# Verification of network end-to-end latencies for adaptive ethernet-based cyber-physical systems

Martin Manderscheid, Gereon Weiss and Rudi Knorr

Fraunhofer Institute for Embedded Systems and Communication Technologies ESK
Hansastrasse 32, Munich, Germany
{martin.manderscheid, gereon.weiss, rudi.knorr}@esk.fraunhofer.de

***Abstract*—As Cyber-Physical Systems (CPS) are evolving towards flexible and smart systems, their dependable communication becomes a decisive factor. In order to still guarantee a predictive and real-time behavior, verifying the network performance of such adaptive systems is vital. Therefore, the performance-verification has to consider the runtime variability while scaling for larger number of applications and networks in CPS. We introduce a novel performance-verification approach with integrated variability enabling the analysis of adaptive Ethernet-based CPS. It incorporates a formal model capturing all relevant characteristics for deriving safe communication bounds. Its soundness has been evaluated in an extensive automotive case study and several changing test setups targeting scalability. The results show that this integrated variability approach is superior to a common static analysis and previously utilized heuristic. In direct comparison it outperforms static analysis by up to 95 percent within the evaluated automotive system. Moreover, the results show that it scales well and provides a profound basis for analyzing larger adaptive networked systems.**

## I. Introduction

Today's cyber-physical systems (CPS) enable a broad range of applications based on software and electronics. To that end, CPS are integrating more and more actuators and sensors with increasing communication demands, such as high definition cameras or laser scanners in the automotive domain. Thus, the needed network bandwidth rises to a level, which can only be satisfied by high bandwidth communication technologies, such as Ethernet[1], [2], [3]. While Ethernet provides high bandwidth to applications, the planning of such interactive networks can no longer rely on static traffic assumptions. CPS have to adapt dynamically to various contexts (cf. [4]). Hence, a static network planning cannot adequately address these flexibility concerns anymore. Modern automobiles for example, may activate different combinations of driver assistance functions, depending on the driving context. For instance, while backing into a parking space the rear view camera function and the park distance control functions are simultaneously activated. During automated highway driving, the activation of the navigation system and the cruise control would rather make sense.

CPS are often produced at high volume while production costs are tried to be kept at a low level. This in turn, has a great impact on the development process. Here, one tries to predict the required hardware resources as exact as possible. At the same time, CPS involve high dependability demands and their functionalities - or also so-called features - must in most cases meet time-critical requirements. For example, when pictures recorded through a rear view camera are sent to the head-unit of an automobile, they have to be displayed within a defined time interval. The fulfillment of those requirements by the network architecture design is verified within the *network performance-verification*. The objective of this step inside the design process of CPS is to meet the requirements of all applications with the lowest possible amount of hardware resources. At this process step, engineers are facing the challenge of mastering the complexity of a vast amount of runtime configurations and an exact network performance-verification for a cost-efficient and dependable network architecture at the same time. In this work, we contribute to this challenge by introducing:

- An improved scalable heuristic model for the network performance-verification of dynamic CPS, which does neither rely on manual runtime configuration definitions nor does it imply artificial restrictions on network topology or medium access methods,
- a corresponding method enabling the calculation of safe bounds for the minimum runtime configuration switching transition delay, and
- an automotive case study representing a typical use-case of real-time CPS proving the applicability of the presented approach.

In the following Section II, we introduce related approaches and background to the introduced network performance-verification. As a running example, we provide a case study of an automotive video system in Section III. In the subsequent Section IV, we present our approach for the network performance-verification with integrated variability for dependable and adaptive CPS. Its validity, applicability and scalability is evaluated within the automotive case study and synthetic test-setups in Section V. Afterwards, we conclude our work in Section VI.

## II. Background and Related Work

Looking a decade or more back, network performance-verification[5], [6], [7] of distributed embedded systems was mainly covering static scenarios. That is, the analytic model covered one static software configuration consisting of a number of tasks competing for shared resources provided

by the system architecture. Since then, distributed embedded systems are becoming more and more dynamic and open, thus, evolving to CPS. This step offered a multitude of possibilities, which made the performance-verification simultaneously very complex. Since CPS are often multi-functional and dynamic, design engineers are facing an immense number of features ending in an exploding number of possible runtime configurations. System designs of CPS often host safety related features and demand high dependability. Thus, worst-case bounds are generally required. This fact makes it difficult to find approximations, since all runtime configurations have to be covered by the performance-verification. To solve this, different kinds of approaches have been proposed.

One popular approach is to abstract from some dynamics of the system for verifying the system's performance in a rather but not completely static way[8], [9], [10], [11], [12], [13]. The idea is, to manually group a set of system-wide software runtime configurations called modes[14] or scenarios[8], each configuration addressing a different context or situation during the runtime of the system. The performance of every runtime configuration has to be verified individually. Depending on the protocol for the configuration change, transitions have to be verified, too (cf. [15]). While those approaches extended the possibilities and allowed more resource-efficient system designs, their limitations are evident. The manual nature of describing runtime configurations limits the designable variability of the systems and thereby the resource-efficiency of the design.

Another branch of approaches (cf. [16], [13]) tries to avoid the abstractions resulting from system-wide modes. Here, each software component owns a set of modes and thus, those approaches allow more dynamisms than system-wide mode approaches. However, for the performance-verification, all runtime configurations have to be covered. Since the overall runtime configurations are obtained by the cross product of the sets of component-modes, this results in a combinatorially explosion and makes it too complex finding an exact solution. In the approaches presented in [16] and [13], this has been solved through restrictions concerning the network topology and media access mechanisms. Both approaches rely on time division multiple access (TDMA). While these solutions help to reduce the complexity of network performance-verification, they entail significant drawbacks concerning resource-efficiency and bandwidth availability.

Additional works have been published [17], [18], [19], [20], [21] that are mainly based on confining the analyzed dynamics to local components. The presented approaches rely on interfaces between classical performance evaluation frameworks such as Real-Time Calculus[6] and more expressive concepts such as Timed Automata[22], Event Count Automata [23] or Lustre [24]. Through this hybrid approach, the combinatorial explosion can be avoided by the price of neglecting system-wide dynamics. Especially for the case of network performance-verification, this is obviously a significant disadvantage.

In summary, the concepts of system-wide modes, component-wide modes, and hybrid performance-verification imply significant drawbacks and leave a gap open that is worth spending a detailed look at. Targeting this opportunity, we will present an approach integrating variability concerns within the network performance-verification of CPS.

## III. Automotive Video System

Automobiles and their complex e/e-architecture are prominent example systems evolving towards adaptive CPS with real-time requirements. Therewith, the in-vehicle video architecture plays an important part. It consists of several time-critical video functions, which all transfer data from different sources to the head-unit display, which visualizes the data. The following features are generally included in such a system:

- *Human Machine Interface (HMI)* feature generates a configuration menu video stream, which is displayed on the head-unit.
- *Navigation* feature calculates driving routes and generates a visualization for the driver.
- *Storage Movie* feature transfers video content from an entertainment electronic control unit (ECU) to the head-unit.
- *Bluray Movie* feature transfers video content from a Bluray Player to the head-unit.
- *Online Stream* feature transfers video content from an Antenna ECU to the head-unit.
- *Top-View* feature aggregates four video streams from four cameras mounted at different positions of the vehicle. Thereof, a merged video stream is generated which shows the vehicle from a bird's eye view.
- *Nightvision* feature sends a video stream from a nightvision camera to the head-unit.
- *Side-View* feature displays two video streams recorded from cameras at the vehicle's mudguards, thus, providing the driver a view into hardly visible areas, such as the case in crossroads.

All of these features have requirements on the maximum end-to-end transmission delay (cf. Table I) and must be integrated in an adequate network architecture. We will introduce our variability-aware network performance-verification approach by further detailing this example in the following.

## IV. The Integrated Variability Approach

Our approach relies on an analysis and performance model, which we will introduce first. Based on these, we will present our method to calculate safe bounds for the minimum runtime configuration switching transition delay.

### A. Analysis and Performance-Models

The introduced work is based on the model presented in [25], [26] which comprises several layers of abstraction (cf. Figure 1).

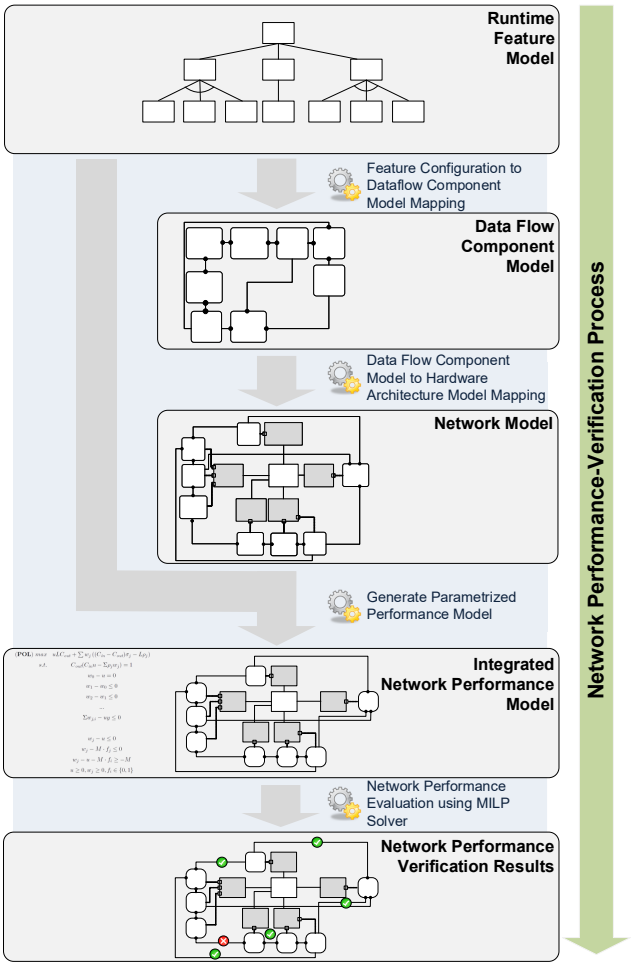


Fig. 1: Overview over the integrated variability approach

*1) Runtime Feature Model:* A runtime feature model describes the dynamics of the system at the abstraction level of features. In general, a feature model represents a system in a tree-like structure, where – originating from a root feature – features are detailed by one or more child-features. The runtime dependencies between features can be modeled by cardinality groups (cf. [27]), i.e., a group of features and cardinalities. The cardinalities characterize in which multiplicities the group of features can occur during runtime. Its formal description is given in Definition 1.

*Definition 1 (Runtime Feature Model):*
A runtime feature model *fm* is a 5-tuple $fm = (F, r, D, \gamma, \Gamma)$. It consists of:

- A set of features $F = \{f_1, ..., f_n\}, n \in \mathbb{N}, n > 0$ and a set of leaf-features $F_{leaf} \subseteq F$.
- A root feature (concept) $r$, where $r$ has no parents. That means $\nexists f \in F : (f, r) \in D$.
- Decomposition Edges $D \subseteq F \times F$, where $D$ forms an acyclic, connected graph, i.e. a tree.
- Cardinality Groups $\Gamma = \{\Gamma_1, ..., \Gamma_n\} \subseteq 2^F \backslash \{\emptyset\}$, $|\Gamma_i| > 1, \forall f, f' \in \Gamma_i : \exists f_p : (f_p, f), (f_p, f') \in D$
- Cardinalities $\gamma(\Gamma_i) = (x, y), \Gamma_i \in \Gamma, x, y \in \mathbb{N}, x \geq 0, y > 0$, where $x$ is the minimum number of features, $y$ is the maximum and $\mathbb{N}$ is the set of natural numbers.

Figure 2 shows an example feature model of the use-case presented in Section III. Here, we have three cardinality groups. The first group consists of four features: *Human Machine Interface (HMI)*, *Navigation*, *Movie* and *Advanced Driver Assistant Systems (ADAS)*. Since the head-unit display

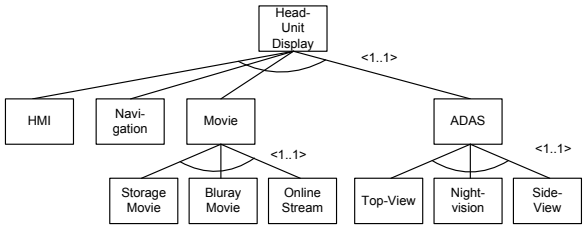


Fig. 2: Runtime Feature-Model of the Automotive Video System

can only show one stream at once, the cardinality of this group is set to 1..1. That is to say, exactly one of these features can be active at the same time. The two remaining feature groups represent refinements of the features *Movie* and *ADAS*. As already mentioned, the *Movie*-feature plays video content from three different sources. These are: *Storage Movie*, *Bluray Movie*, and *Online Stream*. Analogously to the before explained cardinality group, this group allows only exactly one of its features to be present during runtime. This holds also for the last feature group, which details the *ADAS*-feature to three further features: *Top-View*-feature, the *Nightvision*-feature, and the *Side-View*-feature.

*2) Data Flow Component Model:* The data flow component model describes the software architecture, a detailed representation of the data dependencies' traffic characteristics and non-functional requirements. Those are mainly described by so-called *traffic envelopes*, which are worst-case estimations for the data sending behavior of the software components. In detail, this is the maximum burst size $\sigma$ (i.e., the amount of data a component sends at once), the average used bandwidth $\rho$, the maximum data frame size $L_{max}$ on data link layer (DLL), and the application frame size $frame$. The value of $\hat{\delta}$ describes the maximum tolerable end-to-end transmission delay for the relevant data dependency on application level.

*Definition 2 (Data Flow Component Model):*
A data flow component model cm is a triple $cm = (C, R_{dd}, \pi_{data})$. It consists of:

- A set of components $C = \{c_1, ..., c_n\}$.
- Unidirectional data dependencies $R_{dd} \subseteq C \times C$.
- Traffic characteristics and non-functional requirements of data dependencies $\pi_{data} : R_{dd} \rightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}$ $\pi_{data}(rd) = (\sigma, \rho, L_{max}, frame, \hat{\delta})$, where $\mathbb{N}$ is the set of natural numbers and $\mathbb{R}$ is the set of real numbers.

Figure 3 shows the corresponding data flow component model for the use-case introduced in Section III, where data dependencies are represented by lines with a dot-end. The

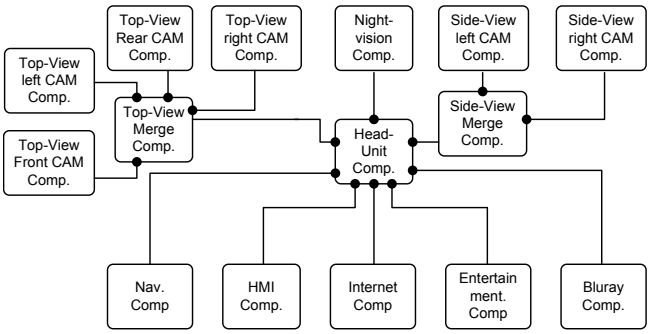


Fig. 3: Software Component Model of the Automotive Video System

latter indicates the receiving component and thereby, the direction of the data flow. The respective traffic characteristics and end-to-end delay requirements are shown in Table I. The first column shows the source and target component of the corresponding data dependency. The following columns include the average bandwidth, the maximum application frame size, the maximum burst data size, the maximum frame size on data link layer, and the maximum acceptable end-to-end delay on application layer (cf. [28], [29], [30], [31], [32]).

| Source- and target-component | Avg. bandw. [Mbits] | Max. size appl.-frame [octet] | Max. Burst [octet] | $L_{max}$ [octet] | Max. delay [ms] |
|---|---|---|---|---|---|
| *SV Merge, SV Right CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *SV Merge, SV Left CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *TV Merge, TV Right CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *TV Merge, TV Left CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *TV Merge, TV Front CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *TV Merge, TV Rear CAM* | 6.1848 | 25,770 | 25,770 | 1526 | 33 |
| *Head-Unit, Nightvision* | 6.1848 | 25,770 | 25770 | 1526 | 33 |
| *Head-Unit, Internet* | 25.7810 | 107,421 | 107,421 | 1526 | 100 |
| *Head-Unit, HMI* | 164.9762 | 687,401 | 687,401 | 1526 | 100 |
| *Head-Unit, Entertainment* | 164.9762 | 687,401 | 687,401 | 1526 | 100 |
| *Head-Unit, Navigation* | 164.9762 | 687,401 | 687,401 | 1526 | 100 |
| *Head-Unit, Bluray* | 164.9762 | 687,401 | 687,401 | 1526 | 100 |

TABLE I: Data dependecies and end-to-end delay requirements of the case study

*3) Feature-to-Component Mapping:* The relation between the feature model and the component model is formalized through a feature-to-component mapping $\mu_{F\times C} : F_{leaf} \rightarrow 2^C$ (cf. Definition 3). Thus, a leaf-feature $f$ is mapped to a set of components $C_f \subseteq C$. In this case, the mappings are disjunct (cf. Equation 2). Furthermore, data dependencies connect solely components within one mapping (cf. Equation 3). These requirements reduce the complexity of the analysis and performance model, as they will enable some useful properties, which will be described in Section IV-A5 in Definition 6 and 7. Although, those formal prerequisites are not fulfilled by runtime feature models and their mappings in general, they can be achieved through component substitutions. This means, whenever there exist overlapped components, those components have to be duplicated and replaced according to the number of overlapping features. The newly generated duplicates can then be assigned to the corresponding feature. If the situation occurs, that there is at least one data dependency between mappings, two steps have to be conducted. First, it has to be determined, for which exact features this data dependency is vital. Afterwards, depending on the concrete mapping, duplicates may have to be created and assigned to the corresponding features. As described in Equation 4, there is no component in the system, which is not represented by a feature.

*Definition 3 (Feature-to-Component Mapping):*
A feature-to-component mapping is a map $\mu_{F\times C}$, mapping from features to components. It is defined as follows:

$$\mu_{F\times C} : F_{leaf} \rightarrow 2^C \quad (1)$$

$$\mu_{F\times C}(f) = C_f, f \in F_{leaf}, C_f \in 2^C, \quad (2)$$

$$\forall f, f' \in F_{leaf}, f \neq f' : \mu_{F\times C}(f) \cap \mu_{F\times C}(f') = \emptyset$$

$$\forall dd \in R_{dd}, dd = (c, c') : \exists f \in F_{leaf}, c, c' \in \mu_{F\times C}(f) \quad (3)$$

$$\bigcup_{f\in F_{leaf}} \mu_{F\times C}(f) = C \quad (4)$$

| Feature | Head-Unit | Top-View Merge | Top-View Right CAM | Top-View Left CAM | Top-View Front CAM | Top-View Rear CAM | Nightvision | Side-View Merge | Side-View Right CAM | Side-View Left CAM | HMI | Navigation | Entertainment | Bluray | Internet |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Component | | | | | | | | | | | | | | |
| *HMI* | x | - | - | - | - | - | - | - | - | - | x | - | - | - | - |
| *Navigation* | x | - | - | - | - | - | - | - | - | - | - | x | - | - | - |
| *Storage Movie* | x | - | - | - | - | - | - | - | - | - | - | - | x | - | - |
| *BluRay Movie* | x | - | - | - | - | - | - | - | - | - | - | - | - | x | - |
| *Online Stream* | x | - | - | - | - | - | - | - | - | - | - | - | - | - | x |
| *Top-View* | x | x | x | x | x | x | - | - | - | - | - | - | - | - | - |
| *Nightvision* | x | - | - | - | - | - | x | - | - | - | - | - | - | - | - |
| *Side-View* | x | - | - | - | - | - | - | x | x | x | - | - | - | - | |

TABLE II: feature-to-component mapping of the Automotive Video System

Table II shows the mapping for the case study introduced in Figure 2. Every leaf-feature is mapped onto a set of components. The mapping is indicated by an 'x'. As we can see, the head-unit component is included in each features' mapping. In particular, each runtime configuration contains this component. Here, we can achieve the properties formulated by Equations

2-3 by replacing the head-unit component by a substitute for each overlapping feature, as described in the former paragraph. For most of the features, the mapping includes, besides the head-unit component, one component that sends video content to the head-unit component. Exceptions are the *Top-View* and the *Side-View*-features. These map to a larger set of components. The *Top-View* feature for example, maps onto six components: four components responsible for generating and transmitting video data, one component (top-view merge) in charge of receiving and merging these video pictures as well as sending the resulting merged pictures to the head-unit component.

*4) Network Model:* The network architecture and the allocations of software components and data dependencies are represented in the network model. It includes descriptions of computing nodes, switches, physical communication links, mappings from software components to computing nodes, as well as allocations of data paths. The formal descriptions are given through Definition 4 and 5.

*Definition 4 (Network Model):*
A network model $nt$ is a four-tuple $nt = (N, IC, \mu_C, \mu_R)$. It is defined as follows:

- Nodes $N = N_c \cup N_{sw}$,
  - Computing Units $N_c = \{nc_1, ..., nc_n\}$,
  - Network Switches $N_{sw} = \{nsw_1, ..., nsw_m\}$,
- Interconnections $IC = \{ic_1, ..., ic_l\}$ between nodes, $IC \subseteq N \times N \times \mathbb{N}, ic_i = (s_i, e_i, ls_i), ic_i \in IC, s_i, e_i \in N$, where $s_i$ is the start node, $e_i$ is the end node and $ls_i$ is the link speed,
- A mapping $\mu_C$ from components to computing nodes as defined in Definition 5,
- A mapping $\mu_R$ from data dependencies onto interconnections as defined in Definition 5.

The allocation of software components to computing nodes and data dependencies to interconnections is described in the *component-to-hardware mapping*. It is assumed without loss of generality, that (i) one component is mapped to exactly one computing node (cf. Equation 7) and (ii) paths of data dependencies are acyclic in general. Whenever a data dependency $dd$ connects any two components which are allocated to the same computing node, the set assigned to $\mu_R(dd)$ is the empty set (cf. Equation 9).

*Definition 5 (Component-to-Hardware Mapping):*
A Component-to-Hardware Mapping $cthw$ is a tuple $cthw = (\mu_C, \mu_R)$ . It is defined as follows:

- A mapping $\mu_C$ from components to computing nodes

$$\mu_C : C \to N_c \tag{5}$$
$$\mu_C(c) = n, c \in C, n \in N_c, \tag{6}$$
$$\exists (c, n) \in \mu_C \to \forall (c, n') \in \mu_C : n = n' \tag{7}$$

- A mapping $\mu_R$ from data dependencies onto interconnections

$$\mu_R : R_{dd} \to 2^{IC} \tag{8}$$
$$\mu_R((c, c')) = \begin{cases} \{ic_{n-1}, ..., ic_0\} & \text{if } \mu_C(c) \neq \mu_C(c') \\ \{\emptyset\} & else, \end{cases} \quad (c, c') \in R_{dd}, ic_i \in IC \tag{9}$$

Figure 4 shows the network model of the automotive example introduced in Figure 2. The core of depicted network architecture is build upon two interconnected switches, which are in turn connected to the computing nodes of the architecture. All physical communication links are 1000 Mbits full-duplex Ethernet links. The allocation of software components is shown as dotted line with a square-end. Since the data paths of different data dependencies are unambiguous (only one path between two computing nodes exists), data path allocations have been omitted in this visualization.

*5) Performance Model:* The performance of the system is modeled by an integrated network performance model, which allows both: the description of performance and possible runtime variability space. The latter denotes the set of all valid runtime configurations. Equations 15 - 20 formalize the main part of the integrated network performance model, which is basically a parametrized network calculus model (cf. [5]). It specifies – depending on the chosen feature activation – the maximum delay a bit can experience in a queue of an egress port of an Ethernet switch. Interpreted as a maximization problem and after a linearization-transformation (cf. [33]), this model can be solved by a *mixed-integer linear program* solver.

Before a detailed description of Equations 15 - 20 is given, some additional mappings are introduced. First, a mapping between features and data dependencies is defined, with Definition 6. Based on this, the inverse mapping of data dependencies to features is defined.

*Definition 6 (Feature-to-Data-Dependency Mapping):*
A Feature-to-Data-Dependency mapping is a map $\mu_{F \times R_{dd}}$, mapping features to data dependencies. It is defined by:

$$\mu_{F \times R_{dd}} : F_{leaf} \to R_{dd} \tag{10}$$
$$\mu_{F \times R_{dd}} = \{(f, dd) | f \in F_{leaf}, dd \in R_{dd}, \tag{11}$$
$$dd = (c, c'), c, c' \in \mu_{F \times C}(f)\} \tag{12}$$

*Definition 7 (Data-Dependency-to-Feature Mapping):*
A Data-Dependency-to-Feature mapping is a map $\mu^{-1}_{F \times R_{dd}}$, mapping data dependencies to features. It is defined by:

$$\mu^{-1}_{F \times R_{dd}} : R_{dd} \to F_{leaf} \tag{13}$$
$$\mu^{-1}_{F \times R_{dd}} = \{(dd, f) | \exists (f, dd) \in \mu_{F \times R_{dd}}\} \tag{14}$$

It can be shown that $\mu^{-1}_{F \times R_{dd}}$ is left-total and right-unique. These two properties are very useful, since they guarantee that for each data dependency there is exactly one feature assigned. This makes the formulation of the maximization problem *POLF* (cf. Equations 15 - 20) less complex, since a unique mapping from data dependencies to features enables an assignment from traffic characteristics to features. Thus, one

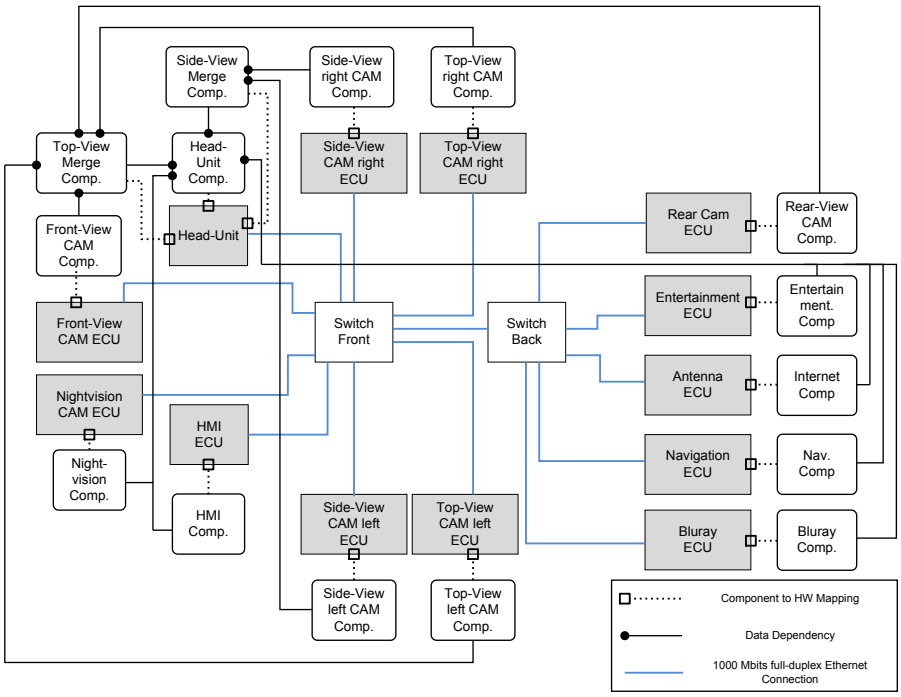


Fig. 4: Network Model of the Automotive Video System

can formulate a parametrized performance model as stated in Equation 15-20.

The input parameters of the maximization problem *POLF* (Equation 15-20) can be obtained from the feature model, from the network model and from the component model. Basically, the values of *POLF* depend on the selected feature-activation $\phi_1, ..., \phi_n, \phi_i \in \{0, 1\}$, where $n$ stands for the overall number of features and value 1 for the activation and 0 for the deactivation of a feature. The de- or activation of a feature influences the values of the sums $\Sigma\phi_j\sigma^r_{jkl}$ respectively $\Sigma\phi_j\rho_{jk}$, where $\sigma^r_{jkl}$ stands for the relaxed burst value and $\rho_{jk}$ for the average bandwidth of a data dependency $dd_{jk}$. The structure of the feature model is given by the constraints of the maximization problem (cf. equations 16–20).

Equation 16 and 17 ensure that the root feature respectively the feature of the considered data dependency are activated. Equation 18 guarantees that a feature can only be activated when its parent feature is activated and Equation 19 assures that maximum multiplicities of the cardinality groups are met. The parameters $C_{in}$ and $C_{out}$ represent the speed of the incoming ports and the speed of the analyzed egress port, where $(C_{in} - \Sigma\phi_j\rho_j > 0)$ and $(C_{out} - \Sigma\phi_j\rho_j > 0)$ have to hold. Since the value of $C_{in}$ would depend on the concrete feature activation, an over-approximation $C^r_{in}$ is used. The variable $\bar{L}$ stands for the sum of the maximum sizes of data link layer frames from the incoming ports. The exact value of $\bar{L}$ would depend on the activated features and that in turn would imply non-linear terms. In this case, we over-approximate this parameter with $\bar{L}^r$, which stands for the maximum possible value of $\bar{L}$ (cf. [25]).

$$\textbf{(POLF)} \quad \max \quad \frac{(C^r_{in} - C_{out})\Sigma\phi_j\sigma^r_j + \bar{L}^r(C_{out} - \Sigma\phi_j\rho_j)}{C_{out}(C^r_{in} - \Sigma\phi_j\rho_j)} \tag{15}$$

$$s.t. \quad \phi_r = 1, \tag{16}$$

$$\nexists(f', f_r) \in D$$

$$\phi_{active} = 1, \tag{17}$$

$$\mu^{-1}_{F \times R_{dd}}(dd_{active}) = f_{active}$$

$$\phi_i - \phi_{pi} \leq 0, \tag{18}$$

$$(f_{pi}, f_i) \in D$$

$$\sum \phi_{j,i} - y \leq 0, \tag{19}$$

$$f_{j_1}, ..., f_{j_n} \in \Gamma_j, \ \gamma(\Gamma_j) = (x, y)$$

$$\phi_j \in \{0, 1\} \tag{20}$$

The solution for the *integer linear problem (ILP)* stated in Equation 21-27 represents an over-approximation $C^r_{in}$ for the $C_{in}$-parameter of *POLF*. In contrast to the heuristic presented in [25], this new heuristic obviously delivers a tighter over-approximation, since it determines the maximum $C_{in}$-value including all relevant runtime configurations. The heuristic in [25] assumed in contrast only the theoretical maximum value for $C_{in}$, no matter if that value can be reached in any relevant runtime configuration.

The objective function (Equation 21) of *PCIN* consists of the sum of the link speeds of incoming links, that have a data dependency crossing the incoming as well as the outgoing link. The link speed is only considered when the relevant incoming link is activated. This is the case, if at least one data dependency is activated on that link. The respective constraint

is modeled by Equation 22. The activation of relevant links $ic^i_{in}$ is represented by a variable $\lambda^i_{in} \in \{0,1\}$, where value 1 stands for the activation and 0 for the deactivation. The constraints implied through the structure of the feature model (Equations 23 – 27) are consistant to those of *POLF*.

$$(\textbf{PCIN}) \max \quad \sum \lambda^i_{in} \cdot ic^i_{in}.ls \tag{21}$$

$$s.t. \quad \lambda^i_{in} - \sum \phi_{ic^i_{in}} \leq 0 \tag{22}$$

$$\phi_r = 1, \tag{23}$$
$$\nexists (f', f_r) \in D$$

$$\phi_{active} = 1, \tag{24}$$
$$\mu^{-1}_{F \times R_{dd}}(dd_{active}) = f_{active}$$

$$\phi_i - \phi_{pi} \leq 0, \tag{25}$$
$$(f_{pi}, f_i) \in D$$

$$\sum \phi_{j,i} - y \leq 0, \tag{26}$$
$$f_{j_1}, ..., f_{j_n} \in \Gamma_j,\ \gamma(\Gamma_j) = (x, y)$$

$$\lambda^i_{in} \in \{0,1\}, \phi_{ic^i_{in}} \in \{0,1\}, \phi_j \in \{0,1\} \tag{27}$$

Based on this formalization, an end-to-end performance-verification can be pursued. Equation 28 describes how to compute a safe bound for the end-to-end delay. The first term stands for the transmission time of the application frame, whereas the second term represents the store-and-forward delays, and the third term sums up delays in the queues of the switch egress ports along the path. The store-and-forward delay $\hat{d}_{sf}$ can be calculated by dividing the maximum frame size through the link speed of the corresponding data link. Delays in the queues of the switch egress port $\hat{d}^r_{pv}$ can thereupon be obtained by solving the maximization problem *POLF*.

$$\begin{aligned} \hat{d}^r_{e2epv}(dd) = \quad & \tfrac{\mu_{data}(dd).frame}{ic_0.ls} \\ & + \sum_{i=0}^{n-1} \hat{d}_{sf}(dd, ic_i) + \sum_{i=1}^{n-1} \hat{d}^r_{pv}(ic_i), \\ & PATH(dd) = (ic_{n-1}, ..., ic_0) \\ & \phi_{active} = 1, \mu^{-1}_{F \times R_{dd}}(dd) = f_{active} \end{aligned} \tag{28}$$

### B. Configuration Switching Analysis

The *configuration change protocol* (or mode change protocol, cf. [34]) defines the process and conditions under which configuration changes have to be conducted. Generally, a configuration change implies that the system has a defined starting configuration $S_a$ at the time $t_0$ when a configuration change to a new configuration $S_z$ is requested. At that point, features can be categorized into three disjunct sets: $F_{az}$ containing features belonging to $S_a$ and $S_z$; $F_a$ including features only belonging to the old configuration $S_a$ and not to $S_z$; and $F_z$ holding solely features from $S_z$ which do not belong to $S_a$. According to [34] configuration change protocols mainly differ in the way they handle feature modifications and their respective tasks or components that belong to $F_a$. There are three possibilities: (i) deactivating those features immediately, (ii) letting those features complete their task and deactivating them afterwards, and (iii) deactivating only partially features of $F_a$ immediately. In the work at hand, the assumption is taken, that all old features are deactivated immediately after a configuration change request.

Figure 5 gives an overview of such a configuration switch at the time $t_0$, when the configuration change was requested and the old features with the corresponding data dependencies are deactivated. The bottom of the figure shows the status in one of the switches at this time. In this excerpt, the queue of the switch output port to the head-unit is still occupied with Ethernet frames belonging to features of the old configuration $S_a$.

Since the performance models generally assume empty queues, it is essential for ensuring the dependability of adaptive CPS to define a minimum transition time $t_{\Delta_{S_a \to S_z}}$, which is needed to bring the network back to a defined state. That is, $t_{\Delta_{S_a \to S_z}}$ time has to elapse at minimum, before the features belonging to $F_z$ can be activated. During this time, all frames belonging to $F_a$ have to be vanished from the network. For a single data dependency, this worst-case time duration can be calculated as stated in Equation 29. This situation occurs, when a configuration change is requested during the transmission of an Ethernet frame corresponding to a data dependency which has to be deactivated. In that case, the frame has to traverse its complete path to the target network node. Analogously to Equation 28, the delay in the queues in the switch egress port $\hat{d}^r_{pv}(ic_i)$ can be obtained by solving *POLF*.

$$\begin{aligned} t^r_{\Delta_{dd}} = \quad & \sum_{i=0}^{n-1} \hat{d}_{sf}(ic_i) + \sum_{i=1}^{n-1} \hat{d}^r_{pv}(ic_i), \\ & PATH(dd) = (ic_{n-1}, ..., ic_0) \end{aligned} \tag{29}$$

For the entire set $F_a$, the minimum transition duration is the maximum value over all relevant data dependencies as stated in Equation 30.

$$t^r_{\Delta_{S_a \to S_z}} = \max_{\mu^{-1}_{F \times R_{dd}}(dd) \in F_a} t^r_{\Delta_{dd}} \tag{30}$$

With this Equation, we are able to calculate the safe minimum transition time for an arbitrary configuration change $S_a \to S_z$ to guarantee a safe configuration switching. The $t^r_{\Delta_{dd}}$-bound holds for all runtime configurations, since $\hat{d}^r_{pv}(ic_i)$ is the maximum delay over all runtime configurations. This offers significant advantages: Firstly, the calculation scales even for large configuration spaces (cf. [25]). Secondly, since we do not need a bound for every single transition, the memory need is reduced dramatically.

## V. Evaluation

In order to evaluate our approach and to analyze the magnitude of those minimum configuration switching times, we applied it to the automotive video system introduced in Section III. The results of the comprehensive evaluation are shown in Table III. In the second and third column, the minimum configuration switching transition times are listed, where the second column contains the results using the heuristic of [25], and the third column holds the results using the heuristic presented in Section IV-A5 (cf. Equations 21 – 27). It can be derived directly that the new heuristic clearly outperforms the one from

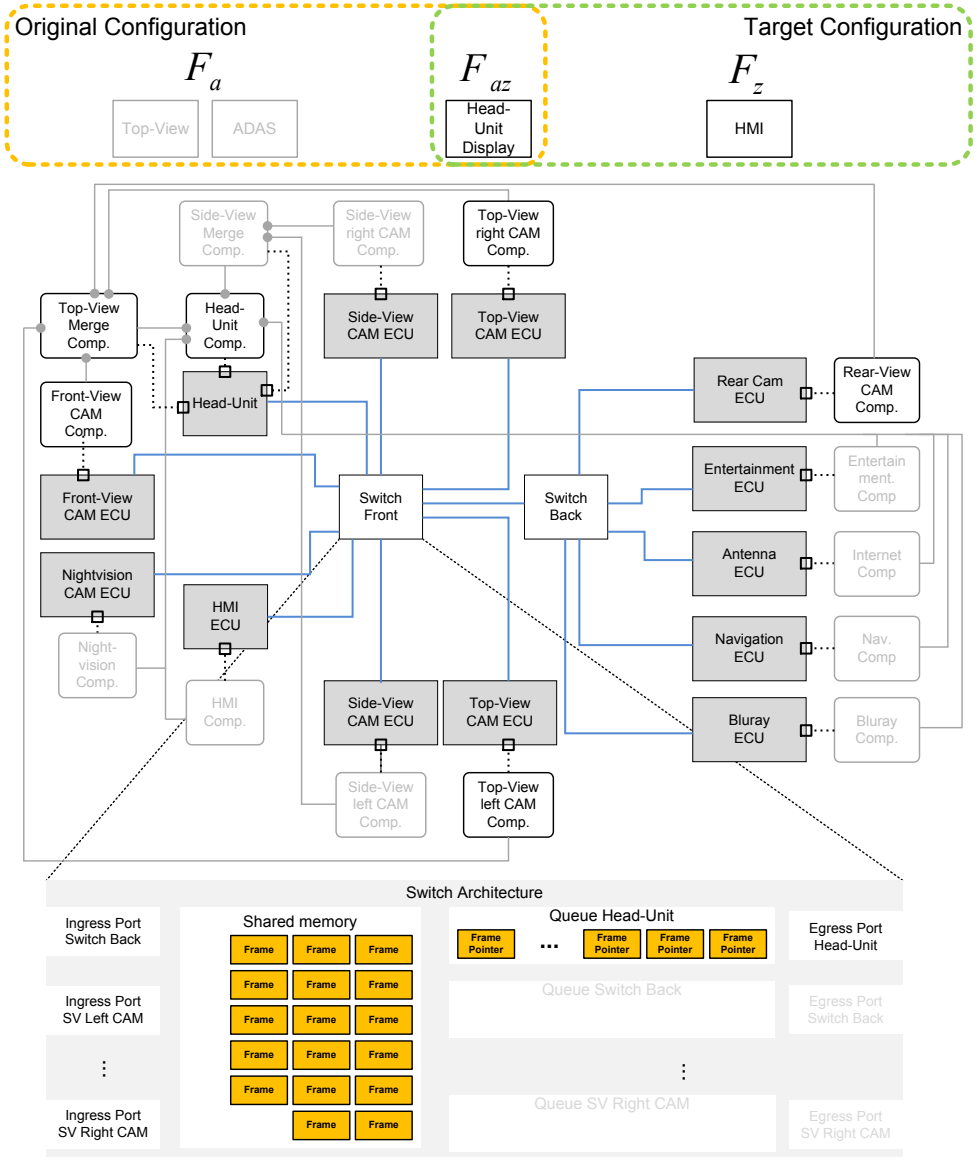


Fig. 5: Illustration of the network architecture within a configuration switch of the Automotive Video System

[25]. The average improvement of the new heuristic lies at approximately 59 percent. Besides the achieved improvement, the values calculated by the new heuristic are all in the order of microseconds, thus, in direct comparison they are very low.

The overall minimum configuration switching time $t^r_{\Delta_{S_a \to S_z}}$ is obtained through the maximum $t^r_{\Delta_{dd}}$ value. Hence, the maximum possible $t^r_{\Delta_{S_a \to S_z}}$-value for this case study is 683 microseconds. Compared to the magnitude of obtained end-to-end latencies, these values are very low. For example, when turning off the *Top-View*-feature and activating the *HMI*-feature, it takes at least[1] $0.683ms + 5.524ms = 6.207ms$ until the first video data of the *HMI* can be displayed (i.e., the minimum transition time takes approximately 11 percent of the overall user-visible feature activation delay).

To summarize the presented findings, one can say, that these configuration switching bounds do not increase the user-visible activation delay of features significantly. However, since they assure the basic assumptions of the performance model, they are clearly not negligible and therefore have to be considered as well in the design of dependable systems.

Besides the minimum configuration switching transition time, the maximum end-to-end latency (cf. Equation 28) has been evaluated. The third, fourth, and fifth column contain the results of a static performance-verification as well as for one using the heuristic from [25], and the novel approach of this work at hand. For the old and new heuristics, we can observe that the delays of all data dependencies are

[1]On application level, there would be additional delays, such as video-decoder delays, but this is the absolute minimum from a network's point of view.

| Data Dependency | Minimum conf. switch transition time $t^r_{\Delta_{dd}}$[ms] old $C_{in}$-heuristic | new $C_{in}$-heuristic | End-to-End Latency $\hat{d}^r_{e2epv}$[ms] static verification | old $C_{in}$-heuristic | new $C_{in}$-heuristic |
|---|---|---|---|---|---|
| *Head-Unit, HMI* | 4.951 | 0.024 | 37.045 | 10.45 | 5.524 |
| *Head-Unit, Navigation* | 10.21 | 0.037 | 52.796 | 15.709 | 5.536 |
| *Head-Unit, Entertainment* | 10.21 | 0.037 | 52.796 | 15.709 | 5.536 |
| *Head-Unit, Bluray* | 10.21 | 0.037 | 52.796 | 15.709 | 5.536 |
| *Head-Unit, Internet* | 1.523 | 0.037 | 48.156 | 2.382 | 0.896 |
| *Head-Unit, Nightvision* | 0.217 | 0.024 | 31.752 | 0.423 | 0.231 |
| *TV Merge, TV Right CAM* | 0.761 | 0.671 | 31.752 | 0.968 | 0.877 |
| *TV Merge, TV Left CAM* | 0.761 | 0.671 | 31.752 | 0.968 | 0.877 |
| *TV Merge, TV Front CAM* | 0.761 | 0.671 | 31.752 | 0.968 | 0.877 |
| *TV Merge, TV Rear CAM* | 0.951 | 0.683 | 47.503 | 1.157 | 0.889 |
| *SV Merge, SV Right CAM* | 0.398 | 0.280 | 31.752 | 0.604 | 0.487 |
| *SV Merge, SV Left CAM* | 0.398 | 0.280 | 31.752 | 0.604 | 0.487 |

TABLE III: Minimum configuration switching transition times and end-to-end latency values of the case study

significantly below their requirements (cf. Table I). In contrast to these approaches, the static performance-verification fails for the data dependency between the *Top-View Merge*- and the *Top-View Rear CAM*-component. The calculated value lies approximately 46 percent above its requirement and thus, the network architecture would be considered to fail guaranteeing dependable communication. For the new heuristic, the average improvement – compared to the static verification – lies by 95 percent.

While in the scope of CPS the question of scalability is decisive, we conducted additional experiments, which where executed on a 64-bit machine with two octa-core Intel Xeon E5-2660 and 128 GB RAM running a Debian 64-bit Linux. The approach is implemented in C using lp_solve[35] 5.5 and the calculations where distributed over 16 threads. Typical modern automotive e/e-architectures of different sizes (cf. [25] for details on the test-setups) were evaluated. In line with present and expected system sizes (cf. [36], [37]), the system architectures vary from the smallest setup with 50 features, 400 software components, 20 electronic control units, 12 Ethernet switches and one runtime configuration to the largest with 500 features, 4000 software components, 200 electronic control units, 143 Ethernet switches and $1,1 \times 10^{15}$ runtime configurations (cf. Table IV). The traffic characteristics have been assigned according to the values and distributions shown in Table V (cf. [28], [32], [29], [30]). The specific assignments have been varied randomly within a range of 20 percent. The transmission speed of the communication links is set to the smallest possible value that results in an maximum utilization smaller than 60 percent, where the speed is at least 10 Mbits and only powers of ten are permitted. For each test-setup and for each $C_{in}$-heuristic 40 test-runs have been conducted. Figure 6 depicts how our approach scales on these test-setups. Both heuristics show comparable results, where the old heuristic performs slightly faster in average. Even for the largest test-setup the new heuristic took only 28 minutes in average to verify the end-to-end delays. Since the term of Equation 29 is also contained in Equation 28, we obtain $t^r_{\Delta_{dd}}$ while computing $\hat{d}^r_{e2epv}$. Thus, the results shown in Figure 6 hold for both values.

The evaluations demonstrate the advantages of our introduced performance-verification approach addressing variability in an integrated way. The novel heuristic performs significantly better than the static or preliminary approach. Moreover, we could highlight that our work scales well and may also be applied for analyzing dependable communication behavior of upcoming larger CPS.

| Scenario ID | #Leaf-Features | Number of Runtime Configurations ±1% over-all | Number of Runtime Configurations ±1% ∅ per Port | #Components | #ECUs | #Switches |
|---|---|---|---|---|---|---|
| SK-1 | 50 | 32 | 1 | 400 | 20 | 12 |
| SK-2 | 100 | $1,0 \times 10^3$ | 10 | 800 | 40 | 22 |
| SK-3 | 150 | $3,3 \times 10^4$ | $3,3 \times 10^2$ | 1200 | 60 | 42 |
| SK-4 | 200 | $1,0 \times 10^6$ | $1,0 \times 10^4$ | 1600 | 80 | 49 |
| SK-5 | 250 | $3,4 \times 10^7$ | $3,4 \times 10^5$ | 2000 | 100 | 67 |
| SK-6 | 300 | $1,1 \times 10^9$ | $1,1 \times 10^7$ | 2400 | 120 | 77 |
| SK-7 | 350 | $3,4 \times 10^{10}$ | $3,4 \times 10^8$ | 2800 | 140 | 91 |
| SK-8 | 400 | $1,1 \times 10^{12}$ | $1,1 \times 10^{10}$ | 3200 | 160 | 110 |
| SK-9 | 500 | $1,1 \times 10^{15}$ | $1,1 \times 10^{13}$ | 4000 | 200 | 143 |

TABLE IV: Test-setups to evaluate the scalability of our approach (cf. [25])

| Category | Bandwidth $\rho$ [Mbit/s] | Appl.-Frame [Octet] | Burst $\sigma$ [Octet] | $L_{max}$ [Octet] | Dist. [%] |
|---|---|---|---|---|---|
| ADAS video | 6.1848 | 25,770 | 25,770 | 1,526 | 30 |
| Control data | 0.0576 | 72 | 7,200 | 72 | 20 |
| Navigation data | 1.65056 | 20,632 | 20,632 | 1,526 | 10 |
| Stereo audio | 1.5072 | 942 | 942 | 942 | 10 |
| Multi-channel audio | 3.814 | 2,384 | 2,384 | 942 | 10 |
| Ultra High Definition Video (h265 5 Main tier) | 25.7810 | 107,421 | 107,421 | 1,526 | 10 |
| Ultra High Definition Video (h265 5.1 High tier) | 164.9762 | 687,401 | 687,401 | 1,526 | 10 |

TABLE V: Traffic characteristics and their distribution for test-setups SK-1 – SK-9

## VI. Conclusions

For future CPS, network performance-verification in variable scenarios is essential to still meet real-time requirements.

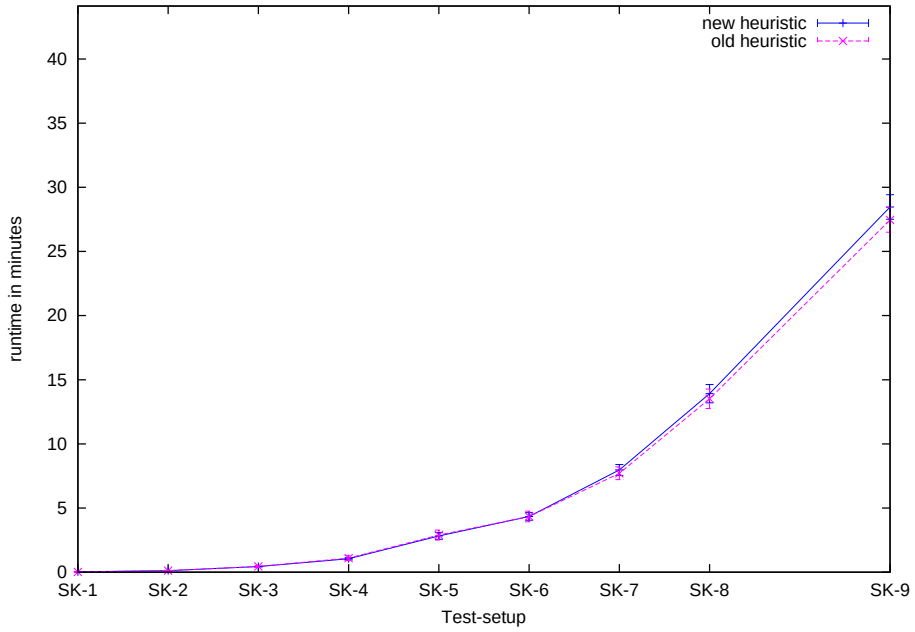


Fig. 6: Average runtime over 40 test-runs of our approach using the new and the old $C_{in}$-heuristic for test setups SK-1 – SK-9 with 95%-confidence Intervals

In the work at hand, we present an approach enabling the network performance-verification of adaptive Ethernet-based CPS. Specifically, we enhance heuristics for superior analyses and present means to determine the minimum transition time bound for configuration switching at runtime. The approach does not rely on the manual definition of modes. Thus, the describable runtime variability becomes more fine-granular and thereby, the potential savings in network resources can be increased. Furthermore, the approach does not imply any artificial architectural restrictions to the network topology, such as assuming time-driven network access. We additionally highlight by an automotive case study that an average improvement of 95 percent can be achieved in comparison to state-of-the-art static analysis. The scalability of the approach has been validated through several test-setups with varying complexity. In these experiments, even the performance of a rather large, dynamic, and complex system with thousands of components and hundreds of network nodes could be verified within reasonable time of half an hour.

## ACKNOWLEDGMENTS

This work has been partially funded by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology.

## REFERENCES

[1] 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.

[2] Jonas Rox and Rolf Ernst, "Formal Timing Analysis of Full Duplex Switched Based Ethernet Network Architectures," in *SAE World Congress*, ser. System Level Architecture Design Tools and Methods (AE318). Detroit, MI, USA: SAE International, 2010.

[3] 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 *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 25:1–25:10.

[4] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr, "Towards Self-Adaptation in Real-Time, Networked Systems: Efficient Solving of System Constraints for Automotive Embedded Systems," in *Proceedings of the 2011 IEEE Fifth International Conference on Self-Adaptive and Self-Organizing Systems*, ser. SASO '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 79–88.

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

[6] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert, "Embedded Software in Network Processors – Models and Algorithms," in *Embedded Software: First International Workshop, EMSOFT 2001 Tahoe City, CA, USA, October 8–10, 2001 Proceedings*, T. A. Henzinger and C. M. Kirsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 416–434.

[7] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," *IEEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[8] R. Henia and R. Ernst, "Scenario Aware Analysis for Complex Event Models and Distributed Systems," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, ser. RTSS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–180.

[9] M. Negrean, M. Neukirchner, S. Stein, S. Schliecker, and R. Ernst, "Bounding mode change transition latencies for multi-mode real-time distributed applications," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, 2011, pp. 1–10.

[10] Mircea Negrean, Rolf Ernst, and Simon Schliecker, "Mastering Timing Challenges for the Design of Multi-Mode Applications on Multi-Core Real-Time Embedded Systems," in *6th International Congress on Embedded Real-Time Software and Systems (ERTS)*, Toulouse, France, 2012.

[11] M. Negrean, S. Klawitter, and R. Ernst, "Timing Analysis of Multi-mode Applications on AUTOSAR Conform Multi-core Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 302–307.

[12] X. Jin, N. Guan, J. Wang, and P. Zeng, "Analyzing Multimode Wireless Sensor Networks Using the Network Calculus," *Journal of Sensors*, 2015.

[13] E. Farcas, "Scheduling multi-mode real-time distributed components," Dissertation, University of Salzburg, 2006.

[14] Phan, Linh T. X., S. Chakraborty, and P. S. Thiagarajan, "A Multi-mode Real-Time Calculus," in *Proceedings of the 2008 Real-Time Systems Symposium*, ser. RTSS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 59–69.

[15] Nikolay Stoimenov and Lothar Thiele, "Poster Abstract: Interface-based design approach for analysis of mode changes in distributed real-time systems," in *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium: Proceedings of Demos and Posters*, San Francisco, USA, 2009.

[16] A. Azim, G. Carvajal, R. Pellizzoni, and S. Fischmeister, "Generation of Communication Schedules for Multi-mode Distributed Real-time Applications," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. Leuven, Belgium: European Design and Automation Association, 2014, pp. 293:1–293:6.

[17] Phan, Linh T. X., S. Chakraborty, P. S. Thiagarajan, and L. Thiele, "Composing Functional and State-Based Performance Models for Analyzing Heterogeneous Real-Time Systems," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, ser. RTSS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 343–352.

[18] K. Lampka, S. Perathoner, and L. Thiele, "Analytic Real-time Analysis and Timed Automata: A Hybrid Method for Analyzing Embedded Real-time Systems," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 107–116.

[19] S. Perathoner, "Modular performance analysis of embedded real-time systems: improving modeling scope and accuracy," Dissertation, ETH Z"urich, 2011.

[20] K. Altisen, Y. Lui, and M. Moy, "Performance Evaluation of Components Using a Granularity-based Interface Between Real-Time Calculus and Timed Automata," in *8th Workshop on Quantitative Aspects of Programming Languages*, Paphos, Cyprus, 2010.

[21] K. Altisen and M. Moy, "ac2lus: Bringing SMT-Solving and Abstract Interpretation Techniques to Real-Time Calculus Through the Synchronous Language Lustre," in *Proceedings of the 2010 22Nd Euromicro Conference on Real-Time Systems*, ser. ECRTS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 207–216.

[22] Rajeev Alur and David L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[23] S. Chakraborty, Phan, Linh T. X., and P. S. Thiagarajan, "Event Count Automata: A State-Based Model for Stream Processing Systems," in *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, ser. RTSS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 87–98.

[24] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.

[25] M. Manderscheid, G. Weiss, and R. Knorr, "Verifying Network Performance of Cyber-physical Systems with Multiple Runtime Configurations," in *Proceedings of the 12th International Conference on Embedded Software*, ser. EMSOFT '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 247–255.

[26] M. Manderscheid and C. Prehofer, "Network Performance Evaluation for Distributed Embedded Systems Using Feature Models," in *Proceedings of the 2013 18th International Conference on Engineering of Complex Computer Systems*, ser. ICECCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 46–55.

[27] 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.

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

[29] M. Manderscheid and F. Langer, "Network Calculus for the Validation of Automotive Ethernet In-vehicle Network Configurations," in *Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, ser. CYBERC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 206–211.

[30] M. Wien, "Profiles, Tiers, and Levels," in *High Efficiency Video Coding: Coding Tools and Specification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 283–289.

[31] Mehrnoush Rahmani, Rainer Steffen, Ktawut Tappayuthpijarn, Eckehard Steinbach, and Giuseppe Giordano, "Performance Analysis of Different Network Topologies for In-Vehicle Audio and Video Communication," *Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International*, pp. 179–184, 2008.

[32] H.-T. Lim, L. Völker, and D. Herrscher, "Challenges in a Future IP/Ethernet-based In-car Network for Real-time Applications," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 7–12.

[33] 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.

[34] J. Real and A. Crespo, "Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal," *Real-Time Systems. The International Journal of Time-Critical Computing Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[35] M. Berkelaar, K. Eikland, P. Notebaert *et al.*, "lpsolve: Open source (mixed-integer) linear programming system," *Eindhoven U. of Technology*, 2004.

[36] M. Broy, "Challenges in Automotive Software Engineering," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 33–42.

[37] 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.