Verifying Network Performance of Cyber-Physical Systems with Multiple Runtime Configurations

Martin Manderscheid, Gereon Weiss and Rudi Knorr
Fraunhofer Institute for Embedded Systems and Communication Technologies ESK
Munich, Germany
{martin.manderscheid, gereon.weiss, rudi.knorr}@esk.fraunhofer.de

ABSTRACT

Modern Cyber-Physical Systems (CPS) must increasingly adapt to changing contexts, like smart cars to changing driving conditions. Thus, design approaches are facing a rapidly growing number of network runtime configurations. With recent approaches this problem can be solved for design space exploration (DSE) by analyzing the network performance of single configurations which are intended to represent the entire runtime variability space. This technique can be applied for DSE since the latter only intends to find an optimized system setup. Yet it does not meet the requirements of network verification, since it does not necessarily find the worst-case for all applications. To solve this, we developed an integrated model, which allows describing runtime variability in the network performance model with a 0-1 linear-fractional program. Thus, we can cover entire runtime variability spaces without analyzing every single network runtime configuration. Although the approach utilizes heuristics, it still guarantees worst-case results. We can show that in comparison to state-of-the-art methods our approach scales for large automotive systems with multiple network configurations. Moreover, our evaluation results highlight the superior capabilities of our method with respect to accuracy and computation time.

1. INTRODUCTION

Several domains, like automotive, railway, or avionic face great challenge in designing and evaluating Cyber-Physical Systems (CPS). Such systems consist of many computing units which are connected through various communication technologies (e.g., CAN, MOST, Flexray, and Ethernet [1]). Since these systems implement applications with real-time constraints, the network architecture must be verified within the design process. This verification process usually consists of a performance evaluation for the designed network architecture carried out late in the design process. The network architecture is only valid if the results of the performance evaluation fulfill the non-functional requirements of the applications.

As the implemented functionality of CPS steadily increases and those systems are more and more dynamic [2], their performance evaluation faces ever new challenges. These dynamic systems may have numerous runtime configurations [3] (also called system scenarios [2]), as the number of configurations grows exponentially with the number of implemented software components. The set of valid runtime configurations is described by the runtime variability space[3]. Since performance evaluation methods must ensure that their predictions cover the entire runtime variability space, scalability becomes an intricated task.

In current research, no approach exists that sufficiently solves this scalability problem while guaranteeing worst-case results. DSE-approaches[4] , focusing on the reduction of the vast number of runtime configurations do not necessarily find the worst-case. Other verification-approaches [5] guarantee worst-case results, but they do not consider scalability sufficiently.

In the work at hand, we tackle the problem of covering the entire runtime variability space while simultaneously guaranteeing worst-case performance predictions. We approach this problem with an integrated model, which describes the runtime variability in the performance model with a 0-1 linear-fractional program. This can then be solved after linearization-transformations, by standard mixed-integer linear programming (MILP) solvers.

The main contributions of this paper are:

- An integrated approach capable of predicting worst-case network aggregation end-to-end delays which integrates runtime variability. This approach consists of:
  - an integrated model which represents the runtime variability of the system in the performance model with a 0-1 linear-fractional program. After relaxations and a reformulation-linearization [6] the model is solvable by standard MILP-solvers such as lp_solve [7].
  - an algorithm making use of this model to calculate the network aggregation end-to-end delays of large systems
- We validated the scalability of the approach by numerous experiments with varying network topology examples of automobile in-vehicle networks.

The remainder of this paper is organized as follows. In the next section, we review related work. In Section 3, we present our novel integrated model along with its performance prediction algorithm. Section 4 discusses evaluation results using the example of modern automobiles. In the last section, we conclude and give an outlook on our future work.

2. RELATED WORK

One crucial step during the verification of network architectures for CPS is the performance evaluation. The objective of this process is the prediction of the system's network performance. Generally, there are three methods to predict the performance: analysis, simulation, and measurement. Typically, simulation[8, 9] and measurement are used to get a picture of the average-case performance. However, for verification purposes worst-case results are needed. These can in general be obtained by analytical methods. One drawback of analytical methods is, that they often lead to over-approximated performance predictions. Therefore, a lot of effort has been spent in last years to develop tight performance models (in particular for Ethernet, cf. [10, 11, 12, 13, 14, 15, 16]). Since those performance models do not capture the runtime behavior or the system application's dynamics, there has been considerable work to describe and analyze the dynamic behavior of CPS.

Gheorghita et al. [2] published a system design approach for dynamic real-time embedded systems. They propose to group system behaviors to system scenarios with respect to similar non-functional properties (e.g., delay or energy consumption). The system scenarios can then be optimized in the design process by exploiting these similarities. Furthermore, the approach provides a methodology to predict and switch between system scenarios during runtime. Since the approach requires to manually define use-case scenarios, it is restricted to small sets of scenarios and thus not suitable to capture variability of multiple runtime configurations.

Due to the fact that the manual definition of multiple runtime configurations is very complex and error-prone, there has been some work on the automatic generation of runtime configurations from dynamic models, such as *Disciplined Dataflow Networks* and *Runtime Feature Models*. In [5], Siyoum et al. present an automated scenario-based approach for analyzing properties of embedded streaming applications. They describe a process which enables the extraction of all possible scenarios of a *Disciplined Dataflow Network*. They present examples of a RVC-MPEG video decoder and a WLAN 802.11a baseband processing implementation. By these, they show that a scenario-based design-time analysis can save resources significantly. However, this does not provide a solution for the exponentially growing number of scenarios. In prior work [3], we presented an approach which is capable of making network performance predictions for dynamic embedded systems at an early design stage. This approach describes the runtime variability of a system in an abstract model. From this model runtime configurations are obtained which are in turn transformed into system configurations. The performance of the system configurations can be analyzed by tools such as network calculus [17]. Since this approach requires analysis of every runtime configuration to guarantee worst-case performance results, it is limited to systems with moderate large runtime variability spaces.

Although latter approaches are able to guarantee worst-case results, they still do not scale for large CPS, since the number of runtime configurations grows exponentially. This issue is addressed by Van Stralen et al. in [4]. They introduce a design space exploration (DSE) approach that addresses the problem of exploding number of runtime configurations. In their approach, they endeavor to optimize task-bindings for MultiProcessor System-on-Chip-based embedded systems with numerous configurations. During the optimization process, they need to evaluate the performance of every candidate. Since a candidate consists of a large number of runtime configurations, they propose a feature selection algorithm that selects only a small set of configurations. This set is intended to be representative for the entire runtime variability space. Since the approach does not necessarily find the worst-case for all applications, it cannot guarantee worst-case performance results.

In summary, it can be said that no current approach solves sufficiently the problem of predicting worst-case performance results for dynamic CPS with multiple runtime configurations. On the one hand, approaches that are capable of guaranteeing worst-case results do not scale for such systems with large runtime variability spaces. On the other hand, approaches that solve the scalability problem cannot guarantee worst-case results. The objective of this work, is to create an approach fulfilling both properties: scalability for systems with large runtime variability spaces and guarantee of worst-case results.

3. INTEGRATED APPROACH

In this section, we present our integrated approach, which aims at tackling the challenge of determining the worst-case network performance of dynamic CPS with multiple runtime configurations. It consists of: i) an integrated network performance model which enables designers to calculate network aggregation end-to-end delays for an entire runtime configuration space, and ii) an algorithm which makes use of the integrated model to calculate the accumulated aggregation delay on the path of a data dependency. As can be obtained from Figure 1, the integrated approach starts with modeling of the runtime behavior of the system through a *Runtime Feature Model (RFM)*. After transforming this model into a *Component/Hardware Network Model (CHNM)* the *Integrated Network Performance Model (INPM)* is generated. With this step the runtime variability of the RFM within the INPM is modeled. By means of this INPM, the performance of the system can be analyzed using standard MILP-solvers.

3.1 Formalisms

First, we introduce the formalisms from [3] which are later used in the integrated model. A RFM represents the runtime variability of the system by means of a feature model (see [18]). On this feature level cardinality groups and corresponding cardinality values (see [19, 20]) describe the system applications' variability at runtime. In a RFM, a cardinality

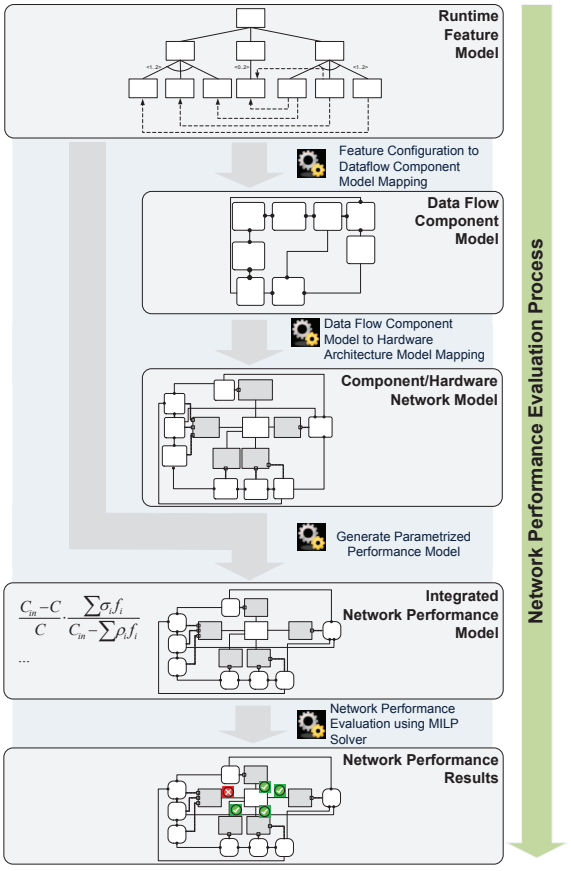

Figure 1: The integrated approach

group describes a set of features which can be active[1] in different multiplicities. An example is a video display feature as depicted in Figure 2. This feature is able to display two video streams at once, while it must show at least one video stream all the time. The user can choose between four different video sources: Human-Machine-Interface (HMI), navigation, storage video server, and Bluray player. Formally, we describe a RFM fm by a 5-tuple consisting of a set of Features F, a root feature r, decomposition edges D, cardinality groups Γ, and their cardinality values γ. The runtime variability space $S(fm)$, of a given feature model fm, consists of all possible valid runtime configurations S_i of the feature model fm.

$$fm = (F, r, D, \Gamma, \gamma) \tag{1}$$

$$S(fm) = \{S_1, ..., S_n\}, S_i \in 2^F \tag{2}$$

The *Dataflow Component Model (DCM)* captures a system on data flow level. Software is described by components and data flows are represented by data dependencies. Formally, a component model cm is a triple consisting of components[2]

[1]Here a feature represents a high-level application which can be activated or deactivated in the system.

[2]In the rest of this text, we mean software component, when ever we use the term component.

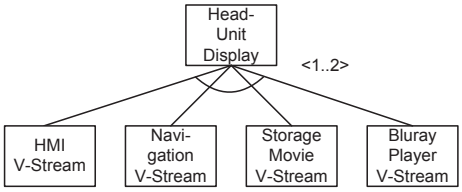

Figure 2: Sample feature group for a video display feature

$C = \{c_1, ..., c_n\}$, a set of data dependencies R_{dd}, and a map π_{data} that assigns a data traffic specification to every data dependency. Such a data traffic specification consists of the maximum amount of data σ a component can send at once, the average data rate ρ, and the maximum data frame size L_{max} on link layer.

$$cm = (C, R_{dd}, \pi_{data}) \tag{3}$$

$$\pi_{data}(rd) = (\sigma, \rho, L_{max}) \tag{4}$$

The mapping of features to software components is modeled by a map which links a leaf-feature to a set of components. This map represents which features are implemented by which components. A leaf-feature has no children. F_{leaf} is the set of all leaf-features, that is defined by $f \in F_{leaf} \rightarrow f$ *has no children.*

$$\mu_{F \times C} : F_{leaf} \rightarrow 2^C, \tag{5}$$

$$F_{leaf} = \{f \mid f \in F, \nexists (f, f') \in D\} \tag{6}$$

$$\mu_{F \times C}(f) = c, f \in F_{leaf}, c \in 2^C \tag{7}$$

In the CHNM, the network topology and the binding of the components to hardware elements are defined. Formally, a CHNM nt is a four-tuple consisting of the set of network nodes $N = N_c \cup N_{sw}$, the interconnections IC, and the mappings μ_C and μ_R. The set N_c consists of computing nodes and N_{sw} represents data switching nodes. These nodes are connected through interconnections $IC = \{ic_1, ..., ic_n\}, ic_i = (s_i, e_i, ls_i)$, where s_i is the start node, e_i is the end node and ls_i is the link speed. The mappings μ_C and μ_R describe the binding of components to hardware nodes with respect to which data flows traverse which interconnections.

$$nt = (N, IC, \mu_C, \mu_R) \tag{8}$$

$$\mu_C : C \rightarrow N_c \tag{9}$$

$$\mu_C(c) = n, c \in C, n \in N_c, \tag{10}$$

$$\exists (c, n) \in \mu_C \rightarrow \forall (c, n') \in \mu_C : n = n' \tag{11}$$

$$\mu_R : R_{dd} \rightarrow 2^{IC} \tag{12}$$

$$\mu_R(rd) = ic, rd \in R_{dd}, ic \in 2^{IC} \tag{13}$$

3.2 Integrated Network Performance Model

To achieve the goal of defining a model which predicts worst-case delays without exploring the entire runtime variability space, we create an integrated model. The idea behind this model is to parametrize the performance model. The parametrization is achieved in the following way. First, we formulate the problem of finding the worst-case aggregation delay for one switch output port for a runtime variability space as a maximization problem. The purpose is to find

the maximum delay within the runtime variability space. In particular, this optimization problem is formulated as a 0–1 linear-fractional program. This can then be solved, after some transformations, by standard MILP-solvers.

Before the parametrization is conducted, we present the performance model. This model is based on the work of [17]. It models the performance of full duplex switched Ethernet networks. Formula 14 calculates the network aggregation end-to-end delay of a virtual bit traversing the network. Since we focus on aggregation delays in this work, the considered path starts at the first switch, which connects the sending computing unit (CU) to the network, and ends at the last switch output port, which connects the receiving CU to the network. The network aggregation end-to-end delay $\hat{d}_{aggr}$ for a data dependency dd consists of the sums of i) the store-and-forward delay $\hat{d}_{sf}$ at the input ports of the traversed switches and ii) the aggregation delay $\hat{d}$ of the switch output ports. The store-and-forward delay depends on the maximum frame size L on the data link layer of the input port, where the corresponding data dependencies must traverse the link ($ic \in \mu_R(dd_i)$), and on the link speed $ic.ls$ of the input port. The delay on a switch output port is calculated according to Formula 16, where C_{in} and C_{out} represent the speed of the incoming links and the speed of the output link. The variables σ_j and ρ_j are the sums of the burst and average data transmission values of the data dependencies traversing the link ($ic \in \mu_R(dd)$). Each of those data dependencies dd belongs to a sending component c_{ji} and a receiving component c_{jk} which are part of the mapping $\mu_{F\times C}$ for a corresponding leaf feature f_j. It is assumed that any two mappings $\mu_{F\times C}(f)$ and $\mu_{F\times C}(f')$ are disjunctive for $f \neq f'$, i.e., there are no overlapping mappings and there are no data dependencies between any two components c and c' of two different mappings, i.e., $\nexists (c,c') \in R_{dd} \mid c \in \mu_{F\times C}(f), c' \in \mu_{F\times C}(f')$. The variable $\bar{L}$ represents the sum of the maximum frame sizes L on the data link layer of the incoming links.

$$\hat{d}_{aggr}(dd) = \sum_{i=1}^{n} \left(\hat{d}_{sf}(ic_{i-1}) + \hat{d}(ic_i)\right), \quad PATH(dd) = (ic_n, ..., ic_0) \tag{14}$$

$$\hat{d}_{sf}(ic) = \frac{L}{ic.ls}, \quad L = \max(dd_1.L_{max}, ..., dd_n.L_{max}), \quad ic \in \mu_R(dd_i) \tag{15}$$

$$\hat{d}(ic) = \frac{C_{in} - C_{out}}{C_{out}} \cdot \frac{\Sigma\sigma_j}{C_{in} - \Sigma\rho_j} + \frac{\bar{L}(C_{out} - \Sigma\rho_j)}{C_{out}(C_{in} - \Sigma\rho_j)}, \quad \sigma_j = \Sigma\sigma_{jk}, \rho_j = \Sigma\rho_{jk}, \quad ic \in \mu_R(dd_{jk}), \pi_{data}(dd_{jk}) = (\sigma_{jk}, \rho_{jk}, ...), \quad dd_{jk} = (c, c'), c, c' \in \mu_{F\times C}(f_j), \quad C_{in} = \Sigma ic_{in}.ls, \quad ic_{in} : \exists dd, PATH(dd) = (..., ic, ic_{in}, ...), \quad \bar{L} = \Sigma L \tag{16}$$

Now, the parametrization of Formula 16 is presented. As stated before, we formulate our problem as a linear-fractional program consisting of an objective function that must be maximized and constraints which must be fulfilled. At first, Formula 16 is extended to Formula 17 by boolean variables $f_1 ... f_n$, $n = |F|$, $f_i \in \{0,1\}$. That is, if a feature is active, then $f_i = 1$; if not, $f_i = 0$. The root feature of the feature model has to be active anyway. Furthermore, since we calculate the worst-case network aggregation delays on a per feature basis (cf. Section 3.5), this feature f_{active} has to be activated, too. The limitations that are given through the cardinality groups and the decomposition edges (see [19]) are modeled in the constraints equations (Formulas 20 – 21). We model a decomposition edge $d_{pi,i} = (f_{pi}, f_i)$ as an inequality $f_i - f_{pi} \leq 0$. The cardinality groups are modeled in the following form: for a cardinality group $\Gamma_j = \{f_{j,1}, ..., f_{j,n}\}$ with $\gamma(\Gamma_j) = (x, y)$ (i.e., there must be at least x and there may be up to y features active), the corresponding constraint inequality is $\Sigma f_{j,i} - y \leq 0$. Thus, we get the following 0–1 linear-fractional program:

$$(\mathbf{POLF}) \max \quad \frac{(C_{in} - C_{out})\Sigma\sigma_j f_j + \bar{L}(C_{out} - \Sigma\rho_j f_j)}{C_{out}(C_{in} - \Sigma\rho_j f_j)} \tag{17}$$

$$s.t. \quad r = 1 \tag{18}$$

$$f_{active} = 1 \tag{19}$$

$$f_i - f_{pi} \leq 0, \quad (f_{pi}, f_i) \in D \tag{20}$$

$$\sum f_{j,i} - y \leq 0, \quad f_{j_1}, ..., f_{j_n} \in \Gamma_j,\ \gamma(\Gamma_j) = (x, y) \tag{21}$$

$$f_j \in \{0,1\}, C_{in} - \Sigma\rho_j f_j > 0 \tag{22}$$

3.3 Reformulation-Linearization

Since the 0–1 linear-fractional program (LFP) formulated in Formulas 17-22 is not solvable with MILP-solvers in this form, we apply the reformulation-linearization method published in [6] by Yue et al.. This transforms the 0-1 LFP into its MILP-equivalent. In a first step, this linearization technique introduces a new variable $u = \frac{1}{C_{in} - \Sigma\rho_j f_j}$. In a second step, further variables $w_i = u \cdot f_i$ are introduced. Thus, $POLF$ can be transformed into the MILP POL which is given below:

$$(\mathbf{POL}) \max \quad u\bar{L}C_{out} + \sum w_j \left((C_{in} - C_{out})\sigma_j - \bar{L}\rho_j\right) \tag{23}$$

$$s.t. \quad C_{out}(C_{in}u - \Sigma\rho_j w_j) = 1 \tag{24}$$

$$w_0 - u = 0 \tag{25}$$

$$w_{active} - u = 0 \tag{26}$$

$$w_1 - w_0 \leq 0 \tag{27}$$

$$w_2 - w_1 \leq 0 \tag{28}$$

$$...$$

$$\Sigma w_{j,i} - uy \leq 0 \tag{29}$$

$$w_j - u \leq 0 \tag{30}$$

$$w_j - M \cdot f_j \leq 0 \tag{31}$$

$$w_j - u - M \cdot f_i \geq -M \tag{32}$$

$$u \geq 0, w_j \geq 0, f_i \in \{0,1\} \tag{33}$$

where M can be determined by heuristics (cf. [6]).

3.4 Relaxations

In this section, we introduce relaxations that preserve the linearity of POL. As can be observed from the Appendix, these relaxations still guarantee worst-case results. Furthermore, they lead to acceptable over-approximation (see Section 4.2).

3.4.1 C_{in}-Relaxation

In order to solve *POL* with a MILP-solver, *POL* must be linear. Since C_{in} depends on the selection of the features (see Formula 16), Formula 23 would be non-linear. To avoid this, we assume C_{in} to be the sum of the speeds of the links which have a data dependency traversing the incoming link and the outgoing link of the analyzed port. Thus, the estimated C'_{in} is always bigger or equal to the real C_{in}. As can be obtained from the Appendix, this relaxation guarantees worst-case results. That means, despite this estimation, we always obtain worst-case results.

3.4.2 σ-Relaxations

The burst-value σ changes after experiencing delay on a network element. According to [17], the new burst-value can be calculated as $\sigma_{new} = \sigma + \rho \cdot d$, where d is the delay of the data flow experienced on the data path so far. To determine the new sigma value, we must calculate all delays on the path. That, in turn would imply that we must calculate all sigma values of all data dependencies interfering with these data dependencies. Note that this relation is transitive. That is to say, a data dependency dd_a interfering with dd_c ($dd_a \sim dd_c$) does not require to have a common path segment with dd_c. It is sufficient if there is a chain $dd_a \sim dd_b \sim dd_c$. This has an impact on two calculations. Firstly, it would prohibit a linear form of *POL* since the replacement of the σ_i variables would lead to non-linear terms. Secondly, for the same reason, $\hat{d}_{aggr}$ cannot be directly formulated as a linear objective function. Therefore, we introduce three heuristics: i) we relax $\sigma_{new} = \sigma + \rho \cdot d$ to Formula 34; ii) we assume $\hat{d}_{sf}$ to be calculated according to Formula 35; iii) we formulate $\hat{d}_{aggr}$ to a form which requires only the calculations of multiple instances of *POL*. As we show in Section 4.2, these relaxations can introduce over-approximation to Formula 36. The reason for this lies in the fact that the single values of $\Sigma\hat{d}_r(ic_j)$ (for Formula 34 and 36) could potentially belong to different runtime configurations since the single instances of *POL* are not synchronized.

$$\begin{aligned} &\sigma_{new} = \sigma + \rho \cdot \Sigma(\hat{d}_{sf_r}(ic_{j-1}) + \hat{d}_r(ic_j)) \\ &ic_j \in PATH(dd) = (.., ic_i, ic_{i-1}, ..., ic_0), 0 < j < i \end{aligned} \tag{34}$$

Since the delay of every network element has an impact on the burst value σ, the store-and-forward delay has an impact, too. That is to say, the σ-value changes to $\sigma_{new} = \sigma \cdot (L/ic.ls)$ after traversing an input port of a switch, where L depends on the selected features. This would introduce again non-linearity to *POL*. Therefore, we relax the determination of this value by selecting the maximum value over all features on that link. The difference between this heuristic and Formula 15 lies in the scope of the max-term. In Formula 15, the maximum over one runtime configuration is determined. In Formula 35, the maximum over all configurations is determined. Thus, $\hat{d}_{sf_r}(ic) \geq \hat{d}_{sf}(ic)$ always holds.

$$\begin{aligned} \hat{d}_{sf_r}(ic) = \; & \frac{L}{ic.ls}, \\ & L = \max(dd_1.L_{max}, ..., dd_n.L_{max}), \\ & ic_{i-1} \in \mu_R(dd_j), ic_i \in \mu_R(dd_j),\ ic_i = ic \\ & PATH(dd) = (.., ic_i, ic_{i-1}, ..., ic_0) \end{aligned} \tag{35}$$

For the estimation of the σ_{new}-value this means that the estimated σ'_{new} is always greater than or equal to the correct σ_{new}. Thus, the same proof as for Formula 34 can be applied to verify the worst-case guarantee of this heuristic. It can be found in the appendix.

Besides calculating store-and-forward delays of input ports and the aggregation delays of single switch output ports, we formulate the equivalent of Formula 14. To avoid non-linearity, we do not formulate a single objective function for $\hat{d}_{aggr}$. Instead of doing so, we calculate $\hat{d}_{aggr}$ by summing up the relaxed aggregation delays $\hat{d}_r$ and the relaxed store-and-forward delays $\hat{d}_{sf_r}$ on the path of the considered data dependency. The values of $\hat{d}_r$ are obtained using *POL* and Formula 34.

$$\begin{aligned} \hat{d}_{aggr\text{-}r}(dd) = \; & \sum_{i=1}^{n} \left(\hat{d}_{sf_r}(ic_{i-1}) + \hat{d}_r(ic_i) \right), \\ & PATH(dd) = (ic_n, ..., ic_0) \end{aligned} \tag{36}$$

As the C_{in}-relaxation, the σ-relaxations do not violate the worst-case guarantee of our approach. The proof can be found in the Appendix.

3.5 Algorithm

In this section, we describe the algorithm which makes use of the previously defined integrated model. As can be obtained for Algorithm 1, the algorithm iterates over all leaf-features F_{leaf}. For each $f_{active} \in F_{leaf}$, the aggregation delays for all links $ic \in IC$ are calculated according to *POL* (see Formulas 23 – 34). After that step, the network aggregation end-to-end delay is calculated for each data dependency that belongs to that feature f_{active}, as the sum of the aggregation delays and store-and-forward delays. Since this value is already worst-case, the performance results can be directly updated with the tuple $(dd, \hat{d}_{aggr\text{-}r}(dd))$.

Algorithm 1 End-to-End Aggregation Delay Calculation

```
1: procedure E2EAggregationDelayCalc
2:     for f_active ∈ F_leaf do                    ▷ For all leaf-features
3:         for ic ∈ IC do                          ▷ For all interconnections
4:             d̂_r(ic) = POL(f_active, ic)
5:         end for
6:         for dd : dd = (c', c), c', c ∈ μ_{F×C}(f_active) do
7:         ▷ For all data dependencies that belong to f_active
8:             d̂_{aggr-r}(dd) = Σ_{i=1}^{n} ( d̂_{sf_r}(ic_{i-1}) + d̂_r(ic_i) )
9:         end for
10:    end for
11: end procedure
```

4. EVALUATION

In this section, we evaluate the performance and accuracy of our approach presented in Section 3 in various experiments of different network sizes.

4.1 Test setups

To examine the scalability and the accuracy of our approach, we define test-setups (see Table 2 and 1). They represent a set of typical CPS as they can be found in current automobiles (cf. [21, 22]). The setups primarily differ in the numbers of features, while the ratios between leaf-features and components, as well as between components and Electronic Control Units (ECU), are fixed. The varying parameter for

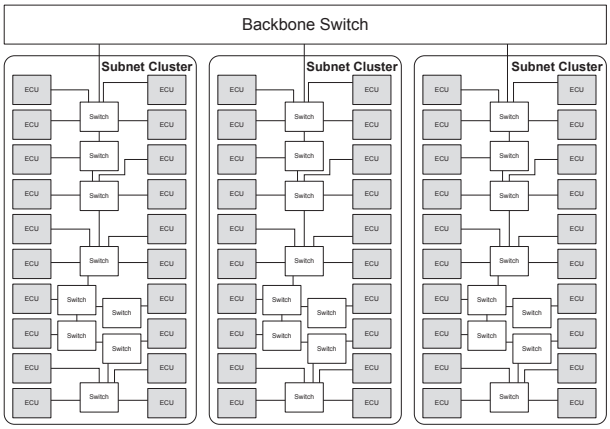

Figure 3: Example of 60 ECUs Topology. The subnets consist of a mixed daisy chain and star topology and are interconnected with a backbone switch.

the different feature models is the number of leaf-features. For capturing the runtime variability, we assume that two variants are expressed by ten leaf-features. Thus, the number of runtime configurations for each test-setup is given by $2^{|F_{leaf}|/10}$. Thereby, the overall number of runtime configurations varies from 32 to 1.1×10^{15}. The cardinality groups of the feature models differ in hierarchy level, i.e., the level in the feature tree. In our scenarios, we have up to 8 hierarchy levels.

In our setups, each leaf-feature maps to a group of 8 components with regard to automotive systems (cf. [22]) and are interconnected randomly (uniformly distributed). Representative data traffic characteristics (cf. [8, 23]) are assigned according to Table 1. We define seven common traffic categories: driver assistant video, control data, navigation bulk data, stereo audio data, storage video data, BluRay video data, and multi-channel audio data. Their σ -and ρ-values are randomly varied (uniformly distributed) within a range of 20 percent. The probability that a data dependency gets assigned to a particular traffic category is documented in column 2 of Table 1.

The ratio between ECUs and components is set to 1:20, derived from [21]. To keep the network topologies scalable, they are created hierarchically (see Figure 3). The smallest segment size consists of 20 ECUs, which are randomly connected in mixed daisy chain and star topology, as expected for upcoming automobiles (see [8, 9]). For slightly larger topologies, these subnet clusters are interconnected by direct links, i.e., no backbone switches are needed. As depicted in Figure 3, three subsegments are interconnected through a backbone switch. If the number of subnet segments exceeds four[3], the backbone clusters are interconnected in daisy chain mode. More than three backbone segments are interconnected by an additional backbone switch. The data transmission rate of the links is set to the smallest value v which results in a maximum link utilization below 40%, where $v \in \{100\,\text{MBits} \cdot 10^n | \; n \in \mathbb{N}_0\}$ and $\mathbb{N}_0$ is the set of natural numbers including 0.

[3]Four subsegments are also interconnected by only one backbone switch since it would make no sense to connect the fourth subsegment to a new backbone switch.

To keep the complexity of the test-setups at a defined level, the parameter *average runtime configurations per switch output port* is introduced (see column *#Runtime Configurations* in Table 2). The number of runtime configurations per switch output port is defined as the number of runtime configurations of the feature model obtained through a data link to feature model projection (see Definition 1 in the Appendix). That is to say, this parameter defines the average complexity of *POL* (see Section 3). The number of bound components for one ECU is in the interval $[1, \frac{2 \cdot |C|}{|N_c|}]$, where C is the set of all components and N_C is the set of Computing Units (i.e., ECUs).

The approach presented in Section 3 is implemented in C using lp_solve 5.5 [7]. For each test-setup the calculations are distributed over 16 threads. All experiments are performed on a 64 bit machine with two octa-core Intel Xeon E5-2660 and 128 GByte RAM. The operating system is a Debian 3.14.4-1 64bit Linux.

Data Traffic Categories	Dist. [%]	ρ [MBits]	σ [octet]	L_{max} [octet]
Driver Assis. Video Data	30	6.3600	28918	1522
Control Data	20	0.0512	6400	64
Navigation Bulk Data	15	1.7046	21308	1522
Stereo Audio Data	15	1.5136	946	946
Storage Video Data	10	15.7070	65446	1522
BluRay Video Data	5	41.6419	173508	1522
Multi-channel Audio Data	5	3.8336	2396	946

Table 1: Distribution of the network traffic characteristics

4.2 Results

Figure 4 shows the computation time needed to determine the worst-case results of test-setups 1 - 9. For the smaller setups 1 - 3, the approach from [3] performs better than the integrated approach. In particular for test-setup 3, it performs approximately twice as fast. For the larger test-setups 4 and 5, the integrated approach needs significantly less computation time. The delay calculation for test-setup 5 – which for instance represents a network with 100 ECU, 2000 components, 250 features and approximately 34×10^6 runtime configurations – took about 3 minutes for the integrated approach. In comparison, the approach from [3] took approximately 3 hours. Test-setups 6 - 9 did not finish within reasonable time applying the approach from [3].

Here, it is clearly observable that the computation time needed for the approach from [3] grows exponentially with the number features (i.e., linear with the size of the runtime variability space). The time needed by our integrated approach scales with the test-setup size. Even for test-setup 9,

Test-setup	#Leaf-features	#Cardinality Groups Hierarchy Level 1	2	3	4	5	6	7	8	#Runtime Configurations overall	average per Switch Port	#Components	#ECUs	#Switches
1	50	-	-	1	1	1	-	-	-	32	1	400	20	12
2	100	-	-	3	3	-	-	-	-	1.0×10^3	10	800	40	22
3	150	-	-	2	5	1	-	-	-	3.3×10^4	$3.3 \times 10^2 \pm 1\%$	1200	60	42
4	200	-	-	4	5	3	-	-	-	1.0×10^6	$1.0 \times 10^4 \pm 1\%$	1600	80	49
5	250	-	1	2	-	9	15	3	-	$3.4 \times 10^7 \pm 1\%$	$3.4 \times 10^5 \pm 1\%$	2000	100	67
6	300	-	1	1	1	7	7	5	-	$1.1 \times 10^9 \pm 1\%$	$1.1 \times 10^7 \pm 1\%$	2400	120	77
7	350	-	1	2	2	8	18	5	2	$3.4 \times 10^{10} \pm 1\%$	$3.4 \times 10^8 \pm 1\%$	2800	140	91
8	400	-	1	4	3	18	22	2	-	$1.1 \times 10^{12} \pm 1\%$	$1.1 \times 10^{10} \pm 1\%$	3200	160	110
9	500	-	1	4	15	22	18	6	-	$1.1 \times 10^{15} \pm 1\%$	$1.1 \times 10^{13} \pm 1\%$	4000	200	143

Table 2: Experimental Setups

which is far beyond the system size of realistic automotive systems, the integrated approach could anyway verify the network performance within 30 minutes.

Since we made some relaxations in the integrated model, we reimplemented the approach from [3] in C to obtain an estimation of the over-approximation of the delay values. As already stated, the approach from [3] did not finish within reasonable time for test-setups 6 - 9. Therefore, we are only able to evaluate the over-approximation of test-setup 1 - 5. Figure 5 shows the 90% percentile and the mean values of the over-approximation of the integrated approach compared to [3]. As can be derived, the over-approximation grows with the size of the test-setups. However, even for test-setup 5 the 90^{th} percentile it is still below 18%.

Thus, we could show that the integrated approach is, in contrast to present approaches, able to verify the network performance of CPS with multiple configurations. Moreover, it scales for large network systems and verifies its performance in reasonable time. The calculated over-approximation is within 18% of an acceptable range. Hence, overall it can be stated that our integrated approach is well-suited for verifying the network performance of dynamic CPS with multiple configurations.

5. CONCLUSION AND FUTURE WORK

In the paper at hand, we presented a novel approach that provides a methodology for the prediction of network aggregation end-to-end delays for CPS with multiple runtime configurations. In contrast to previous work, our approach considers runtime variability while guaranteeing worst-case results at the same time. As shown by the example of varying setups for automobile systems, our approach scales even for large network configurations of complex CPS. The observed over-approximation of network delays through the relaxations is acceptable, particularly, as - to the best of the authors knowledge - no other approach solves this scalability problem while ensuring worst-case guarantees. For future work, we plan to extend the integrated approach with an application end-to-end model.

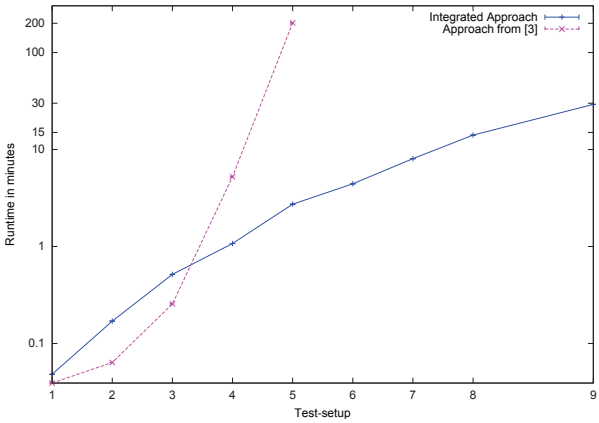

Figure 4: Time to obtain the network performance results for test-setup 1 - 9 in logarithmic scale

6. REFERENCES

[1] "Autosar — the worldwide automotive standard for e/e systems," *ATZextra worldwide*, vol. 18, no. 9, pp. 5–12, 2013. [Online]. Available: http://dx.doi.org/10.1007/s40111-013-0003-5

[2] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, "System-scenario-based design of dynamic embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 3:1–3:45, 2009. [Online]. Available: http://doi.acm.org/10.1145/1455229.1455232

[3] M. Manderscheid and C. Prehofer, "Network performance evaluation for distributed embedded systems using feature models," in *The Eighteenth International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*, 2013.

[4] P. van Stralen and A. Pimentel, "Fast scenario-based design space exploration using feature selection," in *ARCS Workshops (ARCS), 2012*, 2012, pp. 1–7.

[5] F. Siyoum, M. Geilen, J. Eker, C. v. Platen, and

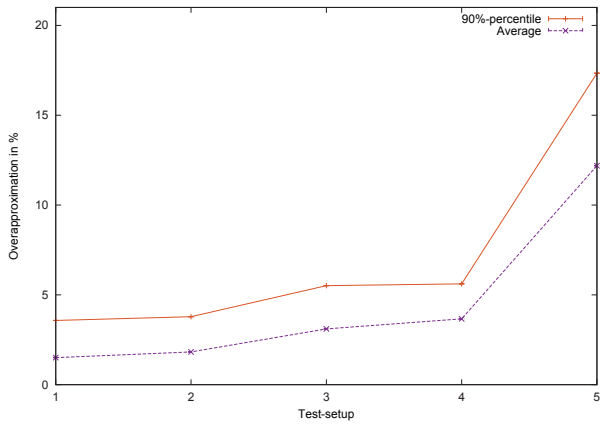

Figure 5: Over-approximation of the integrated approach compared with the approach from [3] for test-setup 1 - 5

H. Corporaal, "Automated extraction of scenario sequences from disciplined dataflow networks," in *Formal Methods and Models for Codesign (MEMOCODE), 2013 Eleventh IEEE/ACM International Conference on*, 2013, pp. 47–56.

[6] D. Yue, G. Guillén-Gosálbez, and F. You, "Global optimization of large-scale mixed-integer linear fractional programming problems: A reformulation-linearization method and process scheduling applications," *AIChE Journal*, vol. 59, no. 11, pp. 4255–4272, 2013. [Online]. Available: http://dx.doi.org/10.1002/aic.14185

[7] "lp_solve version 5.5.1: http://lpsolve.sourceforge.net/5.5/."

[8] H.-T. Lim, L. Völker, and D. Herrscher, "Challanges in a future ip/ethernet based in-car network for real-time applications," in *Design Automation Conference*, 2011.

[9] H.-T. Lim, B. Krebs, L. Volker, and P. Zahrer, "Performance evaluation of the inter-domain communication in a switched ethernet based in-car network," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, 2011, pp. 101–108.

[10] J. Rox, R. Ernst, and P. Giusto, "Using timing analysis for the design of future switched based ethernet automotive networks," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 57–62.

[11] D. Thiele, J. Diemer, P. Axer, R. Ernst, and J. Seyler, "Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, 2013, pp. 1–10.

[12] Jonas Diemer, Jonas Rox, and Rolf Ernst, "Modeling of ethernet avb networks for worst-case timing analysis," in *MATHMOD 2012 - 7th Vienna International Conference on Mathematical Modelling*, Vienna and Austria, 2012. [Online]. Available: http://doi.org/10.3182/20120215-3-AT-3016.00150

[13] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, 2012, pp. 1–10.

[14] F. Reimann, S. Graf, F. Streit, M. Glaß, and J. Teich, "Timing analysis of ethernet avb-based automotive e/e architectures," in *Proceedings of IEEE International Conference on Emerging Technology & Factory Automation (ETFA)*, 2013, p. 8.

[15] Jork Löser, "Low-latency hard real-time communication over switched ethernet," Ph.D. dissertation, Technischen Universität Dresden, 2006.

[16] J. P. Georges, T. Divoux, and E. Rondeau, "Strict priority versus weighted fair queueing in switched ethernet networks for time critical applications," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, p. 141.

[17] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[18] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux, "Feature diagrams: A survey and a formal semantics," in *Requirements Engineering, 14th IEEE International Conference*, 2006, pp. 139–148.

[19] K. Czarnecki, S. Helsen, and E. Ulrich, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, pp. 7–29, 2005.

[20] R. Michel, A. Classen, A. Hubaux, and Q. Boucher, "A formal semantics for feature cardinalities in feature diagrams," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, ser. VaMoS '11. New York and USA: ACM, 2011, pp. 82–89.

[21] K. Venkatesh Prasad, M. Broy, and I. Krueger, "Scanning advances in aerospace & automobile software technology," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 510–514, 2010.

[22] A. Sangiovanni-Vincentelli and M. Di Natale, "Embedded system design for automotive applications," *IEEE Computer*, vol. 40, no. 10, pp. 42–51, 2007.

[23] W. Hintermaier and Eckehard Steinbach, "A novel real-time video data scheduling approach for driver assistance services," in *IEEE Intelligent Vehicles Symposium (IV 2011)*, 2011.

APPENDIX

DEFINITION 1. *Data link - RFM Projection*
A data-link-RFM-projection projects a data link onto a runtime feature model (RFM). Formally, this is a map μ_{proj} *that maps a tuple* (ic, fm) *to a runtime feature model* fm_{ic}*, where* ic *is the projected data link,* fm *the original RFM, and* fm_{ic} *the resulting RFM.*

$$\mu_{proj}(ic, fm) = fm_{ic},$$
$$ic \in IC,\ fm = (F, r, D, \Gamma, \gamma), fm_{ic} = (F_{ic}, r, D_{ic}, \Gamma_{ic}, \gamma_{ic})$$

The elements of the resulting feature model fm_{ic} *are defined as follows. The root feature* r *of the original RFM is also the root feature of* fm_{ic}*. The set of the features* F_{ic} *is a subset of the features of the original RFM. A feature* f *is in* F_{ic}*, if one the following conditions holds: i) the feature* f *maps to any two components* c *and* c' $(c', c \in \mu_{F\times C}(f))$ *that are*

part of a data dependency dd and dd traverses the data link ic ($ic \in \mu_R(dd)$); or ii) the feature f is a parent feature of a feature f' for which condition i) holds. D^+ is the transitive closure of the set of decomposition edges D of the original RFM. A feature f is a parent feature for f', if $(f, f') \in D^+$.

The set of decomposition edges D_{ic} contains only edges from D, for which both features f and f' are in F_{ic}.

$$D_{ic} = \{d \mid d = (f, f') \in D, f \in F_{ic}, f' \in F_{ic}\}$$

The cardinality groups $\Gamma_{i_{ic}}$ are the subsets of the correspondent cardinality groups Γ_i. The cardinality values $\gamma_{ic}(\Gamma_{i_{ic}}) = (min_{ic}, max_{ic})$ depend on the size $|\Gamma_{i_{ic}}|$ of the cardinality groups. If the size of $|\Gamma_{i_{ic}}|$ is less than the size of the original values $\gamma(\Gamma_i).min$ or $\gamma(\Gamma_i).max$, then those values must be adapted.

$$\Gamma_{ic} = \{\Gamma_{1_{ic}}, ..., \Gamma_{n_{ic}}\}, \Gamma_{i_{ic}} = \Gamma_i \backslash \{f | f \notin F_{ic}\}$$
$$\gamma_{ic} = \{(\Gamma_{i_{ic}}, (min_{ic}, max_{ic})) \mid min_{ic} = \min(\gamma(\Gamma_i).min, |\Gamma_{i_{ic}}|),$$
$$max_{ic} = \min(\gamma(\Gamma_i).max, |\Gamma_{i_{ic}}|)\}$$

PROOF. The C_{in}- and $\bar{L}$-relaxations are worst-case preserving.
Let $\hat{d}$ be the worst-case delay of a switch output port according to Formula 16. According to Section 3.4.1 the C_{in}-relaxation is a over-estimation of the C_{in}-parameter. That means the estimated $C'_{in} = C_{in} + \epsilon_C, \epsilon_C \geq 0$. Analogously, the $\bar{L}$-relaxation leads to a overestimated $\bar{L}' = \bar{L} + \epsilon_L, \epsilon_L \geq 0$. We now show that for every over-estimated $C'_{in} = C_{in} + \epsilon_C, \epsilon_C \geq 0$ and $\bar{L}' = \bar{L} + \epsilon_L, \epsilon_L \geq 0$ the delay $\hat{d}_r$ is always greater than or equal to the correspondent $\hat{d}$.

$$\hat{d}_r = \frac{(C'_{in} - C_{out})\sigma + \bar{L}'(C_{out} - \rho)}{C_{out}(C'_{in} - \rho)}$$

$$\hat{d}_r \geq \hat{d} \Leftrightarrow \hat{d}_r - \hat{d} \geq 0$$

$$\hat{d}_r - \hat{d} = \frac{(C'_{in} - C_{out})\sigma + \bar{L}'(C_{out} - \rho)}{C_{out}(C'_{in} - \rho)} - \frac{(C_{in} - C_{out})\sigma + \bar{L}(C_{out} - \rho)}{C_{out}(C_{in} - \rho)}$$

$$= \frac{(C_{in} - \rho)\left[(C'_{in} - C_{out})\sigma + \bar{L}'(C_{out} - \rho)\right]}{C_{out}(C'_{in} - \rho)(C_{in} - \rho)} - \frac{(C'_{in} - \rho)\left[(C_{in} - C_{out})\sigma + \bar{L}(C_{out} - \rho)\right]}{C_{out}(C'_{in} - \rho)(C_{in} - \rho)}$$

$$= \frac{(C_{in} - \rho)(C'_{in} - C_{out})\sigma - (C'_{in} - \rho)(C_{in} - C_{out})\sigma}{C_{out}(C'_{in} - \rho)(C_{in} - \rho)} + \frac{\bar{L}'(C_{in} - \rho)(C_{out} - \rho) - \bar{L}(C'_{in} - \rho)(C_{out} - \rho)}{C_{out}(C'_{in} - \rho)(C_{in} - \rho)}$$

$$= \frac{\sigma\left[(C_{in} - \rho)(C_{in} + \epsilon_c - C_{out}) - (C_{in} + \epsilon_c - \rho)(C_{in} - C_{out})\right]}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)} + \frac{(C_{out} - \rho)\left[(\bar{L} + \epsilon_L)(C_{in} - \rho) - \bar{L}(C_{in} + \epsilon_c - \rho)\right]}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)}$$

$$= \frac{\sigma\epsilon_c(C_{out} - \rho)}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)} + \frac{(C_{out} - \rho)(\epsilon_L(C_{in} - \rho) - \bar{L}\epsilon_c)}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)}$$
$$= \frac{(C_{out} - \rho)(\sigma\epsilon_c + (\epsilon_L(C_{in} - \rho) - \bar{L}\epsilon_c)}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)}$$
$$= \frac{(C_{out} - \rho)(\epsilon_c(\sigma - \bar{L}) + \epsilon_L(C_{in} - \rho))}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)}$$

Since $C_{out} > \rho, C_{in} > \rho, \sigma \geq \bar{L}$:
$\hat{d}_r - \hat{d} = \frac{(C_{out} - \rho)(\epsilon_c(\sigma - \bar{L}) + \epsilon_L(C_{in} - \rho))}{C_{out}(C_{in} + \epsilon_c - \rho)(C_{in} - \rho)} \geq 0$ holds. □

PROOF. σ-relaxations are worst-case preserving.
In this proof we show that relaxations in Formulas 34 - 36 are worst-case preserving. Let $\hat{d}$ be the worst-case delay of a switch output port according to Formula 16. According to Section 3.4.2 the σ-relaxations base on an over-estimation of the σ-parameter. That means the estimated $\sigma' = \sigma + \epsilon, \epsilon \geq 0$. We now show that i) for every over-estimated $\sigma' = \sigma + \epsilon, \epsilon \geq 0$ the delay $\hat{d}_r$ is always bigger than or equal to the correspondent $\hat{d}$, and ii) that the resulting $\hat{d}_{aggr-r}$ is always greater than or equal to the actual $\hat{d}_{aggr}$.
i:

$$\hat{d}_r = \frac{(C_{in} - C_{out})(\sigma + \epsilon) + \bar{L}(C_{out} - \rho)}{C_{out}(C_{in} - \rho)}$$

$$\hat{d}_r \geq \hat{d} \Leftrightarrow \hat{d}_r - \hat{d} \geq 0$$

$$\hat{d}_r - \hat{d} = \frac{(\sigma + \epsilon)(C_{in} - C_{out}) + \bar{L}(C_{out} - \rho)}{C_{out}(C_{in} - \rho)} - \frac{\sigma \cdot (C_{in} - C_{out}) + \bar{L}(C_{out} - \rho)}{C_{out}(C_{in} - \rho)}$$

$$= \frac{(C_{in} - C_{out})(\sigma + \epsilon) - \sigma \cdot (C_{in} - C_{out})}{C_{out}(C_{in} - \rho)} = \frac{(C_{in} - C_{out})(\sigma + \epsilon - \sigma)}{C_{out}(C_{in} - \rho)} = \frac{\epsilon \cdot (C_{in} - C_{out})}{C_{out}(C_{in} - \rho)}$$
Since $C_{in} > \rho$ and $C_{in} > C_{out}$:
$\hat{d}_r - \hat{d} = \frac{\epsilon \cdot (C_{in} - C_{out})}{C_{out}(C_{in} - \rho)} \geq 0$ holds.
ii: Since $\hat{d}_r \geq \hat{d}$ and $\hat{d}_{st_r} \geq \hat{d}_{st}$, $\hat{d}_{aggr-r} = \sum\left(\hat{d}_{st_r} + \hat{d}_r\right) \geq \hat{d}_{aggr} \geq \sum\left(\hat{d}_{st} + \hat{d}\right)$ also holds. □