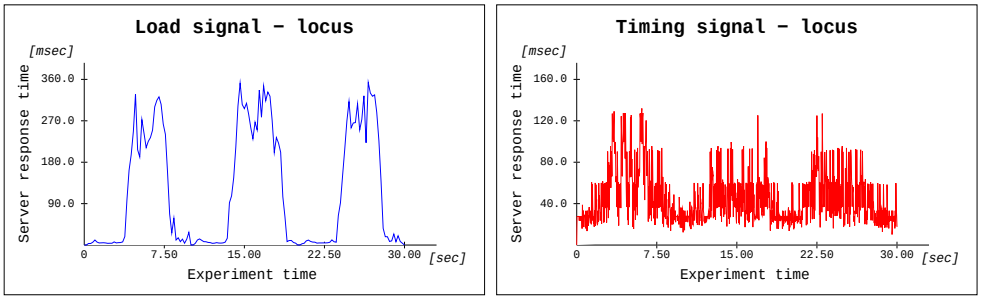


Figure 6.3: Basic experiment (direct cabling) – Signals

Preprocessing this data by means of median filtering as described in Sect. 3.2.1 significantly reduces the amount of noise in the signal which is clearly visible in Fig. 6.4. On that account the frequency analysis and the corresponding rating techniques produce more explicit results, but are also amplifying minor variances. Therefore comparing a series of experiments should be arranged with unfiltered results, while achieving the best possible detection for a particular experiment is better performed with the examination of filtered data. Further analyses on that matter can be found in Sect. 6.1.4.

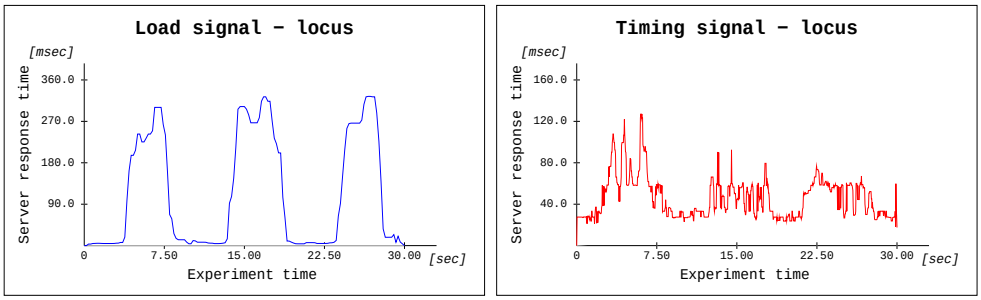


Figure 6.4: Basic experiment (direct cabling) – Filtered signals

With the given load of 120 samples at the peak steps, the webserver's average response times are in the magnitude of 300 milliseconds, which would still be bearable even for productive use. As for the timing measurement, 30 connection attempts per second don't provoke any significant workload at all and remain on a constant level (further considered in Sect. 6.1.4), but the database's internal load caused by the webserver's queries is clearly visible on top of it. Effects of additionally introducing more database workload with higher samplerates getting to the ssh daemon's limit are examined in Sect. 6.4.1. The final results for this first experiment are summarized on the next page.

Result 1

Start with direct cabling

With configured parameters as described in Tables 6.1 and 6.2 and a full direct cabling setup, an excellent load signal with an additionally introduced workload lower than expected could be exerted on the webserver. Moreover, this signal is unambiguously propagated to the database and can already be verified in the time domain with the naked eye (Fig. 6.3).

By making use of frequency analysis, the result is getting even more explicit. The targeted frequency of `0.10 Hz` shows by far the highest amplitude in value. Figure 6.5 thereby depicts the frequency spectrum of the unfiltered timing signal; a comparison to the filtered results along with further details can be found in the corresponding reports. With the introduced rating concept qualitative metrics for measuring the significance of the detection are provided, which in this case result in:

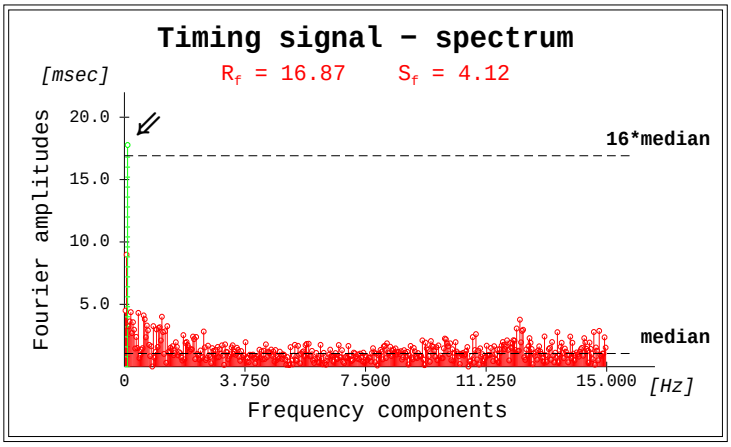


Figure 6.5: Basic experiment (direct cabling) – Result

Median rating:	**16.87**
Inharmonic rating:	**4.12**

These values are for the time being taken as the maximum amount of quality to be generally achievable. All results of further experiments gradually introducing more and more variables will be compared with them. Allowing for reasonable comparison thereby implies stable results, which will be further examined in Sect. 6.1.4.

Appendix:

Figure A.1: `experiment__2009-07-21__19.12.41`
Figure A.2: `experiment__2009-07-21__19.12.46`

Introduction of a semi-switched setup

In practical scenarios it is unlikely for systems hosting services to be directly wired or having direct interfaces for a straight connection to their services. Accessing these services should be possible from all over the network, so an advanced topology has to be considered. However, the steps to the simulation of a live network are performed by degrees. Figure 6.6 shows a setup where all clients query the services over a common network switch, while the targeted system still communicates with its *Association* over an independent line. This experiment is considered to be of rather theoretical nature, as there would be no obvious reason for a direct connection *and* a link to the switched network for the systems to be analyzed. Nevertheless, the setup allows the examination of parallel load and timing signals making use of an existent network infrastructure.

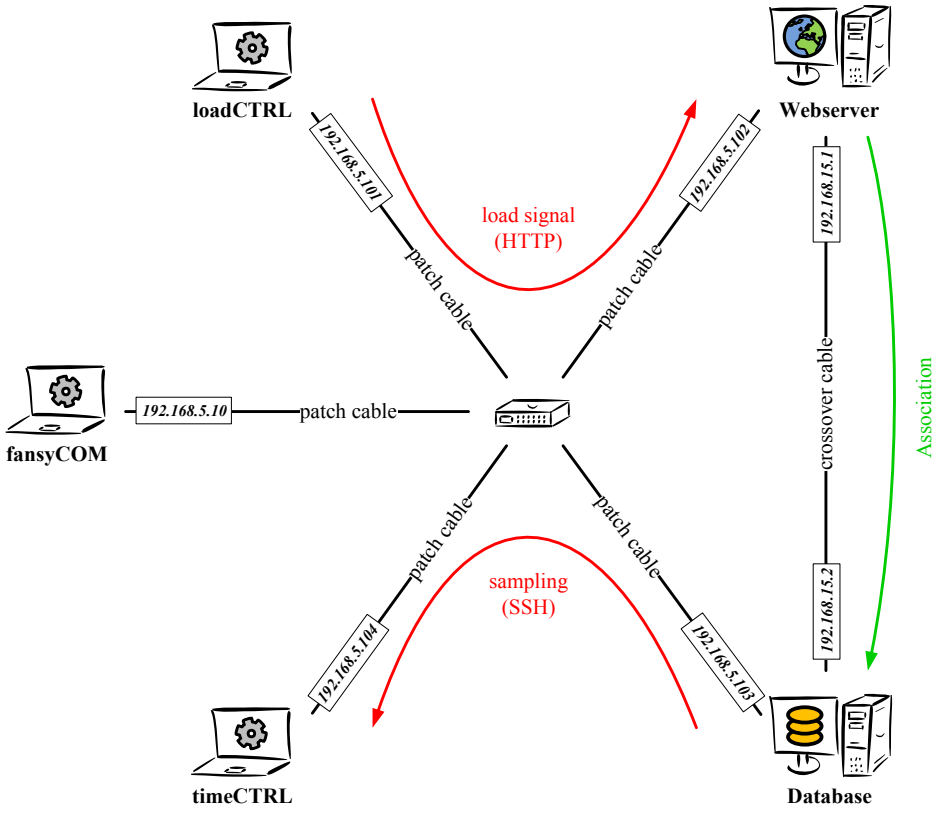


Figure 6.6: Basic experiment (semi-switched)

With this topology, the same results as in the previous experiment are expected. In principle, switches are capable of providing separate full speed links for multiple

connections. Even if several connections would exhaust the maximum bandwidth of 100 Mbit, the switch were still reliable due to the use of an internal high-speed data bus. Getting to its limit with approximately 9,000 samples for the load signal and not even 1,000 timing samples in the time period of 30 seconds is thereby very unlikely. Hence the load and timing signals shouldn't change their characteristics or interfere with each other. Besides, studies on higher traffic volumes charging the switch more significantly are organized in Sect. 6.2.1. The following diagrams (Fig. 6.7) resulting from the same settings as described in Tables 6.1 and 6.2 confirm the assumption.

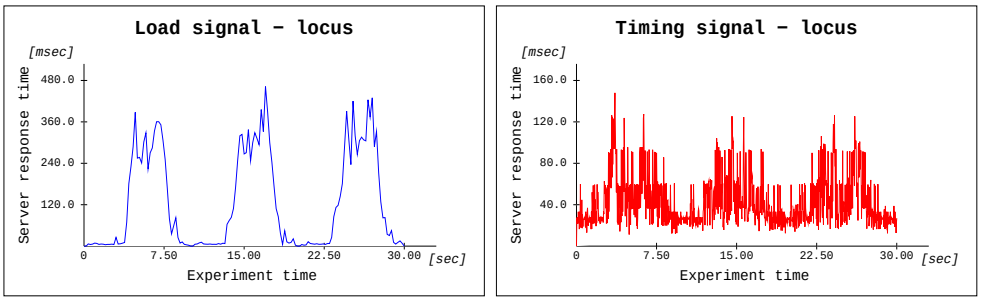


Figure 6.7: Basic experiment (semi-switched) – Signals

The webserver's response times are still in the magnitude of about 300 milliseconds, which indicates that both the webserver and the database communicate in the same undistorted way as for the direct cabling setup. But more important, the timing signal isn't influenced by the parallel load signal either. It exactly matches the signal for the direct cabling. The filtered diagrams in Fig. 6.8 confirm these observations in an even more explicit way.

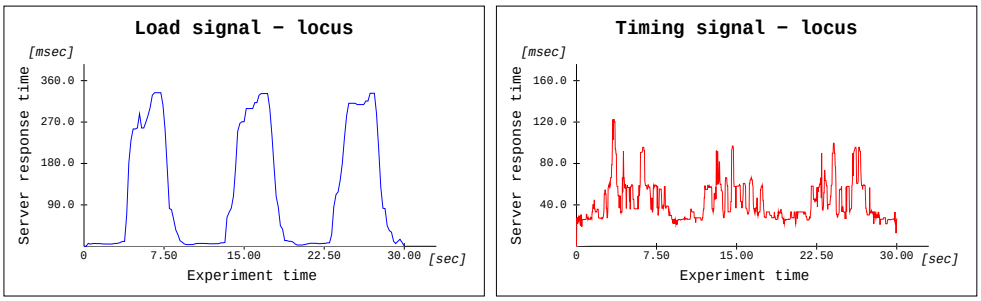


Figure 6.8: Basic experiment (semi-switched) – Filtered signals

The next page exposes the final result by applying frequency analysis and the corresponding rating techniques to the measured timing signal.

Result 2

Introduction of a semi-switched setup

Just as expected, the introduction of a semi-switched topology for the `loadCTRL` and `timeCTRL` components didn't influence their sampling at all. It is exactly in the same magnitude as for the direct cabling. Nevertheless, the frequency spectrum of the timing signal potentially revealing inconsistencies is worth to be examined.

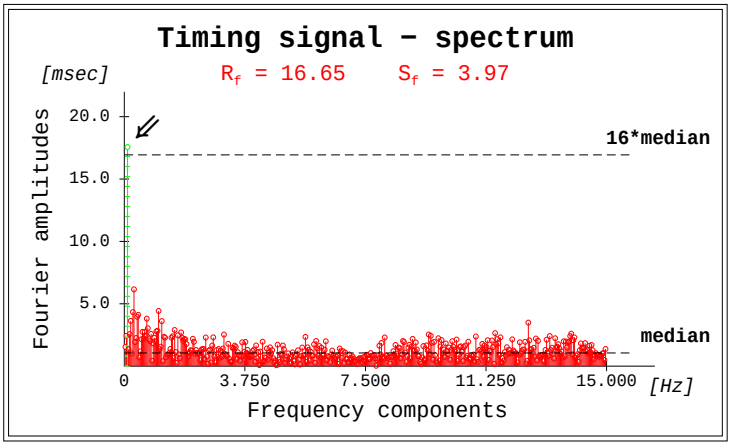


Figure 6.9: Basic experiment (semi-switched) – Result

The dominant frequency part is still the targeted frequency of `0.10 Hz`. The median rating R_f thereby achieves an outcome of almost `99%` compared to the previous experiment, which is legitimately considered to be equal. For the inharmonic rating representing the intensity of noise in the signal, its value shows still more than `96%` regarding the direct cabling result, which can according to Sect. 6.1.4 also be regarded virtually identical. The corresponding rating values obtained from Fig. 6.9 are explicitly stated below, allowing for a direct comparison with the previous result.

Median rating:	**16.65**
Inharmonic rating:	**3.97**

To sum up, it could be verified that performing experiments over a common switch doesn't influence the outcome in a significant way, as long as it is independent from the dependency link. Further generalizations are reasonable at this point.

Appendix:

Figure A.3: `experiment__2009-07-21__20.12.26`
Figure A.4: `experiment__2009-07-21__20.12.31`

Extension to a fully switched topology

With possible side-effects of the network topology itself eliminated, the influence of shared usage of the same network interfaces for both the sampling procedures and the dependency communication are worthwhile to examine. The next step is to connect all involved components to the central switch resulting in a starlike setup commonly used in productive environments (Fig. 6.10). This is considered to be a very general case as it perfectly simulates the use of an existing network structure and the collective use of the targets' dedicated service interfaces. Basically this proceeding represents the analysis of two machines with unknown connection properties by simply plugging in the detection components somewhere into the network.

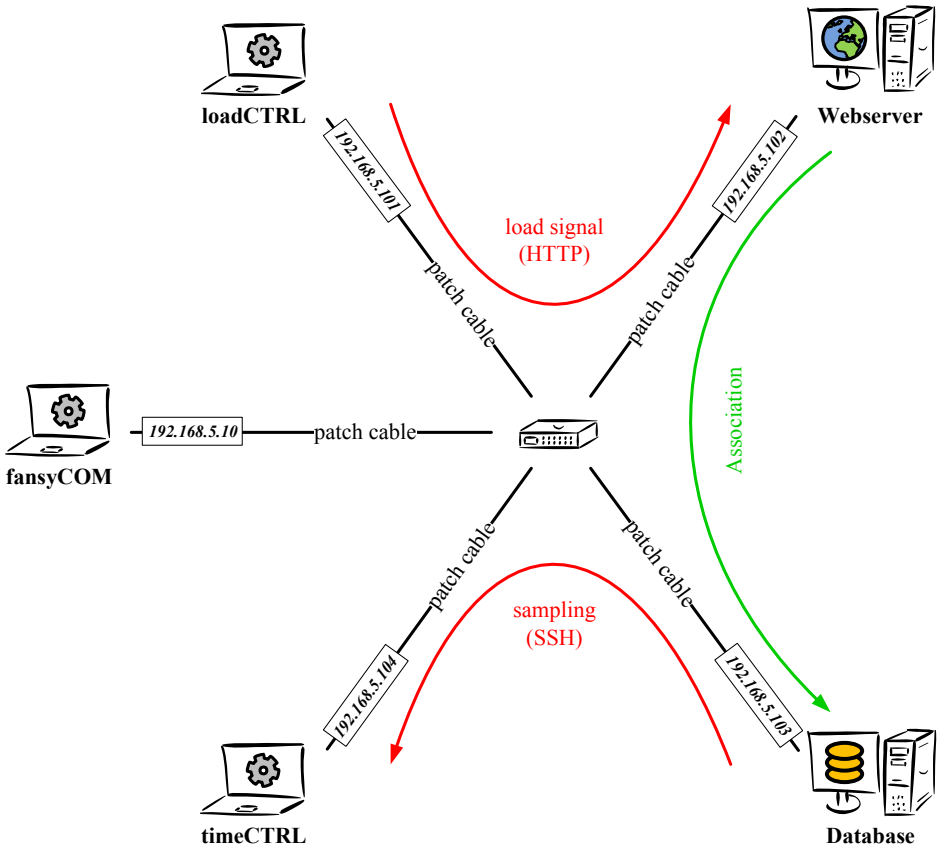


Figure 6.10: Basic experiment (switched)

The already mentioned common use of the targets' standard network interfaces for the load generation and timing measurement could possibly have a significant influence on the measured signals. It potentially leads to some kind of interferences

and additional noise or at least to higher response times in the signals, which are examined in the following. At first, Fig. 6.11 shows the corresponding unfiltered time domain diagrams.

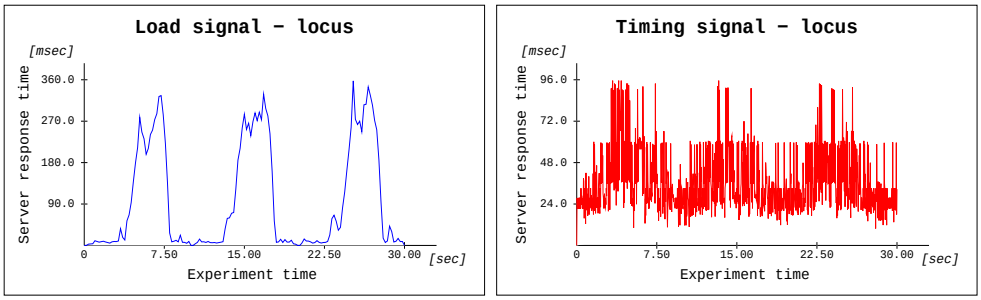


Figure 6.11: Basic experiment (switched) – Signals

Surprisingly, at first sight no additional load can be recognized in comparison to the previous experiments. Both the webserver's response times and the detected load in the database's sampling times remain in the same range. By filtering these signals, Fig. 6.12 reveals no visible difference.

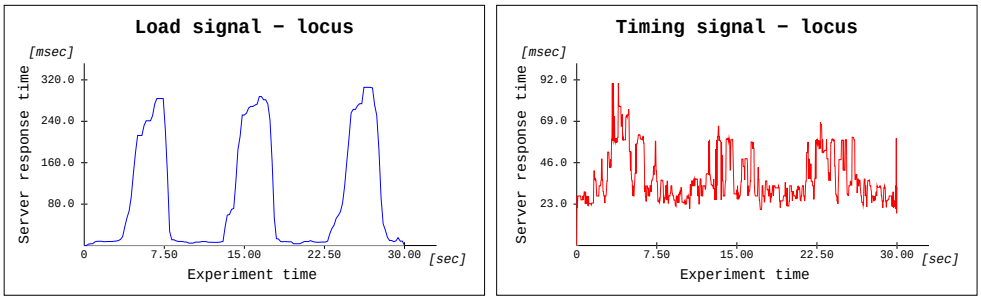


Figure 6.12: Basic experiment (switched) – Filtered signals

Obviously the additional load resulting from the completely shared use of the network switch and the target's corresponding communication interfaces doesn't influence the measured signal in a recognizable manner. However, a detailed examination of the frequency spectrum on the next page remedies remaining doubts. Additionally, Sects. 6.1.4 and 6.2.1 perform further studies on the subject of switched topologies.

Result 3

Extension to a fully switched topology

With also directing the dependency communication over the central switch, no significant changes are introduced in the measured signals' time domains. However, this applies for the corresponding spectrum-based ratings only in a limited form.

The median rating for the unfiltered data decreases by nearly 20% compared to the direct cabling experiment. For the inharmonic rating, the decrease is even slightly higher. Although these differences could be considered stochastically insignificant, Sect. 6.1.4 shows that this is not the case. Greater series of experiments proof that there is a definite relation of lower rating results and the fully switched topology, even though it doesn't influence the overall detection success in a significant way. The achieved absolute rating values (also depicted in Fig. 6.13) are given below:

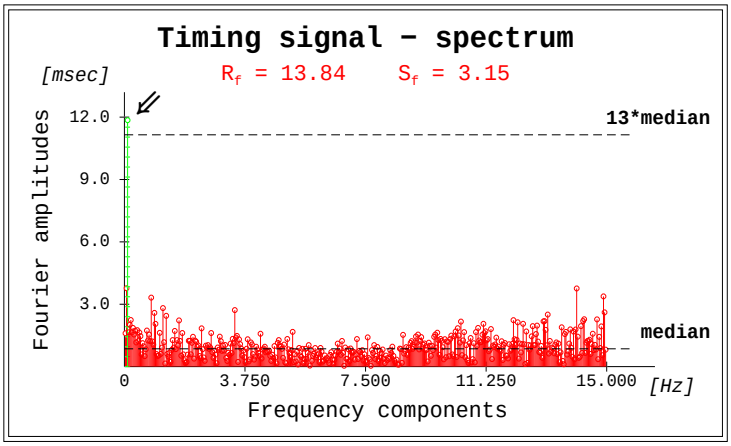


Figure 6.13: Basic experiment (switched) – Result

Median rating:	**13.84**
Inharmonic rating:	**3.15**

It has to be pointed out that these slightly weaker detection results are presumably the effect of random variances stemming from additional stochastic delays introduced by the switch or the common use of the network interfaces. Studies on higher network workloads actively disturbing the measurement are therefore justified and performed in Sect. 6.2.1.

Appendix:

Figure A.5: `experiment__2009-07-21__21.04.09`
Figure A.6: `experiment__2009-07-21__21.04.13`

6.1.4 Stability of results

To conclude this section on basic feasibility, studies on the general stability of experiment results are conducted. For that, series of experiments with the same configuration are performed, allowing the determination of basic variances resulting from random timing fluctuations introduced by inconstant server workloads and network delays. Therefor the direct cabling setup as well as the fully switched topology are examined. Both experiments are configured with the settings as used so far as shown in Tables 6.1 and 6.2. The bulk of corresponding reports is attached to the software package of this thesis included in Table A.4.

Before analyzing the reproducibility of the overall results, the stability of the underlying signal verification is examined. In doing so, the variance of timing samples for the database while idle has to be investigated. The following diagrams (Fig. 6.14) show the time domains of a signal constantly sampling the database for the direct cabling setup, with raw data (left diagram) and applied filter (right diagram).

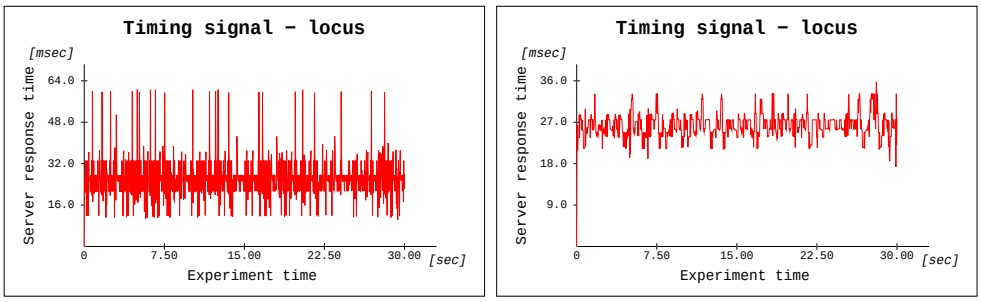


Figure 6.14: Sampling of database while idle (direct cabling)

Disregarding some sporadic outliers in the unfiltered case, the sample times are considered to be basically stable. Table 6.3 provides corresponding statistical data supporting this conclusion.

Average response time:	26.60 msec
Standard deviation:	8.17 msec
Average response time (filtered):	26.32 msec
Standard deviation (filtered):	2.62 msec

Table 6.3: Stability of timing measurement (direct cabling)

By interpreting these statistics, the measured timing signal for the direct cabling setup shown in Fig. 6.3 respectively Fig. 6.4 (depicting peaks in the range of 40–120 milliseconds) is perfectly confirmed, leading to a positive verification of the initial load signal without any ambiguity. For the more general case of a switched setup, Fig. 6.15 shows the corresponding timing diagrams.

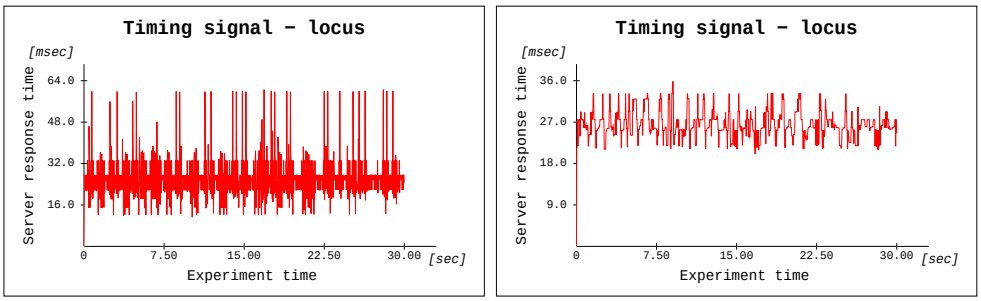


Figure 6.15: Sampling of database while idle (switched)

Again, apart from a few outliers in the unfiltered data, the sampling still shows very small variances with no relation to the actual results obtained in previous experiments. Its statistical stability is calculated accordingly, presented in Table 6.4.

Average response time:	26.78 msec
Standard deviation:	8.48 msec
Average response time (filtered):	26.46 msec
Standard deviation (filtered):	2.87 msec

Table 6.4: Stability of timing measurement (switched)

Here again, the statistics imply an authentic detection of the previously measured signals. Although these values are of the same magnitude as for the direct cabling, it has to be noted that all four of them are slightly higher than the accordant counterparts. Considering the fact that these statistics are based on an experiment with a samplerate of 30 Hz over 30 seconds (which means a base of 900 samples), it can be very well stated that a small amount of additional noise is introduced by the usage of a shared switch respectively of shared network interfaces. However, the previous experiments already showed that this issue has no significant influence on the resulting positive signal verification itself.

Both experiments were also performed with active load generation, but a disconnected link between the webserver and database (i.e. no hardware link for the direct

cabling respectively no software link for the switched setup). This proves that the load generation process for itself has no influence on the timing measurement.

In order to further prove the stability of results, their reproducibility has to be analyzed. For that purpose, series of 50 experiments for each topology are performed and the variances for the relevant results are extracted. Figure 6.16 gives a comprehensive overview with corresponding graphs for the introduced rating concepts (Sect. 3.3) carried out for the direct cabling setup.

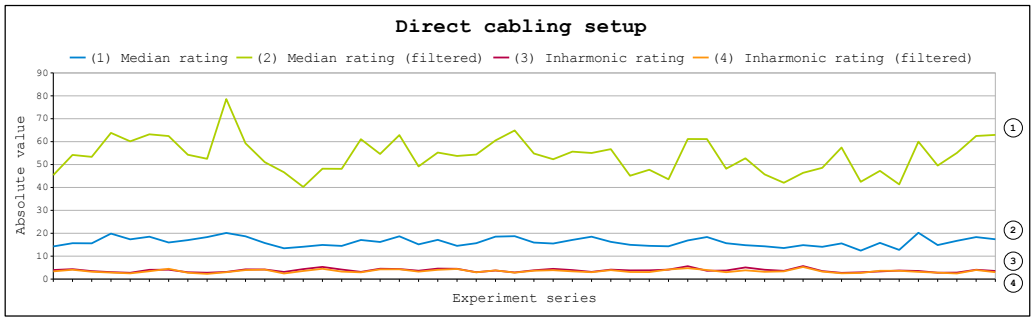


Figure 6.16: Reproducibility of results (direct cabling)

First it can be observed that filtering amplifies all local median rating maxima to a significant extent, but also leads to higher variances. However, the absolute values (1, *green*) are consistently about three times as high as for the unfiltered rating (2, *blue*) and thus provide better absolute results for single experiments. For the comparison of multiple experiments, the more stable unfiltered rating should be preferred though. In contrast, the inharmonic rating concept which is operating relative to the spectrum is hardly affected by the filtering at all (3+4, *red/orange*) as the signal characteristics are fully preserved. Table 6.5 determines more concrete statistical parameters.

	unfiltered	filtered
Average median rating:	16.19	53.86
Standard deviation:	1.93	7.65
Average inharmonic rating:	3.80	3.46
Standard deviation:	0.73	0.67

Table 6.5: Stability of rating concepts (direct cabling)

The statistical analysis confirms the conclusions drawn from the graphical evaluation. On the one hand the increase of the median rating's absolute values is accompanied by an even higher multiplication of the expected distortion when considering multiple experiments. The inharmonic rating on the other hand is nearly immune to the filtering, however a decrease of about 10% has to be accepted. For the switched topology, the situation is similar as shown in Fig. 6.17.

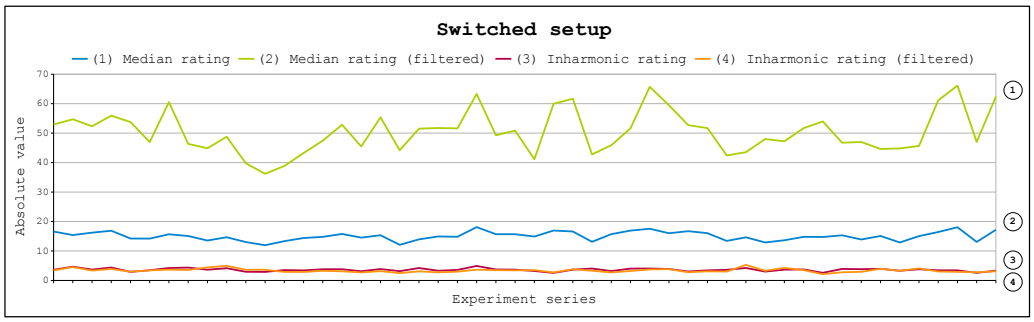


Figure 6.17: Reproducibility of results (switched)

Again, the same conclusions on the stability of results can be drawn as above for the direct cabling. Regarding unfiltered data, the median rating (2, *blue*) produces constant results while the variance is increased when applying the filter (1, *green*). The inharmonic rating (3+4, *red/orange*) by contrast remains unaffected. However, the absolute values of both rating techniques are slightly smaller than for the direct cabling, which can be better observed in the following statistical analysis (Table 6.6).

	unfiltered	filtered
Average median rating:	15.05	50.48
Standard deviation:	1.49	7.13
Average inharmonic rating:	3.61	3.37
Standard deviation:	0.52	0.61

Table 6.6: Stability of rating concepts (switched)

Compared to the previous analysis, small but nevertheless clearly existent differences are noticeable. The average median rating decreases by about 7% independent from the use of the filter, while the reduction for the inharmonic ratings turns out to be only slightly smaller. This decrease accounts for the fact of additional noise

in the switched setup as mentioned earlier. Although the standard deviations are also decreasing by up to 30%, this is considered to be of coincidental origin as there is no obvious reason for the switched setup to be more stable against overall result variances. Future work however should focus on performing considerably larger series of experiments to gain a broader statistical base for drawing further conclusions on that matter. Nevertheless, the stability for allowing the comparison of different experiments regardless of the actual setup is – at least when making use of unfiltered results – given to the full extent.

With the profound verification of a basic feasibility and moreover the fundamental stability of achievable results at hand, further investigation of other variables potentially influencing the analysis is reasonable at this point. The next section therefor carries out supplemental studies on parameters which cannot be controlled in productive use and could thus impose certain limitations for the approach as a whole.

6.2 External influences

Although the quality of the results of the experiment can be actively influenced by a variety of configuration settings combined with different deployment strategies, there are potentially other factors having an uncontrollable effect on the outcome. This section focuses on two different cases analyzing such influences. At first, the studies on switched network topologies are extended to the simulation of additional unrelated traffic being present on the network. After that, examinations on the influence of differing queries being sent from the webserver to the database are conducted, which are generally unknown in productive use and thus could provoke arbitrary load. Especially modest queries are of further interest as they possibly invoke too low workloads not being detectable by the sampling procedure any more.

6.2.1 Cross-traffic

The previous experiments performed in Sect. 6.1 already introduced a fully switched network and lead to the conclusion that a common switch reduces the verification quality in an insignificant but nevertheless measurable way. However, in live environments a large increase of network load is anticipated, so analyzing different kinds of so-called *"cross-traffic"* appears necessary. Figure 6.18 shows the possibilities.

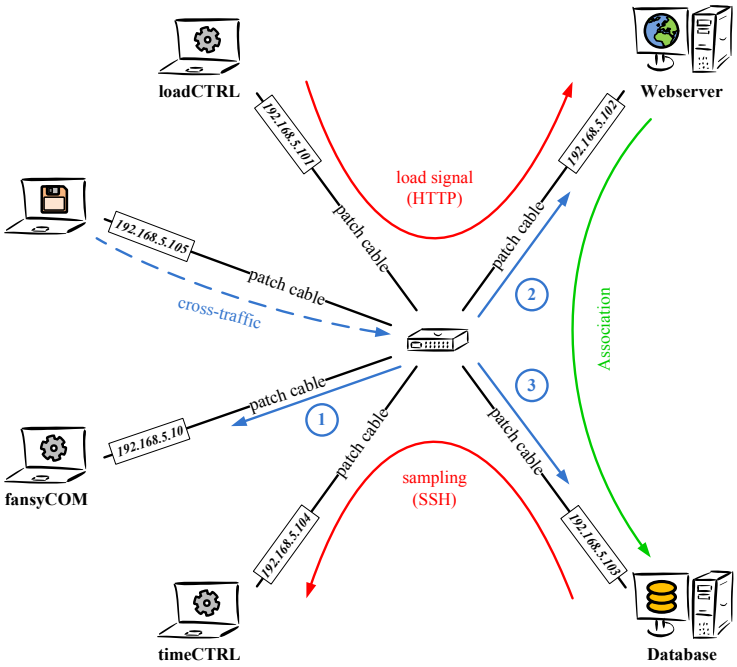


Figure 6.18: Studies on cross-traffic

For generating the different forms of additional traffic, a supplemental system also directly connected to the central switch is used. From there, large amounts of data are sent into the network, divided into three stages:

① **Background cross-traffic** Initially the switch itself is being analyzed. A full speed data transfer to the controlling machine is thereby loading the switch more significantly, but doesn't influence the clients nor the targeted systems.

② **Webserver cross-traffic** By constantly sending data to the webserver while being unrelated to the experiment, a higher workload is simulated. This setting can be interpreted as the concurrent utilization of the webserver by other consumers, the database however experiences still the same load.

③ **Database cross-traffic** Analogue to the previous setup, the database's internal load and therefore its response times are increased simulating to be simultaneously queried by other systems as well.

Background cross-traffic

As already explained in Sect. 6.1, the switch can host multiple parallel connections due to its internal high-speed data bus. For scenarios where network loads tax switches to their limits leading to significant increases in response times, improvements in capacity management are obviously more important for that environment than operational dependency detection, so this setting is omitted. Testing the technique on higher workloads is nevertheless reasonable for verifying that the switch itself has only an insignificant effect on the outcome. Figure 6.19 shows the corresponding result for an additional full speed link to be handled by the switch.

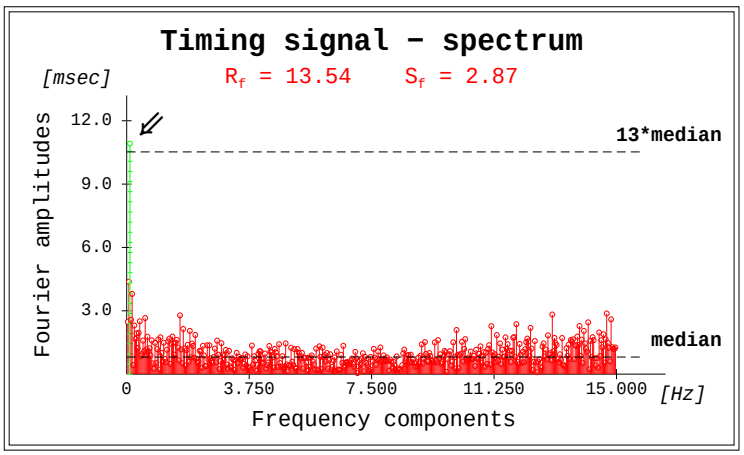


Figure 6.19: Background cross-traffic – Result

Both rating values are exactly in the magnitude of the switched setup performed without any additional traffic (Fig. 6.13), and are thus also slightly lower than for the direct cabling setup. Regarding the full reports

Figure A.7: `experiment__2009-07-22__03.39.58`
Figure A.8: `experiment__2009-07-22__03.40.12`

attached to the appendix, it can be also stated that no changes in the webserver's or database's response times are visible. The experiment is apart from small variances covered by the statistical evaluation in Sect. 6.1.4 virtually identical to the corresponding previous analysis on switched networks in Sect. 6.1. According to that, these facts are not considered as a new result, but again confirm that the use of a network switch doesn't have any significant impact on the overall dependency detection, even if it is taxed with higher workloads.

Webserver cross-traffic

Compared to unrelated background traffic, additional load directed towards the webserver is of much higher importance. First, other networked services probably running on that machine could lead to interference with the load generation. Furthermore, the web content itself may be queried during an experiment in an unknown scale. Studying this kind of cross-traffic is again simulated by simultaneously transfering large amounts of data to the webserver, putting its network interface under heavy load. Figure 6.20 shows the resulting signal plots.

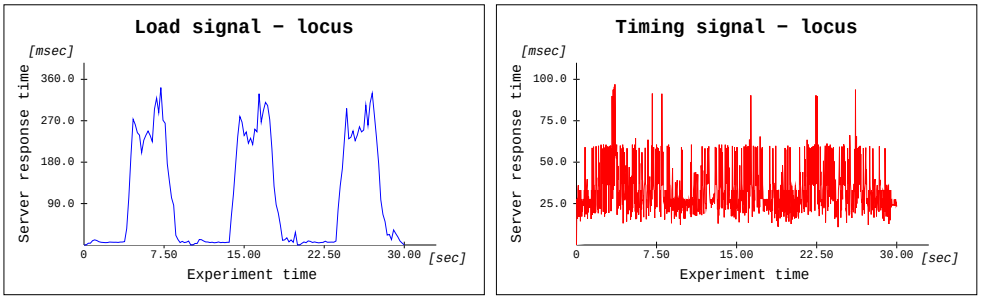


Figure 6.20: Webserver cross-traffic – Signals

Surprisingly, the webserver's response times are still in the same magnitude of 300 milliseconds as previously observed. This implies that most of the time until the response is received is spent on the database querying its content. The times for sending packets over the link and the additional network workload are thereby obviously insignificant for the overall response times. However, this fact doesn't eliminate the possibility of higher traffic influencing the exact load signal generation by delaying incoming load queries as well as outgoing database requests. Figure 6.21 supports this assumption as the timing signal is superposed with significant noise.

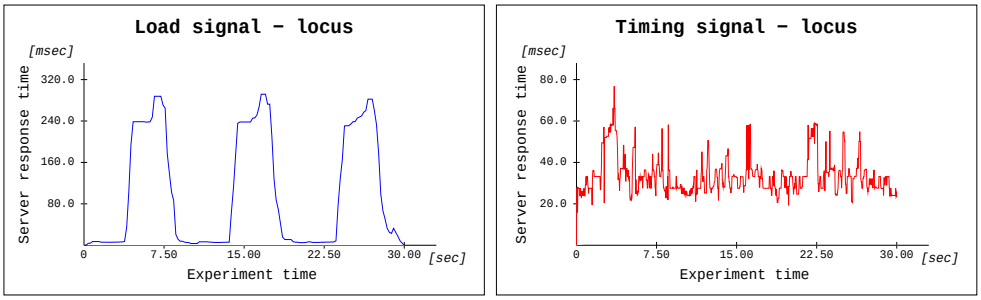


Figure 6.21: Webserver cross-traffic – Filtered signals

Result 4

Webserver cross-traffic

Although heavy secondary network load directed towards the webserver didn't affect its response times at all as they are apparently influenced mainly by the database's performance, the timing signal showed serious distortion effects. The reason for that are presumably small delays in the webserver's input and output traffic.

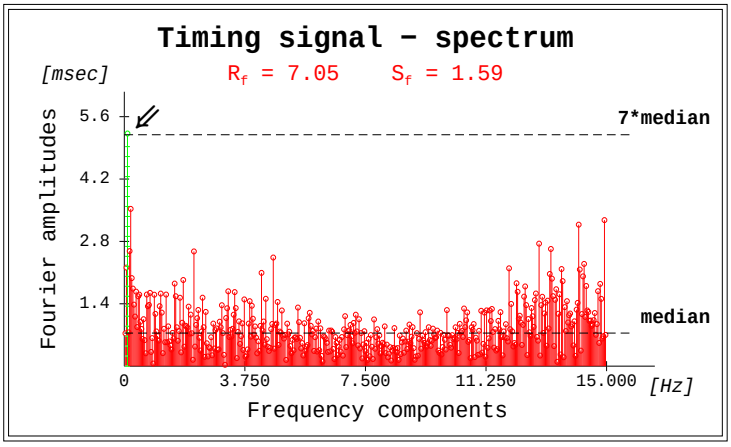


Figure 6.22: Webserver cross-traffic – Result

The frequency spectrum of the timing signal shows far more noise than observed so far, which also originates in the webserver's delayed queries. However, the actual targeted frequency is effectively recognized and still dominant in the signal. Nevertheless, the rating concepts show an almost halfed outcome compared to the switched setup while idle, for both the median and the inharmonic rating. This supports the assumption of a blurred load signal generation as the individual samples are pushed out of place leading to strong high-frequency parts in the peaks (recognized by the inharmonic rating) and an overall decrease in intensity (verified with the median rating). The absolute values are:

Median rating:	**7.05**
Inharmonic rating:	**1.59**

Even though the results are less significant, the frequency verification itself is definitely positive. Thereby the signal processing techniques reveal their strength as a manual investigation in the time domain would already come to its limit.

Appendix:

Figure A.9: `experiment__2009-07-22__08.05.40`
Figure A.10: `experiment__2009-07-22__08.05.45`

Database cross-traffic

Even more interesting than the already studied webserver cross-traffic is the examination of additional load on the database. As the previous experiment revealed, the webserver's response times are mainly depending on the database's load. Therefore, higher workloads on it should effect both systems alike. The simulation of secondary traffic is thereby again achieved by simultaneously transfering huge amounts of data onto the database. Figure 6.23 presents the corresponding measured signals.

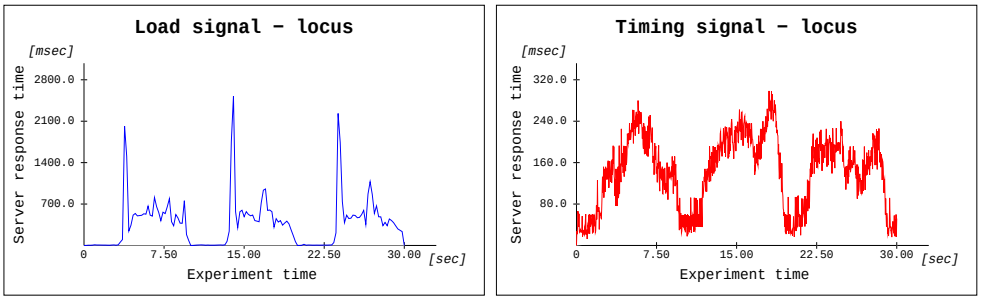


Figure 6.23: Database cross-traffic – Signals

Clearly, the increase in traffic has severe influences on both signals measured. Just as expected, the webserver's response times rise, in fact by about 100% with outliers in the magnitude of several seconds. The timing signal is increased even more dramatically and gets four times as high as for the setup without additional load. The filtered diagrams below (Fig. 6.24) exhibit these facts more obviously.

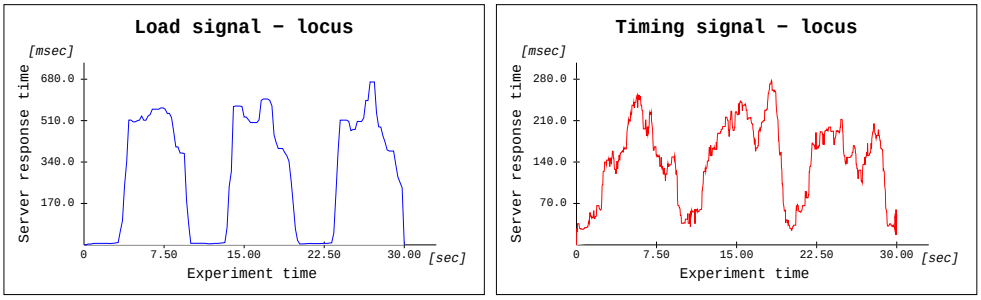


Figure 6.24: Database cross-traffic – Filtered signals

The reason for this dramatic change are most certainly differing optimization strategies. While a webserver is tuned to handle as much requests as possible, databases are by contrast optimized for queries on large amounts of complex data.

Result 5

Database cross-traffic

The influence of increasing traffic on the database is far more significant than for the webserver in the previous experiment. Both systems are affected at the same time, which implies that the database is the bottleneck of the dependency relation. Secondary network load thereby results in an amplification of the internal load, resulting in severe delays for the measured timings.

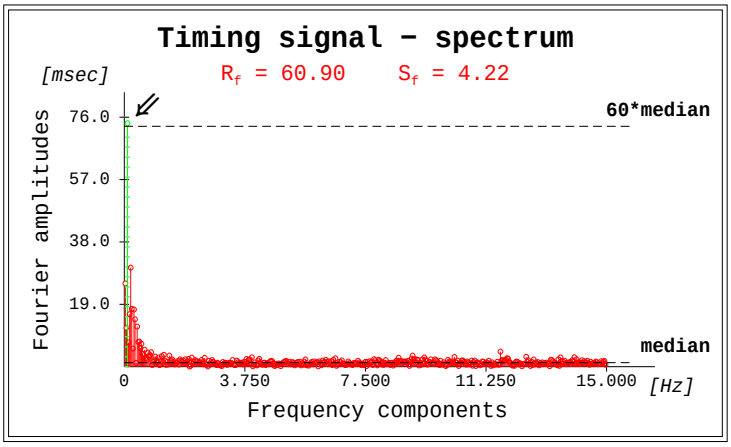


Figure 6.25: Database cross-traffic – Result

With the additional load, the signals can be measured more effectively due to greater variations in the response times. The frequency spectrum reveals other frequencies being still in a similiar ratio than for the unstressed setup (measured by the inharmonic rating); however, the targeted frequency gets amplified intensely. An additional outcome of about 400% is achieved for the median rating, while the inharmonic rating remains still in the same magnitude. Details can be found in the Appendix.

Median rating:	**60.90**
Inharmonic rating:	**4.22**

Although the impact on the analysis has to be considered dramatic, the verification still succeeds for this experiment, ironically in a much better way.

Appendix:

Figure A.11: `experiment__2009-07-22__08.53.19`
Figure A.12: `experiment__2009-07-22__08.53.23`

Summary

In summary, the experiments on different kinds of cross-traffic provided interesting insights on the behaviour of single components. First, the switch itself was examined by charging it more significantly with traffic unrelated to the experiment. Previous results showing only insignificant influence could thereby be verified accordingly. With this information at hand, productive deployments independent from the underlying network topology are considered basically viable.

As for the subsequent analyses on cross-traffic geared towards the systems to be scanned, a few remarks have to be made. At first, the database could be identified as the bottleneck of the dependency relation. This implies that the database and therefor the measured signal is far more susceptible to cross-traffic, even if it is directed towards the webserver. However, it has to be pointed out that these results are dependency-specific, which means other services could possibly exhibit a different behaviour. Furthermore, the simulation of load is achieved by transfering large amounts of data being completely unrelated to the corresponding services. Although this de facto increases the load on the targeted system, another case has to be considered. It is possible that other consumers query the webserver or the database in the same manner as the actual load generation, potentially leading to further kinds of distortion, which is together with studies on other services suggested for future work.

To conclude, cross-traffic in its different variants has a noticeable effect on the dependency discovery, however even with maximum distortion significant results can still be obtained without any difficulty. The next subsection focuses on another uncontrollable parameter presumably having an impact on the outcome as well.

6.2.2 Database caching

Generating load on the webserver is basically achieved by determining a dynamic website supposingly being dependent on other services. This dependency however is considered as a *"black box"*, which means no knowledge is available on the actual constitution of the information exchange. Obviously, queries to a huge database making use of complex searching algorithms are expected to produce recognizable load on the corresponding system. However, if the database applies sophisticated caching techniques or is only filled with a small amount of data, the situation is possibly to change.

The next experiment analyzes the influence of the database query by simulating a significant caching process. For that, the same switched setup and identical configuration parameters as described in Tables 6.1 and 6.2 are utilized again. More important, the caching is simulated by querying no information at all. This means, the webserver connects to the database, but immediately closes the connection after success. With this procedeeding, the minimal possible response times for a dynamic website requesting cached information are obtained. Figure 6.26 show the results.

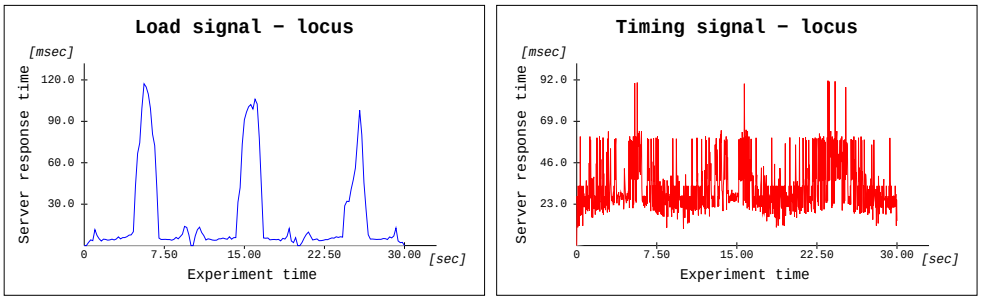


Figure 6.26: Database caching – Signals

Apparently, the faster webserver response times meet the expectations to the full extent. In fact, compared to the experiments so far, the timings decrease by more than 50%. As for the database sampling, no significant changes are recognized, except for fewer outliers in the magnitude of 90 milliseconds. This implies that the connection establishment with its user authentication procedure itself consumes most of the computation time. Accordingly, the nature of the actual query is of hardly any consequence. For completeness, Fig. 6.27 shows the corresponding filtered diagrams, the frequency analysis is performed on the next page.

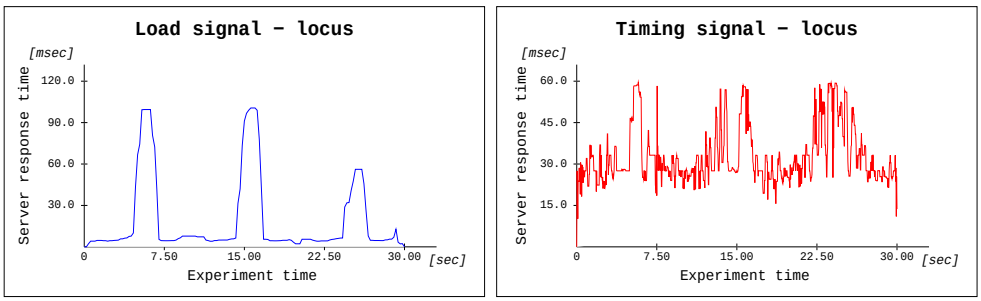


Figure 6.27: Database caching – Filtered signals

Result 6

Database caching

With the simulation of a caching algorithm providing requested information in zero time, the webserver's response times were more than halfed. However, as the experiments on cross-traffic already demonstrated these nevertheless still significant response times originate on the database for the most part.

This implies that the connection procedure is an invariant reliably invoking enough load for the dependency detection, regardless of the actual query. The load signal's frequency analysis thereby shows only small decreases for the absolute rating values. A decrease of about 15% for the median rating respectively 25% for the inharmonic rating compared to the standard switched experiment in Sect. 6.1 is considered plausible. With less computation time for each request, the overall load is reduced resulting in a weaker signal and more significant variances. The corresponding rating values are stated below:

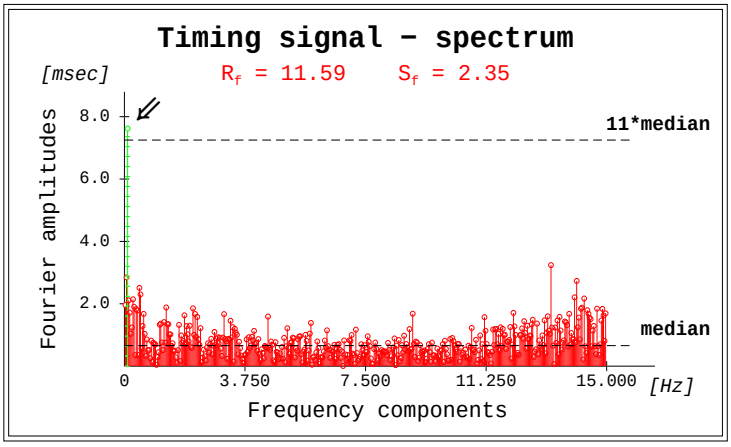


Figure 6.28: Database caching – Result

Median rating:	**11.59**
Inharmonic rating:	**2.35**

It could be determined that a considerable part of the overall load is solely based on the initial connection establishment. This result is promising for productive use as no premises to the actual dependency constitution have to be made. Future work however should focus on the examination of different service relations as well.

Appendix:

Figure A.13: `experiment__2009-07-22__15.29.20`
Figure A.14: `experiment__2009-07-22__15.29.25`

6.3 Advanced experiments

With the experiments performed throughout the last sections, a solid base for examining advanced setups is formed. As already shown in Chps. 4 and 5, there is a variety of interesting constellations regarding different dependency relations. At first, all experiments so far focused on the detection of *Associations* (Sect. 3.1), i.e. that machines which are essential for the system in question to function properly. However, the inverse scenario – finding *Affiliations* respectively that machines being dependent on the targeted system – is to be studied as well. Furthermore, for productive use it could be valuable to perform multiple experiments simultaneously. Therefore, two conceivable side-effects are closely investigated. Finally, the propagation of load is further examined, potentially "hopping" from one system to another in a chain of dependencies.

All experiments performed in this section are designed to determine further capabilities of the active dependency discovery. However, additional work regarding implementation, automation and stability would still be necessary for productive use. On that account these experiments have to be considered as proof of concept only and should be extended in future work. Nevertheless, fundamental insights on advanced fields of application are already provided guiding the way to a prospectively flexible and comprehensive dependency discovery operating on complex constellations and network topologies as well.

6.3.1 Reverse direction

In many cases the method for detecting dependencies as constituted in previous experiments is perfectly sufficient. With directing the load towards a particular system and verifying that signal on possible dependencies (i.e. *Associations*), outstanding detection results could be obtained. For detecting depending systems respectively *Affiliations* relying on the system in question the same procedure can also be applied, however this would require the generation of load on many primary systems while repeatedly verifying it on the target. A more practical approach for this scenario is to specifically put the target under load and measure timings on its parent systems, as depicted in Fig. 6.29. For that, the previously used `SSH` timing technique is obviously insufficient due to the fact that the dependency relation wouldn't be invoked at all. On that account, a service-specific timing procedure engaging the dependency communication is to be studied.

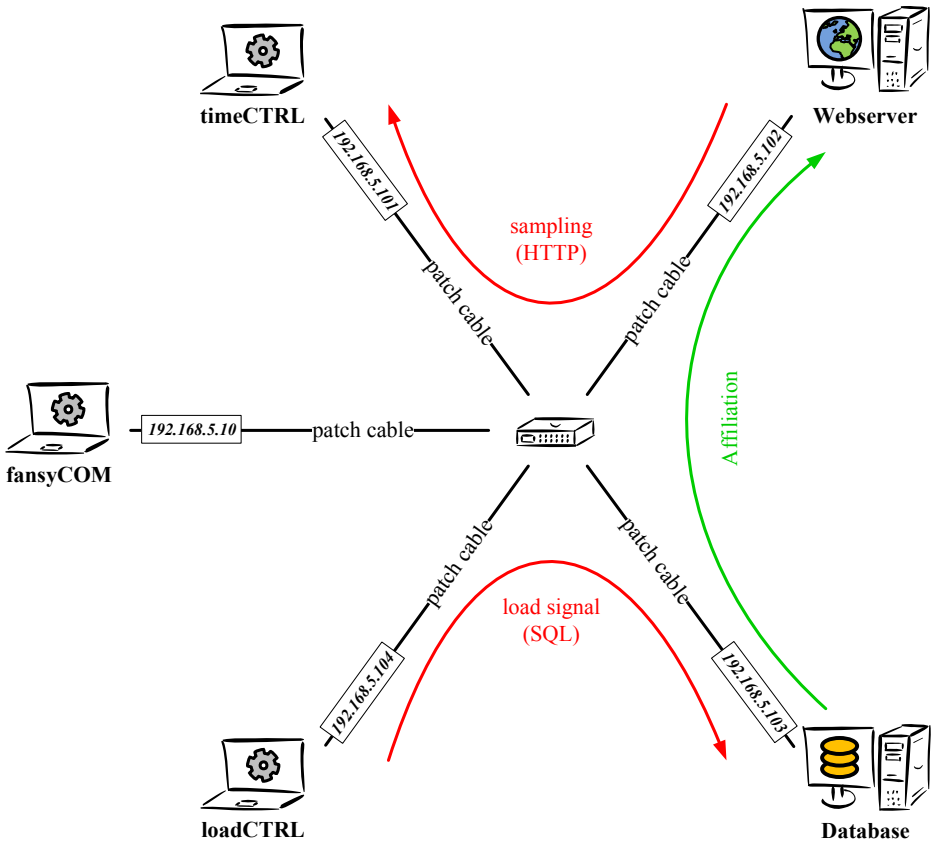


Figure 6.29: Reverse direction

For this experiment, a webserver-database relation is examined again. The load is now directed towards the database generated by specific `SQL` queries (Sect. 5.3.2), while the timing on the webserver is measured by making use of the `HTTP` module (Sect. 5.4.2). For generating the load, proper access credentials have to be provided, whereas further information on the data itself is not mandatory as the query of standard tables is also possible. The corresponding configuration is listed in Table A.2.

This approach significantly reduces the overall load to be generated by taxing the dependency itself whereas all depending systems are only moderately queried in the constant sampling process. Accordingly all measured systems are confronted with a minimum of additional load influencing the existing services to a negligible extent. However, this benefit is traded in for the necessity of additional knowledge about connection details for querying the database, which could be unavailable in certain environments. Therefore the application of this reverse direction procedure has to be considered depending on the actual situation.

With configuration settings similar to previous experiments performed in Sect. 6.1, the load signal on the database could be clearly generated. It has to be noticed thereby that the load is only achieved by querying the MySQL standard database `mysql` and more specific its table `user`. This kind of information request – along with the connection establishment itself – produces enough load for verifying the signal with the constant HTTP sampling. In fact, a samplerate of `2 Hz` is perfectly sufficient for unambiguously detecting the signal, as shown in Fig. 6.30.

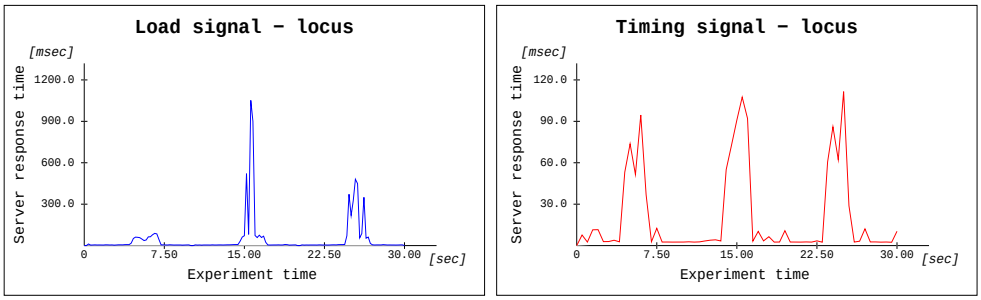


Figure 6.30: Reverse direction – Signals

Applying the median filter reduces nearly all outliers and thus shows less intense peaks for both measured signals (Fig. 6.31). Compared to the previous experiments (e.g. Fig. 6.12), these peak response times are in the same magnitude which means higher times for the load and low times for the corresponding verification measurement. But most important, the systems are interchanged for this setup, so the load is not generated on the primary system, but coming from its dependency.

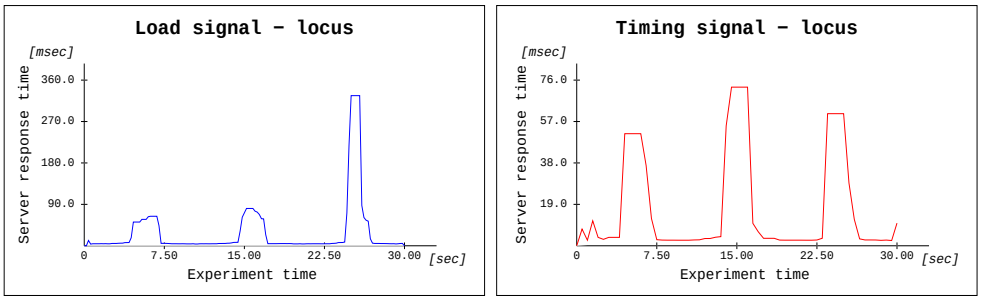


Figure 6.31: Reverse direction – Filtered signals

The next page focuses on further signal processing including the application of rating techniques and summarizes the result.

Result 7

Reverse direction

Apparently the technique of reverse scanning allows a significant decrease of load on main systems introduced with the sampling process while still obtaining unamibguous signal detection due to considerable load directly generated on the dependency. Figure 6.32 provides the corresponding frequency analysis for the timing signal.

The frequency spectrum shows the targeted frequency of `0.10 Hz` holding the highest amplitude in value as already expected by the time domain examination above. Thereby the corresponding rating concepts are not directly comparable to previous experiments as the sampling is performed with a significantly lower samplerate of `2 Hz`. Nevertheless, the results perfectly confirm the verification, with absolute values as stated below. The corresponding reports can be found in the Appendix.

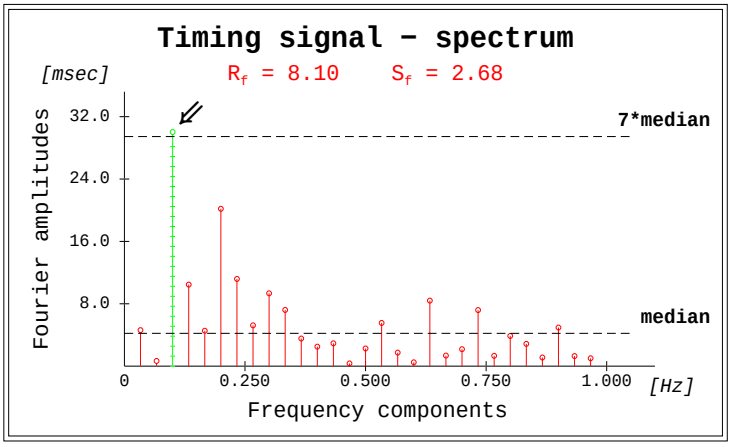


Figure 6.32: Reverse direction – Result

Median rating:	**8.10**
Inharmonic rating:	**2.68**

To conclude, when directly querying the database for generating the load is possible, the influence of the search for *Affiliations* on primary systems can be reduced to a insignificant minimum also bearable in sensitive environments. However, the propagation of this service-specific load signal has to be examined for other services as well, which is highly suggested for future work.

Appendix:

Figure A.15: `experiment__2009-07-27__11.17.48`
Figure A.16: `experiment__2009-07-27__11.17.52`

6.3.2 Frequency overlays

For a complex network topology, constellations like *Fuzzy Association* or *Shared Affiliation* as described in Sect. 3.1 could probably occur. Basically these relationships cover the fact that a single information request can affect a particular system in multiple manners, especially if it is hosting more than one service or a fundamental service of the network. Considering the *"domain name system"* (DNS) for instance, this issue is getting more clear. When targeting a certain host, it potentially initiates a DNS query before requesting information from its actual dependency. That dependency itself could possibly query the DNS service as well, which would in summary lead to the overlay of two identical signals on the system hosting the DNS. To study such a behaviour, an adequate simulation setup as shown in Fig. 6.33 is used.

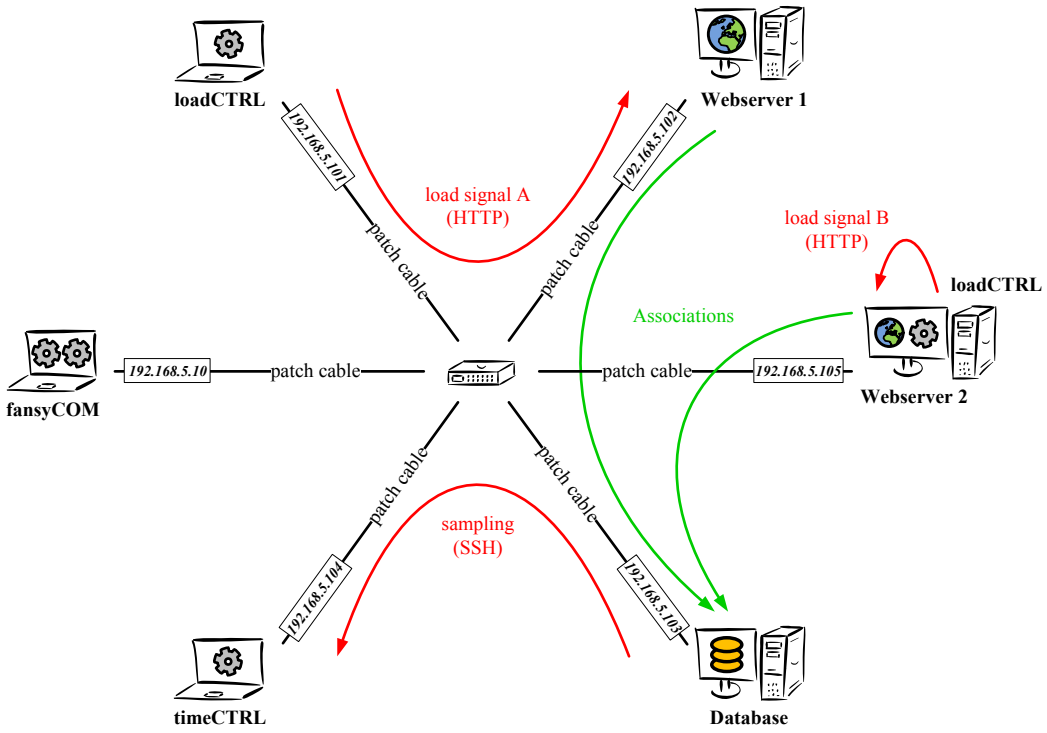


Figure 6.33: Frequency overlays

In general, the basic setup and the configuration settings are identical to the switched setup as described in Sect. 6.1. However, an additional webserver is used for simultaneously generating an identical signal. Therefor the `loadCTRL` application is deployed directly onto the second webserver, while it is operated by an additional

`fansyCOM` instance on the controlling system. A secondary timing measurement is thereby irrelevant, so only one `timeCTRL` client is in use. Both webservers communicate with the database over the central switch, but previous studies in Sect. 6.2.1 already showed that this issue has no influence on the outcome. Nevertheless, both signals could interfere with each other on the database, resulting in higher load and thus higher response times for the webservers as well. Figure 6.34 shows the corresponding measurement diagrams.

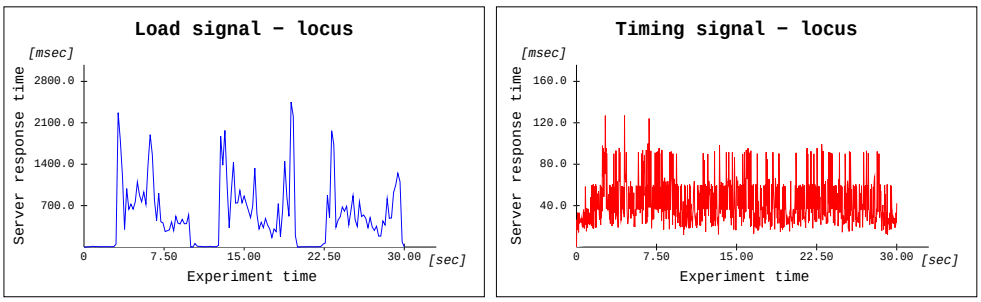


Figure 6.34: Frequency overlays – Signals

The effect of two identical signals querying the database is significant. While the webserver response times are more than doubled, the timing measurement hardly allows the recognization of the corresponding load signal. This applies for the filtered diagrams in Fig. 6.35 as well. For a clear view, the load signal diagrams of the second webserver are left out as they show exactly the same behaviour.

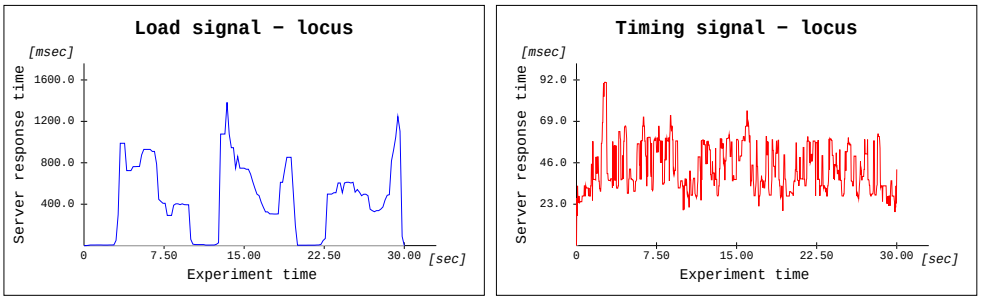


Figure 6.35: Frequency overlays – Filtered signals

Apparently, the application of two load signals leads to significantly higher response times on the webservers, originating in higher load on the database. In addition, the timing signal gets more indistinct with broader peaks resulting from the interference of both signals or possibly also from a slight displacement. The corresponding frequency analysis is performed on the next page.

Result 8

Frequency overlays

Although constellations leading to frequency overlays are considered to occur only in very complex environments, such a setup is simulated nonetheless. Thereby the response times for the webservers are more than doubled, while the timing measurement produces more obscure results. However, the frequency analysis again qualifies the amount of disturbance and the overall success of verification more clearly.

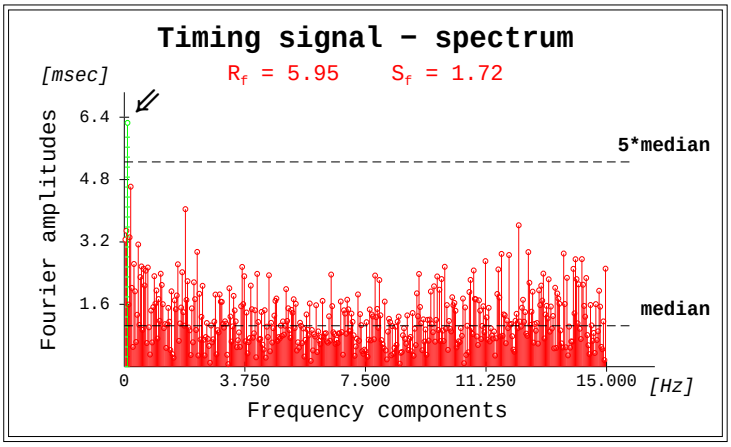


Figure 6.36: Frequency overlays – Result

While the targeted frequency is still dominant, the extent of noise compared to previous experiments is significantly increased. In relation to the common switched setup as described in Sect. 6.1, the outcome of the rating techniques are approximately halfed. This perfectly matches the fact of two parallel load signals and its corresponding doubled response times. A linear correlation could be suspected, however further studies on that subject would be necessary. But for all that, the absolute rating values positively verify the frequency detection without any doubt and are given below:

Median rating:	**5.95**
Inharmonic rating:	**1.72**

In spite of these rather serious side-effects possibly to occur in large environments, the frequency verification itself is nevertheless still viable to the full extent, so this subject is not to be considered of further importance at the moment.

Appendix:

Figure A.17: `experiment__2009-07-27__13.31.38`
Figure A.18: `experiment__2009-07-27__13.31.42`

6.3.3 Multi-signal interference

In large environments and especially for obtaining full dependency maps as described in Sect. 4.1.3 parallel scanning of multiple systems is reasonable for reducing time expenses and administrative efforts. Although the previous experiment already showed that multiple identical signals could have significant influence on the outcome while still allowing unambiguous dependency verification, the utilization of different load patterns is to be studied for that scenario as well. Basically the setup is similar (as given in Fig. 6.33), however the two load signals differ in its characteristics. With this approach multiple hosts could be scanned for dependencies in parallel allowing an explicit mapping of particular dependency connections to specific signals. For the case of several systems querying the same dependency, this experiment provides an effective simulation of the interference to occur.

The common signal parameters as listed in Table 6.1 are again used for the load generation. In addition, a second signal is generated with half the amplitude and an inharmonic frequency of `0.1666 Hz` superposing the original signal. This weaker high-frequency signal is thus expected to overlay the common stronger low-frequency part. Both theoretical signals are given below (Fig. 6.37).

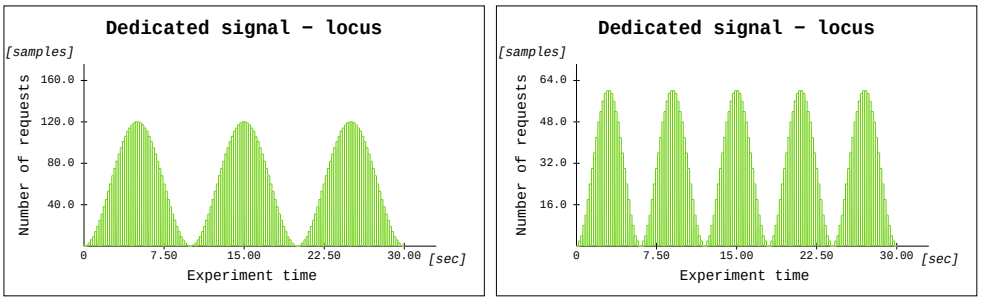


Figure 6.37: Multi-signal interference – Analytical signals

The left diagram shows the low-frequency signal with an absolute peak amplitude of 120 samples, while the right diagram represents the overlay with doubled frequency and halfed intensity. Figure 6.38 shows the corresponding webserver response times and database timing measurements. Due to the areas of sparse load samples between the wave forms in the left diagram the additional peaks of the second signal wouldn't be visible in the webserver timings at all. Therefore Fig. 6.38 shows that second signal's response times, while the timing measurement on the database refers to the left experiment analysis as it is performed only once.

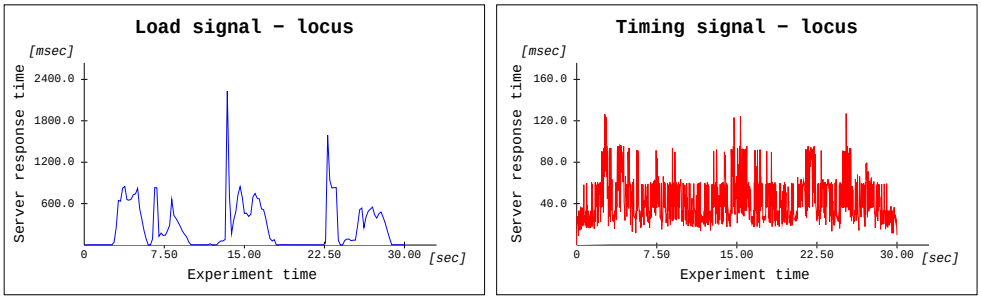


Figure 6.38: Multi-signal interference – Signals

Although the targeted signal contains three wave forms, the influence of the second high-frequency signal is clearly visible. The superposition of this secondary signal leads to a total of five wave forms contained in three areas resembling the original signal. The database measurement by contrast allows no distinct recognition, but the corresponding frequency parts should nevertheless be present in the frequency spectrum. Figure 6.39 shows the diagrams with applied median filtering.

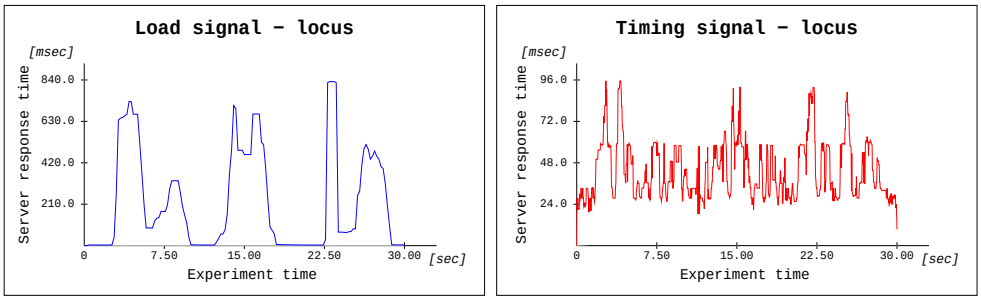


Figure 6.39: Multi-signal interference – Filtered signals

The webserver response times are approximately doubled compared to a single-signal experiment due to the additional load present on the database. However, the webserver itself isn't put under load more than in previous experiments. In fact, the load signal's amplitude is set to half for the second signal, leading to even less workload for the corresponding webserver. The additional delays completely originate on the database, as already pointed out in Sect. 6.2.1. The timing measurement by contrast doesn't produce higher response times, but as in the previous experiment the peaks are again significantly broader resulting in a more indistinct signal.

For the following analysis, the rating concepts refer to the frequency spectrum of the common low-frequency signal (Fig. 6.37, left plot) as they are initiated with the timing measurement corresponding to that signal.

Result 9

Multi-signal interference

The time domains of the measured signals above already lead to the assumption that directing the load towards several hosts in parallel while making use of varying load patterns allows concurrent dependency verification even if several hosts depend on the same system. Thereby a careful examination of the corresponding frequency spectrum is necessary for obtaining measurable results supporting the detection.

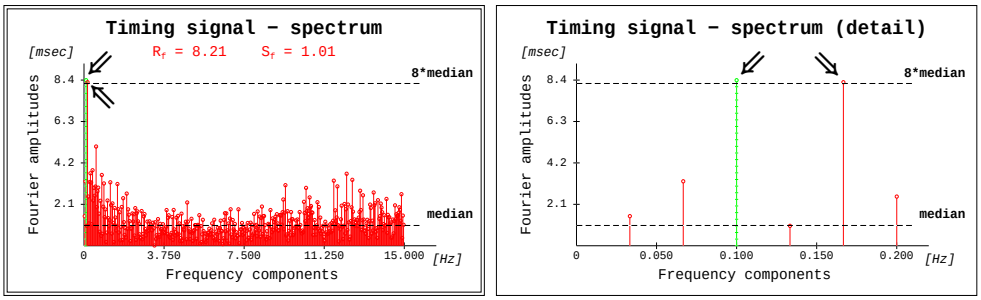


Figure 6.40: Multi-signal interference – Result

Although the targeted frequency of `0.1 Hz` is significantly present in the spectrum, it is clearly contrasted with the frequency part of `0.1666 Hz`. This result was already anticipated by the examination of the time domains, however the frequency analysis and its corresponding rating concepts provide a vital base for explicitly detecting multiple signals in a single sampling. The achieved values are:

Median rating:	**8.21**
Inharmonic rating:	**1.01**

This astonishingly unambigous result implies that scanning techniques with multiple signals could very well be further automated by filtering out single frequencies (and their harmonics) to detect specific dependencies between different hosts simultaneously. Further studies and improvements for the prototype based on this successful proof of concept are therefore highly suggested.

Appendix:

Figure A.19: `experiment__2009-07-27__15.23.07`
Figure A.20: `experiment__2009-07-27__15.23.11`

6.3.4 Multi-level dependencies

Finally another interesting constellation for large environments is to be examined. When targeting a certain system, its dependency can be accordingly verified as all previous experiments already confirmed. However, this dependency itself could be dependent on another system, leading to a chain of dependencies. Unfortunately this second dependency connection is potentially only invoked on request of the initial system, so a separate examination of this constellation is probably impossible. On that account, the propagation of load over multiple hosts is studied, simulated with the setup presented in Fig. 6.41.

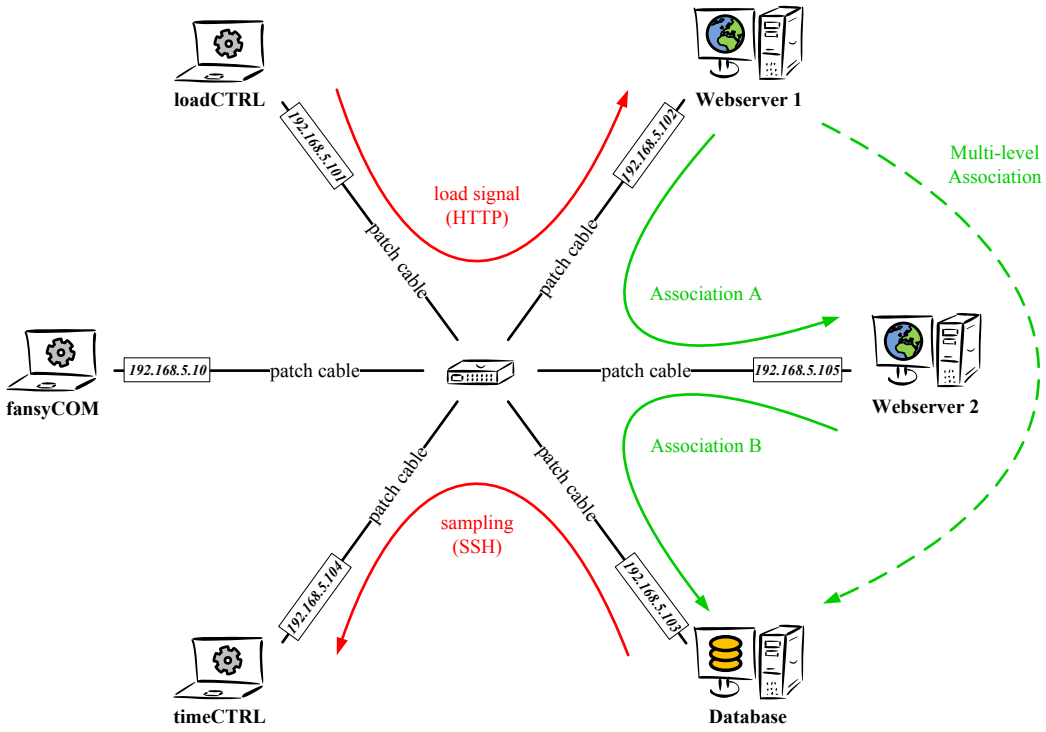


Figure 6.41: Multi-level dependencies

In addition to the basic switched setup in Sect. 6.1, a second webserver is introduced representing the dependency for the first system. Furthermore, this second webserver depends on the database, so a query to the original system "hops" along the dependency chain and eventually results in a query to the database, where it should basically be detectable as well. However, a small increase in the targeted webserver's response times and a slight degradation of the overall verification quality is expected. Figure 6.42 shows the corresponding measurement diagrams.

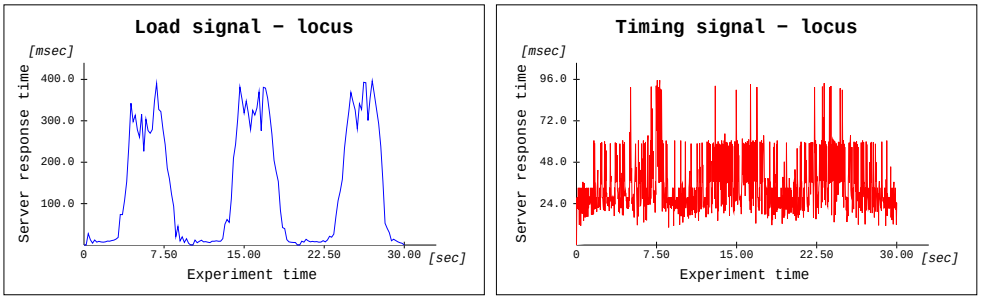


Figure 6.42: Multi-level dependencies – Signals

In fact, the webserver response times slightly increase by about 10% compared to the standard switched setup, while the timing signal doesn't reveal any differences at all. This applies also for the filtered signals depicted in Fig. 6.43.

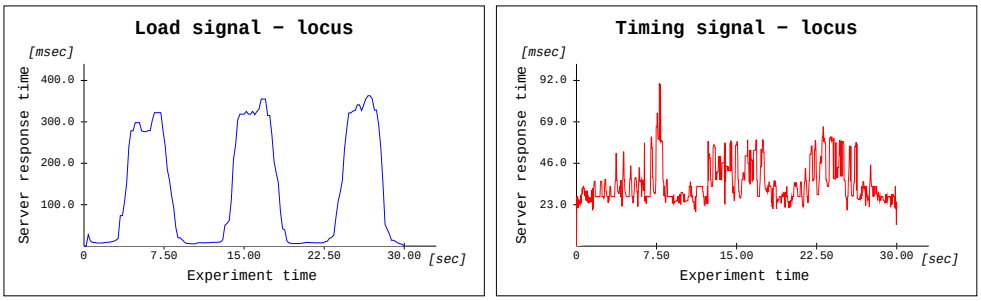


Figure 6.43: Multi-level dependencies – Filtered signals

Apparently the introduction of an additional step in the dependency connection doesn't have any significant influence on the measured signal except for the anticipated small increase in the webserver timings. In general, the basic feasibility of detecting multi-level dependencies can thus be confirmed. However, there are further parameters which could have a significant impact on the outcome. The most important variable is thereby the actual constitution of the second webserver. If for instance caching techniques are in use, the database could be queried less frequently. Furthermore, the second webserver could take measurable computation time before querying the database, which has to be considered as well. Another problem when detecting dependency chains is that it cannot be determined how many links are utilized, only the dependency itself is verified. For that, further techniques like actively distorting a single link could lead to the desired result. Further studies on this subject are suggested.

Result 10

Multi-level dependencies

When analyzing large scale environments dependency chains are likely to occur. For the basic simulation however, the measured signals are only insignificantly affected. Nevertheless, the examination of the frequency spectrum is reasonable.

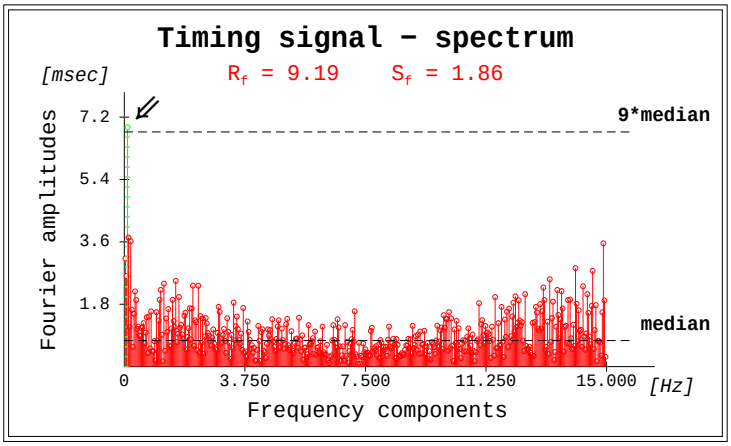


Figure 6.44: Multi-level dependencies – Result

Just as for the basic switched setup performed in Sect. 6.1, the targeted frequency of `0.1 Hz` has by far the greatest amplitude in value. However, the amount of noise present in the signal is considerably increased for the multi-level dependency constellation. This is also covered by the decrease of the inharmonic rating value by about `40%`, which represents the intensity of the highest distortional frequency part contained in the signal. As for the median rating, its value is decreased in the same magnitude. The absolute achieved results for both rating concepts are given below:

Median rating:	**9.19**
Inharmonic rating:	**1.86**

To conclude, the detection feasibility of multi-level dependencies is confirmed in the scope of a proof of concept. However, further studies on the influence of different variables as mentioned above are recommended for future work.

Appendix:

Figure A.21: `experiment__2009-07-27__16.51.10`
Figure A.22: `experiment__2009-07-27__16.51.15`

6.4 Studies on further interrelations

Up to this point, it can be stated that the presented approach is basically feasible. Furthermore, its stability is verified and potential side-effects from external influences are ruled out. Advanced considerations for sophisticated application scenarios complete the studies on operational use. In addition, this section provides further insights on different components involved in the analysis process. At first, the webserver and database are examined on potential repercurssions resulting from the fundamental load generation idea behind the approach of active dependency discovery. Therefore corresponding workload minima for reducing the influence on productive environments are determined. Beyond that, the implementation is checked for correctness like the constituted independence of certain parameters. Finally, large studies on the influence of different parameter settings for the prototype are performed, with the objective of finding potential maxima in the detection results.

An approximation of the number of possible parameter combinations clarifies this approach. Considering only the parameters `amplitude`, `resolution`, `frequency` and `samplerate` with reasonable limits and partitions, the total number N_{conf} can be estimated as follows:

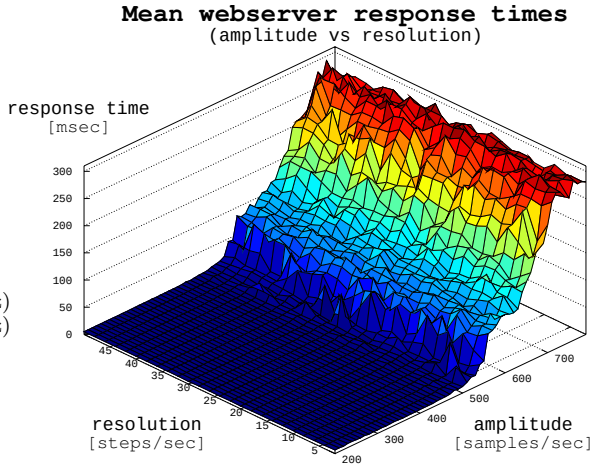
$$\begin{aligned} N_{conf} &= |\texttt{amplitude}| \cdot |\texttt{resolution}| \cdot |\texttt{frequency}| \cdot |\texttt{samplerate}| \approx \\ &\approx |[200, ..., 800]| \cdot |[1, ..., 50]| \cdot |[0.033, ..., 1.000]| \cdot |[1, ..., 30]| \approx \\ &\approx 600 \cdot 50 \cdot 30 \cdot 30 = \underline{27 \cdot 10^6} \end{aligned}$$

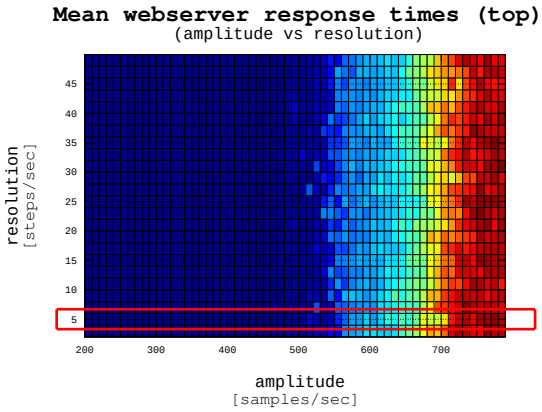
This huge value clearly points out the magnitude of the cardinality for the space of parameter combinations, although it is already simplified and some parameters are still left out. It is therefore reasonable to conduct large studies regarding the interdependence of certain parameters or potentially discovering local maxima or minima in different fields of interest.

6.4.1 Workload analyses

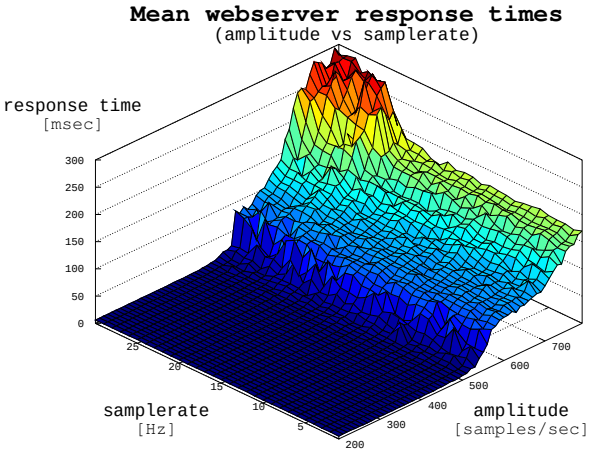
Actively interfering with productive environments is a highly sensitive task. In principle, the influence on main systems have to be held as small as possible. For that, studies on average workloads are performed in the following. The objective thereby is to determine decisive differences resulting from the use of various parameter combinations. To start with, Fig. 6.45 shows the webserver analysis series.

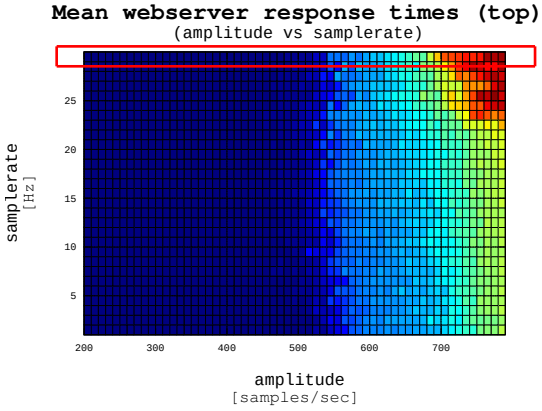
Invariants:
samplerate (30Hz)
frequency (0.1Hz)



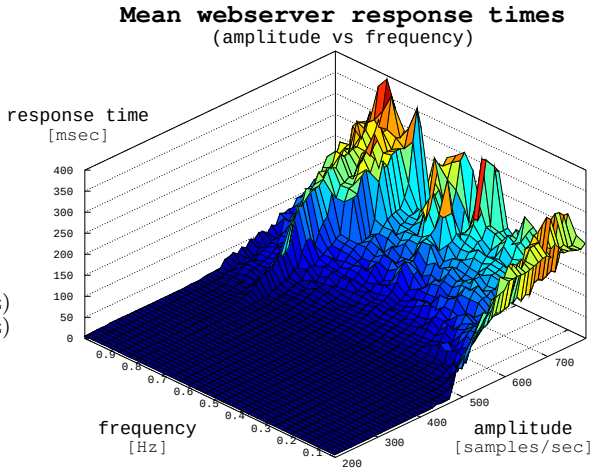


Invariants:
resolution (5Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
samplerate (30Hz)



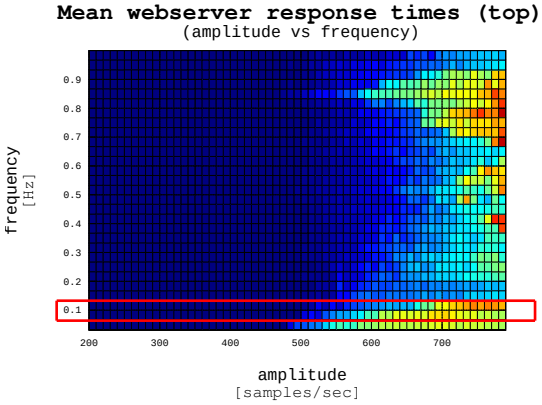


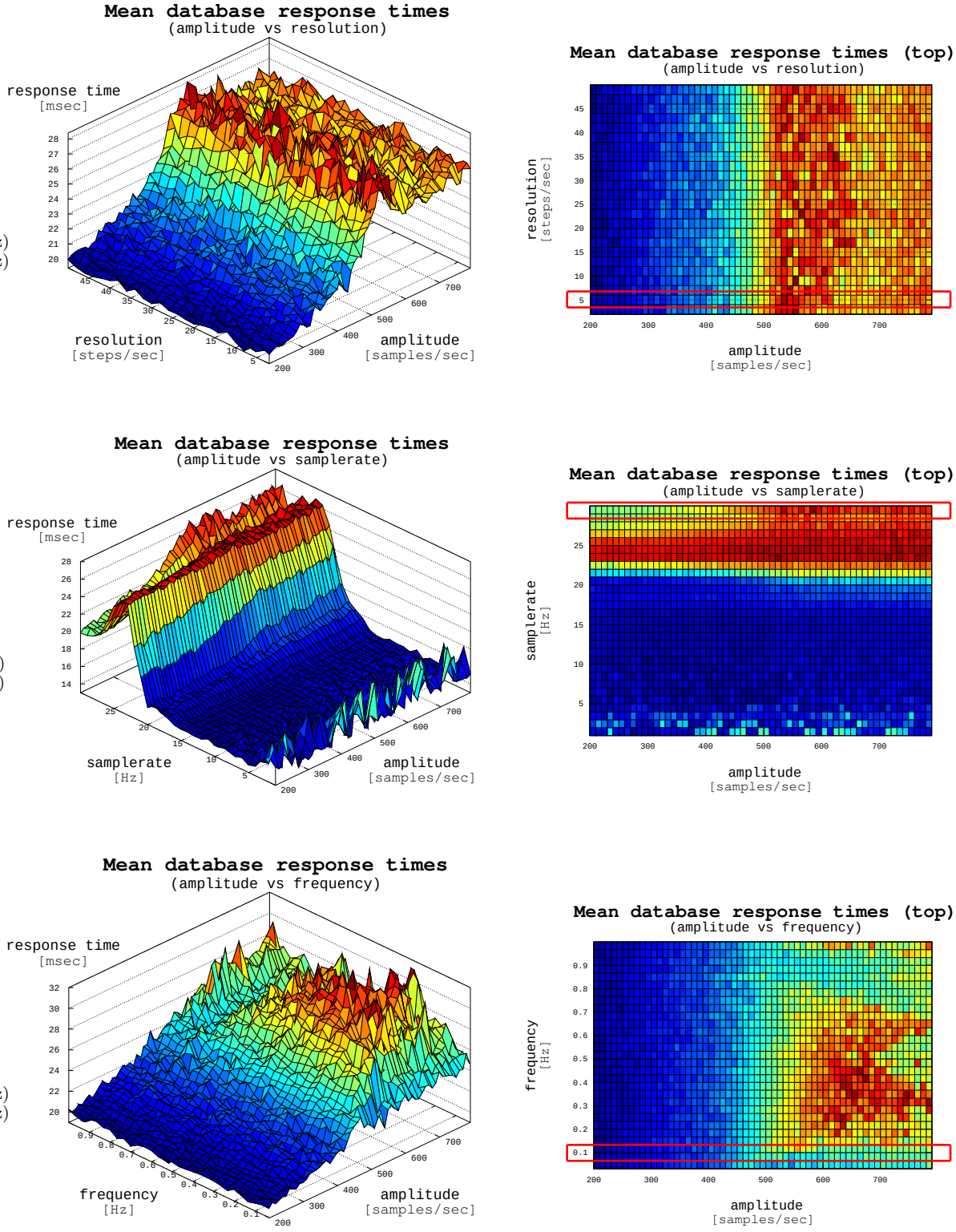
Figure 6.45: Webserver load examination

Each row in Fig. 6.45 represents the study of a specific parameter. As the amplitude is the vital parameter influencing all workloads and detection results, it is considered in every experiment series and listed as x-axis. The y-axis thereby varies between resolution, samplerate and frequency, leading to three distinct parameter studies. On that account, every analysis row expands in a different dimension spanned in the 4-dimensional parameter space, with a common fixed line depicted in the top views. The fifth dimension respectively the z-axis is used for displaying various metrics, which is in this case the mean webserver response times.

For calculating these response times, all obtained webserver timings per time step are averaged, giving a good indication of the mean overall load. As earlier explained in Sect. 6.1.3, all sample times of a particular discretization step are already averaged out in order to reduce outliers and prevent the distortion of the corresponding timing diagrams due to the inconsistently distributed samples. For the analyses in this section, these discrete average sample times are again averaged out leading to a single mean value representing the overall load in a very stable form as random outliers don't have a significant influence on the outcome.

All three analysis rows basically show increasing response times for increasing amplitudes. However, the influence of the comparison parameter is different for each experiment. First, the resolution doesn't have any noticeable impact on the shape of the timing curve, which is just like expected as it is only improving the cosine signal, but not producing any significant changes in the sample count or its distribution. The common fixed line in the database samplerate analysis shows (inevitably) the same behaviour, but interestingly for decreasing samplerates, the load on the webserver reaches a local minimum. This implies that above a samplerate of about `20 Hz` the load on the database significantly increases, while below that threshold the samplerate doesn't influence the database workload at all. For varying frequencies, three areas of high load in the same magnitude as for the other series can be identified. However, for frequencies between `0.1 Hz` and `0.3 Hz` respectively around `0.6 Hz`, the webserver workload is significantly lower, which suggests that in these areas the database is under less load. Figure 6.46 supports these assumptions.

After all, it has to be noticed that increasing webserver response times mostly originate in the corresponding additional database load, which has been already pointed out in Sect. 6.2. However, with amplitudes beyond `650` samples per second, this effect is considered to be outweighed by the webservers own workload as the response times increase disproportionately. Nevertheless, the selection of compatible parameters can prevent additional load on top of this level, which accounts for the database analysis in Fig. 6.46 as well.



Invariants:
samplerate (30Hz)
frequency (0.1Hz)

Invariants:
resolution (5Hz)
frequency (0.1Hz)

Invariants:
resolution (5Hz)
samplerate (30Hz)

Figure 6.46: Database load examination

The database workload analysis is similarly arranged as for the webserver timings above. Three series of experiments compare different parameters to increasing amplitudes with a common fixed line combining them. Thereby the response times are again averaged out over the timeframe, allowing a stable view of the actual workload. Although the results show other characteristics than for the webserver, they perfectly support the corresponding assumptions made in the previous analysis.

At first it can be stated that the database response times increase with raising amplitudes just as for the webserver. However, above an amplitude of about `650` samples per second the timings remain on a constant level slightly lower than the peak at `550` samples per second. This confirms the conclusion that in this area the webserver workload increases significantly, leading to delayed database queries and thus to the observed decrease. Considering further increasing amplitudes, the webserver's queries are apparantly retarded accordingly, resulting in a nearly constant level of the database workload. The resolution parameter by contrast has no significant influence on the timing curve as already anticipated previously.

Regarding the samplerate, a completely different behaviour is recognized. While the outliers at very low samplerates are considered to be random as there are alltogether only a few samples to average, the increase of database response times is dramatical for samplerates beyond `15 Hz`. The actual amplitude thereby only plays a secondary role, however the contour lines show exactly the same behaviour of a beginning increase with a constant level at the end. At about `25 Hz`, the peak load regarding the samplerate is reached, followed by a significant decrease. The reason for that is unknown, but multiple repeats have confirmed this result. Finally, the fixed line at `30 Hz` shows an identical behaviour as obtained for the first row analysis. In summary, samplerates above `20 Hz` are considered to provide best results, whereas values below `20 Hz` significantly reduce the introduced workload.

For varying frequencies, further conclusions can be drawn. While the response times globally increase with raising amplitudes, there are nevertheless local minima. First, the marked line at `0.1 Hz` shows the same curve as in the first two rows (i.e. increasing response times followed by a constant level). However, this does not account for higher frequencies up to `0.7 Hz`. There, the timings increase undamped, assuming the database to be under more load. The areas of lower response times thereby match the corresponding levels of higher timings for the webserver and vice versa. Apparently, the frequency choice directly impacts the actual load dispersion, where frequencies below `0.1 Hz` and above `0.7 Hz` invoke lower load on the database, while the area in between generates lower load on the webserver. With increasing amplitudes however, the load dispersion can be shifted, too.

For completeness, the actual number of generated samples is closer examined. As already described in Sect. 5.3, it is depending on the integral of a modified cosine curve. However, as the integral value is scaled to the peak amplitude specified with the configuration, side-effects could possibly occur leading to an inconsistent sample distribution for varying parameters. Although it was mathematically evidenced that the calculation is approximately independent from the resolution (and correspondingly from the frequency), it has to be practically verified as well. Therefore, the sample distribution diagrams especially for the parameters resolution and frequency are of further interest, which are presented in Fig. 6.47.

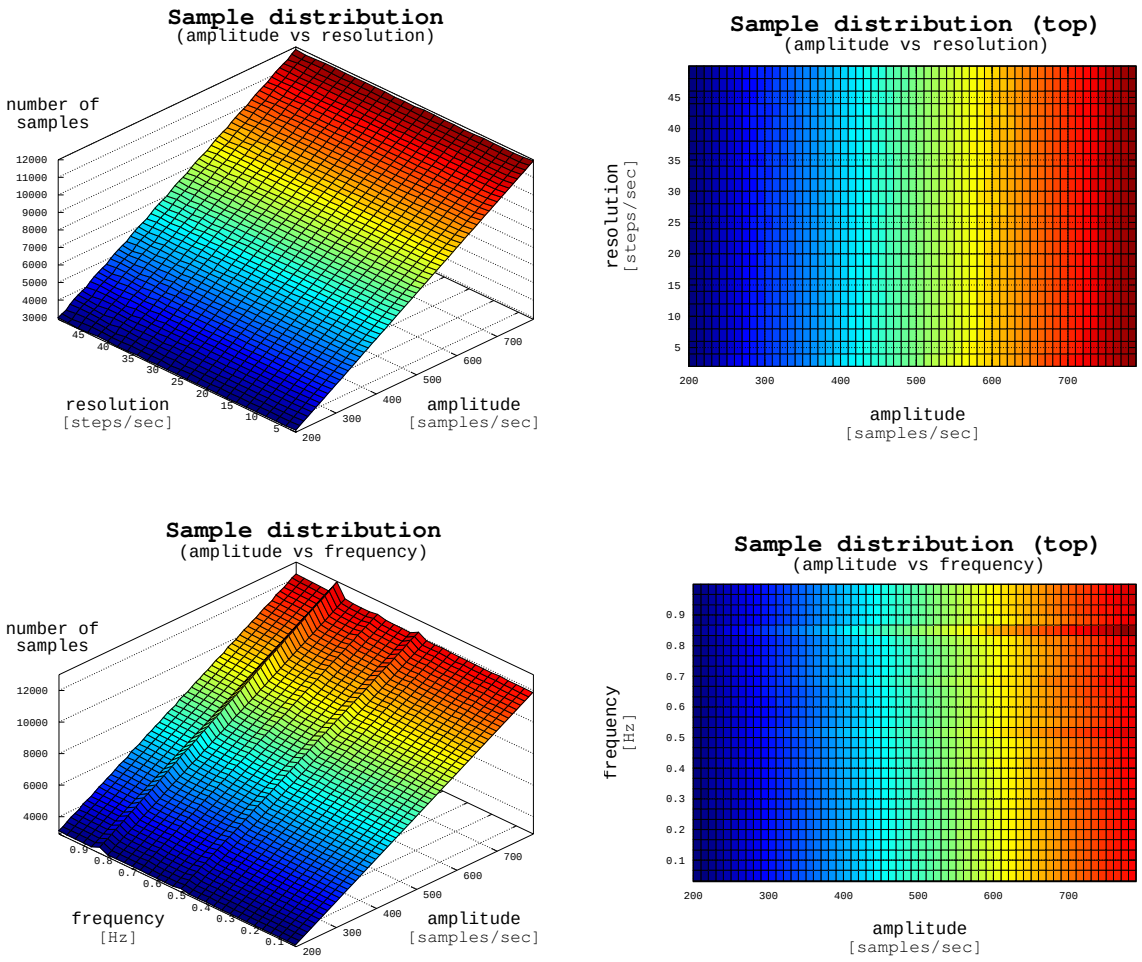


Figure 6.47: Sample distribution examination

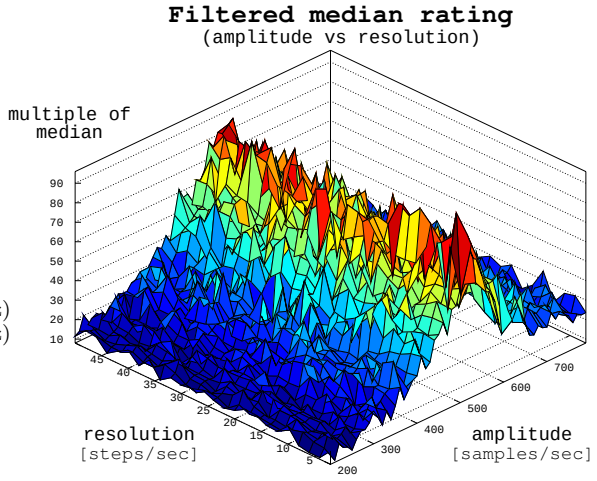
Considering the first row, i.e. the influence of the resolution parameter on the overall sample distribution, no significant irregularities are recognizable. Apparently, the number of samples is nearly constant for a specific amplitude and thus independent from the actual resolution. The second analysis row however shows some erratic peaks when using particular frequencys higher than `0.5 Hz`. Nevertheless, such high frequencies were not used throughout this thesis, so for earlier experiments this behaviour can be completely ignored. Regarding the previous (and also following) large comparison series which examine different metrics for varying frequencies, these peaks don't have any recognizable influence either. On balance, the discrepancy in the sample count is in the magnitude of five percent and only occurs sporadically.

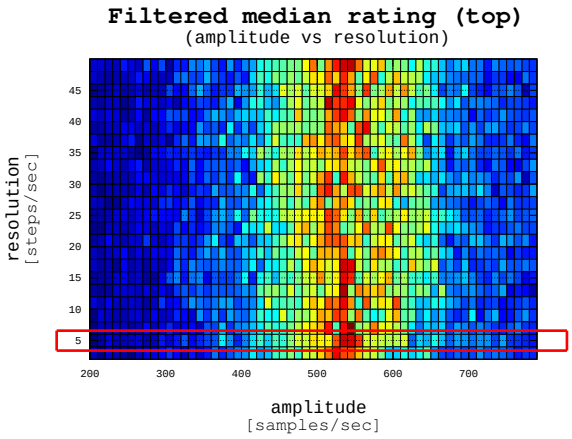
Altogether it can be confirmed, that the number of samples (and thus the invoked load) is in fact independent from the parameters resolution and frequency, apart from single outliers without any noticeable effect on the results. This independence obviously accounts for the database samplerate parameter as well, as it is inherently disconnected from the load generation procedure.

6.4.2 Rating analyses

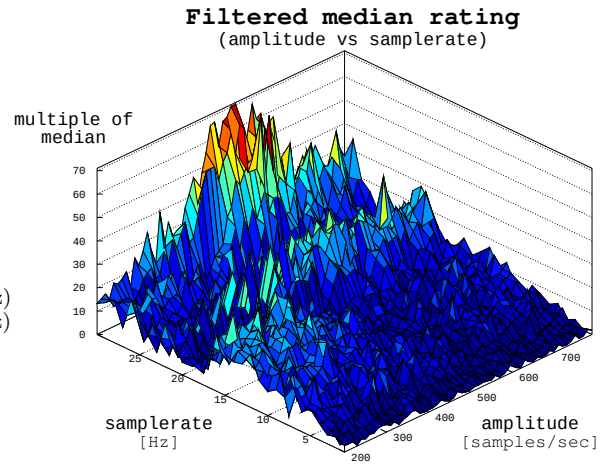
Although previous analyses already showed that specific parameters other than the amplitude parameter can have a significant influence on the overall workload for both the webserver and the database, further considerations have to be made regarding their impact on the detection results. It is on the one hand conceivable that lower overall workloads lead to a weaker detection quality or even provide no distinct results at all below certain thresholds. On the other hand however, it is possible that specific workload minima could be accompanied by areas of higher detection significances just as well. For that, the rating techniques have to be studied accordingly.

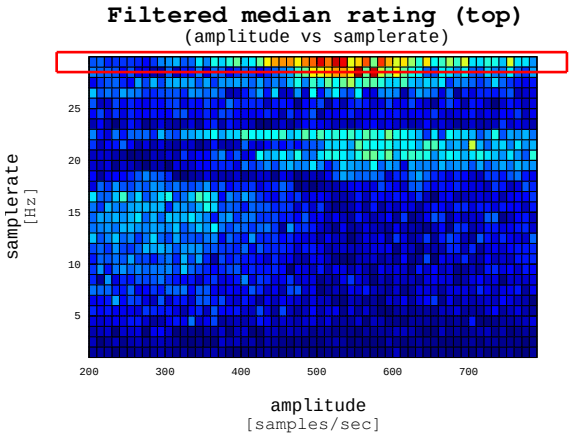
Such coherences of minimum workloads and local maxima of rating results are superiorly qualified for productive use, as the influence on live environments can be minimized while still being able to obtain significant results. This section therefore provides studies similar to the previous analyses for further investigating the median and inharmonic rating concepts. All diagrams shown below refer to the filtered rating techniques as the objective is to obtain the highest possible absolute values. The corresponding unfiltered results are given in the Appendix. Figure 6.48 shows the first series of analyses for the median rating on the next page.



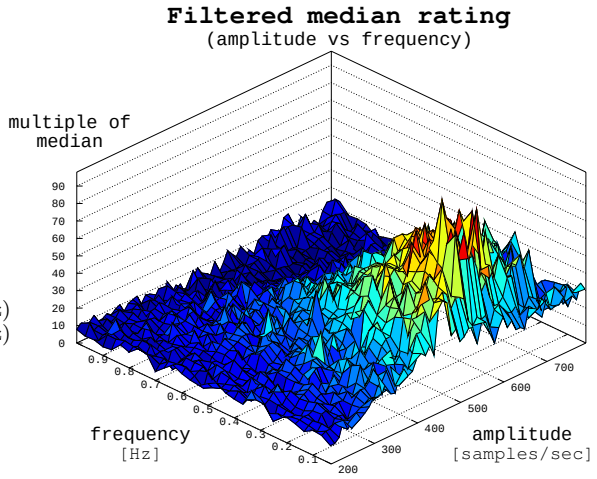


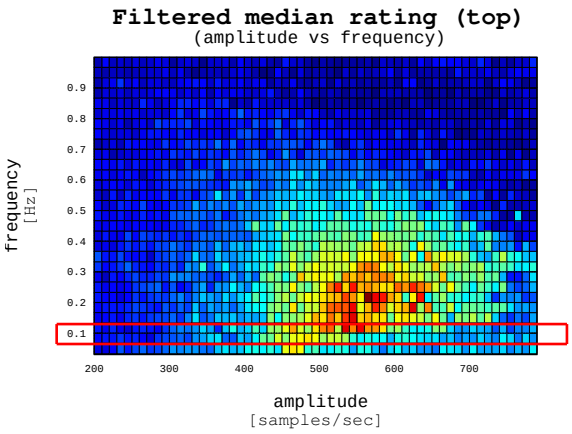
Invariants:
samplerate (30Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
samplerate (30Hz)

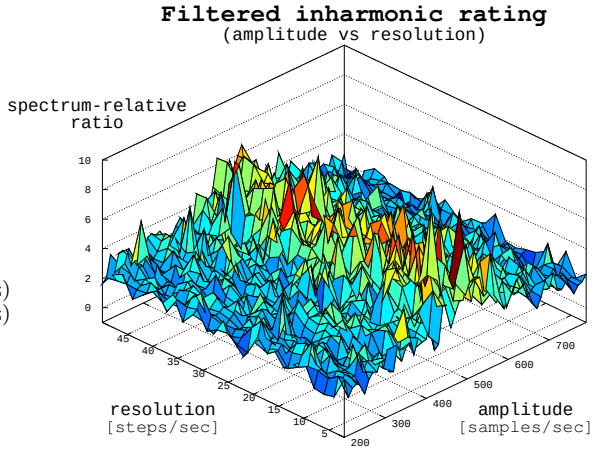
Figure 6.48: Median rating examination

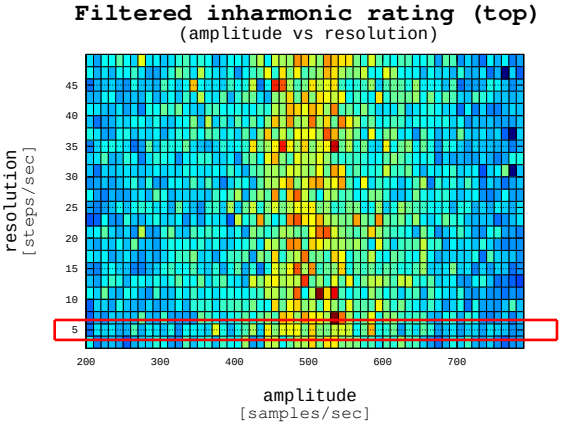
Analyzing achievable detection results is as important as considering minima in the workload for the webserver and database. For that, the same series of experiments have to be closely investigated on most significant values for the median rating (Fig. 6.48). The corresponding results thereby represent the intensity of the targeted frequency overlaying the already filtered database timings. Figure A.27 provides the related unfiltered diagrams, which in fact show similar results.

The first row exhibits a significant peak in the absolute rating values at an amplitude of about `550` samples per second. Compared to the database timings in Fig. 6.46, this matches exactly the area of highest load. As for the webserver response times (Fig. 6.45), the corresponding sector shows an extremely low workload barely recognizable at all, while areas of higher webserver workloads (i.e. higher amplitudes) result in lower detection results. The peak rating values are almost at `90` times the median, which is definitely outstanding. Absolute values of `20` or even below would confirm the dependency just as well, so lower amplitudes decreasing the database workload can be chosen without any problems. In fact, with the given samplerate and frequency excellent results can be obtained regardless of which amplitude. Increasing resolutions were expected to improve the results, but this is not the case. Apparently already one step per second is enough for producing a clear cosine curve unambiguously detectable by the frequency analysis.

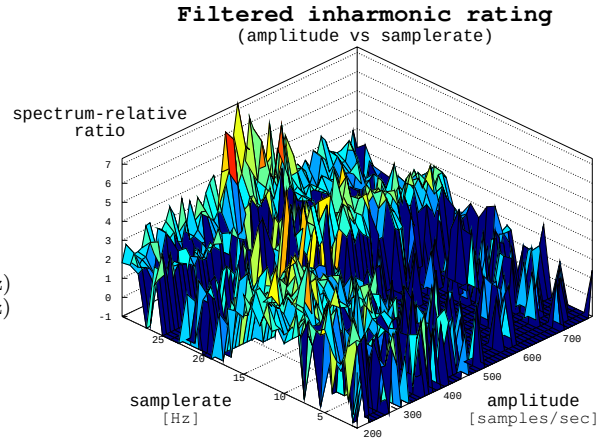
Considering lower samplerates, the detection decreases rapidly. Furthermore, between `23 Hz` and `26 Hz`, no distinct results can be obtained. The reason for that is definitely related to the corresponding database behaviour depicted in Fig. 6.46, which shows disproportionately high load in this area. Below that threshold the rating values are raising again, up to peaks of `40` times the median. This area is increasing to the right, directly linked to the amplitude and also to the database load. Finally, the left field of low samplerates and low amplitudes obviously allows ratings of about `20` times the median, which is still perfectly sufficient for verifying the dependency. So in general, this area should be perferred as it is invoking extremely low webserver and database workloads.

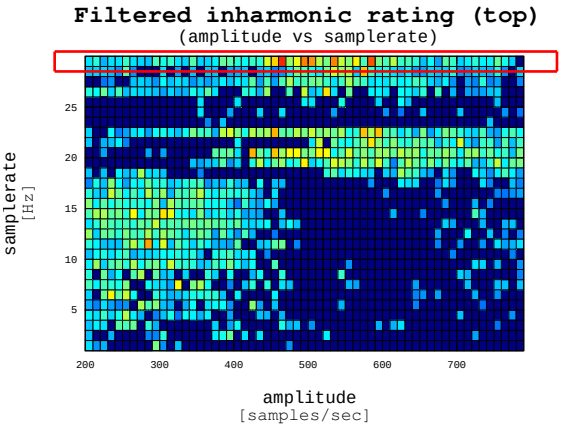
Apart from that, the frequency parameter doesn't have any significant influence on the outcome, at least when set to `0.5 Hz` and below. Nevertheless, higher frequencies still allow detection results in the magnitude of `20` times the median, especially when used with low amplitudes. Compared to the webserver timings in Fig. 6.45, it is evident that areas of highest load are linked to corresponding areas of weak detection results. This again confirms the conclusion that above a threshold of about `650` samples per second the webserver workload has a significant influence on the outcome. Figure 6.49 on the next page focuses on the inharmonic rating.



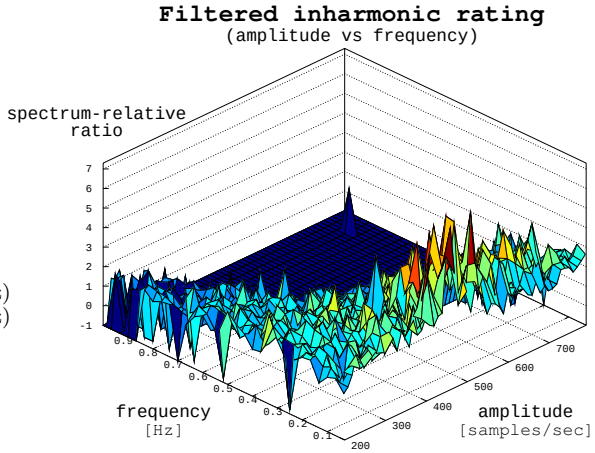


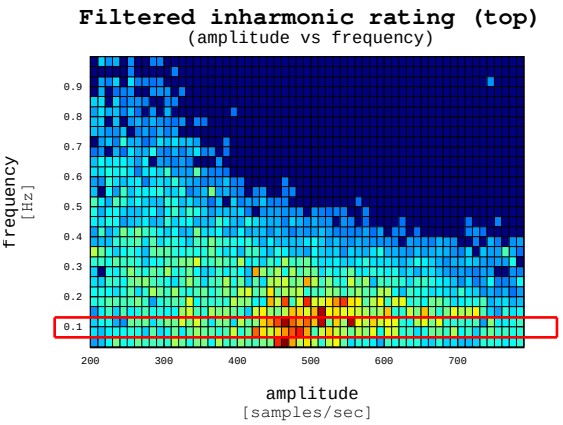
Invariants:
samplerate (30Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
samplerate (30Hz)

Figure 6.49: Inharmonic rating examination

Although the median rating provides a good initial estimation of the overall detection result, it has to be supported by the inharmonic rating (Sect. 3.3.2). This technique thereby provides further information on the targeted frequency. On the one hand, it determines if it holds the highest amplitude in value compared to the whole frequency spectrum. Values of `-1` indicate that other frequency parts are dominant. On the other hand, it measures the protrusion relative to the frequency with the second highest amplitude in value. Again the filtered versions of the analyses are considered, the corresponding unfiltered diagrams can be found in Fig. A.28.

The first row in Fig. 6.49 shows extraordinary results. There is virtually no occurence of the targeted frequency not holding the highest amplitude in value. Furthermore nearly all amplitudes are twice as high as the frequency with the second highest amplitude in value, with peaks up to `10` times. These values are all considered to verify the dependency unambiguously, which perfectly matches the analysis for the median rating (Fig. 6.48). Interestingly the actual peak appears for slightly lower amplitudes of about `500 Hz`. This accounts for the fact that high median ratings do not necessarily depict the best detection results. Nevertheless, they are a good indication and always occur in the same magnitude as high inharmonic ratings. This issue can be confirmed by considering the second row. All peak areas are exactly arranged as previously, although the virtual peak values again appear for about `10%` lower amplitudes. However, these second diagrams show distinct areas of negative detection results, which are not recognized by the corresponding median rating. All this accounts for the last row of analyses as well, whereas the absolute peaks are additionally shifted slightly down the frequency axis. It can thereby be clearly observed that for increasing frequencies only decreasing amplitudes still provide positive results. The reason for that is presumably a blurring of the load signal for frequencies above `0.5 Hz` combined with high amplitudes, which leads to significant noise present in the signal. For lower amplitudes by contrast, this effect is flattened.

In summary, the inharmonic rating allows the determination if the targeted frequency is dominant in the spectrum, and furthermore results in similar detection results compared to the median rating. Although it is preferable to obtain high absolute values for the inharmonic rating, it has to be stated that even negative results don't eliminate the possibility of a dependency relationship if exorbitant high distortion is to be expected. As long as the median rating shows a significant increase in the amplitude of the corresponding frequency part, closer examination with possibly higher workloads or samplerates is reasonable. Another interesting series of analyses is given in Fig. 6.50. Thereby the diagrams show if the frequency part with the second highest amplitude in value is a harmonic of the targeted frequency. Examining harmonics could infere the dependency relation from another point of view.

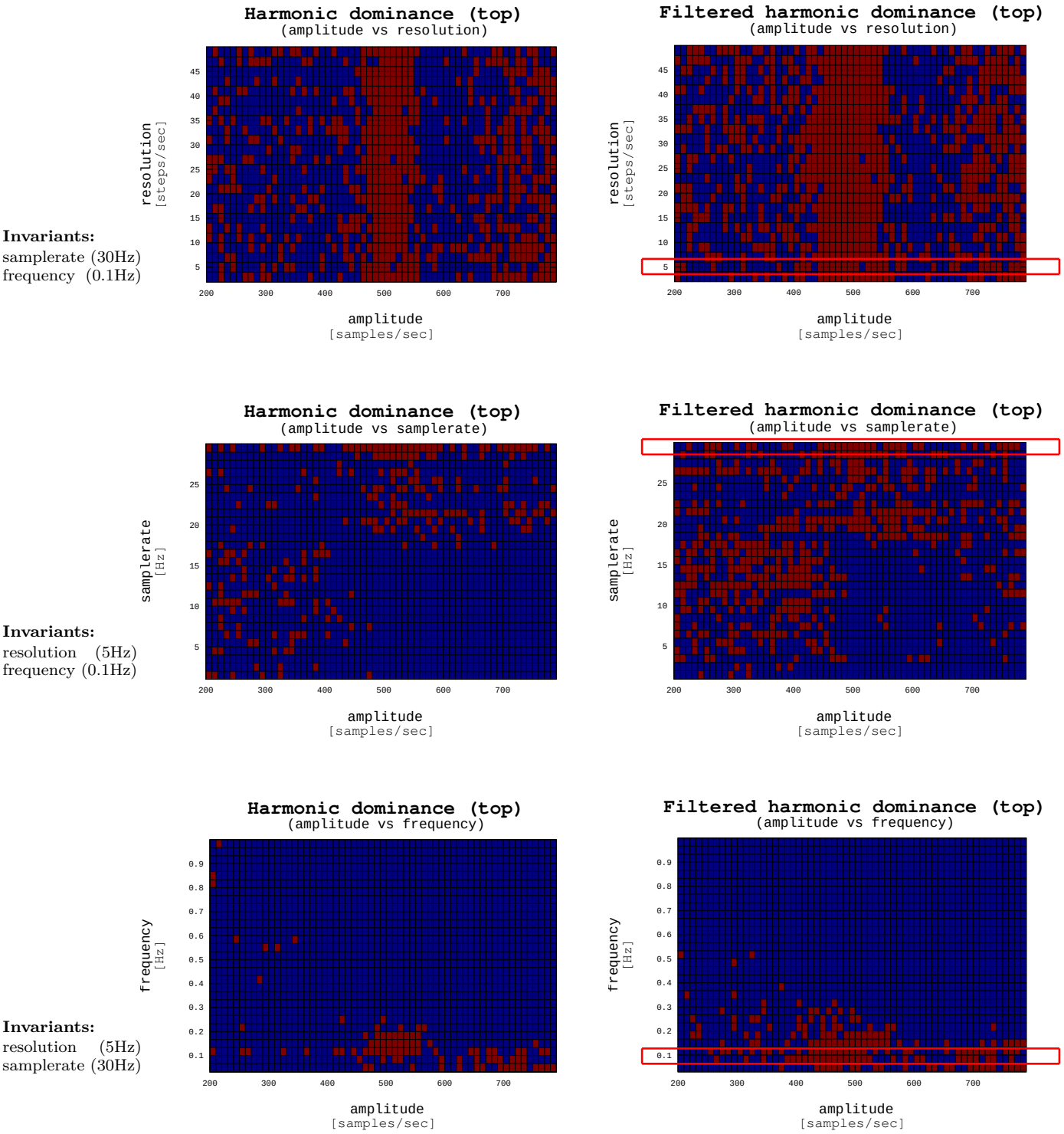


Figure 6.50: Harmonic dominance examination

As already pointed out in Sect. 3.3.2, particular frequencies in signals encountering noise and distortion categorically produce additional harmonic components in the frequency spectrum. For further investigating this issue, Fig. 6.50 shows diagrams similar to the previous analyses, which depict whether the frequency part with the second highest amplitude in value is a harmonic of the targeted frequency. The left column thereby represents the unfiltered signal, while the right diagrams show the corresponding filtered results. Although the provided information is of primarily scientific interest, it could also support the dependency detection. Future work could thereby focus on the additional evaluation of harmonics for the targeted frequency, allowing a decisive verification even for extremely weak signals or extraordinary high noise where the actual frequency part would be overwhelmed by the noise.

At first it can be noticed that the unfiltered diagrams show exactly the same peak areas as the inharmonic rating (Fig. 6.49), i.e. slightly shifted towards lower amplitudes respectively frequencies compared to the median rating (Fig. 6.48). The filtering thereby enlarges these areas, which confirms that the median filter reduces overall noise leading to more harmonics dominant in the spectrum.

Another interesting conclusion can be drawn for high amplitudes in the first row. Although the previously observed rating results are relatively weak compared to the peak area (as the webserver workload significantly increases, Fig. 6.45), the harmonic analysis shows that even there most of the experiments result in dominant harmonics, so the positive verification is effectively confirmed. For the area in between, other frequencies overtop the harmonics, although the median rating shows its highest values there. Apparently this area of maximum database load (Fig. 6.46) produces significantly more noise. Regarding low amplitudes, the results become more indistinct as for lower workloads random delays probably have a greater impact on the frequency spectrum.

The second row again confirms conclusions drawn in the previous analyses. Most important, the area of low amplitudes and low samplerates shows a surprisingly high amount of dominant harmonics, so this area is defintely suitable for performing dependency detection, and especially for productive use as the corresponding webserver and database workloads are reduced to a significant minimum.

Finally, the last row clarifies that although both rating concepts resulted in viable detection results for high frequencies with low amplitudes, virtually no experiment shows a dominant harmonic. This was already anticipated as high frequencies lead to blurred signals introducing great additional noise. Nevertheless, dependency detection in this area is feasible as both rating techniques produce reasonable results.

6.4.3 Summary

With the above analyses, it could be categorically shown that the approach of active dependency discovery is feasible in a large scale as well. The comparison of more than 5,000 experiments thereby lead to profound conclusions significantly extending the knowledge gathered in previous sections. To finally close this section on the examination of further interrelations, the most important results are briefly recapitulated below.

1. **Workload analyses.** At first, it could be shown that the webserver and database workloads educe reciprocally. This fact is immensely useful as adequate configuration settings allow the load to be shifted towards a specific system in respect of on-site peculiarities and the actual use case. Furthermore, a connection between overall workloads and the database samplerate as well as the load signal's frequency could be evidenced, which permits even further differentiation of the effective workload spreading. Finally, large areas of insignificant load for both involved systems were determined, providing a fundamental base for productive usage even in sensitive environments.

2. **Rating analyses.** Both introduced rating techniques proved to produce similar conclusions. The combination of them thereby provides significant detection results independent from statistical randomness or noise present in the signals. Besides, it could be discovered that the actual database workload is decisive for the rating results and scales accordingly. More important, areas of high detection significance in combination with low parameter values were determined. Combining this information with the preceding workload analyses, practical feasibility even for sensitive fields of application can be concluded. Finally, another information source based on the target frequency's harmonics was introduced, showing great promise for the dependency detection even under adverse conditions.

The approach is herewith conclusively investigated and proves viable to the full extent, while further interrelations for improved deployment strategies are revealed. Although there are most certainly more fields of interest requiring further detailed studies, the objective of this thesis is satisfyingly fulfilled. Nevertheless, an additional last validation is performed in the next section. Therefore the prototype is deployed in a productive environment, facing all previously addressed influences, side-effects and phenomena and beyond that, introducing the lack of knowledge about the underlying communication structure.

6.5 Productive deployment

Ultimately verifying the active dependency discovery approach implies a productive field test in a live environment. For that, the prototype was deployed in a medium-sized network spanning across multiple countries. At the same time a real use case was studied. Figure 6.51 shows the corresponding setup.

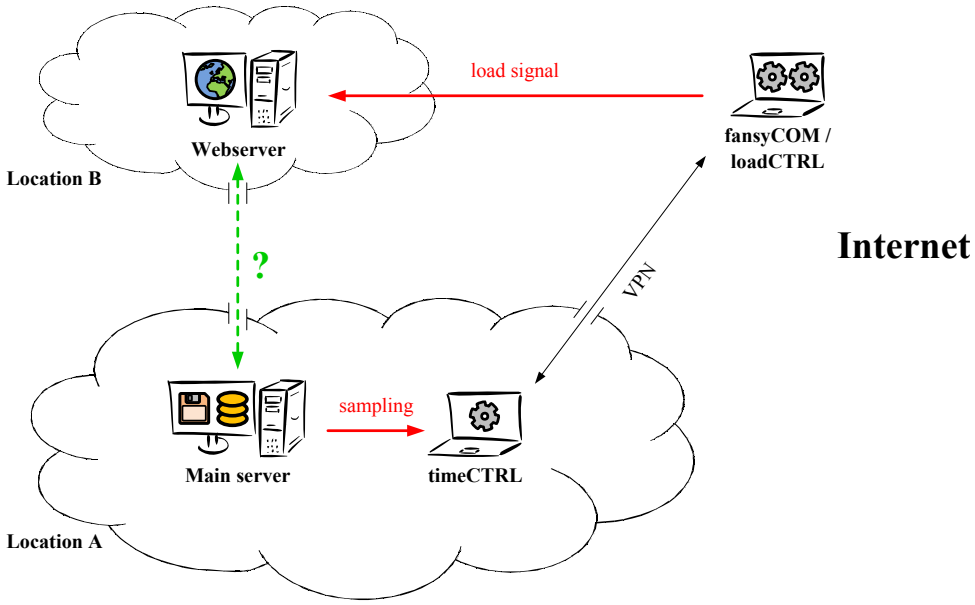


Figure 6.51: Productive deployment

Location A hosts a main server providing various services for different subnets. Each subnet thereby covers another location in a different country. The communication between these subnets is based on a *virtual private network* (VPN). For maintenance reasons, the main server had to be shut down, which raised the issue if the webserver in location B could be affected of any kind. The previous knowledge on both systems was very limited, in fact only the internet address of the webserver and the internal IP address of the main server were obtained from a simple asset management sheet. Regarding the prototype deployment, a dedicated client notebook was used for the response time sampling procedure. Thereby it was simply plugged into a common office switch inside location A with no further knowledge of the actual topology. This location hosts approximately 50 clients and 10 specialized servers, so the infrastructure can be considered as moderately complex. The load generation and controlling applications were deployed on a public notebook with

a direct connection to the internet. While the load generation procedure accessed the webserver directly over the public interface of location B, the timing mechanism was controlled over the VPN access. The setup for itself is relatively straightforward, however the underlying topology involving the internet and also locations in different countries obviously imposes severe difficulties. In addition, the actual constitution of the dependency communication – if existent at all – was completely unknown. For getting a basis of comparison, the timing procedure was initially performed without any present load. Figure 6.52 shows the (unfiltered) results.

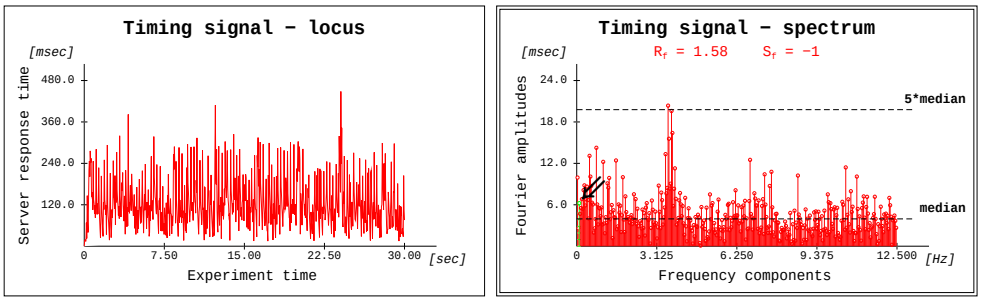


Figure 6.52: Productive deployment – Idle timing

Although the timing was performed with a samplerate of `25 Hz`, which is slightly smaller than for the experiments in previous sections, the response times are nearly ten times as high. The reason for that is on the one hand the far more complex network topology. On the other hand cross-traffic from other clients (mainly accessing the network per VPN as the maintenance was scheduled on a Saturday) and especially from other servers is suspected. The frequency analysis thereby shows huge amounts of noise with peak amplitudes significantly higher than for the test setups used so far. The targeted frequency of `0.1 Hz` (which wasn't applied on the webserver for this experiment yet) shows a value in the magnitude of the median, so it is completely random. Figure 6.53 displays the filtered results as well.

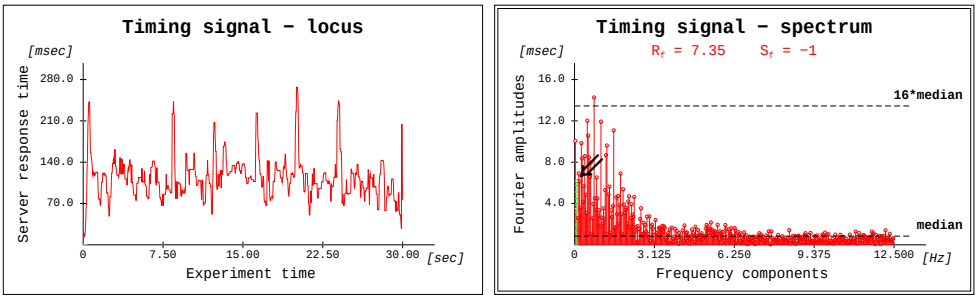


Figure 6.53: Productive deployment – Filtered idle timing

It is obvious that the filtering reduces the overall noise to a significant extent and thereby illustrates low-frequency components more clearly, but the targeted frequency is still overtopped by plenty of other frequency parts. The median rating however increases significantly.

Figure 6.54 shows the load signal used for the second experiment. The amplitude was thereby set to only 50 samples per second as the load on the webserver had to be held as low as possible. Nevertheless, the response times were in the magnitude of several seconds since the corresponding requests had to be directed over the internet towards location B, and – assuming the dependency relationship – resulted furthermore in queries to the main server at location A. However, the actual cosine-shaped load signal is clearly recognizable.

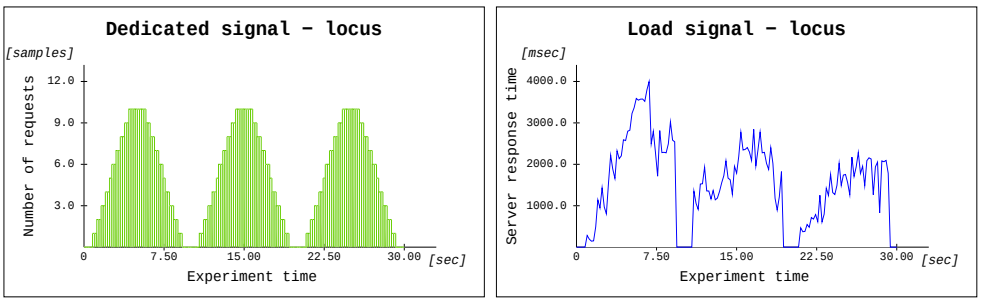


Figure 6.54: Productive deployment – Load signal

The corresponding configuration file is listed in Table A.3, whereas the overall detection results obtained from the timing measurement are depicted below in Fig. 6.55. Thereby the left diagram shows the unfiltered result, whereas the right diagram represents the corresponding filtered analysis.

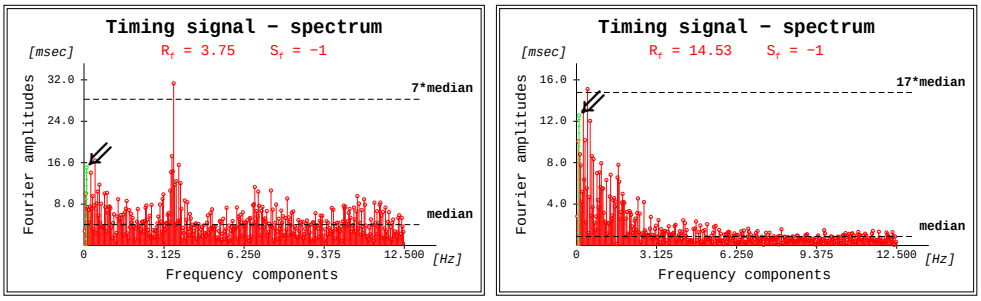


Figure 6.55: Productive deployment – Result

Apparently the absolute median ratings raised by almost 140% for the unfiltered data respectively by about 100% for the filtered case. This huge increase is considered

as a positive verification success. Regarding the tremendously complex setup, it is astonishing that the load propagation could be detected even across country boarders with only moderately generated load. Full reports including additional details can be found in the appendix:

Figure A.23: `experiment__2009-07-25__13.23.02`
Figure A.24: `experiment__2009-07-25__13.23.04`
Figure A.25: `experiment__2009-07-25__13.30.00`
Figure A.26: `experiment__2009-07-25__13.30.03`

After this experiment, the actual webserver configuration was closer examined by the system administrator. The corresponding system turned out to be mounting a remote file system from the main server at location A necessary for the webserver operations, so the positive verification was fully justified. As a consequence, the administrator took the website offline with a short notice of ongoing maintenance.

This final experiment brings the evaluation chapter down to a round figure. Although the dependency relationship was not as perfectly recognized as for the test setups, it could provide significant facts for making qualified assumptions in spite of facing adverse conditions. Nevertheless there is enough space for further improving the prototype. Especially for large productive environments, the already mentioned harmonic detection technique (Sect. 6.4) is most promising. A complete outlook on future work is given in the final chapter.

Chapter 7

7 Conclusion and outlook

"The last act is bloody, however fine the rest of the play. They throw earth over your head and it is finished forever."

Blaise Pascal

Active dependency discovery by means of generating and detecting synthetic load fluctuations is a new approach for supporting the process of configuration management. While current techniques rely on human expert knowledge or focus on narrow fields of application, this thesis's approach proved to reliably deliver practical results even under adverse conditions. The method of active interference on the one hand lead to a fully deterministic, reproducible and unambigous detection of dependency relationships among networked services. On the other hand, a careful implementation combined with reasonable deployment strategies reduced the influence on productive environments to a negligible amount. Moreover, distinct series of experiments demonstrated general feasibility and profound capabilities for coping with complex scenarios as well. Finally, a productive field test disclosed the full strength of the introduced technique by detecting dependencies on short notice, even across network and country borders. In summary, the developed approach met and actually exceeded all expectations, suggesting further work on this topic without reservation.

Improvement considerations can be split into two parts. First, the approach itself shows great potential for further extensions. Although the utilized load signal proved to deliver a fully satisfying dependency detection, the investigation of different signals promises further reduction of workloads as well as better resistance to noise effects. The corresponding rating techniques already formed a solid base for qualitative result evaluation and automatable detection mechanisms, however sophisticated concepts like signal-to-noise ratio could further improve their significance.

In addition, other fields of future research are conceivable. While this thesis mainly focused on the evaluation of common network services on dedicated machines, advanced experiments already demonstrated different fields of application. Anticipated effects when dealing with current virtualization technology or abstracting from the actual infrastructure were simulated accordingly and generally proved to allow active dependency discovery. Nevertheless, advanced studies on that matter are suggested. Finally, the combination with related work – mostly for administering detected dependencies in global maps or dedicated applications – is considered to be an important step towards a practical solution and seamless integration into configuration management systems.

The second area of possible improvements deals with the developed prototype itself. As it was desgined to be modular and extendable from the beginning, further upgrades are planned. At first, an interface for making use of existent asset management systems is inevitable. With that, the process of targeting particular services for generating load becomes automatable as well. Furthermore, the output mechanism for final detection results could easily be extended to meet various different specifications. As already mentioned, consolidation efforts with related work show great promise for an extensive configuration management, which is anticpated for the prototype due to its modular design. In addition, the support for analyzing further services is to be extended. This includes techniques for scanning multiple systems in parallel by making use of the corresponding simulative results provided with this thesis. Finally, the implementation is ready for an improved graphical user interface, allowing interactive dependency discovery for operational as well as strategical use.

Leaving the scope of configuration management, a new form of communication based on a typical sender recipient relationship as described in [1] has been created. Thereby specific signals can be reliably transmitted and received between arbitrarily correlated systems, regardless of the underlying connection properties. Possible applications are manifold as communication interrelations always are and therefore left to the imaginative reader.

Bibliography

[1] C. E. Shannon, A Mathematical Theory of Communication, in *The Bell System Technical Journal*, volume 27, Short Hills, NJ, USA, 1948, American Telephone and Telegraph Company.

[2] *IT Infrastructure Library©*, http://www.best-management-practice.com/.

[3] L. Klosterboer, *Implementing ITIL© Configuration Management*, IBM Press, 2008.

[4] S. Lacy and I. Macfarlane, *ITIL© – Service Transition*, The Stationery Office, 2007, Office of Government Commerce, UK.

[5] C. Probst, *Signalbasierte Analyse von Abhängigkeiten bei Netzwerkdiensten*, Tübingen, BW, GER, 2008, Universität Tübingen.

[6] G. Kar, A. Keller, and S. Calo, Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis, in *Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium*, pp. 61–74, Honolulu, HI, USA, 2000, IEEE Press.

[7] A. Keller and G. Kar, Determining Service Dependencies in Distributed Systems, in *IEEE International Conference on Communications, 2001*, volume 7, pp. 2084–2088, Helsinki, SF, FIN, 2001, IEEE Press.

[8] Extensible Markup Language (XML) 1.0, 5th Edition, http://www.w3.org/TR/xml/, 2008.

[9] *HP OpenView*, http://openview.hp.com/.

[10] *Microsoft Operations Manager*, http://www.microsoft.com/mom/.

[11] *Novell ZENworks*, http://www.novell.com/products/zenworks/.

[12] *Red Hat Network*, https://www.redhat.com/rhn/.

[13] *IBM Tivoli*, http://www-01.ibm.com/software/tivoli/.

[14] P. Barham, R. Black, M. Goldszmidt, R. Isaacs, J. MacCormick, R. Mortier, and A. Simma, *Constellation: automated discovery of service and host dependencies in networked systems*, Cambridge, CAM, UK, 2008, Microsoft Research Technical Report.

[15] 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 *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 13–24, New York, NY, USA, 2007, ACM Press.

[16] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, *Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions*, in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, 2008*, pp. 117–130, San Diego, CA, USA, 2008, USENIX Association.

[17] S. Kandula, R. Chandra, and D. Katabi, What's going on?: learning communication rules in edge networks, in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pp. 87–98, New York, NY, USA, 2008, ACM Press.

[18] S. Basu, F. Casati, and F. Daniel, Web Service Dependency Discovery Tool for SOA Management, in *IEEE International Conference on Services Computing, 2007*, pp. 684–685, Salt Lake City, UT, USA, 2007, IEEE Press.

[19] A. Brown, G. Kar, and A. Keller, An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Environment, in *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 377–390, Seattle, WA, USA, 2001, IEEE Press.

[20] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, A break in the clouds: towards a cloud definition, in *SIGCOMM Computer Communication Review*, volume 39, pp. 50–55, New York, NY, USA, 2009, ACM Press.

[21] W. K. Pratt, *Digital Image Processing: PIKS Inside, 3rd Edition*, Wiley InterScience, Malden, MA, USA, 2001.

[22] S. Perreault and P. Hebert, Median Filtering in Constant Time, in *IEEE Transactions on Image Processing*, volume 16, pp. 2389–2394, New York, NY, USA, 2007, IEEE Press.

[23] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing, 2nd Edition*, Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[24] J. W. Cooley and J. W. Tukey, An Algorithm for the machine calculation of complex Fourier series, in *Mathematics of Computation*, volume 19, pp. 297–301, Providence, RI, USA, 1965, American Mathematical Society.

[25] T. B. Barker, *Engineering quality by design: interpreting the Taguchi approach*, M. Dekker, New York, NY, USA, 1990.

[26] I. Sommerville, *Software Engineering, 8th edition*, Addison-Wesley, New York, NY, USA, 2006.

[27] B. W. Boehm, *Software Engineering Economics*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[28] I. Jacobson, *Object-oriented software engineering: A use-case driven approach*, ACM Press, New York, NY, USA, 1992.

[29] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory: 1736-1936*, Clarendon Press, Oxford, OXF, UK, 1976.

[30] The Perl Foundation, http://www.perlfoundation.org/.

[31] ISO/IEC 9899:1999 – Programming language C, http://www.open-std.org/JTC1/sc22/wg14/www/standards, 1999.

[32] GNU make, http://www.gnu.org/software/make/.

[33] RFC 707: A High-Level Framework for Network-Based Resource Sharing, http://tools.ietf.org/html/rfc707/, 1976.

[34] A. D. Birrell and B. J. Nelson, Implementing Remote procedure calls, in *SIGOPS Operating Systems Review*, volume 5, New York, NY, USA, 1983, ACM Press.

[35] D. R. Butenho, *Programming with POSIX threads*, Addison-Wesley, Reading, MA, USA, 1997.

[36] Doxygen, http://www.doxygen.org/.

[37] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, Reading, MA, USA, 1995.

[38] The XML C parser and toolkit of Gnome, http://xmlsoft.org/.

[39] C. A. R. Hoare, Quicksort, in *The Computer Journal*, volume 5, pp. 10–15, Oxford, OXF, UK, 1962, British Computer Society.

[40] FFTW – Fastest Fourier Transform in the West, http://www.fftw.org/.

[41] The plotutils Package, http://www.gnu.org/software/plotutils/.

[42] RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, http://tools.ietf.org/html/rfc2616, 1999.

[43] RFC 791: Internet Protocol, http://tools.ietf.org/html/rfc791, 1981.

[44] MySQL, http://www.mysql.com/.

[45] RFC 4251: The Secure Shell (SSH) Protocol Architecture, http://tools.ietf.org/html/rfc4251, 2006.

[46] RFC 4253: The Secure Shell (SSH) Transport Layer Protocol, http://tools.ietf.org/html/rfc4253, 2006.

[47] RFC 4252: The Secure Shell (SSH) Authentication Protocol, http://tools.ietf.org/html/rfc4252, 2006.

[48] Apache HTTP Server Project, http://httpd.apache.org/.

[49] PHP, http://www.php.net/.

List of Figures

List of Figures

List of Tables

List of Tables

Appendix

Table A.1: Basic experiment – XML configuration

```
<!- example_config.xml ->

<configuration>

    <fansyCOM
        host             = "192.168.5.10"
        writeToFile      = "1"
        filter           = "1"
        debug            = "0" >
    </fansyCOM>

    <loadCTRL
        inUse            = "1"
        debug            = "0"
        host             = "192.168.5.101"
        target           = "192.168.10.2"
        loadstyle        = "HTTP-GET"
        wavestyle        = "cosine"
        timeframe        = "30"
        resolution       = "5"
        frequency        = "0.1"
        amplitude        = "600" >

        <httpGET_LOADMODULE
            url          = "http://192.168.10.2/direct.php" >
        </httpGET_LOADMODULE>

    </loadCTRL>
```

continued on next page

continued from previous page

```
    <timeCTRL
        inUse             = "1"
        debug             = "0"
        host              = "192.168.5.104"
        target            = "192.168.10.3"
        timingstyle       = "telnet"
        timeframe         = "30"
        samplerate        = "30" >

        <telnet_TIMINGMODULE
            port          = "22"
            burst-length  = "1" >
        </telnet_TIMINGMODULE>

    </timeCTRL>

</configuration>
```

Table A.2: Reverse direction – XML configuration

```
<!- reverse_config.xml ->

<configuration>

    <fansyCOM
        host              = "192.168.5.10"
        writeToFile       = "1"
        filter            = "1"
        debug             = "0" >
    </fansyCOM>

    <loadCTRL
        inUse             = "1"
        debug             = "0"
        host              = "192.168.5.104"
        target            = "192.168.5.103"
        loadstyle         = "SQL"
        wavestyle         = "cosine"
        timeframe         = "30"
        resolution        = "5"
        frequency         = "0.1"
        amplitude         = "650" >

        <SQL_LOADMODULE
            port          = "3306"
            user          = "XXX"
            password      = "XXX"
            database      = "mysql"
            table         = "user"
            burst-length  = "1" >
```

continued on next page

continued from previous page

```
        </SQL_LOADMODULE>

    </loadCTRL>


    <timeCTRL
        inUse            = "1"
        debug            = "0"
        host             = "192.168.5.101"
        target           = "192.168.5.102"
        timingstyle      = "HTTP-GET"
        timeframe        = "30"
        samplerate       = "2" >

        <httpGET_TIMINGMODULE
            url          = "http://192.168.5.102/switched.php" >
        </httpGET_TIMINGMODULE>

    </timeCTRL>

</configuration>
```

Table A.3: Productive deployment – XML configuration

```
<!- productive_config.xml ->

<configuration>

    <fansyCOM
        host             = "127.0.0.1"
        writeToFile      = "1"
        filter           = "1"
        debug            = "0" >
    </fansyCOM>

    <loadCTRL
        inUse            = "1"
        debug            = "0"
        host             = "127.0.0.1"
        target           = "X.X.X.X"
        loadstyle        = "HTTP-GET"
        wavestyle        = "cosine"
        timeframe        = "30"
        resolution       = "5"
        frequency        = "0.1"
        amplitude        = "50" >

        <httpGET_LOADMODULE
            url          = "http://X.X.X.X/index.php" >
        </httpGET_LOADMODULE>

    </loadCTRL>
```

continued on next page

continued from previous page

```
    <timeCTRL
        inUse             = "1"
        debug             = "0"
        host              = "10.1.1.100"
        target            = "10.1.1.5"
        timingstyle       = "telnet"
        timeframe         = "30"
        samplerate        = "25" >

        <telnet_TIMINGMODULE
            port          = "22"
            burst-length  = "1" >
        </telnet_TIMINGMODULE>

    </timeCTRL>

</configuration>
```

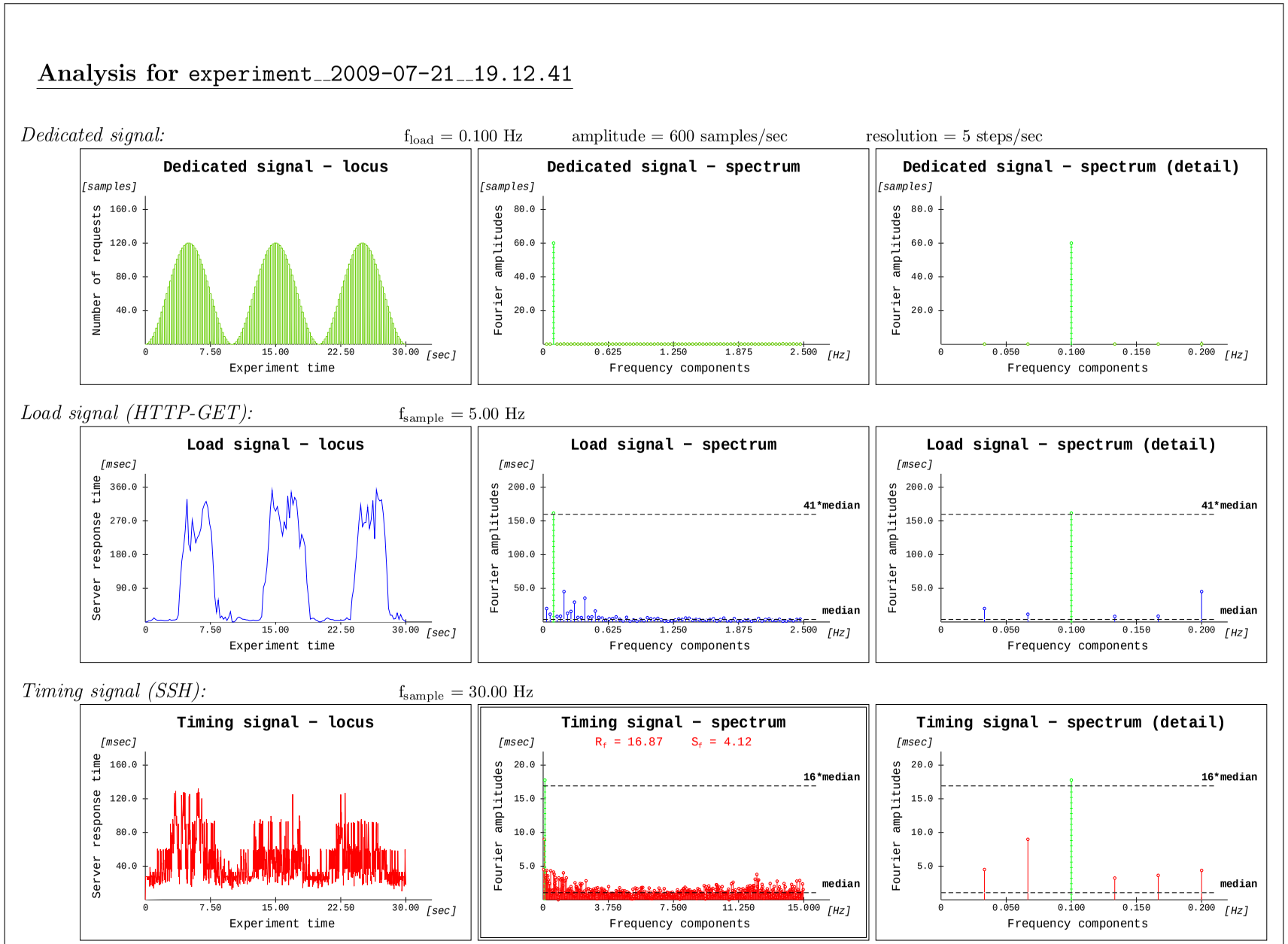


Figure A.1: Basic experiment (direct cabling) – Report

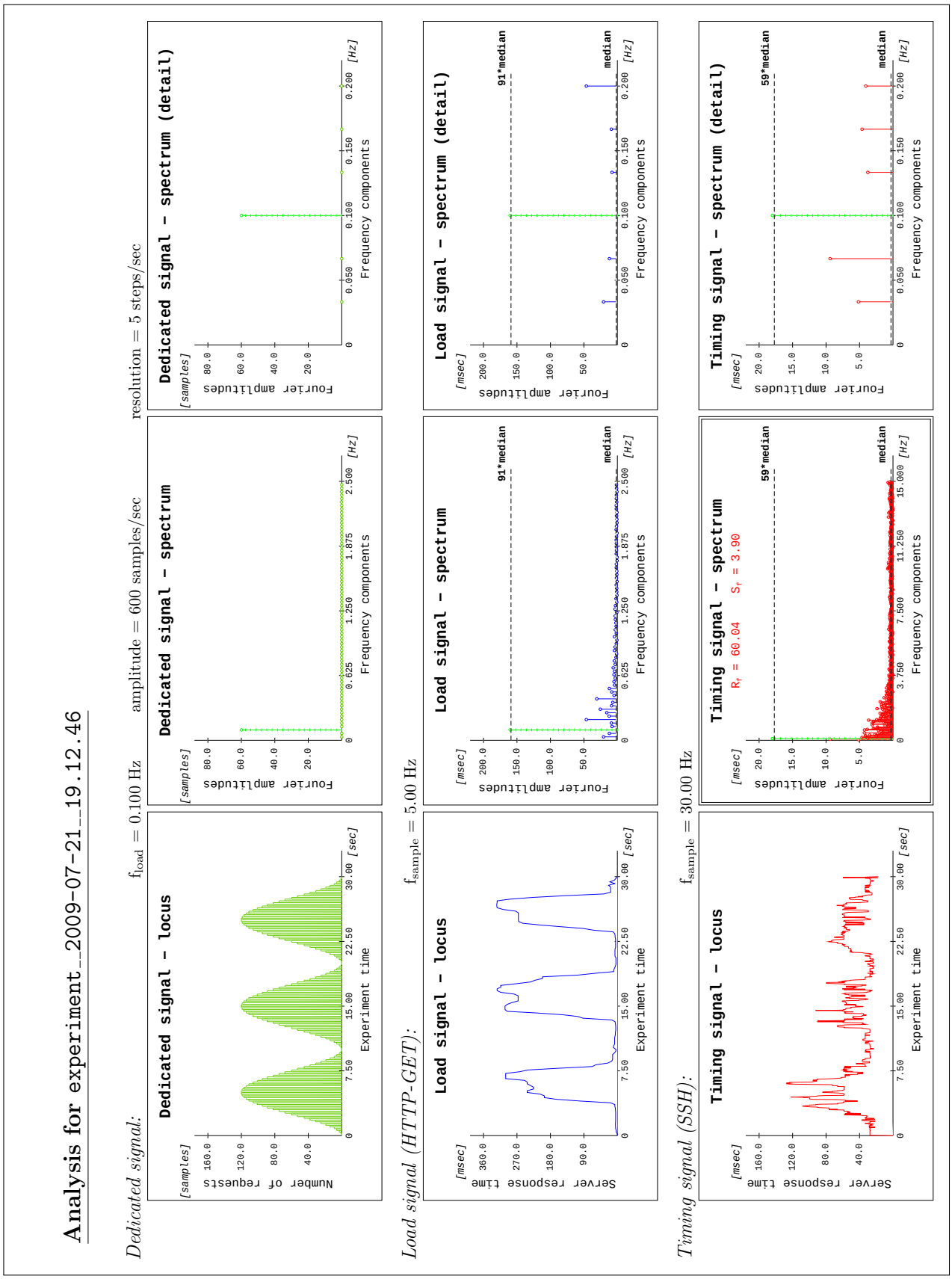


Figure A.2: Basic experiment (direct cabling) – Filtered report

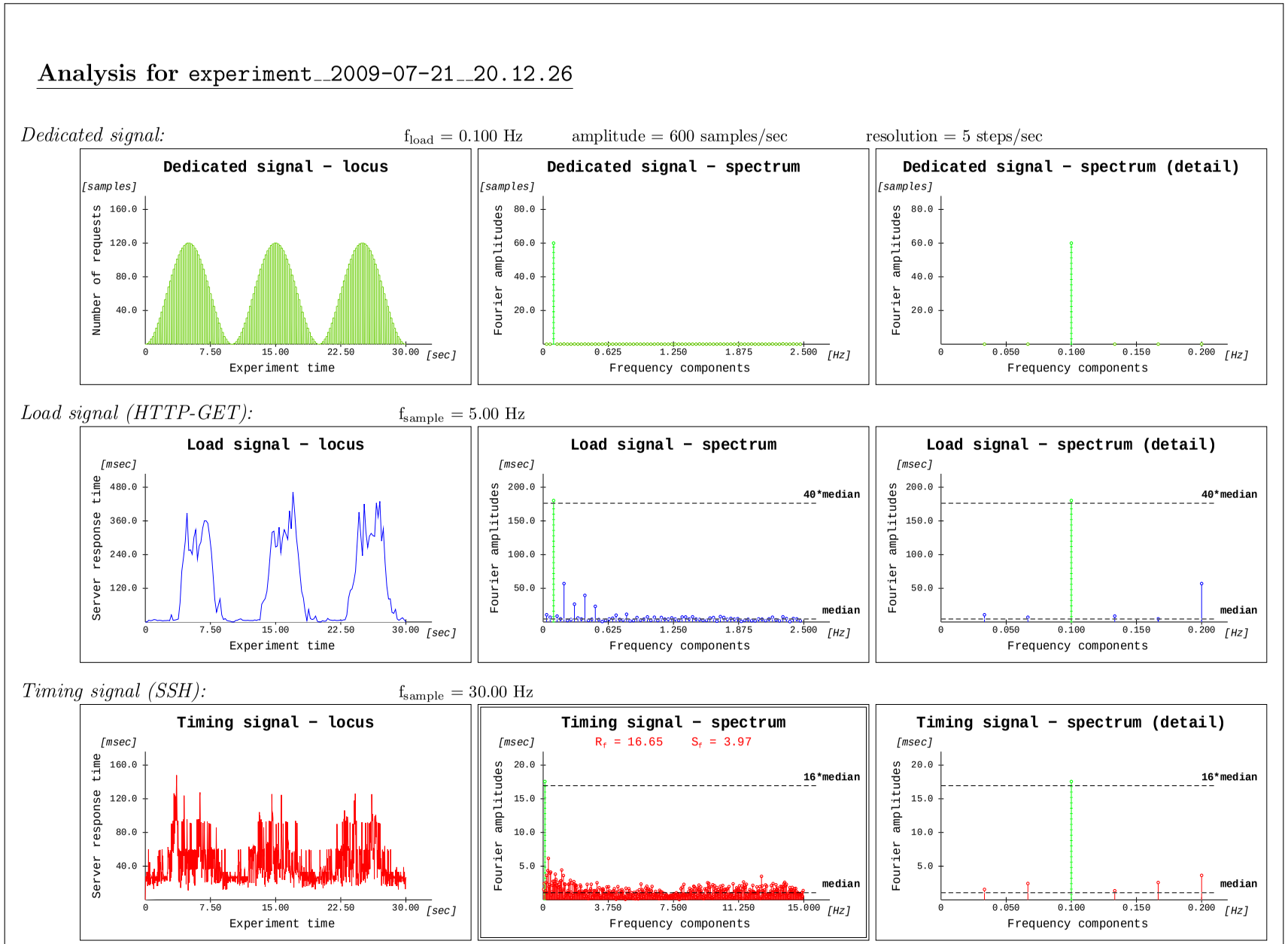


Figure A.3: Basic experiment (semi-switched) – Report

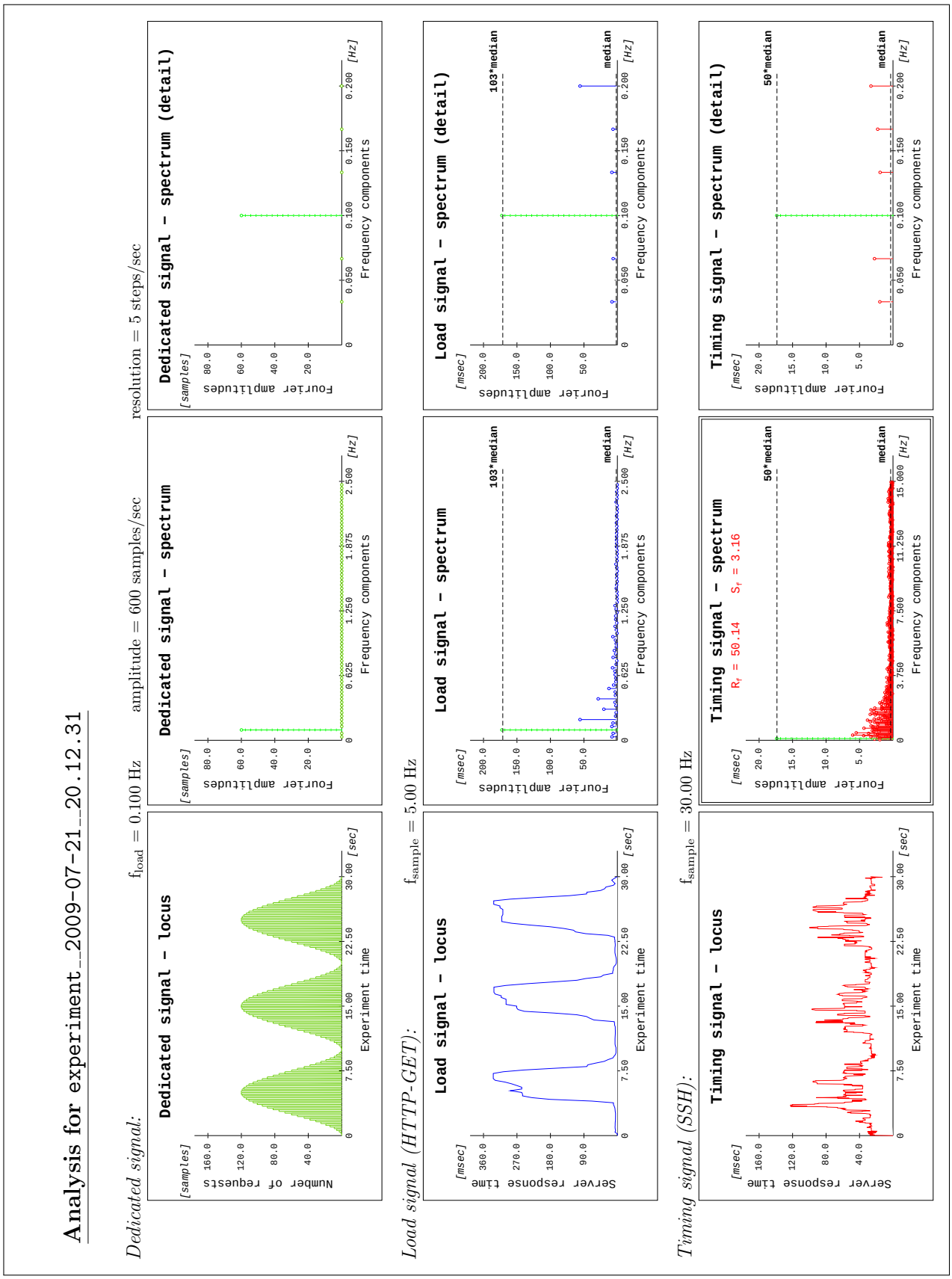


Figure A.4: Basic experiment (semi-switched) – Filtered report

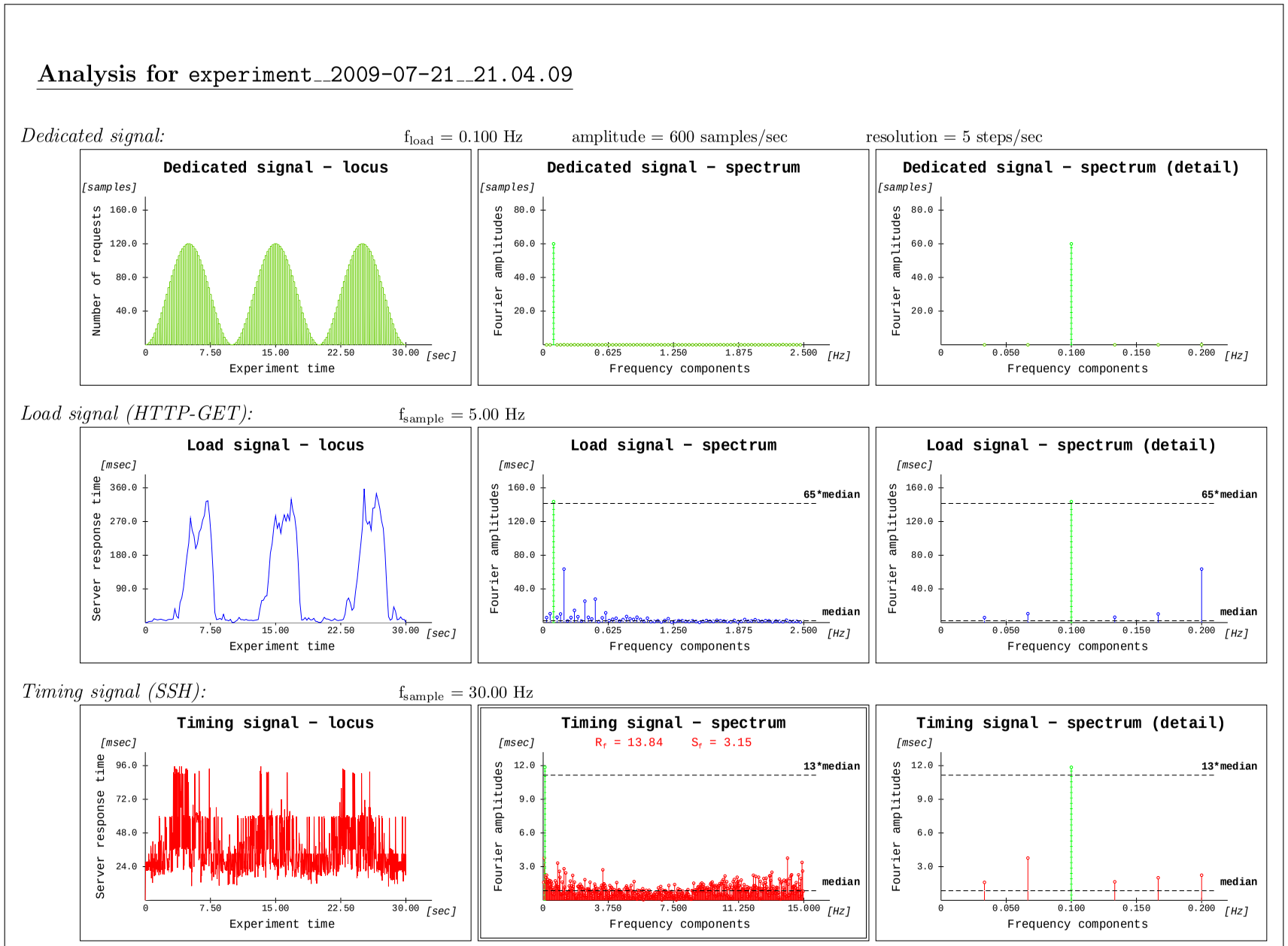


Figure A.5: Basic experiment (switched) – Report

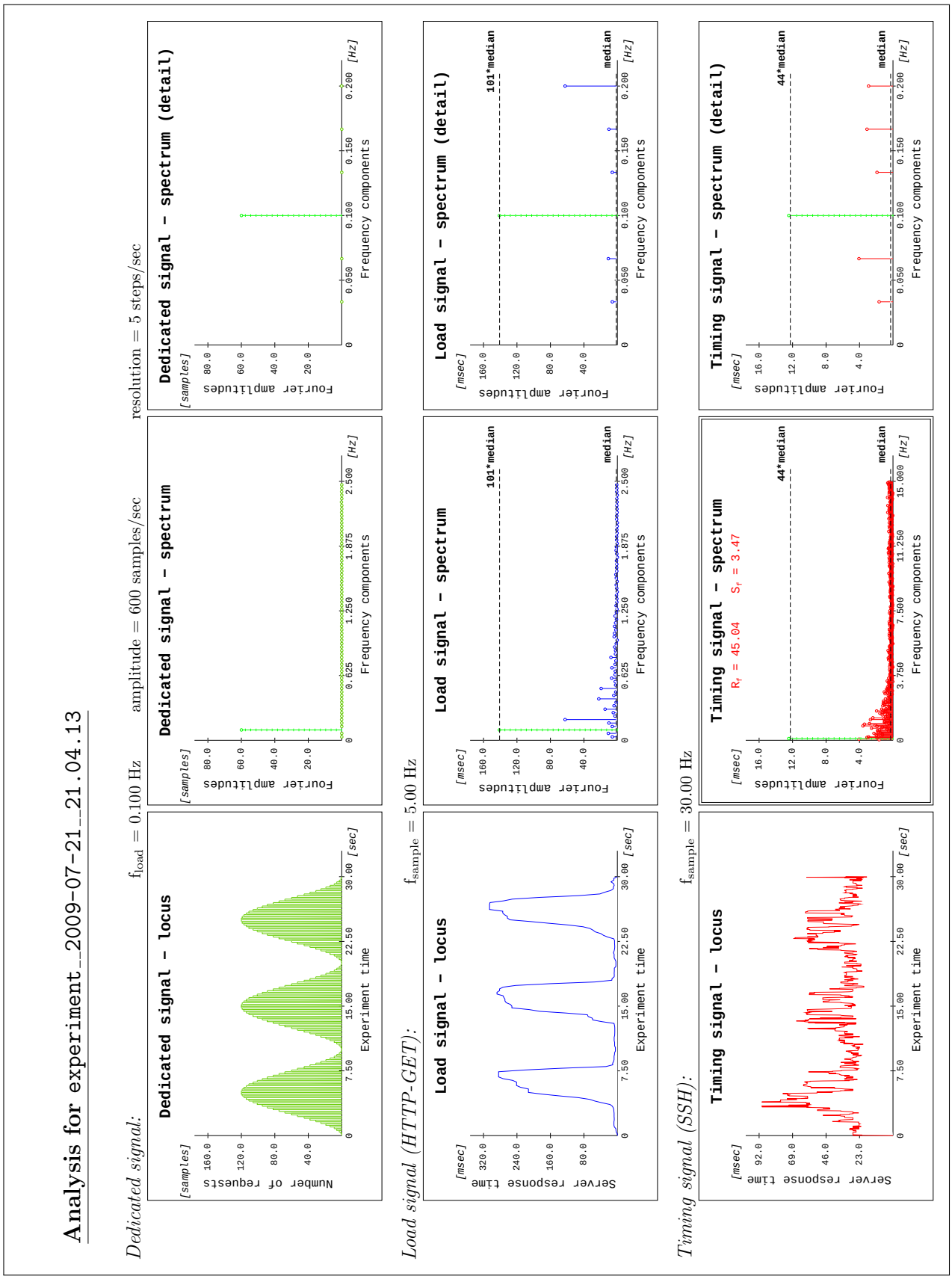


Figure A.6: Basic experiment (switched) – Filtered report

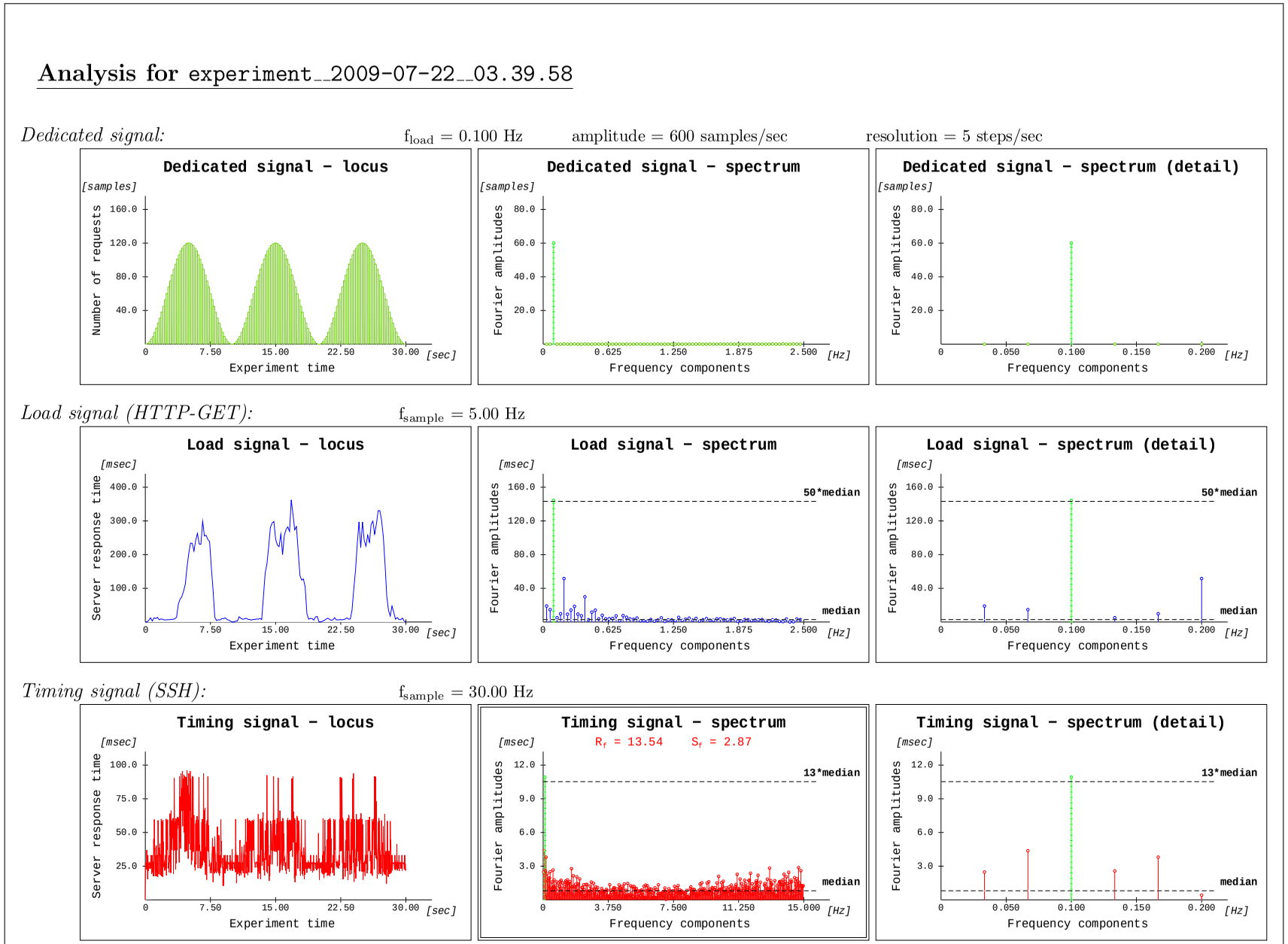


Figure A.7: Background cross-traffic – Report

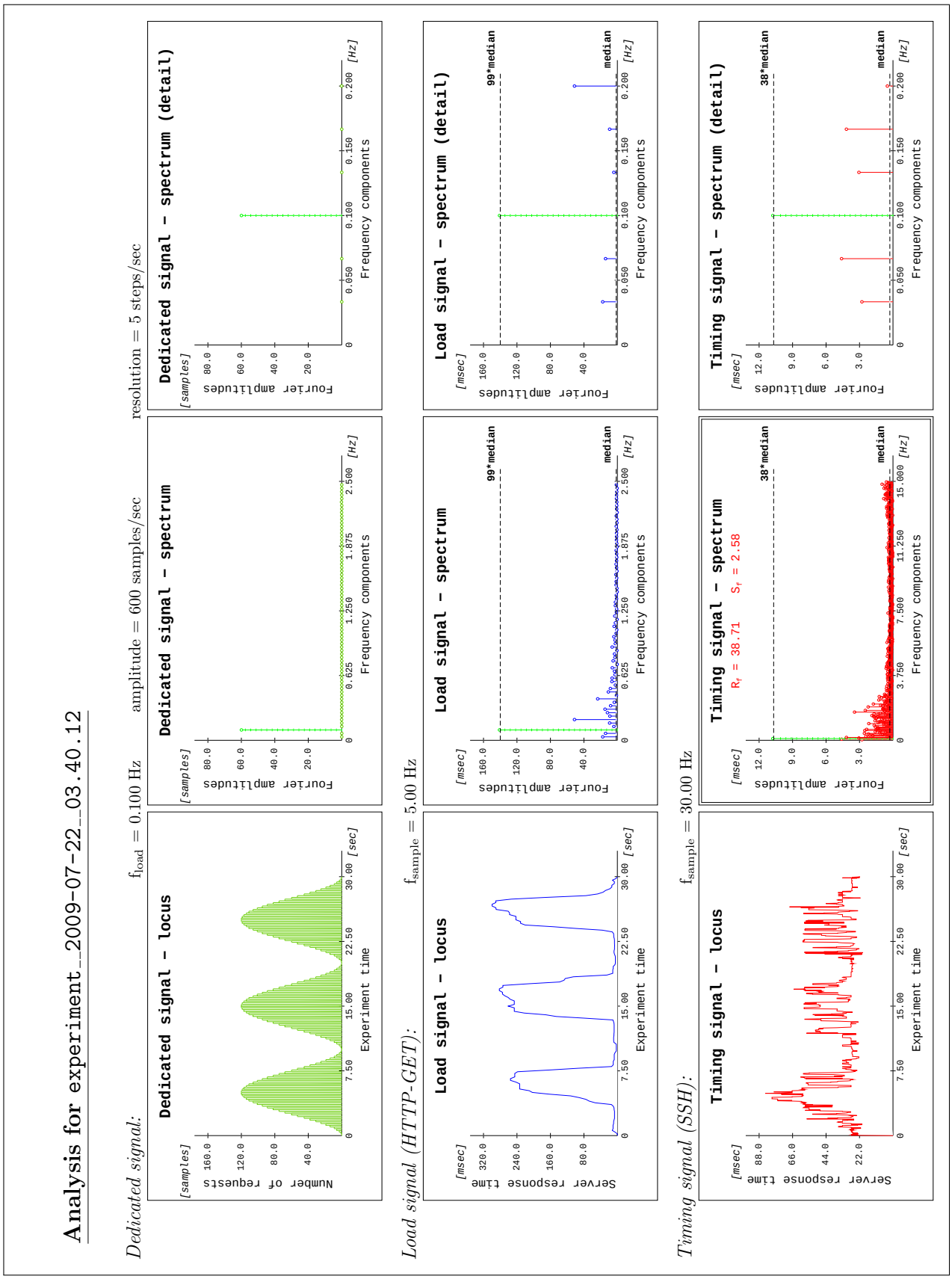


Figure A.8: Background cross-traffic – Filtered report

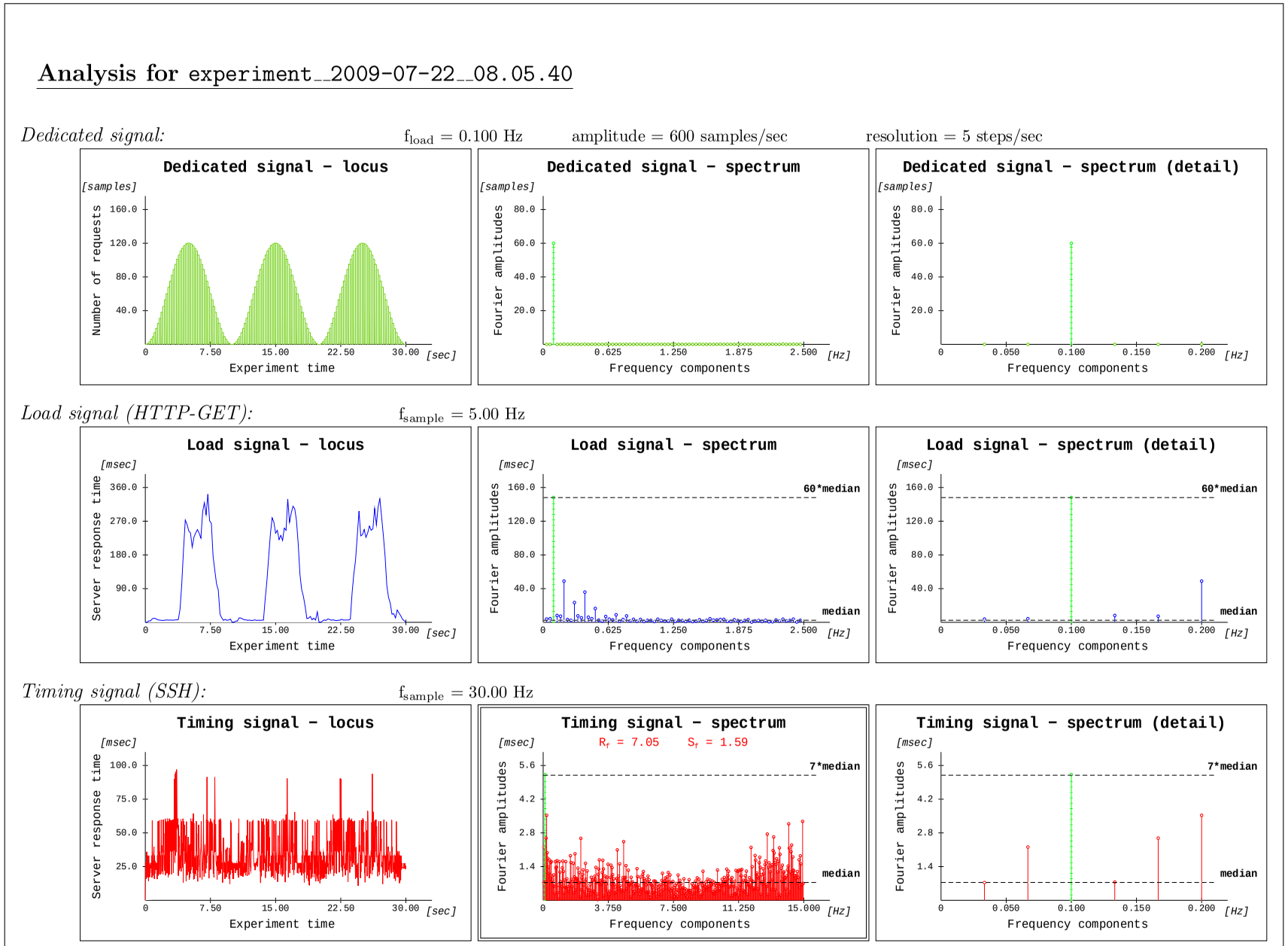


Figure A.9: Webserver cross-traffic – Report

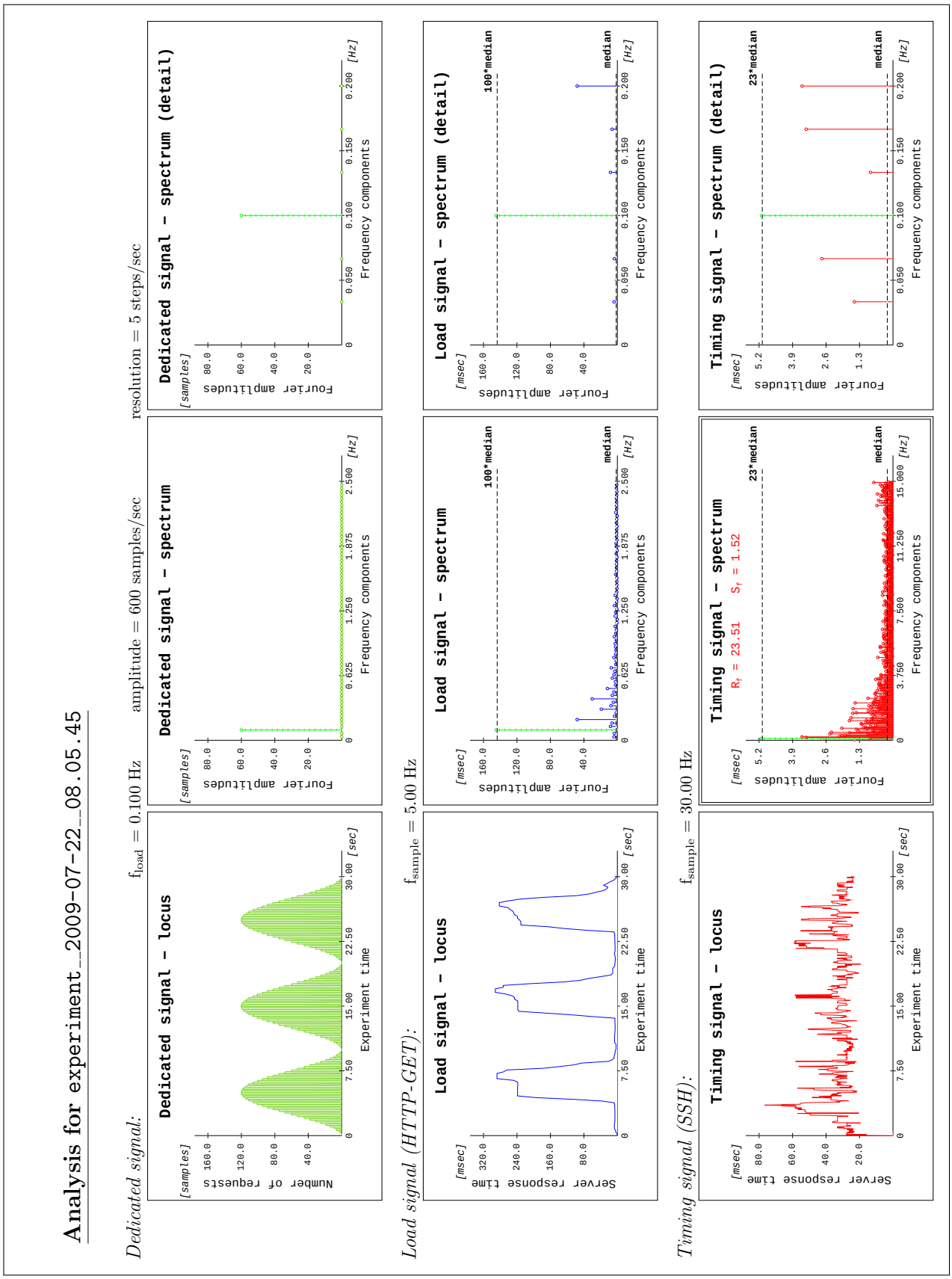


Figure A.10: Webserver cross-traffic – Filtered report

A Appendix

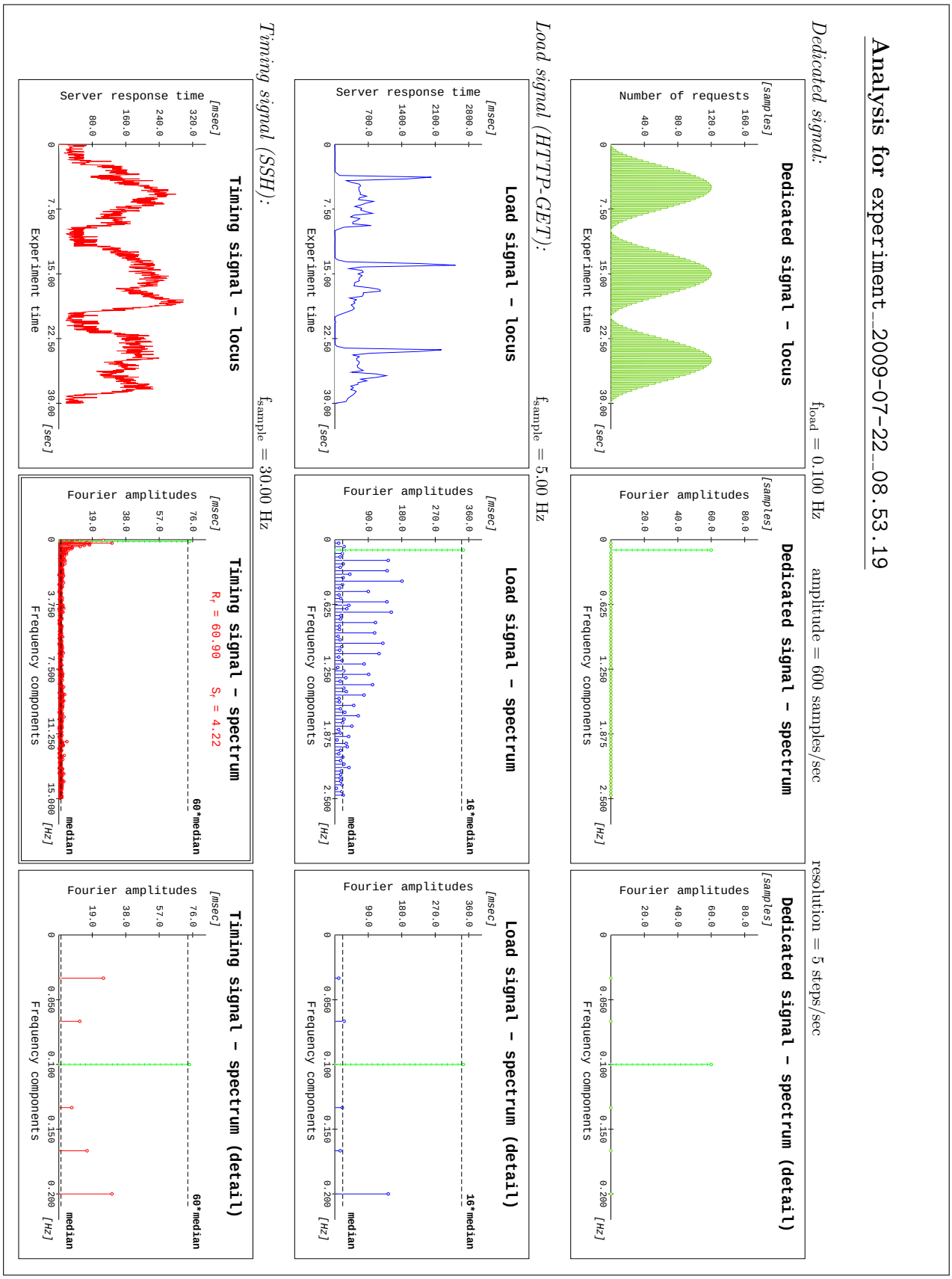


Figure A.11: Database cross-traffic – Report

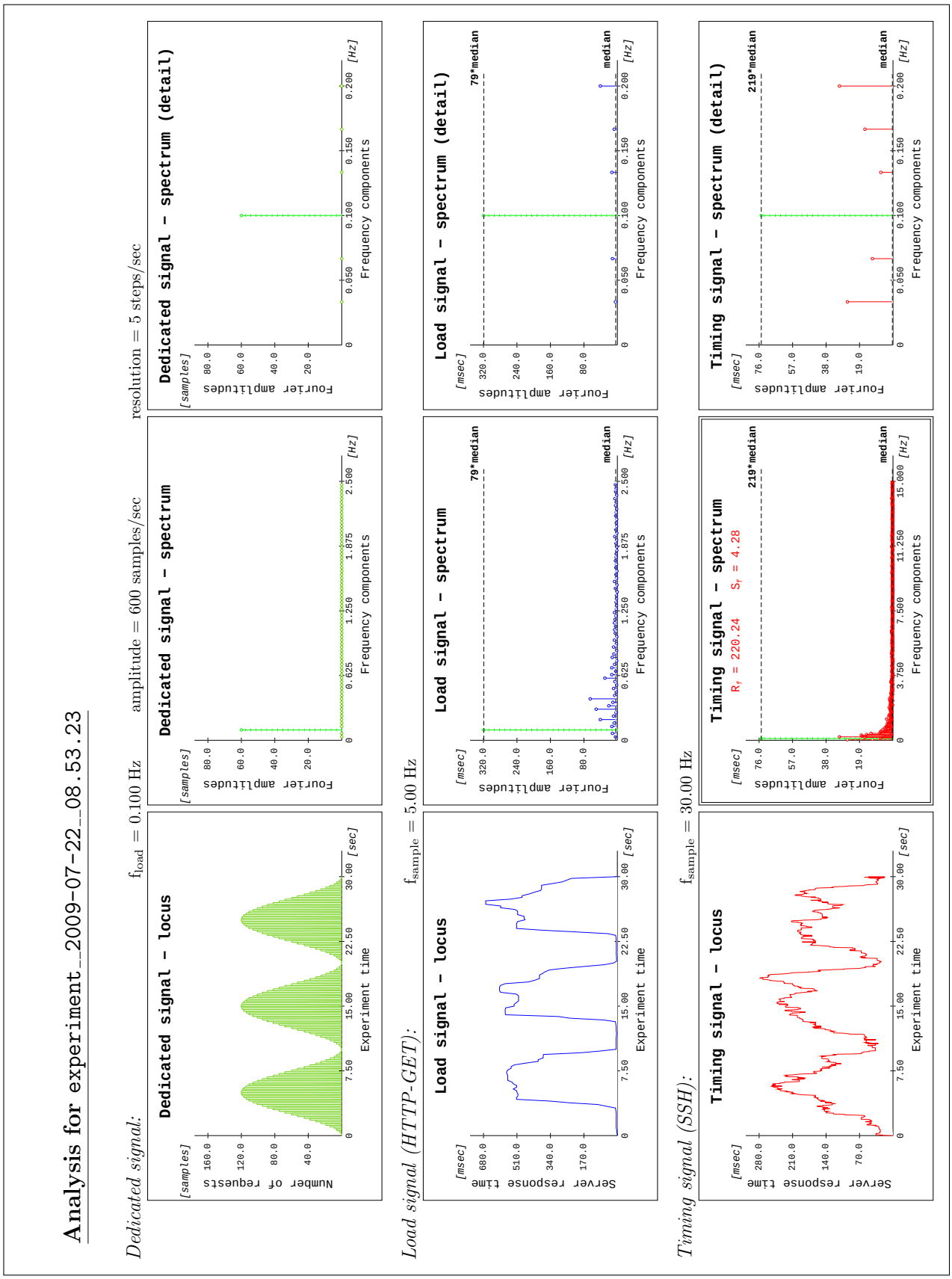


Figure A.12: Database cross-traffic – Filtered report

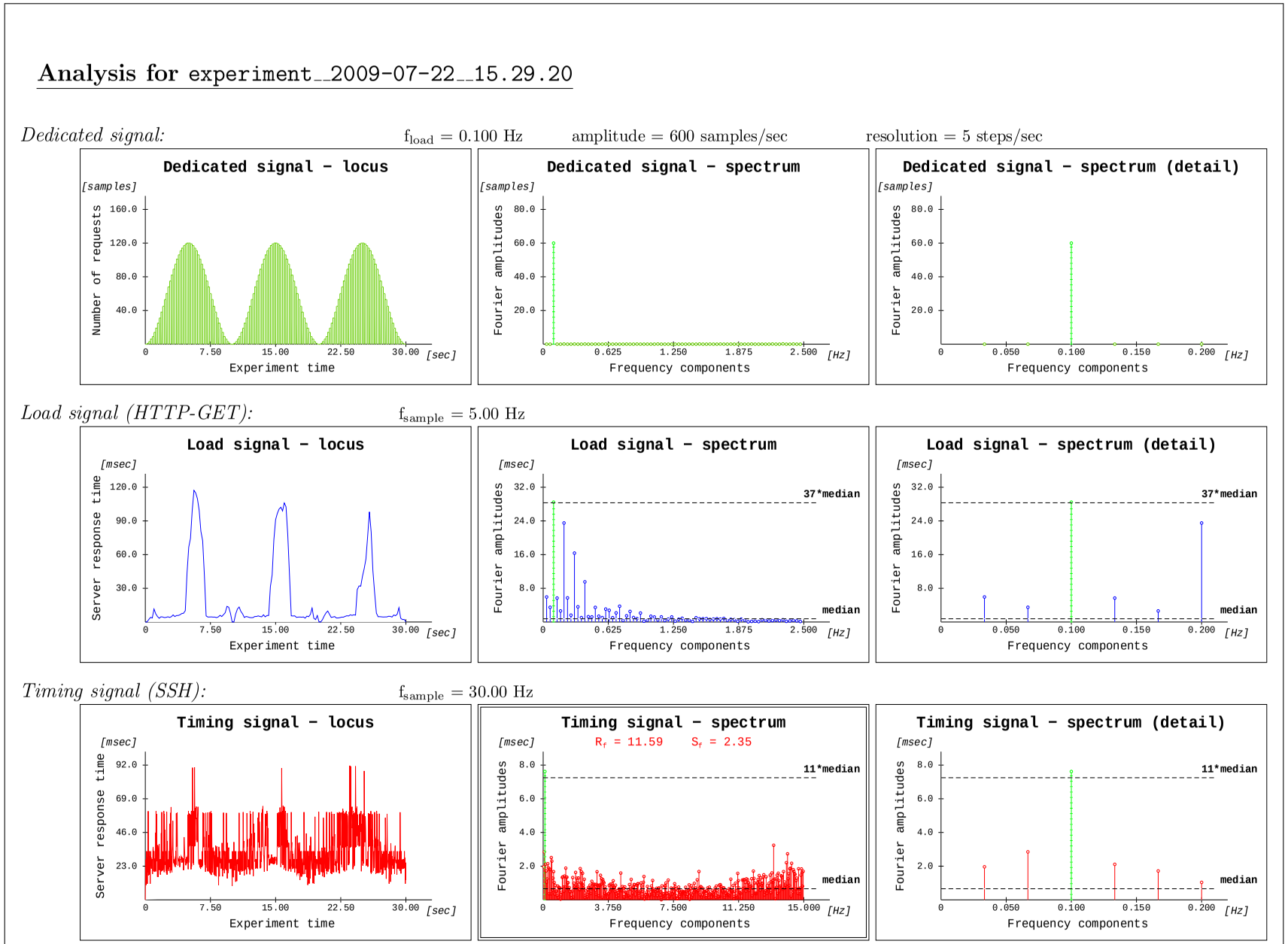


Figure A.13: Database caching – Report

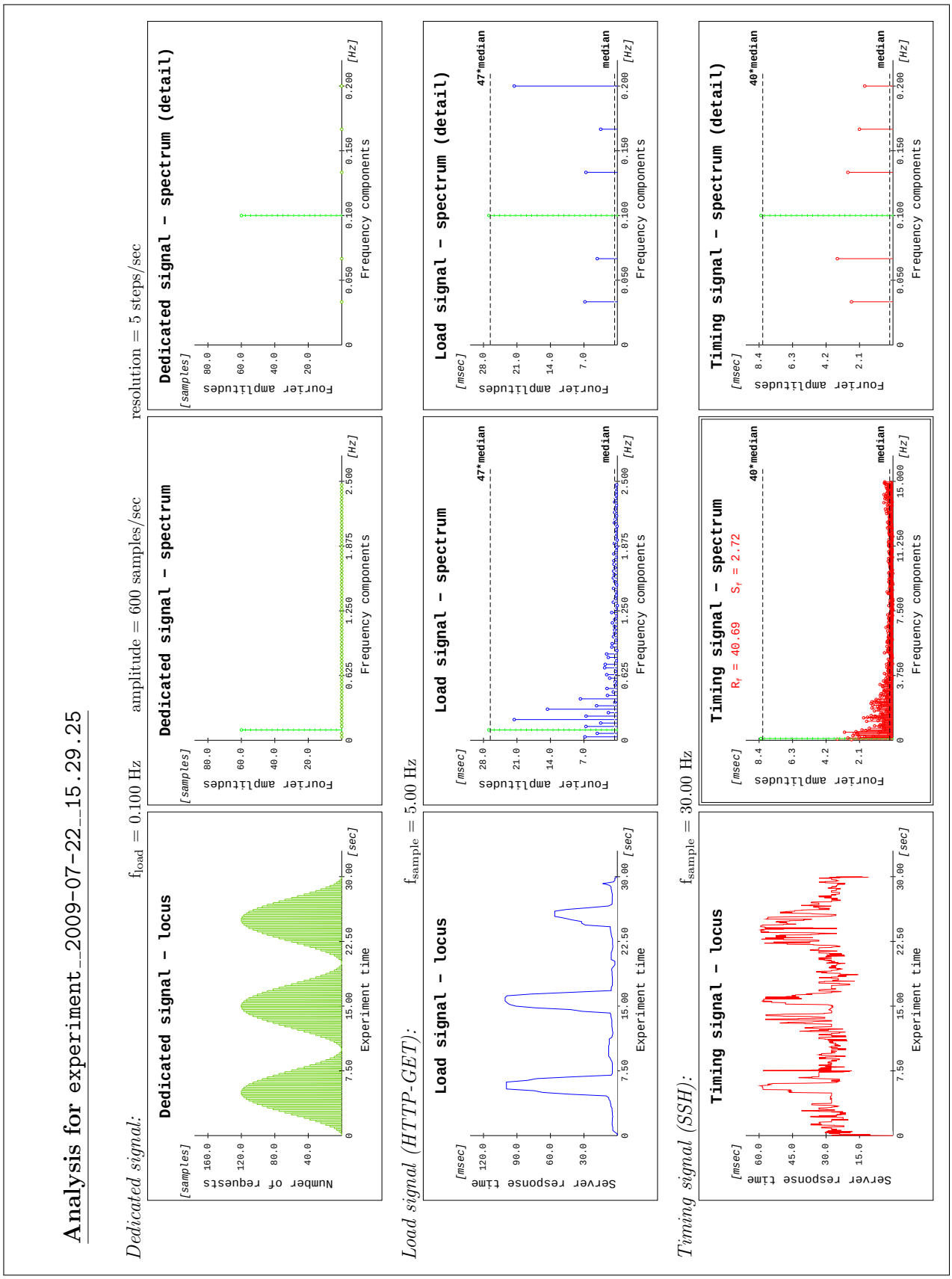


Figure A.14: Database caching – Filtered report

A Appendix

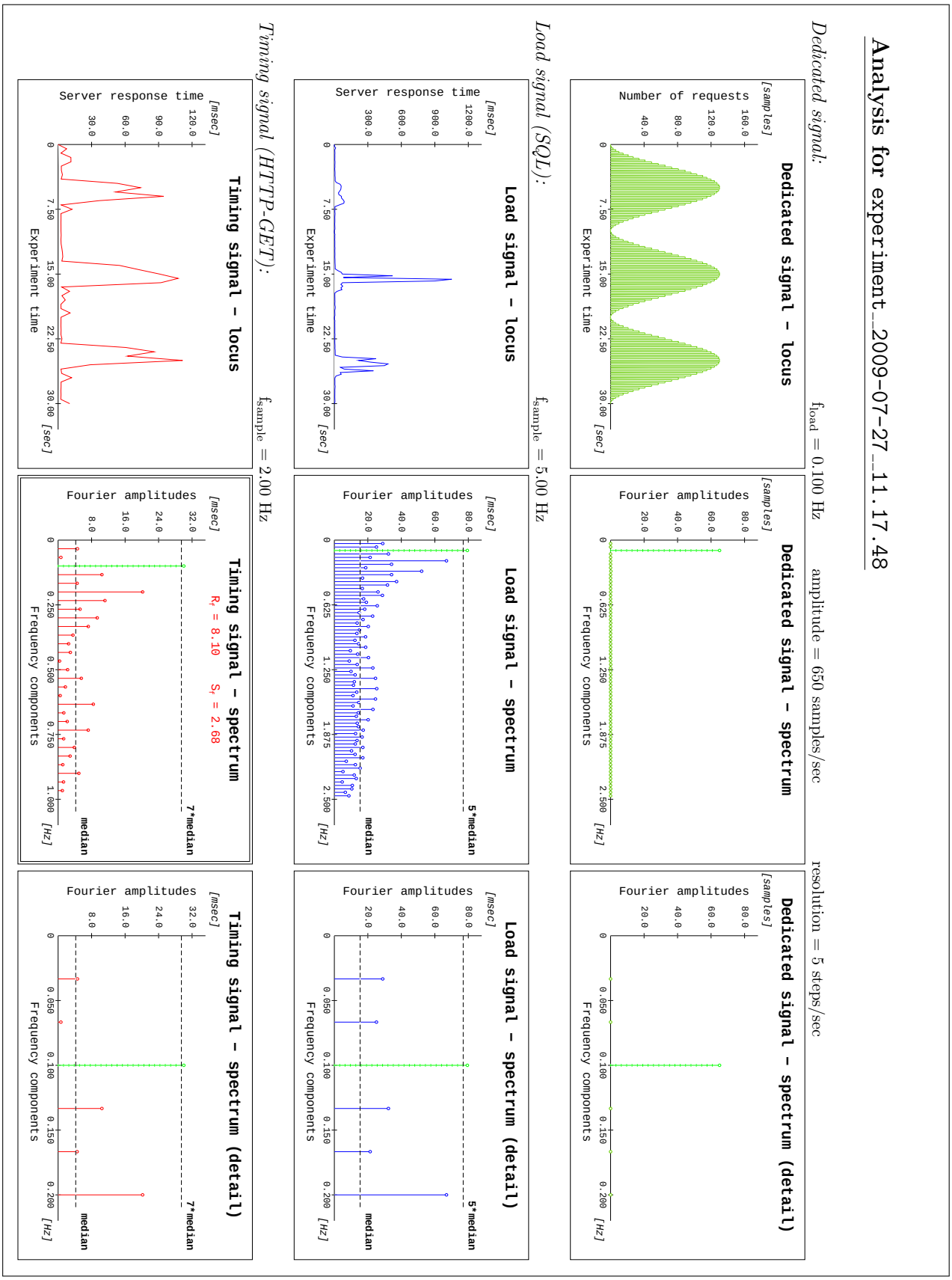


Figure A.15: Reverse direction – Report

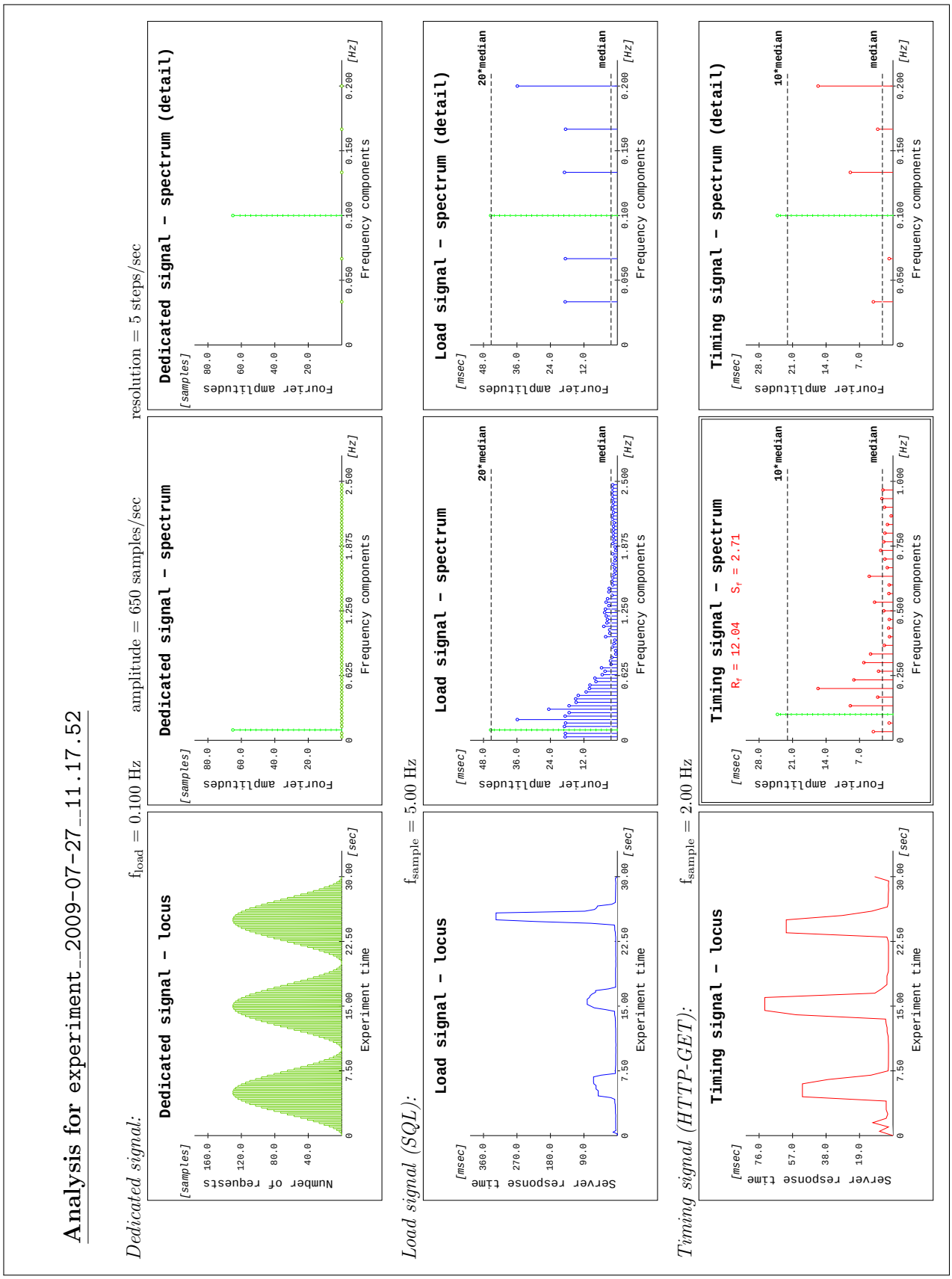


Figure A.16: Reverse direction – Filtered report

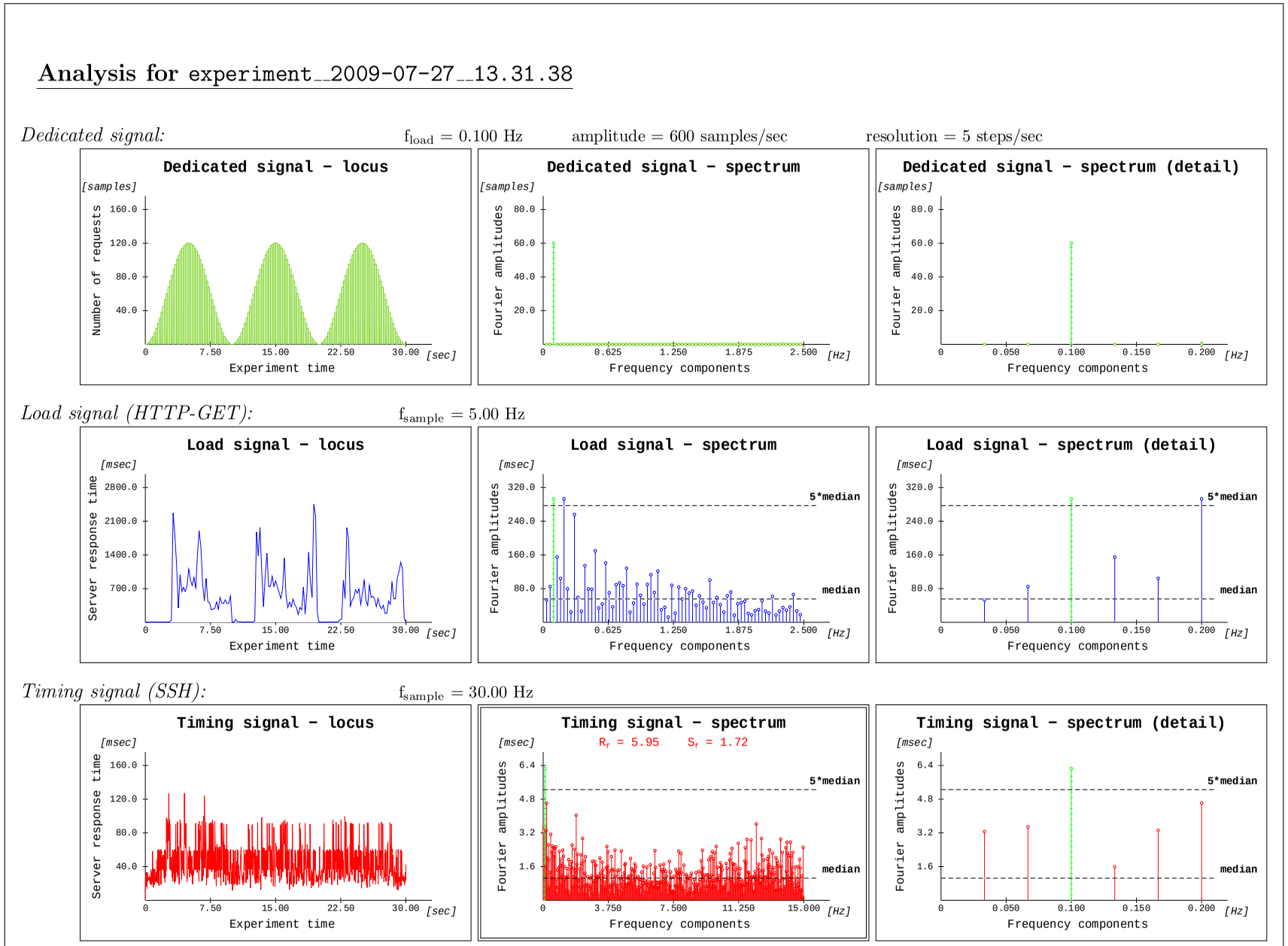


Figure A.17: Frequency overlays – Report

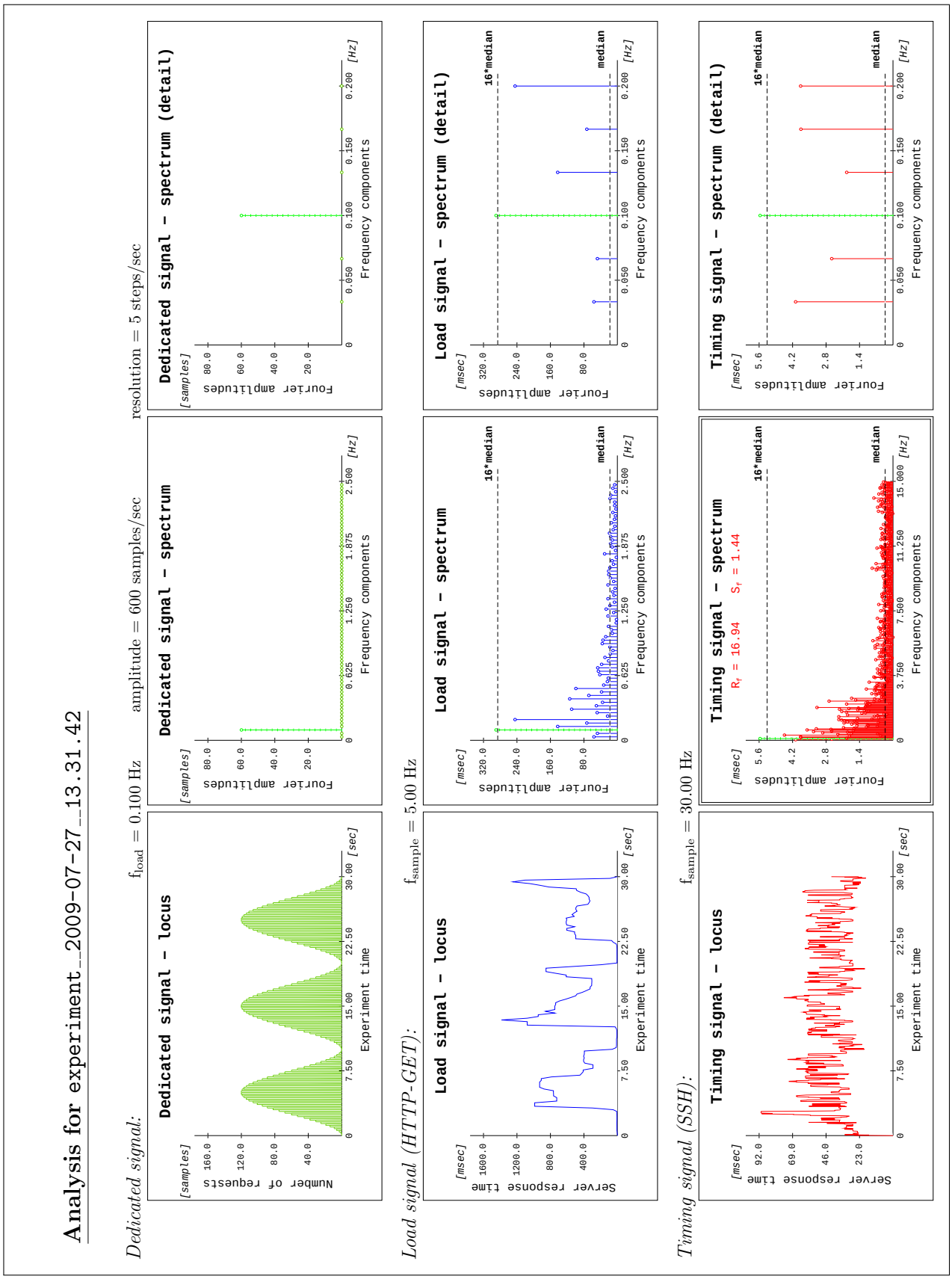


Figure A.18: Frequency overlays – Filtered report

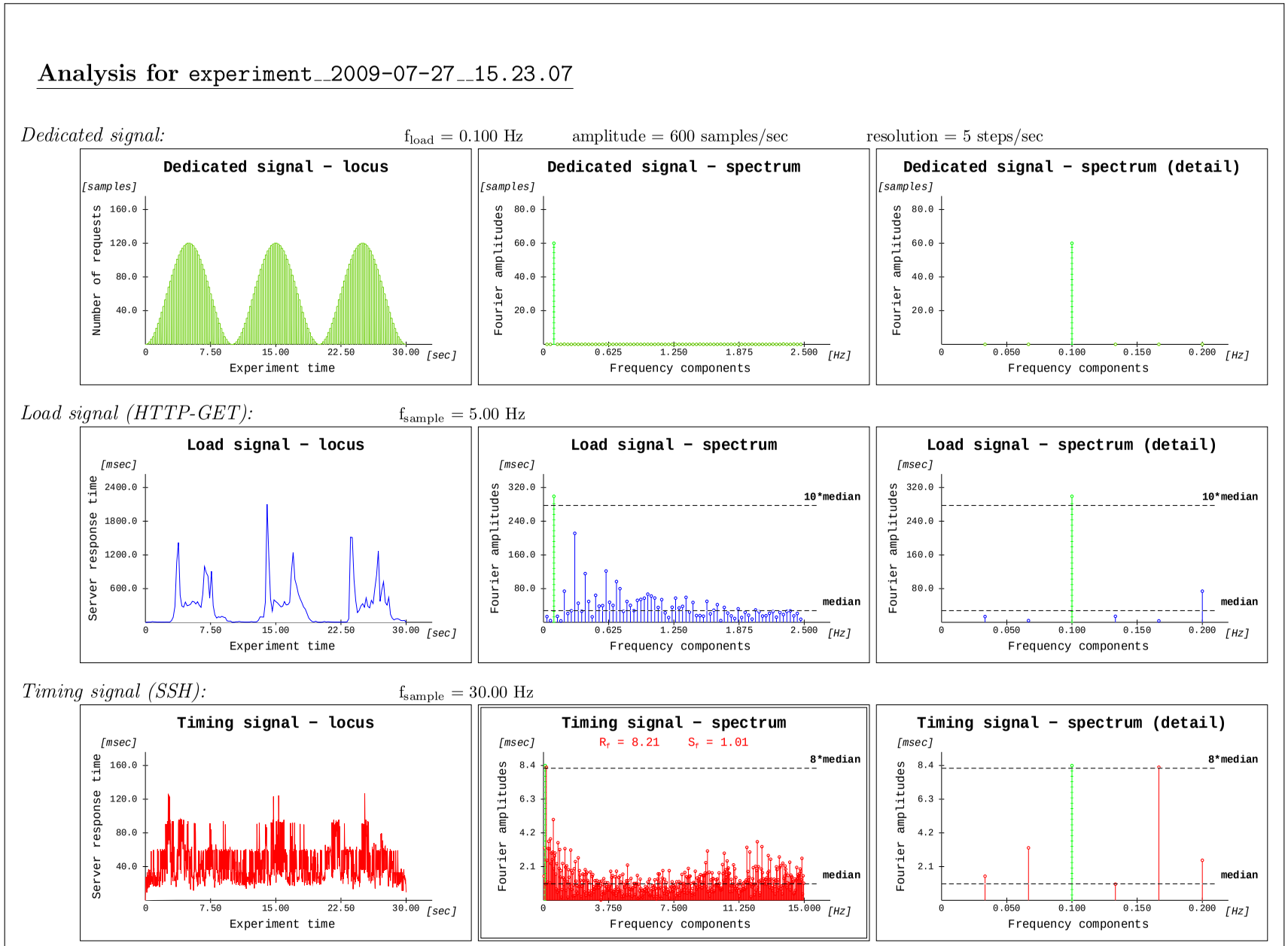


Figure A.19: Multi-signal interference – Report

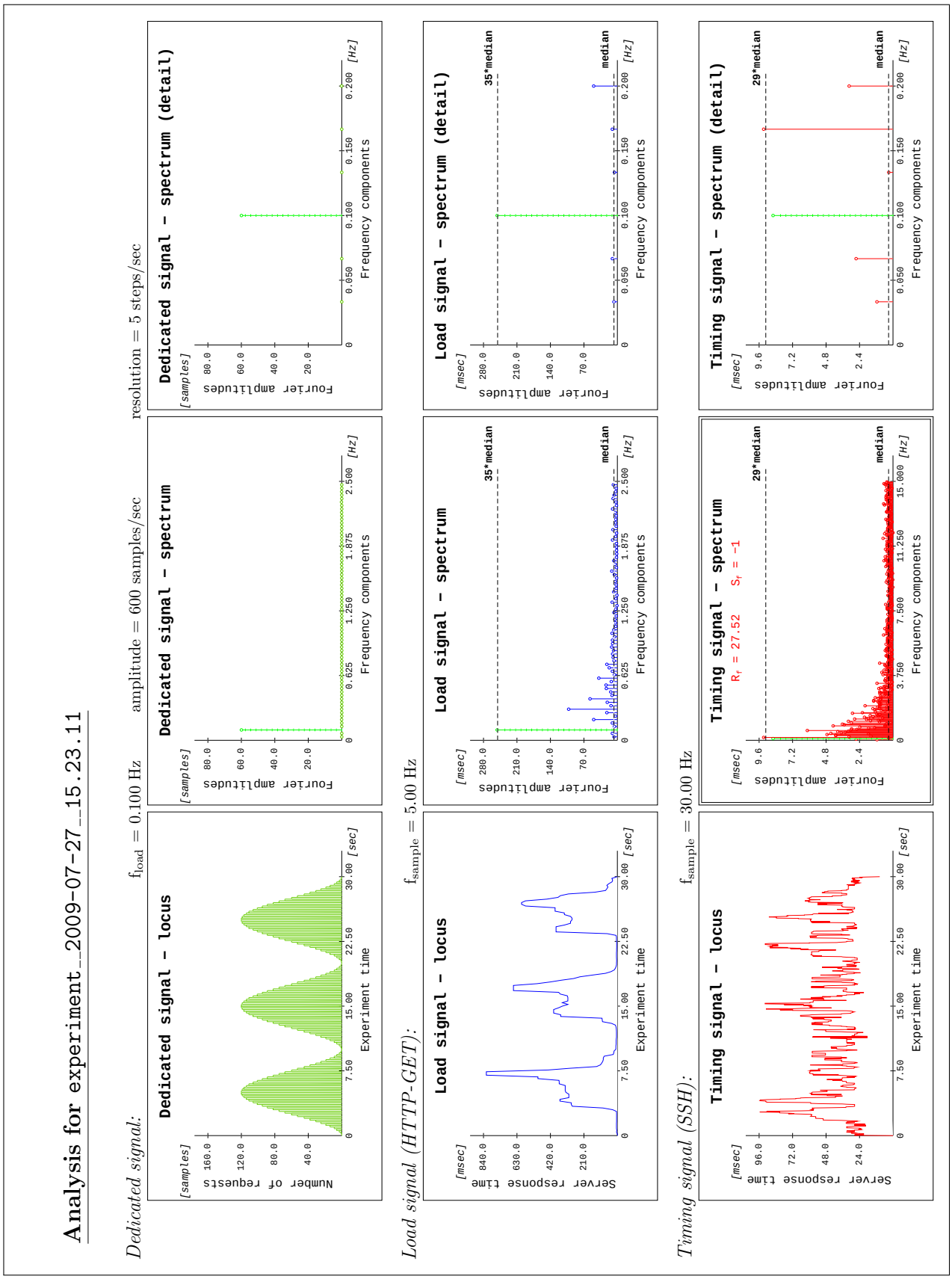


Figure A.20: Multi-signal interference – Filtered report

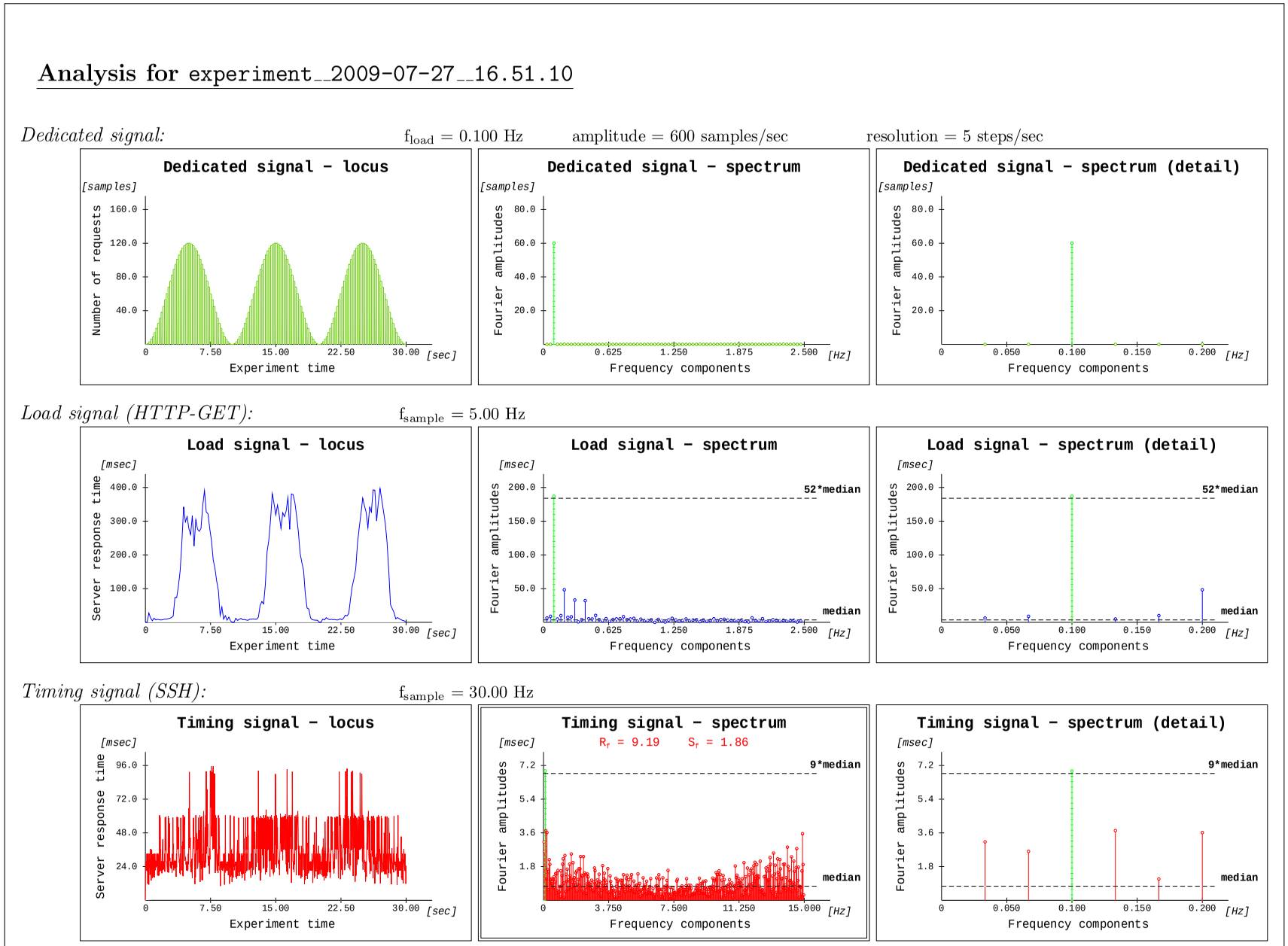


Figure A.21: Multi-level dependencies – Report

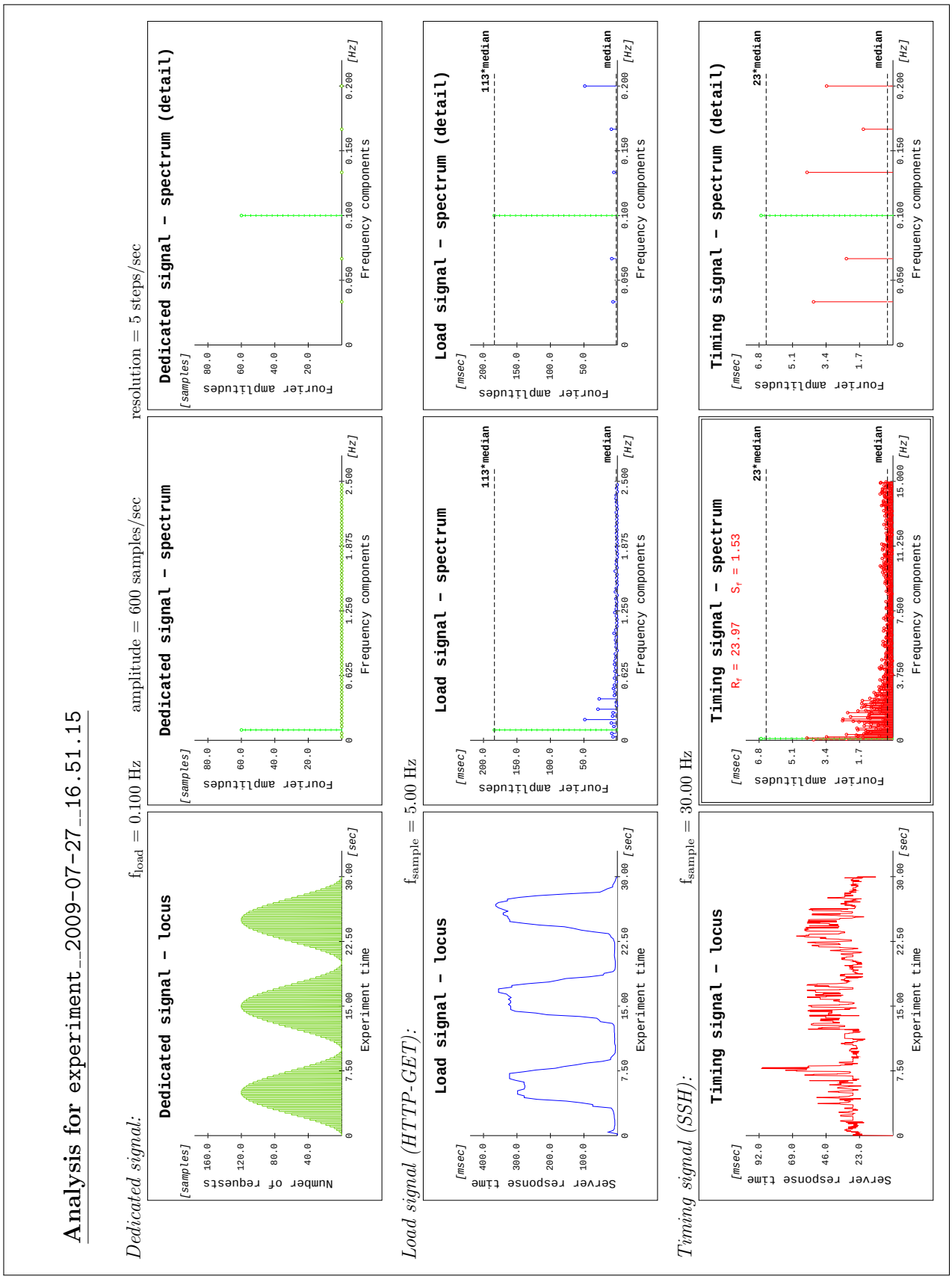


Figure A.22: Multi-level dependencies – Filtered report

A Appendix

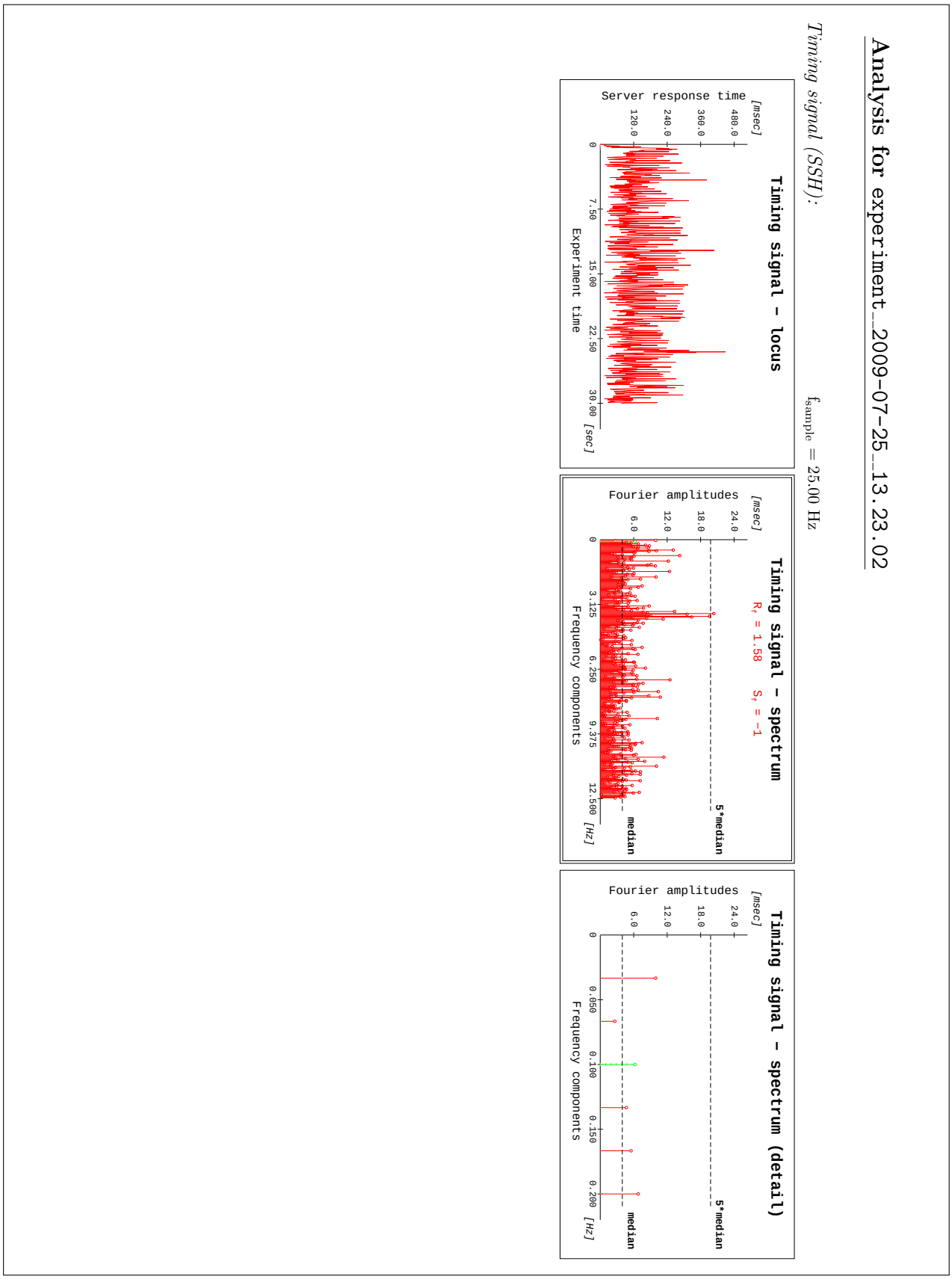


Figure A.23: Productive deployment (idle timing) – Report

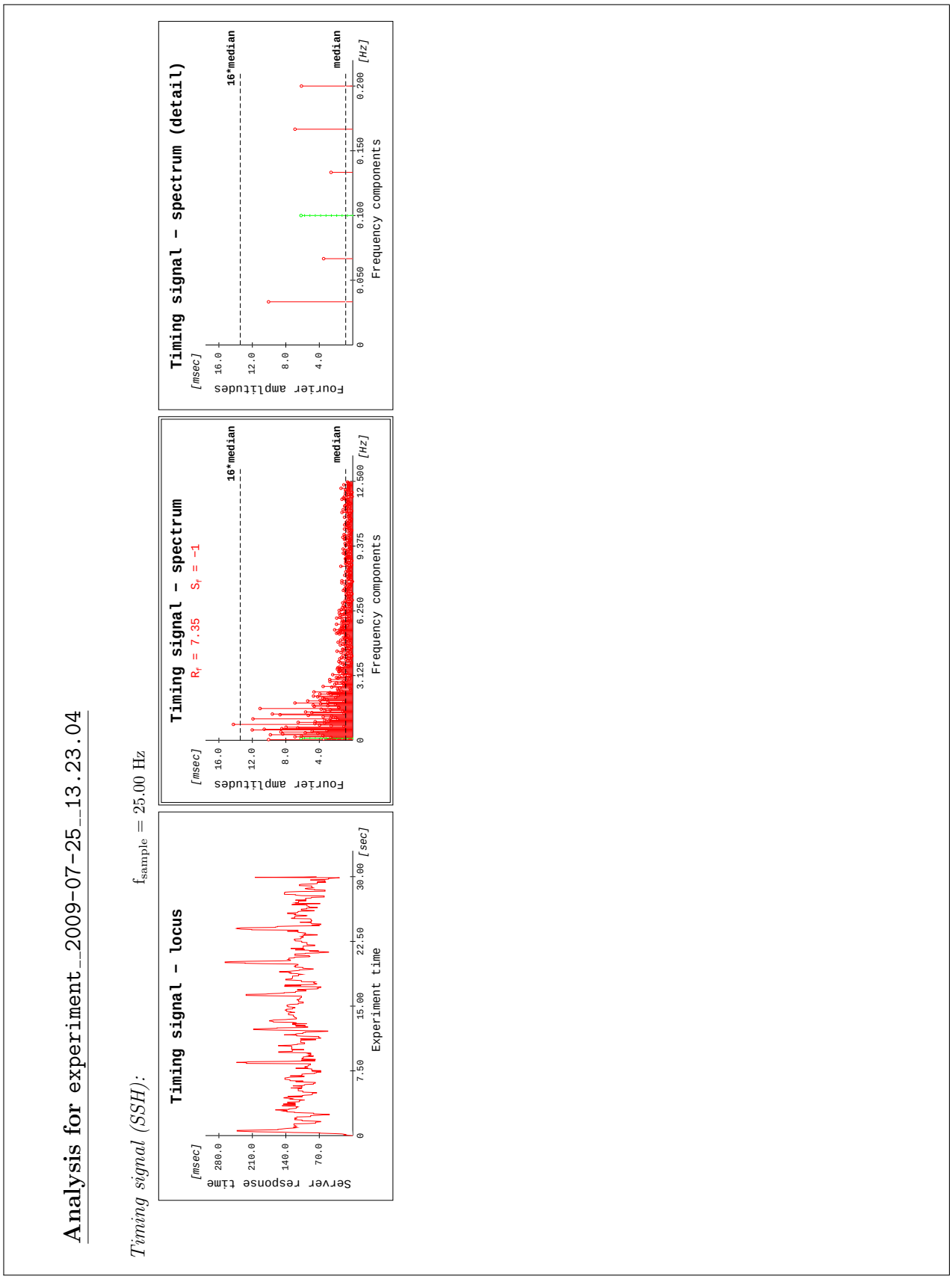


Figure A.24: Productive deployment (idle timing) – Filtered report

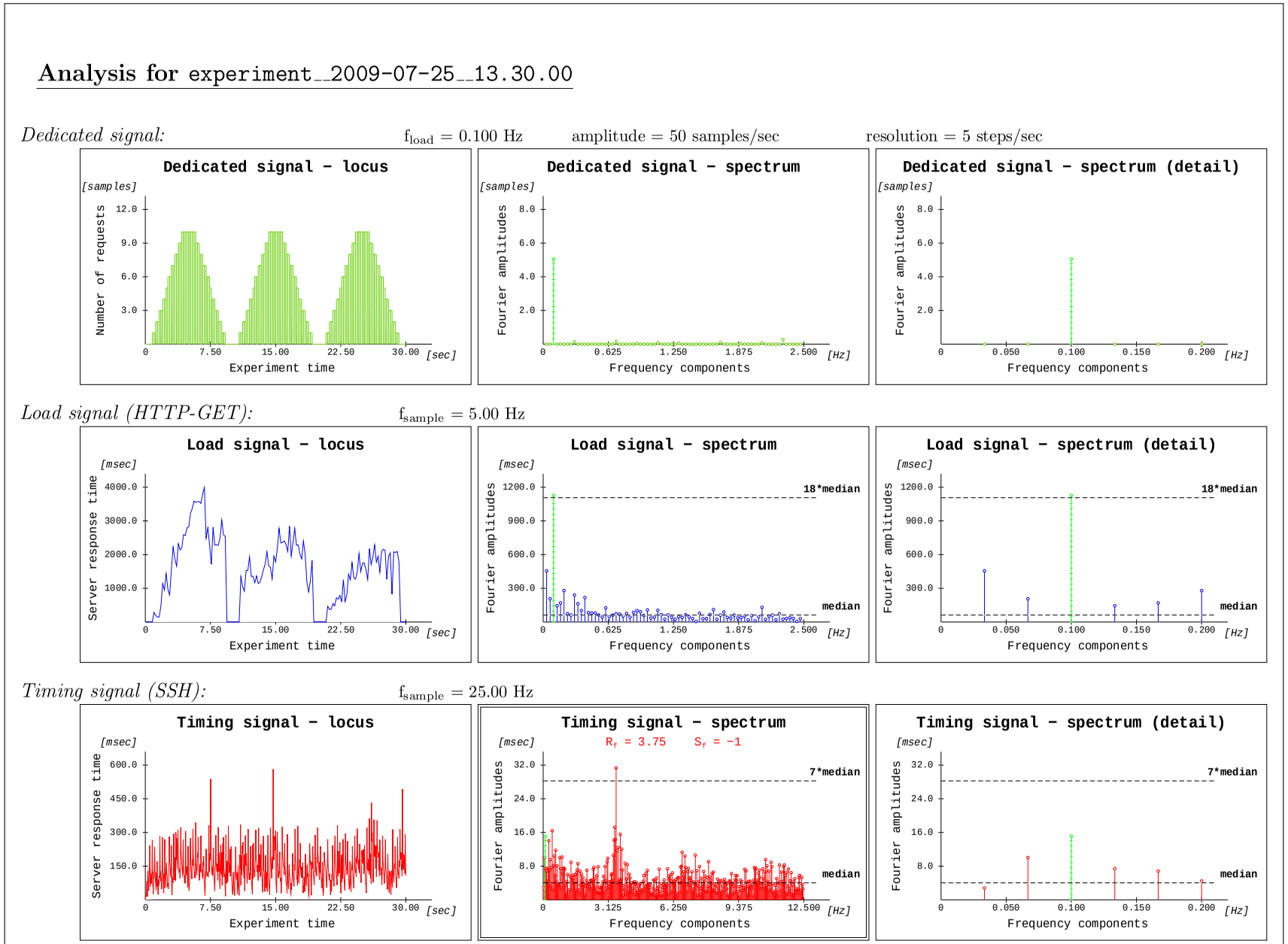


Figure A.25: Productive deployment (operational use) – Report

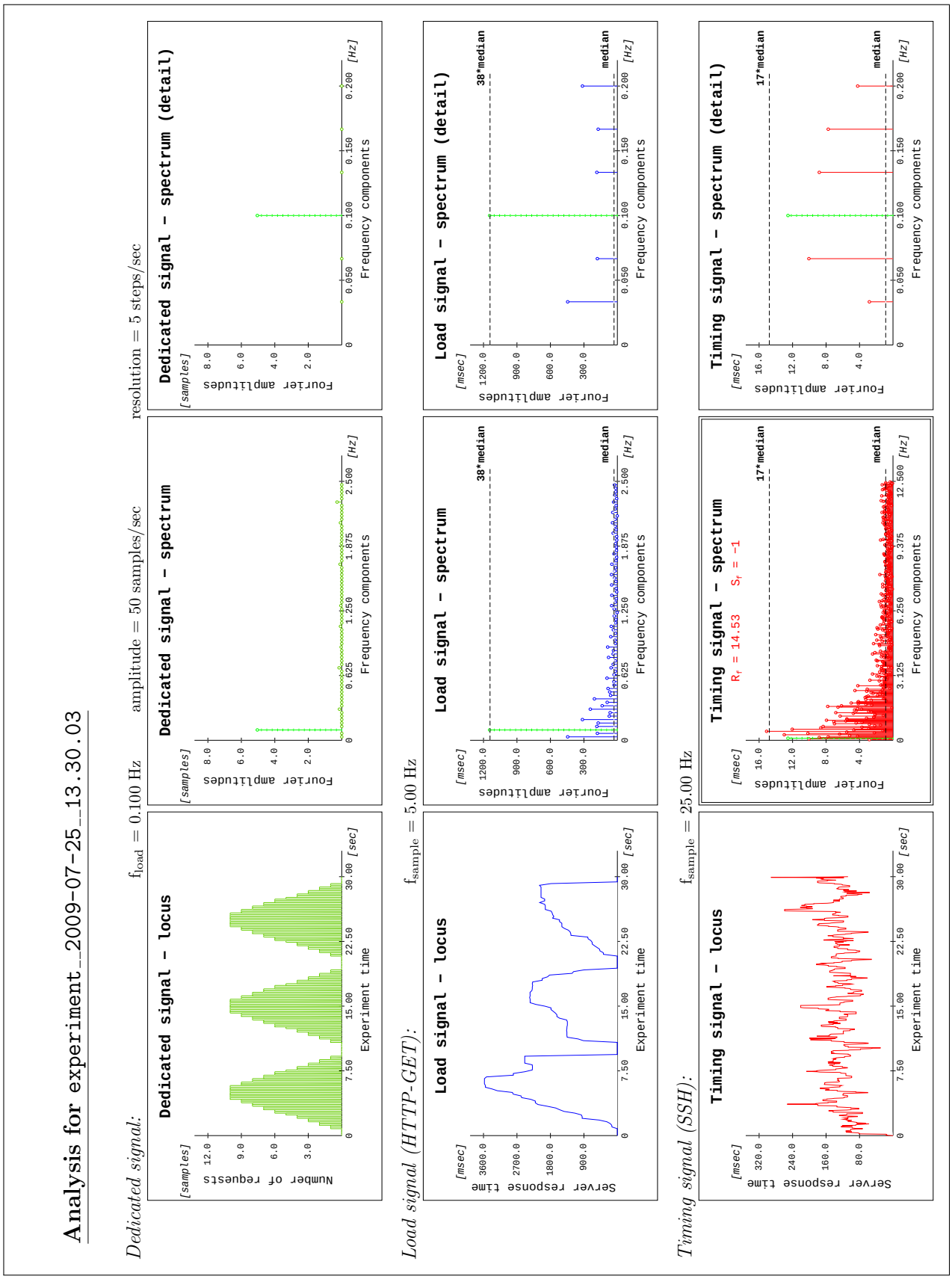
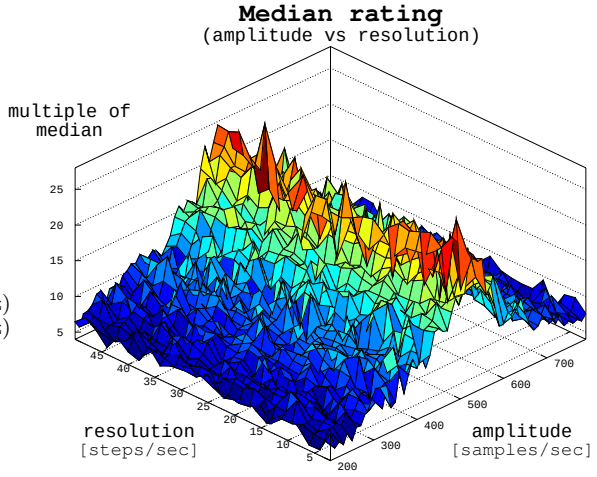
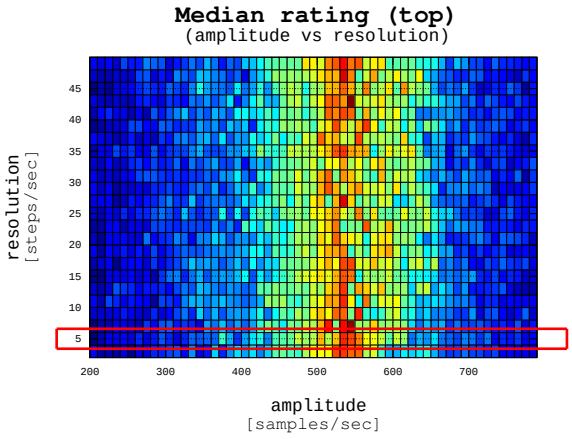


Figure A.26: Productive deployment (operational use) – Filtered report

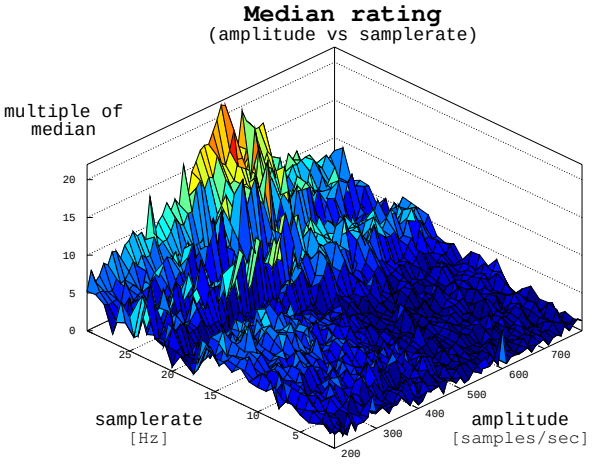
A Appendix

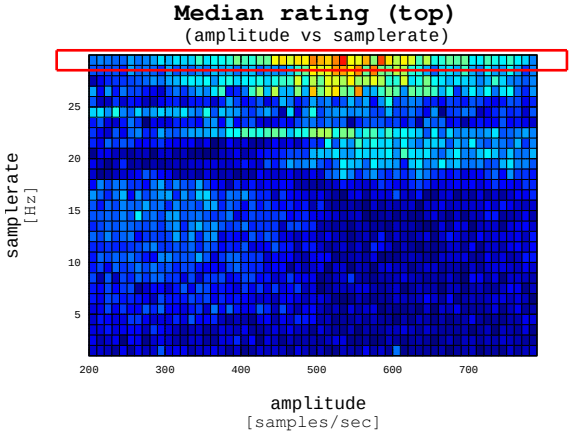
Invariants:
samplerate (30Hz)
frequency (0.1Hz)



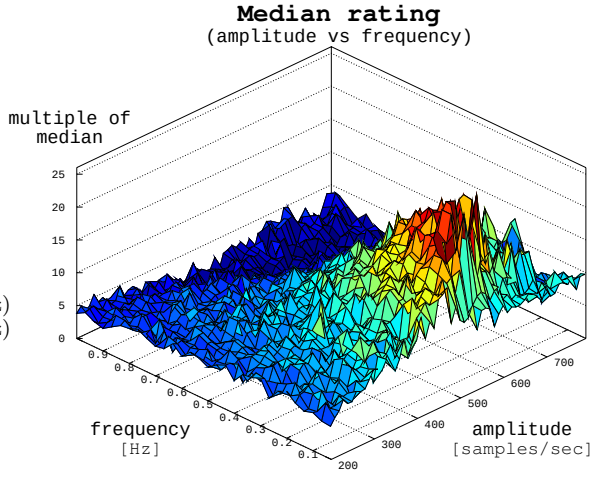


Invariants:
resolution (5Hz)
frequency (0.1Hz)





Invariants:
resolution (5Hz)
samplerate (30Hz)



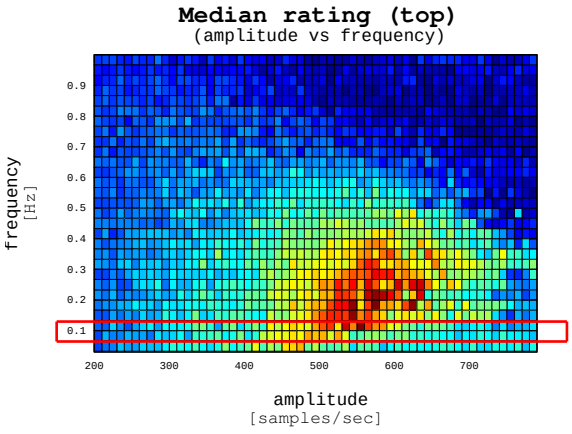


Figure A.27: Unfiltered median rating examination

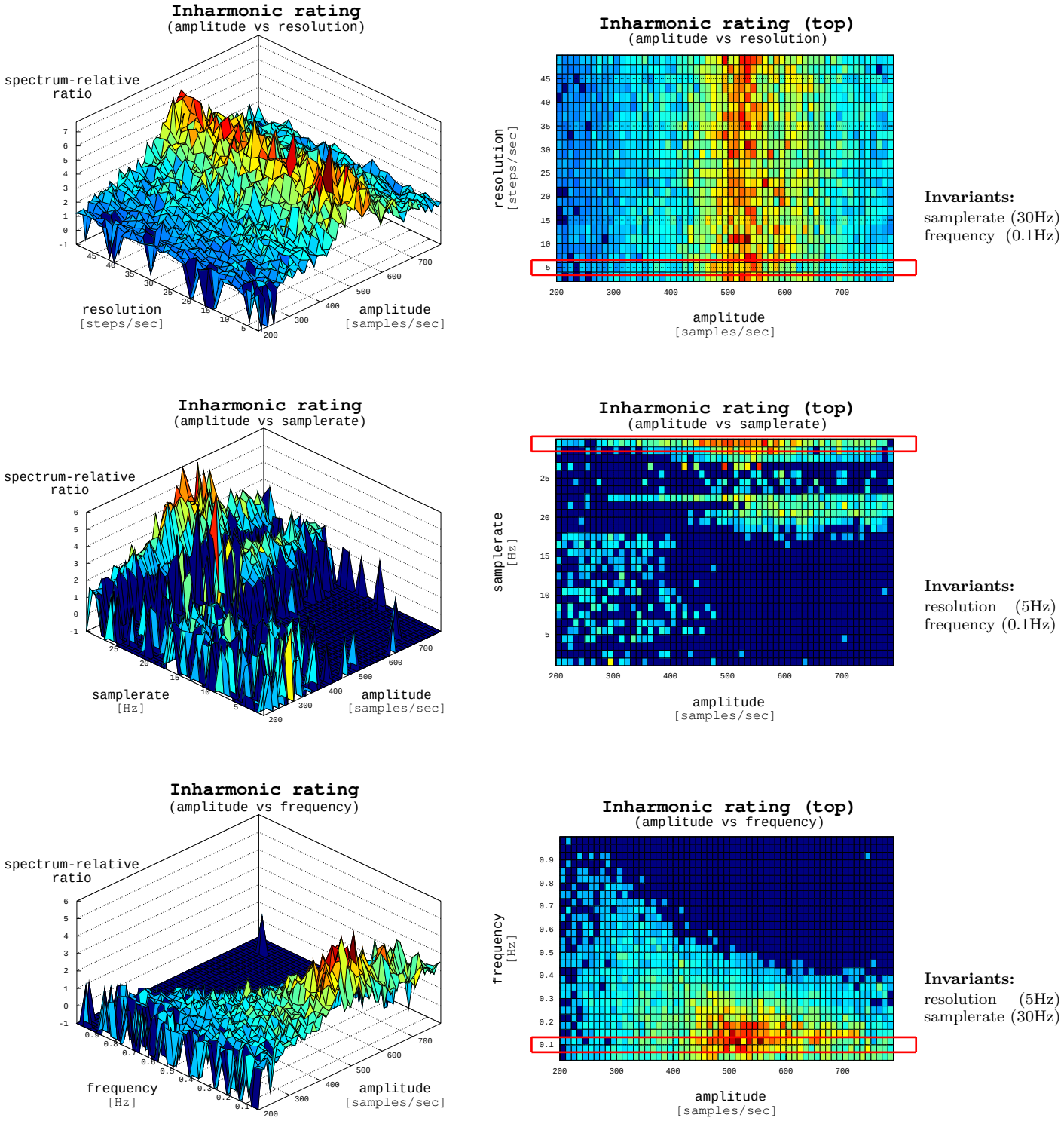


Figure A.28: Unfiltered inharmonic rating examination

A Appendix

DVD missing?
→ schlamp@in.tum.de

Table A.4: DVD with source code and experiment data

"...And chances are if he asks for a glass of milk, he's going to want a cookie to go with it."