Automated Extraction of Network Traffic Models Suitable for Performance Simulation

Piotr Rygielski
Institute of Computer Science
University of Würzburg,
Germany
piotr.rygielski
@uni-wuerzburg.de

Viliam Simko
FZI Forschungszentrum
Informatik
Haid-und-Neu-Straße 10–14,
76131 Karlsruhe, Germany
simko@fzi.de

Felix Sittner
Zentrum für Telematik e.V.
97218 Gerbrunn, Germany
felix.sittner
@telematik-zentrum.de

Doris Aschenbrenner
Zentrum für Telematik e.V.
97218 Gerbrunn, Germany
doris.aschenbrenner
@telematik-zentrum.de

Samuel Kounev
Institute of Computer Science
University of Würzburg,
Germany
samuel.kounev
@uni-wuerzburg.de

Klaus Schilling
Institute of Computer Science
University of Würzburg,
Germany
schi@informatik.uni-
wuerzburg.de

ABSTRACT

Data centers are increasingly becoming larger and dynamic due to virtualization. In order to leverage the performance modeling and prediction techniques, such as Palladio Component Model or Descartes Modeling Language, in such a dynamic environments, it is necessary to automate the model extraction. Building and maintaining such models manually is not feasible anymore due to their size and the level of details. This paper is focused on traffic models that are an essential part of network infrastructure. Our goal is to decompose real traffic dumps into models suitable for performance prediction using Descartes Network Infrastructure modeling approach. The main challenge was to efficiently encode an arbitrary signal in the form of simple traffic generators while maintaining the shape of the original signal. We show that a typical 15 minute long *tcpdump* trace can be compressed to $0.4 - 15\%$ of its original size whereas the relative median of extraction error is close to 0% for the most of the 69 examined traces.

1. INTRODUCTION

Modern IT data center systems have increasingly complex layered architectures composed of loosely-coupled components deployed in virtualized environments. The use of virtualization provides increased flexibility and efficiency by enabling the sharing of resources among independent applications. The computing resources are connected by virtualized network infrastructure that spans across the virtual machines and servers. However, management of the performance of such complex infrastructures is challenging. It is difficult to follow the dynamic changes of the virtual infrastructures without any proper representation or model.

The following questions may arise during operation: How the user-caused workloads affect the load of the data center network? How such operations as e.g., migration of VMs, provisioning and deprovisioning of extra resources affect the performance of data center network? Which part of the data center network is a bottleneck if any? How a network reconfiguration would affect the Service Level Agreements (SLAs) provided to the end-user? To answer such questions it is necessary to predict the network performance during operation and observe the influences of the network to the other parts of the system when the workloads change.

Existing approaches to network performance prediction are mostly based on coarse-grained performance models, that treat the network components as black-boxes additionally abstracting the link to the computing infrastructure and the architecture of the applications running in the data center. On the other hand, there exist many fine grained simulation models for network performance but such models usually capture only selected specific aspects of the network infrastructure whereas other aspects are abstracted. The Descartes Network Infrastructures (DNI) modeling approach [11] addresses the gap between the coarse-grained and simulation models. However, the process of model extraction remains challenging for a DNI model as well as for the other network performance models [20].

Usually, a performance model must represent the structure of the modeled system, its performance-relevant variables, and the workload. Building such performance models that capture different aspects of system behavior is a challenging, error-prone, and time-consuming task when applied manually to real-size systems [8]. The data about the network topology, configuration, and deployment of software may be fragmented between different engineers, obsolete or not documented at all. Fortunately, both, the static or slowly-changing parameters, such as: topology, configuration, and software deployment can be automatically extracted out of a running system assuming that the access to the system is permitted. However, the model of the network workload (network traffic) remains the challenging part mainly due to the high dynamism and large volume of the data transmitted via network in every second.

Techniques for automated model extraction based on observation of the system at run-time are highly desirable given the high costs of manual modeling of the network traffic, high dynamism of network traffic profiles, large volumes of data, and the importance of the traffic model for performance modeling. To address

ICPE'16, March 12-18, 2016, Delft, Netherlands

DOI: http://dx.doi.org/10.1145/2851553.2851570

the challenges of network traffic model extraction, in this paper we contribute:

- a novel **tool** for workload characterization that is based on *Multi-scale time-series decomposition* (MSD),
- a **flexible algorithm** (MSD) for extracting traffic profiles from *tcpdump* traces (practically, from any time series),
- **optimization routine**, so that the level-of-details of the extracted model can be tuned,
- **set of tools** that implement the presented approach and automatically create instances of the DNI model (the traffic part) out of a set of *tcpdump* traces.

The main novelties of our approach are the following:

- we do not extract aggregated statistical models, but a set of deterministic *traffic generator* models that can be understood by non-experts,
- the granularity of the extracted model can be defined manually to fit the level of details required by a given use-case,
- the extracted and optimized traffic model is much compacter than the original trace so it can be simulated faster and even manually corrected by human operator if needed,
- the extraction and optimization process (compacting the extracted model) is automated and does not require human attention,
- the extracted models can be directly used for performance prediction using automatically-generated simulations (using DNI model),
- using the extracted information in the *traffic generator* form can be used for the purpose of traffic patterns analysis, network dimensioning, or debugging.

The rest of this paper is organized as follows. In Section 2, we introduce the traffic generator model that we use as an intermediate step of the extraction. Section 3 describes in detail the MSD (Multi-Scale-Decomposition) algorithm and the optimizations applied to the extracted model. The description is enriched with toy examples and partially evaluated to demonstrate the effects of various fine-tuning parameters. In Section 4, we present the robot telemaintenance case study that we use for evaluation of the proposed approach. Also in Section 4, we describe the evaluation procedure, characterize the traces and present selected representative extraction results along with discussion. Section 5 presents the related work in the field of network traffic models and model extraction. Finally, we conclude the paper and present possible future work directions in Section 6.

2. EXTRACTED TRAFFIC MODEL

In our approach, the extraction of a network traffic model is divided into two coarse steps: (a) extraction and optimization of the traffic generators, and (b) transformation of the traffic generators into an instance of the DNI meta-model. The first step is tailored to simplify the second step, however the output of the first extraction step is not bound to the DNI model and can be used for other models as well. The details about the extraction and optimization using the proposed Multi-scale Decomposition (MSD) algorithm are presented later in Section 3.

2.1 Network Traffic Generator Model

Before we explain the extraction pipeline and signal decomposition, it is necessary to summarize important assumptions about the network traffic. Typically, in a network traffic dump, we can identify multiple communicating parties based on IP addresses, ports or other parameters. For example, inside the communication from a single server we can identify two applications by the TCP port, albeit the IP address is always the same. In this paper, we assume that a pre-processing step takes place to isolate individual sessions that are worth modeling. We assume that the network traffic is represented as a univariate time-series with one-second time granularity and positive values (packets with negative size do not make sense).

In reality, we may observe regular and periodic data transfers but also irregular and highly asymmetric signals depending on the application. Therefore, our decomposition algorithm has to handle irregular signals well and to take advantage of regularities whenever possible.

The last assumption in our approach concerns the DNI model and the simulation models that can be automatically generated from a DNI model. Network traffic in simulators generated by DNI (e.g., SimQPN[14] or OMNeT++) has to be modeled as a collection of traffic generators emitting packets with a certain frequency. The information about packets emitted is encoded into DNI *flows* that are later transformed into entities used in the respective simulation (e.g., packets in OMNeT++ or tokens in SimQPN). There is not much room for implementing arbitrary real-value functions, which also hinders the application of Fourier and Wavelet transforms as explained further in Section 3.1.

2.1.1 Model of a simple traffic generator

In this paper, we assume the model of a simple traffic generator as depicted in Figure 1.

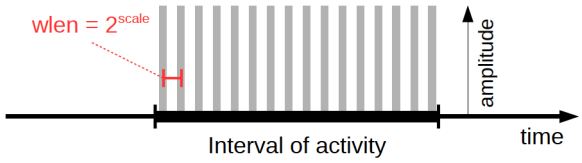


Figure 1: Model of a simple traffic generator

A simple traffic generator is a tuple:

$$Generator = (scale, amplitude, begin, end)$$

where:

- **scale**: How frequently the generator emits packets; every $wlen = 2^{scale}$ seconds.
- **amplitude**: Size of the packet to be emitted.
- **begin**: Beginning of the interval when the generator is active.
- **end**: End of the activity interval.

The extracted set of generators is saved in a text format for further processing. We parse the generator descriptions and build a DNI model but the data can be used also for other models. Figure 2 demonstrates the idea behind the decomposition into activity of simple traffic generators and the corresponding fragment of the DNI model.

2.1.2 Examples of Network Traffic

Figure 3 shows four examples of a recorded network traffic. The first time-series represents data transfer during 60 minutes of video streaming with occasional caching. The second example represents an irregular data transfer with a heat-up phase containing several small data transfers, followed by a continuous transfer reaching the link capacity. The third example represents a signal with several outliers — an example of multiple applications transferring data in parallel. The fourth time-series represents a regular signal.

2.1.3 Clustering of the traffic

We observe that the traffic rates can be efficiently clustered because an application usually transfers data in packets of certain

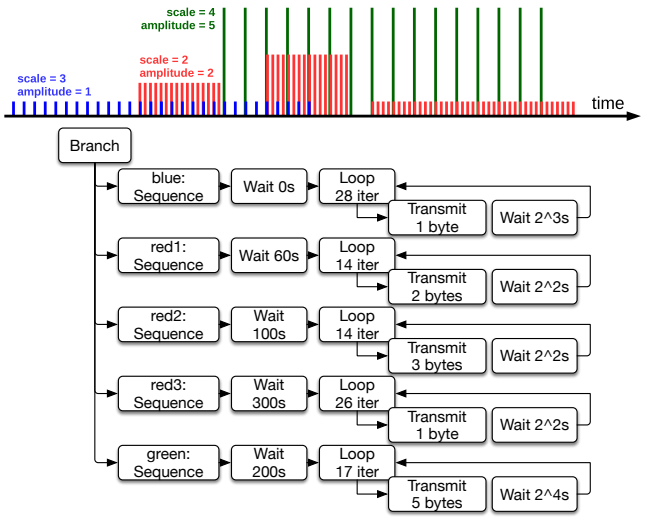


Figure 2: A toy example: decomposition into traffic generators at different scales and amplitudes and the corresponding instance of a DNI traffic model.

sizes. This can be seen in Figure 4. We plotted the kernel density plot for each of our example traffic dumps showing the amount of different transfer sizes. In order to compare all the densities, we applied normalization and plotted them together in a single diagram. Each local maximum in the diagram represents a cluster candidate. From Figure 4 we observed that a good estimate for our clustering is between 3 and 7 clusters.[1] Similar behavior can also be observed in other network traces, such as those made available publicly by the LBNL/ICSI Enterprise Tracing Project.[2]

2.2 Traffic Model in DNI

The generators extracted from the traffic traces are meant to be represented in DNI model. Here, we revise how the DNI meta-model represents the network traffic. More information about the DNI meta-model can be found in [11, 12, 10].

In the DNI meta-model, network traffic is generated by traffic sources originating from software components that are deployed on so-called end nodes (e.g., servers, VMs). Each traffic source generates traffic flows that have exactly one source and possibly multiple destinations. The flow destinations are software components that are located in nodes and can be uniquely identified by a set of protocol-level addresses. Flows can be composed in a workload model that defines how each flow is generated (e.g., with sequences, loops, or branches). For the purpose of this paper, we describe a flow by specifying the amount of transferred data (so called *GenericFlow*). Graphically, the traffic part of the DNI meta-model is depicted in Figure 5.

Besides the network traffic information, the DNI model represents also the network topology and its configuration. We say, that the extracted model is partial because we focus on traffic model extraction. In fact, we derive simplified models of network topology

[1]We also provide an mechanism for estimating the optimal number of clusters automatically, tailored for a given signal.

[2]Packet header traces of LBNL's internal enterprise traffic: ftp://ftp.bro-ids.org/enterprise-traces/hdr-traces05/, last accessed January 8, 2016

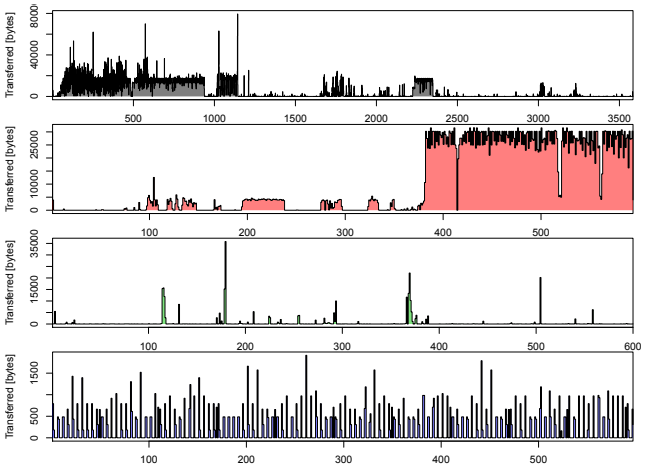


Figure 3: Four examples of typical traffic dumps (60, 10, 10, 10 minutes). X-axis represents time in seconds, Y-axis represents bytes transferred per second.

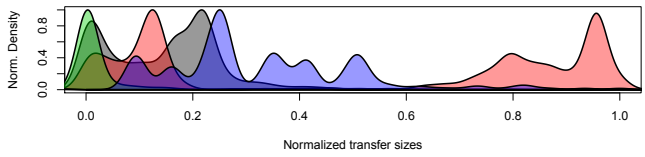


Figure 4: Normalized kernel density plots of transferred bytes from the example traffic dumps.

and the deployment of software on nodes, but this information is incomplete as its extraction is not the goal of this paper (e.g., in the extracted model we assume that all nodes are connected with a star topology which is not always the case in general).

3. APPROACH

In Figure 6, we present the high-level overview of the approach, i.e. the extraction pipeline. The process starts by recording real traffic on multiple interfaces within the network (e.g., using *tcpdump*). A single traffic dump (here depicted as *tcpdump.log*) is treated as a univariate irregular time-series and goes into the *Multi-scale decomposition* step (explained in detail in Section 3.1), which in turn produces the *Decomposition Matrix*.

Each row of the matrix is a separate regularized time-series. First column represent the original signal (for debugging purposes), while the other columns represent packets emitted by simple traffic generators operating at different frequencies and amplitudes. The Decomposition Matrix is then further transformed into the *Configuration for generators*. This is a sequence of tuples in the form of $(scale, amplitude, begin, end)$ as already explained above. Finally, the DNI model is produced which reflects the decomposition.

3.1 Multi-scale time-series decomposition

Our *Multi-scale time-series decomposition* (MSD) is loosly inspired by Wavelets, in particular by Discrete Wavelet Transform (DWT) [16]. In DWT, the signal is processed at multiple scales: $Scales = (1, \ldots, maxscale)$ where: $maxscale = log_2|signal|$ The DWT software packages, such as `wavethresh` in R, expect the input signal length to be a power of two. For each scale $s \in Scales$, the signal has got half the size of the signal at the pre-

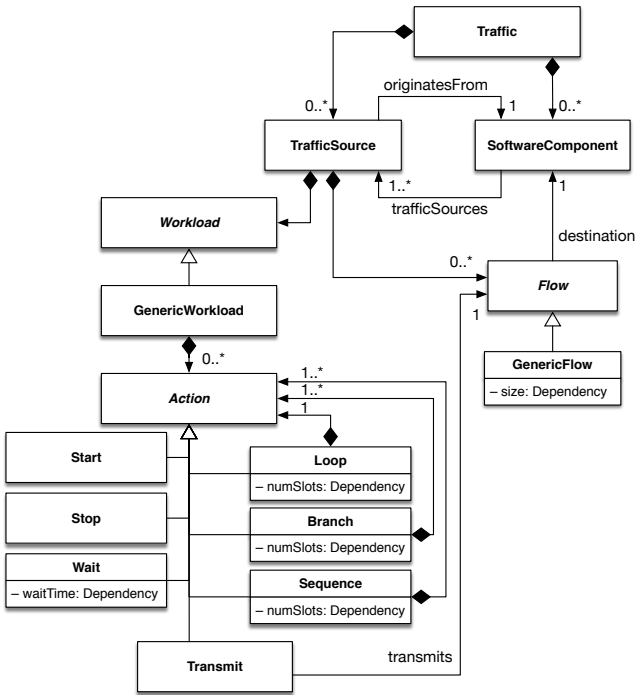


Figure 5: DNI Meta-model of network traffic

vious scale $(s-1)$, i.e.: $|signal_s| = |signal_{s-1}|/2$. Scale 0 represents the original signal, i.e. $signal = signal_0$. Roughly speaking, at each scale $s \in Scales$, DWT computes the interference between the $signal_s$ and a particular wave (stretched and shifted *mother wavelet*). This yields a vector of coefficients $Coef_s = (c_1^s, \ldots, c_{|signal_s|}^s)$. All scales together form a pyramid which can be represented as a single linear vector:

$$Coef = \bigcup_{s \in Scales} Coef_s$$

The DWT coefficients represent the original signal in a new space of functions. It should be noted that the time complexity of DWT is $O(|signal|)$, the length of the original signal and the length of $Coef$ vector are the same:

$$|Coef| = |signal|$$

In a similar way, the original signal can be reconstructed by a process which uses DWT coefficients $Coef$ and stretched/shifted wavelets in an inverse manner.

The applications of DWT usually involve manipulation of DWT coefficients, such as removing all coefficients from a particular scale $Coef_s$ (e.g. for compression). However, there are many other application areas of science and engineering – noise reduction, edge detection, time/frequency analysis, ...

Unfortunately, for the purpose of traffic modeling with DNI[3], we realized that DWT is not particularly useful. Although it makes frequency analysis at different bands possible, DWT coefficients cannot be directly mapped to simple traffic generators. Remember that a DWT coefficient represents the *strength* of interference between the signal and a wavelet at a particular time (shift) and frequency band (scale). A single coefficient can well be a negative

[3]Currently, DNI supports automated transformation to OMNeT++ simulation and Queueing Petri nets (using SimQPN simulator [14])

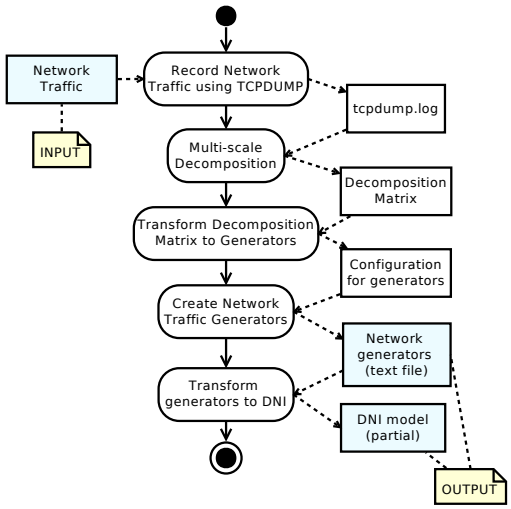


Figure 6: The overview of the extraction pipeline. (Rectangles represent data, ovals represent actions)

number. This makes sense for inverse DWT, when multiple functions are summed together forming the original signal, but has little use when modeling a simple traffic generator that can only produce *positive* traffic. Even after several attempts, we were not able to get convincing results with DWT, thus we opted for a custom DWT-inspired transformation instead — the aforementioned MSD.

Similarly to DWT, the notion of scales is preserved in MSD, however, the scales are processed in reversed order: $maxscale, \ldots, 1$. Instead of a pyramid of DWT coefficients, MSD transforms the signal into a matrix of *Emits*. Each scale s corresponds to a vector $Emits_s$ of equal length:

$$|Emits_1| = \ldots = |Emits_{maxscale}| = |signal|$$

Before the actual MSD loop, the input time-series needs to be preprocessed as follows:

1. Regularization, i.e. every second should be represented by a single value. Gaps in the signal are replaced by zeroes, while multiple values are aggregated using the *sum* function. (Figure 7: *Read Input*)

2. The signal length is padded with zeroes to the nearest power-of-two (e.g. signal with the size of 4000 samples is padded to 4096 samples)

3. Values in the signal are clustered. As already explained, we can efficiently apply clustering of values using k-means. We can either manually set the number of clusters (e.g. 6 clusters) or we can use an automated estimation approach. (Figure 7: *Estimate Optimal Number of Clusters*) We compute total within-cluster sum of squares for up to 25 clusters. Then we pick the lowest number such that a higher number would only differ by less than a preselected cutoff value (10% by default). After k-means clustering of values, a new signal is generated where original values are replaced by the cluster centers.

Such a preprocessed signal then goes into the MSD loop, where it is passed through a sieve which subtracts certain sub-signal in each iteration until only "noise" remains. The subtracted sub-signal (a vector of Emits) is such a traffic that can be generated by a simple

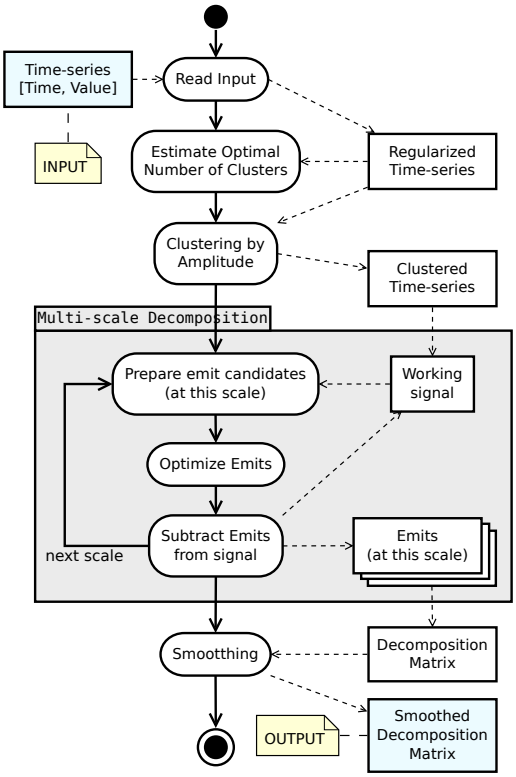


Figure 7: Multi-scale decomposition algorithm. (Rectangles represent data, ovals represent actions)

traffic generator with a frequency corresponding to the particular scale.

An example decomposition of a short signal is depicted in Figure 8 (without any clustering nor optimization). Signal in this example consists of 32 samples which is then processed at 5 scales. To derive the Emits at each scale, we have to:

1. Split the remaining signal into equal intervals.

2. Extract a single emit candidate from each interval. (Figure 7: *Prepare emit candidates*)

3. Optimize the vector of emit candidates. (Figure 7: *Optimize Emits*)

Ad. 1, at a given scale s, the working signal is split into intervals $(I_1, \ldots, I_{wtimes})$ of equal length $wlen$.

$$\begin{aligned} wlen &= 2^{scale} \\ wtimes &= |signal|/wlen \end{aligned}$$

Ad. 2, for each interval I_i, a single emit candidate is selected that has the highest value such that it appears only once inside the interval.

$$Emits = \bigcup_{i \in 1}^{wtimes} select(I_i)$$

If no such a candidate exists, the *select* function returns 0 as a candidate.

Ad. 3, the $Emits$ vector is then optimized as follows:

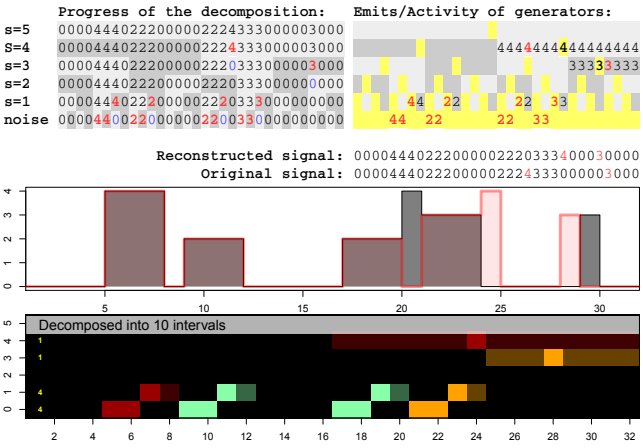


Figure 8: Example decomposition explained step-by-step.

1. Small sequences of equivalent consecutive emits are removed. The reason for this is to prevent isolated peaks to become emits at higher scales. Example:

$$00\mathbf{11}00\mathbf{2}01110 \rightarrow 00\mathbf{00}000\mathbf{0}01110$$

2. Small gaps are removed in order to achieve higher compression. Example:

$$000111\mathbf{2}11100 \rightarrow 000\mathbf{1111111}00$$

Time complexity of MSD is $O(n.log(n))$, $n = |signal|$. After Emits from all scales were collected, vector of Emits at each scale can be further smoothed (This is an optional step, Figure 7: *Smoothing*).

3.2 Decomposition of a real signal

We have demonstrated in Figure 8 the result of decomposition on a toy signal consisting of 32 samples. Let us now take a look at decomposition of a real network traffic (10 minutes sampled at 1 second, total 600 samples). This signal has already been presented in Figure 3 (second diagram from the top). Without any optimization nor clustering, MSD decomposes the signal into 253 intervals as depicted in Figure 9.

The top diagram shows the original and reconstructed signal combined. To provide more insight into the decomposition algorithm, we use three visual tools for comparing signals from the frequency-content point of view:

1. *Periodogram* [3] compares spectra of the original signal(solid line) and reconstructed signal (dashed line) as given by the Fourier Transform. From left to right are shown densities of all frequencies (from lower to higher) in the signals' spectra.

2. *Squared Coherency* [3] diagram estimates the percentage of variance in the original signal that is predictable from the reconstructed signal at the same frequency band. The higher the value, the more similar the reconstructed signal to the original is.

3. *Scaleograms* [17] represent signals in a time/frequency domain as given by the continuous wavelet transform using the *Paul* mother-wavelet.

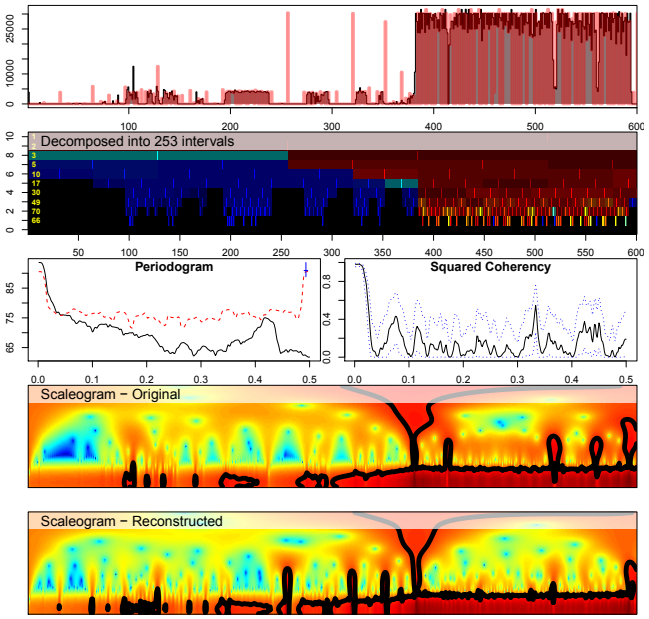


Figure 9: Decomposition example: 10 minutes of traffic without optimization.

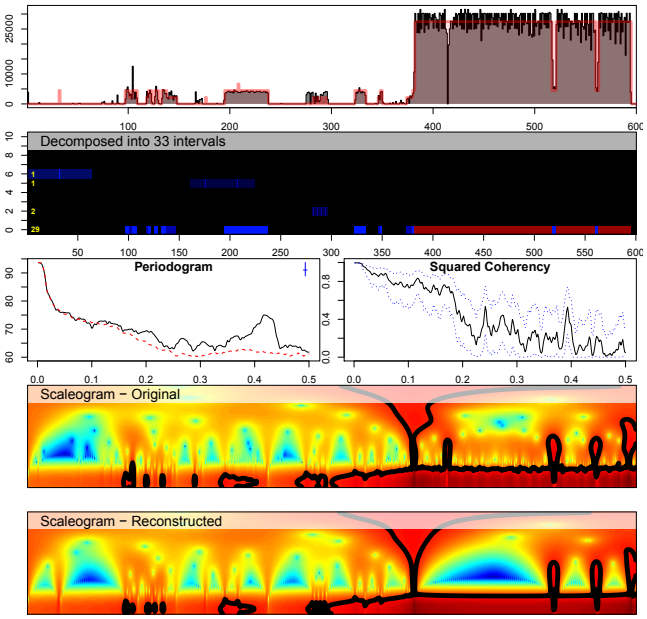


Figure 10: Decomposition example: 10 minutes of traffic with moderate optimization.

When we reconstruct the signal, we can see that it matches its original with an exception of few isolated peaks and some additional high-frequency content. Frequency, however, is not the main criterion for evaluating the quality of the decomposition. More important is the overall shape of the signal that plays the major role later in the simulation phase.

With a moderate optimization, as depicted in Figure 10, we were able to reduce the number of intervals to 33 (i.e., 13%). At the same time, the reconstructed signal still preserves the shape of the original. The tradeoff between the reconstruction accuracy and the model size can be fine-tuned using the following parameters: (1) Number of clusters, (2) reduction of gaps and isolated peaks in emit vectors, and (3) final smoothing of emit vectors.

In Section 4, we set the parameters of the extraction process and further evaluate the quality of extraction on different traffic traces using a real-world case study.

4. EVALUATION

In Section 3, we evaluated the extraction and optimization method using selected traces from online repositories to demonstrate the features of the approach. In this Section, we evaluate the proposed approach using traces from the robot telemaintenance scenario. The traces include mainly two types of network traffic: small control instructions or sensor readings and large flows of video streams from cameras that observe the work of a robot. The traces have been captured in a data center where the devices were controlled from. We have recorded the *tcpdump* trace from the server that acting as a switch (see Fig/ 11b). It was equipped with a four port network card. For the need on the experiment the server was replacing a standard gigabit data center switch.

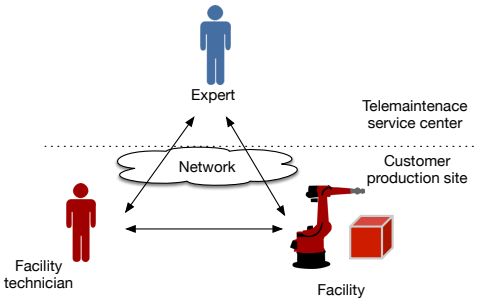
4.1 Robot Telemaintenance Scenario

The Industry 4.0 initiated by German government[4] comprises among other things the introduction of the Internet to the manufacturing process. In the research project *MainTelRob*[2] different industry 4.0 approaches have been tested for telemaintenance of a plant in a working production line [4]. Figure 11 depicts the project setting. The costumer production site of Braun, which belongs to Procter & Gamble, consists of a six-axis Cartesian industrial robot by KUKA Industries, a two-component injection molding system and an assembly unit. The plant produces plastic parts for electric toothbrushes. On the upper half there is a telemaintenance center from which an engineer—the *expert*—who provides technical expertise to the local repair personnel (the facility technician). Next to the plant, the facility contains telemaintenance equipment: a computer, multiple cameras for video streaming and a mobile device. Center and facility are connected over the Internet. The main prerequisite is to provide the expert with a good view of the situation on-site. This insight can be offered by a specifically orchestrated combination of services: Remote access to machinery data in combination with video streaming and communication services, e.g., text chat and Voice-over-IP (VoIP). In addition, visual Augmented Reality (AR) overlays inserted into the camera pictures or video view are used to provide guidance. As the targeted environment includes the service technician repairing the machinery on-site, the industrial telemaintenance system should additionally provide modern means of communication.

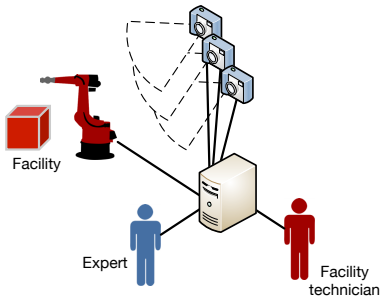
4.2 Traffic Modeling in the Telemaintenance

For the presented scenario, the proposed extraction method delivered insight in the patterns of the data exchange between the components of the system. Although the main goal of the proposed method is to use the extracted models for performance prediction, here, the robot operator can observe the patterns of traffic. The analysis of the data exchange patterns may be used (additionally to the performance prediction) for the following purposes: (1) analyzing compact representation of control signals for debugging purposes (instead of, e.g., analyzing textual *wireshark* traces), (2) understanding the data exchange for network dimensioning.

[4]Similarly *Industrial Internet* was framed in the US



(a) Scenario overview



(b) Network topology with central server, robot, surveillance cameras, and maintenance staff.

Figure 11: Robot telemaintenance scenario. The remote expert and the local facility technician use mobile devices to observe the production facility.

4.3 Assessing Model Extraction Errors

We evaluate the model extraction accuracy by computing the relative errors for each pair of time intervals in the extracted model instance and the original trace. We stress, that the maximal model accuracy and the size of the extracted model are the trade-offs and the model should not tend towards one of these extremes. During evaluation of the size of the extracted models, we assume that the original trace represents as many *transmit* actions as many lines the original *tcpdump* file contains.

We selected 69 traffic traces from the robot telemaintenance scenario. Average trace size was similar among the four types and amounts about 22000 data samples. The traces were captured within a 15-minute measurement period. For each trace we picked arbitrary parameters of the extraction algorithm (smoothing 0, maximal number of clusters 25, cut-off for clustering 0.1, and intervals reduction parameter to 0.0004). We repeated the analysis of the 69 files 30 times and collected the following metrics: (1) median of relative error; the error was calculated between the original and extracted trace for every second (the aggregation function over one-second bins was sum) (2) mean relative error, (3) relative error of the total data transmitted in a trace (4) compression of the modeled signal (the modeled signal requires only x% of generators of the original signal). For the four metrics, we calculated the 5^{th}, 50^{th}, and 95^{th} percentiles and presented the results in Table 1.

We divided the traces into four types based on the values of the obtained metrics. To the first type we account the traces with the three relative errors lower than 10%. There are 43 traces of this type what shows that the proposed method can extract models of the most traces with good accuracy and good compression—the extracted models are 40–200 times smaller than original (compressed to 0.4%–2.3% of the original size).

The traces assigned to the second type are characterized with good relative median error and relative total data error but higher values of the relative mean error. We investigated the discrepancies among the traces and the extracted models. The higher mean error rates are caused mainly by shift in a extracted signal—a workload peak shifted in time doubles the error: first because a real peak is not discovered and second because an artificial peak is produced whereas no peak exists in a real signal. An example of the described situation is depicted in Figure 12a). Few peaks of type 2 were also influenced by non-ideal set of extraction parameters of extracted model (see traces categorized as type 3).

The traces of the third type are described by low errors of relative total data error but higher relative median and mean. We name this type "extraction parameters" as the parameters of the extraction and optimization process were not optimal for the traces (reminder: for the whole scenario we select parameter values arbitrarily). The higher errors are caused mainly by two factors: time-shifted peaks and outliers in the extracted signal caused by lower fit of the extracted model. We selected one trace affected by lower quality extraction and depicted it in Figure 12b). We observe, that the extraction was generally correct, but there are several periods where the data is not generated although the original trace behaves differently. This low extraction accuracy is caused by the fixed set of parameters selected for the method. One could optimize the accuracy by fine tuning the algorithm parameters or use less optimization for the compression. Although the median and mean errors vary from 13% to 30% the total amount of transmitted data is accurate and the compression ratio is good 7–125. Comparing the traces of type 2 against type 3, we observed that the ratio of time-shifted peaks to lower quality of extraction is higher in type 2; in type 3 the situation is opposite: there are more cases of lower quality extraction.

The fourth traces of type four do not fit to any other type. In this experiment the relative mean error and total data error are above the arbitrarily selected 10% threshold. The relative median error is low and the compression ratio vary from 7 to 12. There were only two traces of type 4 in the dataset. We depicted a selected trace of type 4 in Figure 12c). The errors are mainly caused by outliers introduced by suboptimal selection of extraction method parameters. We observe that the main shape of data trace was extracted correctly.

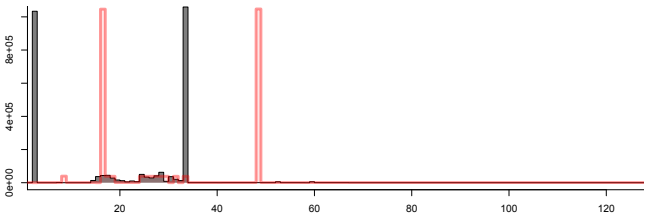
5. RELATED WORK

As noticed by Adas, "Traffic models are at the heart of any performance evaluation of telecommunications networks" [1]. On the other hand, the authors of [7] claim that "there is not much work on measurement, analysis, and characterization of datacenter traffic" suggesting that more focus should be put to modern data centers and the intra-data-center traffic characterization.

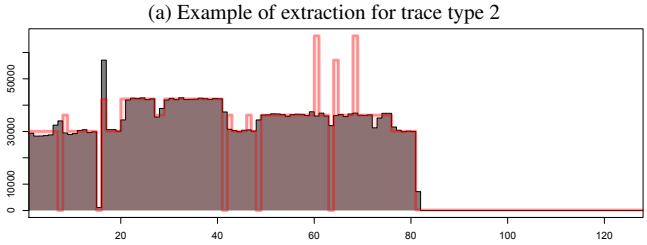
There exist many related work on modeling general network traffic [5, 9, 6, 13]. However, most of the works focus on probabilistic models that were meant to approximate the characteristics of network traffic when aggregated or to preserve self-similar nature of the traffic. Goals of this work are different. We want to represent the traffic deterministically to analyze the traffic exactly from the time it was recorded and not to generalize the model to larger time scales. Due to that, we: (1) decompose the traffic profile into a set of generators (on-off traffic sources) with defined start and end of their activity, (2) flexibly compress the model of the network traffic at the same time being able to control the loss of the charac-

Table 1: Results for the 69 analyzed traces divided into 4 groups

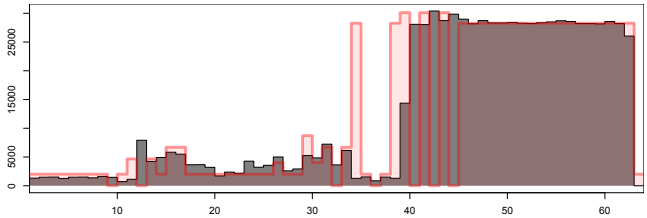
Type	Traces	Relative median error %			Relative mean error %			Relative total data error %			Compressed to %		
		percentile			percentile			percentile			percentile		
		.05	.50	.95	.05	.50	.95	.05	.50	.95	.05	.50	.95
1	43	0	0	2.9	0	0.1	8.4	−1.2	0	1.7	0.4	0.8	2.3
2 (shifted peaks)	13	0	0	7.5	12.6	16.4	36.7	−4.4	1.5	6.6	1.6	7.8	15
3 (extraction parameters)	11	13.5	21.8	32.4	13	17.1	23	0	0.3	2.6	0.8	1.6	13.3
4 (rest)	2	0.8	1.2	1.7	24.1	26.1	28.2	10.9	11.7	12.5	8.1	10.9	13.7



(a) Example of extraction for trace type 2



(b) Example of extraction for trace type 3



(c) Example of extraction for trace type 4

Figure 12: Representative traces of type, 2, 3, and 4 (in grey) and their extracted models (red).

teristics of the original trace (some simulators cannot accept large detailed inputs, for example: SimQPN [14]), (3) support any traffic aggregation interval, whereas the trace driven simulations use usually packet as a smallest unit of traffic and due to that produce fine-grained models with predefined, constant granularity.

The authors of [19] propose a similar approach to ours. They propose a tool that extract workload profiles (not network traces but rather service requests) and decompose them into patterns. The decomposed traces are stored in an ecore-based models [15] and are used mainly for workload forecasting and replaying modified traces in benchmark environments (e.g., replaying an original trace but with amplified burstiness). Although the approach is similar, the details clearly separate the work from ours. First, the workload model that LIMBO extracts is different to DNI Traffic model so the extraction procedure needs to be different. We extract sets of traffic generators whereas LIMBO looks for patterns like, for example: seasonal, trend, burst. Second, we focus on network traffic models extraction by considering time series of data transferred; LIMBO defines workload at the level of requests that can be mapped to various data sizes. And finally, LIMBO depends strongly on seasonality of the workload as the first step of their extraction procedure searches for data seasonal patterns (e.g., sine-shape). Our approach also supports seasonal patterns (see network traffic generator model in Section 2.1), however any other traffic characteristic can be modeled as well using the network traffic generator representation.

Regarding the approaches to model extraction, the most of the approaches calculate traffic statistics from the traces and represent the traffic statistically, for example using packet size distributions and packet inter-arrival times. As already discussed, such approaches cannot be applied in our case, as we aim to discover relatively compact set of traffic generators to represent the trace. Other works, for example [18], do model the structural information about the traffic, but this is usually represented as users, sessions, connections and packets causing the approach to be application-specific. Additionally, in [18], there is no intention for flexibility of representation of the traffic so that the trade-off between model size and accuracy of representation cannot be selected. In our approach, we allow the user to tune the parameters of the MSD algorithm to provide different level of details in the representation.

6. CONCLUSION AND FUTURE WORK

The nature of the data center network traffic is still not well known due to large variety of virtualized applications running in modern environments that can be provisioned and deprovisioned with resources on the fly. In this paper we addressed the problem of microscopic deterministic analysis of the network traffic traces for performance modeling purposes. We provided a flexible algorithm (MSD — Multi-Scale-Decomposition) for extracting traffic profiles from any time series (e.g., *tcpdump* traces). Moreover, we showed that the extracted traffic models, represented as set of traffic generators, can be optimized to reduce the size of the model but sill accurately model the characteristics of the original trace. We showed that the model with reduced size can be as small as about 0.5% of the original while still accurately represent the original traffic characteristics (relative errors can be as low as 0.1%). Furthermore, the extraction and the optimization procedure can be fine-tuned to extract even long and complicated traces—without the fine tuning, we may observe relative extraction errors up to 30% in the worst case. The parametrized extraction causes that the procedure is flexible with respect to the demanded level of details in the extracted models. The models can be larger but more detailed or more compressed but coarser. Finally, we implemented the method as a set of tools that can be freely used for extraction of the traffic generator models as well as the DNI meta-model instances. We stress, that the extracted model in form of the *traffic generator* model is not explicitly designed for DNI, but may be also used *as-it-is* for traffic pattern analysis, debugging, or network dimensioning purposes. The auxiliary materials (including the MSD decomposition) are available online under http://go.uni-wuerzburg.de/aux.

As a part of our future work, we see several possible directions.

First is the improvement of the automatic selection of extraction parameters (described in Section 3.2) so that more traces can be analyzed with lower extraction errors automatically. Second, we think that the approaches to automatic model extraction should be compared with the manual-built models crafted by humans. In such evaluation the human caused errors could be compared with the imperfections of the MSD extraction algorithm. Such evaluations are challenging and time consuming (mainly due to the involvement of the human factor), however the results would be interesting. Finally, the proposed approach was tailored (but not limited) to extract traffic models in a DNI-friendly format. The scope could be extended and also the extraction of other traffic models could be added.

7. REFERENCES

[1] A. Adas. Traffic Models in Broadband Networks. *Communications Magazine, IEEE*, 35(7):82–89, Jul 1997.

[2] D. Aschenbrenner, F. Sittner, M. Fritscher, M. Krauss, and K. Schilling. Teleoperation of an Industrial Robot in an Active Production Line. In *Proceedings of 2nd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT)*, 2015.

[3] P. Bloomfield. *Fourier Analysis of Time Series: An Introduction*. Wiley, 1976.

[4] S. Chowdhury and A. Akram. E-Maintenance: Opportunities and Challenges. In *Proceedings of the 34th Information Systems Research Seminar in Scandinavia (IRIS)*, pages 68–81, 2011.

[5] V. Frost and B. Melamed. Traffic Modeling for Telecommunications Networks. *Communications Magazine, IEEE*, 32(3):70–81, March 1994.

[6] A. Grzech and P. Świątek. Parallel Processing of Connection Streams in Nodes of Packet-switched Computer Communication Systems. *Cybernetics and Systems*, 39(2):155–170, 2008.

[7] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: Measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[8] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, July 2006.

[9] L. Qian, A. Krishnamurthy, Y. Wang, Y. Tang, P. Dauchy, and A. Conte. A new traffic model and statistical admission control algorithm for providing qos guarantees to on-line traffic. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 3, pages 1401–1405 Vol.3, Nov 2004.

[10] P. Rygielski and S. Kounev. Descartes Network Infrastructures (DNI) Manual: Meta-models, Transformations, Examples. Technical Report v.0.3, Chair of Software Engineering, University of Würzburg, Sep. 2014.

[11] P. Rygielski, S. Kounev, and P. Tran-Gia. Flexible Performance Prediction of Data Center Networks using Automatically Generated Simulation Models. In *Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques (SIMUTools 2015)*, August 2015.

[12] P. Rygielski, S. Kounev, and S. Zschaler. Model-Based Throughput Prediction in Data Center Networks. In *Proceedings of the 2nd IEEE International Workshop on Measurements and Networking (M&N 2013)*, pages 167–172, October 2013.

[13] M. Z. Shafiq, L. Ji, A. X. Liu, and J. Wang. Characterizing and Modeling Internet Traffic Dynamics of Cellular Devices. *SIGMETRICS Perform. Eval. Rev.*, 39(1):265–276, 2011.

[14] S. Spinner, S. Kounev, and P. Meier. Stochastic Modeling and Analysis using QPME: Queueing Petri Net Modeling Environment v2.0. In S. Haddad and L. Pomello, editors, *Proceedings of the 33rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2012)*, volume 7347 of *Lecture Notes in Computer Science (LNCS)*, pages 388–397, June 2012. Springer.

[15] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[16] M. Stéphane, editor. *A Wavelet Tour of Signal Processing (Third Edition)*. Academic Press, Boston, 2009.

[17] C. Torrence and G. P. Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79:61–78, 1998.

[18] K. V. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 111–122, New York, NY, USA, 2006. ACM.

[19] J. von Kistowski, N. R. Herbst, D. Zoller, S. Kounev, and A. Hotho. Modeling and Extracting Load Intensity Profiles. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, May 2015.

[20] F. Willnecker, M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, and H. Krcmar. Comparing the Accuracy of Resource Demand Measurement and Estimation Techniques. In *Computer Performance Engineering — Proceedings of the 12th European Workshop (EPEW 2015)*, volume 9272 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2015.