Formal Analysis of the Startup Delay of SOME/IP Service Discovery

Jan R. Seyler*, Thilo Streichert*, Michael Glaß†, Nicolas Navet‡, Jürgen Teich†
*Mercedes-Benz Cars Development, Networking & Standard Software – Daimler AG
Email: {jan.seyler|thilo.streichert}@daimler.com
†Department of Computer Science 12 – University of Erlangen-Nuremberg
‡Faculté des Sciences, de la Technologie et de la Communication – Université du Luxembourg

Abstract—**An automotive network needs to start up within the millisecond range. This includes the physical startup, the software boot time, and the configuration of the network. The introduction of Ethernet into the automotive industry expanded the design space drastically and is increasing the complexity of configuring every element in the network. To add more flexibility to automotive Ethernet networks, the concept of Service Discovery was migrated from consumer electronics to AUTOSAR within the SOME/IP middleware. A network is not fully functional until every client has found its service. Consequently, this time interval adds to the startup time of a network. This work presents a formal analysis model to calculate the waiting time of every client to receive the first offer from its service. The model is able to determine the worst case of a given parameter set. Based on this, a method for calculating the total startup time of a system is derived. The model is implemented in a free-to-use octave program and validated by comparing the analytical results to a timing-accurate simulation and an experimental setup.**
In every case the worst-case assumption holds true – the gap between the maximum of the simulation and the presented method is less than 1.3 %.

I. Introduction

This work introduces a method to calculate the worst-case configuration time of a system using Automotive Ethernet with service discovery as defined in AUTOSAR [1]. It is particularly important to know this time as the total startup time from power down to full communication consists of the physical boot time and the configuration time of the system and needs to be in the millisecond range. The configuration time is finished as soon as every node in the system is able to fully communicate.

Ethernet was introduced in the automotive industry because of its high bandwidth and high grade of standardization [10]. In comparison to todays automotive bus systems like CAN, LIN, and FlexRay, Ethernet is a switched network with full-duplex links. The current in-car communication is defined by the senders whereas the receivers can filter in hardware which data units (*PDU*) they want to receive. As a result, Electronic Control Units (*ECU*s) need to be organized in *communication clusters*. These clusters contain all ECUs that share communication flows and consist of a bus or several buses that are connected over gateways. Every participant receives each PDU that is sent within this cluster. This so-called broadcast communication is still possible with switched Ethernet networks but this results in a poor efficiency because every frame needs to be received and filtered above OSI layer 4. Since an automotive network consists of embedded systems where resource usage is crucial, communication and buffer efficiency are important parameters. As a result, the communication in embedded point-to-point networks should mainly consist of uni- and multicast messages. This can only be realized by defining the sender/receiver relations and statically configuring ports as well as IP-addresses. Automotive networks are mostly engineered and static which means that it is in principal possible to store all configurations which are needed in the AUTOSAR system description [2]. In this case, the OEM needs to manage the topology, the traffic, and the configuration of every network in every variant of each car. This results in an extremely difficult and complex network administration. Additionally, this approach is very static. The migration of functionality from one ECU to another would result in a high amount of reconfiguration, making the process very expensive for already produced cars.

In consumer electronics, the typical local area network is not static. Ethernet and its higher protocol layers – especially IP on the network layer – are designed with the premise of an always running network to which nodes are connected and removed dynamically. The newly added nodes need to register in the network and find out what functionality the other participants in the network can offer. This is typically done by using the Service Location Protocol and Service Discovery [3, 7]. This principle was moved into the automotive industry with the Scalable Service-Oriented Middleware on IP (SOME/IP) on top of switched Ethernet. But SOME/IP introduces overhead within the startup and at runtime.

This work presents, to the best of our knowledge, the first analytical approach on the startup time of SOME/IP-SD and provides a free-to-use program that derives the worst-case startup time and a worst-case parameter-set for a given Ethernet network.

II. Related Work

The arbitration of Ethernet output ports typically follows a strict-priority and non-preemptive (SPNP) scheduling approach, for which local formal analysis is well understood [4]. Switched Ethernet combines several data streams of the same priority ordered in FIFOs. The usage of Ethernet in industrial automation, avionics, and the automotive industry requires a formal worst-case analysis. The two most well known methods are real-time calculus (RTC) [20] and compositional performance analysis (CPA) [9, 16, 17]. Both have been successfully applied to switched Ethernet [6, 14, 15] and [5, 15, 18, 19]. To evaluate the real-time behavior of a system, not only the communication delay due to the arbitration in the output ports but an application-to-application delay is important. As a result, the timing of each OSI layer must be considered. The timing of the network startup phase is of special interest because the user expects a fast response time from the car. Miller [13] determined that the limit of the response time giving the user the feeling that a system is reacting instantaneously is about 100 milliseconds. One of the first things that a user perceives when entering a car is the interior light, where no difference between the turning on of the different light sources should be palpable. As a result, the demanded startup time of todays bus systems is below 150 ms. It is foreseeable that the limits for Ethernet will also be within the millisecond range.

As a remedy, this paper introduces a technique to analyze the startup timing of a system using SOME/IP Service Discovery on the application layer to set up all communications paths within the system. A startup delay of every client in the system will be calculated which is defined as the time between the client being ready and receiving the first offer from the service, it wants to subscribe to.

III. Functional Description of SOME/IP Service Discovery

SOME/IP [1] is a middleware that is developed with automotive usecases in mind. In contrast to consumer electronics, the focus is not the location of services but the availability of services. The premise is, that all clients and services are already connected to the network. The network is initially powered down. After a startup event, a subset of all nodes will wake up and the services and clients on these nodes will start. SOME/IP Service Discovery (*SOME/IP-SD*) can be used to set up communication paths in this phase. A service as well as a client has three phases: the *Initial Wait Phase*, the *Repetition Phase*, and the *Main Phase*. A service sends *offer* messages within the network. These broadcast messages are used to offer services within a network. They consist of a type, service ID, instance ID, version, and a time-to-live field. A client uses *find* messages to locate services in a network. A find message is also a broadcast message containing the service and instance ID and version of the searched service [1]. This section describes the timing relevant behavior of every phase for a client and a service which is depicted in Figure 1.

A. The Initial Wait Phase

This is the first phase a client or service enters after the DOWN phase. It is introduced to prevent initial bursts that could jam the system.

1) Client: A client enters the Initial Wait Phase if it is requested from within the application layer. It stays within this phase for a random time between *SdClientTimerInitialFindDelayMin* and *SdClientTimerInitialFindDelayMax* [cf. 1, ch. 7.7.2]. Within this phase, the client stays silent, i.e. no *find* messages are sent. If an offer of the service, that the client wants to find, is received within this phase, the client skips its Repetition Phase and transits directly to its Main Phase.

2) Service: A service enters the Initial Wait Phase if its state is changed to *available*. Much like the client, the service remains in this phase for a random time between *SdServerTimerInitialOfferDelayMin* and *SdServerTimerInitialOfferDelayMax* and the service also stays silent. But unlike the client, any find message received in this phase is ignored. There is no chance in which the service skips its Repetition Phase.

B. The Repetition Phase

This phase enables a short startup delay with short periods between the find and offer messages whereas it is also possible to keep the additional network load caused by the SD module as low as possible in the Main Phase.

1) Client: As soon as it enters the Repetition Phase, a client sends its first multicast find message. Every sent message is counted in *FndCnt* starting with 0 and the time between two succeeding find message is calculated by

$$t_p = 2^{FndCnt} \cdot CltRepDel$$

until $FndCnt \geq CltRepMax$.

When the client receives an offer message from the service it is searching, it stops sending find messages, even though the maximum number of *find* messages *CltRepMax* was not reached.

2) Service: A service behaves almost exactly like a client with the exception of the reaction to a received find message. In this case, the service waits a random time between *SdServerTimerRequestResponseMinDelay* and *SdServerTimerRequestResponseMaxDelay* [cf. 1, ch. 7.5.3] and sends a unicast *offer* to the sender of the find message. This will, however, not influence the normal behaviour within the Repetition Phase and the service continues to send multicast offer messages until the maximum number of *offer* messages SvcRepMax is reached.

C. The Main Phase

The purpose of this phase is to keep the network load produced by the SD module as low as possible in normal operation. But the availability of a service should still be signalized. As a result, a client does not send any find messages in its Main Phase. A service continues to send offer messages with a period of *SvcCycDel* to show its availability. It also continues to answer received find messages as described in Section III-B2.

The following section derives a set of formulas to calculate the startup delay t^W caused by SOME/IP-SD for a client within a given system. $t^W_{i,j}$ describes the startup delay of a client on node i that wants to consume service j. The interval starts directly after $CltBootDel_{i,j}$ and finishes as soon as the client receives an offer from service j.

IV. Analysis of the SOME/IP-SD Startup Phase

This section describes the main contribution of this paper: A set of formulas to calculate the startup delay of a system using SOME/IP-SD. The following assumptions are made:

Assumptions:

- A system consists of several nodes which are connected using switched Ethernet.
- A node can be both a client of several remote services and server for several services.
- Each service is identified in the system via an unique number j.
- Every service only has one instance within the system.
- For one specific service there can be at most one client on each node.
- All nodes have varying boot times and get activated asynchronously.

After the premises are set, let $CltBootDel_{i,j}$ be the time node i needs to boot and start the Initial Wait Phase of a client that wants to find service j within the network. Further, let $SvcBootDel_{n,j}$ be the boot time of node n on which service j is running. Additionally, the following parameters of service j on node n are needed for the calculation:

Definition IV.1

$SvcInitDel_{n,j}$: The duration which service j on node n will stay in the Initial Phase.
$SvcRepDel_{n,j}$: The base interval with which the service j sends offer messages in the Repetition Phase.
$SvcRepMax_{n,j}$: Maximum number of offer messages sent in the repetition phase of service j.
$SvcCycDel_{n,j}$: The interval of the offer messages in the main phase.
$SvcAnsDel_{n,j}$: Maximum time until a service sends an answer to a received find message.

For a client on node i consuming service j the following definitions are made:

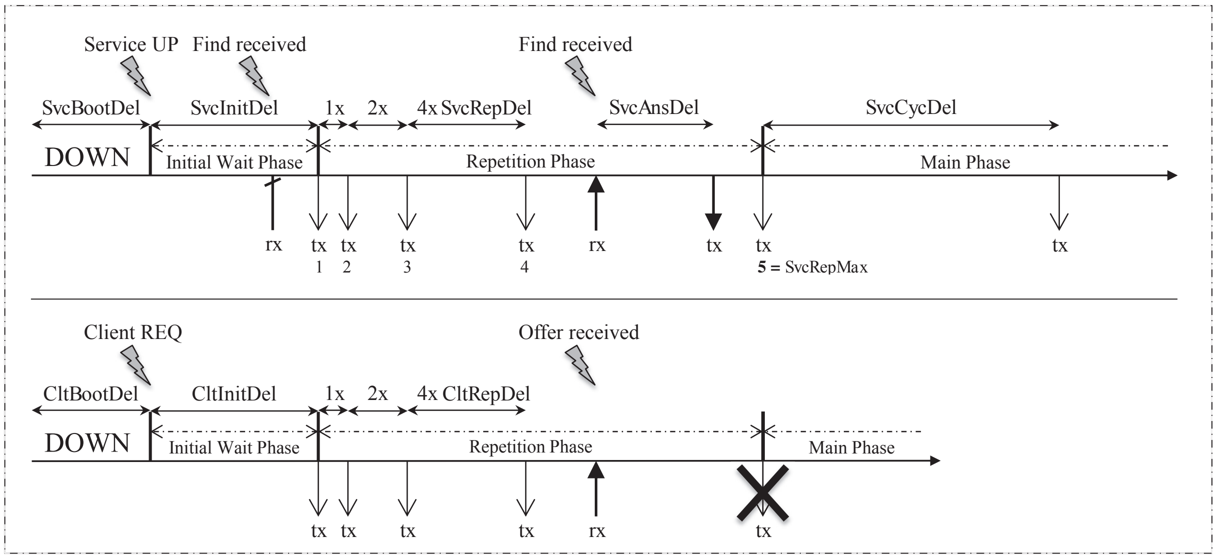


Figure 1 – The startup phase of a SOME/IP Service (upper) and Client (lower).

Definition IV.2

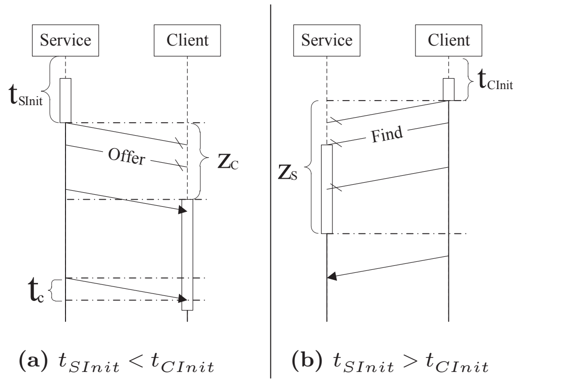
$CltInitDel_{i,j}$: The duration which the client on node i will stay in the Initial Phase.
$CltRepDel_{i,j}$: The base interval with which the client sends find messages for service j in the Repetition Phase.
$CltRepMax_{i,j}$: Maximum number of find messages sent in the repetition phase of every client.

Now, the scope is to calculate the startup delay of a system of nodes and the waiting time $t^W_{i,j}$ of each client in the system. $t^W_{i,j}$ describes the timespan that a client running on node i needs to find the service j to which it wants to subscribe to. The timespan starts as soon as the client enters its Initial Wait Phase and stops as soon as the client receives the first offer from service j. Every client has two possible states: either *request* mode, where it will send *find* messages in the Repetition Phase or *listen* mode, where it will only wait for offer messages. Likewise, a server can either be in *offer* mode, where it will send offer messages in the Repetition Phase and the Main Phase or in *silent* mode, where it will only react to find messages. $t^W_{i,j}$ does consist of the software-related delay calculated out of the parameters defined in Definition IV.1 and Definition IV.2, and the communication delay t_c(n,i) of the transmitted messages from node n to node i, see Figure 2. This delay is the worst-case response time (wcrt) of the connected communication technology, i.e. Ethernet, used in this system. For this approach, the wcrt as calculated in [18] is used.
To make the formula more readable, the following definitions are made:

Definition IV.3

$$t_{S_{n,j}Init} := \left(SvcBootDel_{n,j} + SvcInitDel_{n,j}\right)$$
$$t_{C_{i,j}Init} := \left(CltBootDel_{i,j} + CltInitDel_{i,j}\right)$$
$$z_S := t_{S_{n,j}Init} - t_{C_{i,j}Init}$$
$$z_C := CltBootDel_{i,j} - t_{S_{n,j}Init}$$

To determine t^W, the number of already received find (X_S) and offer (X_C) messages within the Repetition Phase are needed. In the following *RepDel* is synonym for $SvcRepDel_{n,j}$ or $CltRepDel_{i,j}$, z for z_S or z_C, and respectively X is a synonym for X_S or X_C. As the interval between two succeeding messages



(a) $t_{SInit} < t_{CInit}$ **(b)** $t_{SInit} > t_{CInit}$

Figure 2 – Representation of the variables in Definition IV.3.

in the Repetition Phase is doubling with every regularly sent message, the following inequation needs to hold:

$$z - t_c(n,i) \leq \sum_{k=0}^{X} 2^k \cdot RepDel \tag{1}$$

$X \in \mathbb{N}^0$ can then be calculated using the properties of geometric series:

$$\begin{aligned} z - t_c(n,i) &\leq \sum_{k=0}^{X} 2^k \cdot RepDel = \left(2^{X+1} - 1\right) \cdot RepDel \\ &\Leftrightarrow \frac{z - t_c(n,i)}{RepDel} + 1 \leq 2^{X+1} \qquad |RepDel \neq 0 \\ &\Rightarrow X = \left\lceil \log_2\left(\frac{z - t_c(n,i)}{RepDel} + 1\right)\right\rceil - 1 \quad |X > 0 \end{aligned} \tag{2}$$

If a service exceeds *SvcRepMax* offer messages, it will transit from its Repetition Phase to its Main Phase which sends offer messages with the interval *ScvCycDel*. In this case, the number of already sent periodic offer messages $Y \in \mathbb{N}^0$ is also important for the calculation:

$$\begin{aligned} &Y \cdot SvcCycDel \geq \\ &\quad z_C - t_c(n,i) - \left(2^{SvcRepMax+1} - 1\right) \cdot SvcRepDel \\ &\Leftarrow Y = \left\lceil \frac{z_C - t_c(n,i) - \left(2^{SvcRepMax+1} - 1\right) \cdot SvcRepDel}{SvcCycDel} \right\rceil \end{aligned} \tag{3}$$

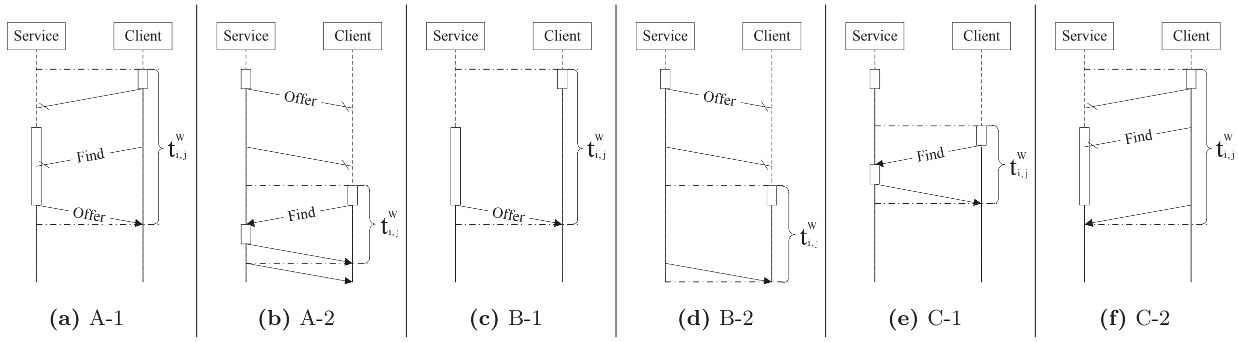


Figure 3 – Sequence charts of the possible SOME/IP SD startup configurations.

To display the formula in a more compact way, the following distinction is necessary:

Definition IV.4

$$\hat{X} = \begin{cases} X_C, & \text{if } X \le SvcRepMax \\ SvcRepMax, & else \end{cases} \tag{4}$$

$$\hat{Y} = \begin{cases} 0, & \text{if } X \le SvcRepMax \\ Y, & else \end{cases} \tag{5}$$

With this knowledge it is now possible to develop the formulas to calculate the pre-subscription time of a client within a system. The mathematical formulation depends on the state of the system, which leads to a case distinction on the state of the service and the client. All cases are depicted in Figure 3.

In reality, $CltBootDel_{i,j}$ and $SvcBootDel_{n,j}$ will not be well defined but within a range of possible values. Finding the worst-case parameter set requires to determine $t_{S_{n,j}Init}$ and $t_{C_{i,j}Init}$ such that $t^W_{i,j}$ is maximum. The setting of the parameters does vary in the following cases and is thus done separately.

A. Service in Offer Mode and Client in Request Mode

The service will send offer messages in its repetition and its main phase and the client will send find messages in its repetition phase. To find the worst case, $CltBootDel_{i,j}$ needs to be chosen minimal and

$$t_{S_{n,j}Init} = \max\left(SvcBootDel_{n,j} + SvcRepDel_{n,j}\right).$$

After these parameters are set, another distinction is neccessary:

1) $t_{S_{n,j}Init} + t_c(n,i) \ge CltBootDel_{i,j}$:

$$t^W_{i,j} = z_S + CltInitDel_{i,j} + t_c(n,i) \tag{6}$$

2) $t_{S_{n,j}Init} + t_c(n,i) < CltBootDel_{i,j}$:

$$t^W_{i,j} = \min\left\{ \begin{array}{c} \left(\begin{array}{c} \left(2^{\hat{X}+1} - 1\right) SvcRepDel_{n,j} + \\ \hat{Y} \cdot SvcCycDel_{n,j} + t_c(n,i) - z_C \end{array} \right) \\ CltInitDel_{i,j} + 2\,t_c(n,i) + SvcAnsDel_{n,j} \end{array} \right\} \tag{7}$$

B. Service in Offer Mode and Client in Listen Mode

Because a service does not process find messages received in its initial phase, this case can be derived from case IV-A. For the case displayed in Figure 3c, $t^W_{i,j}$ is calculated as in formula (6). If $t_{S_{n,j}Init} + t_c(n,i) < CltBootDel_{i,j}$ as shown in Figure 3d, $t^W_{i,j}$ is calculated by the first row of formula (7):

$$\begin{aligned} t^W_{i,j} = \; & \left(2^{\hat{X}+1} - 1\right) SvcRepDel_{n,j} + \\ & \hat{Y} \cdot SvcCycDel_{n,j} + t_c(n,i) - z_C \end{aligned} \tag{8}$$

C. Service in Silent Mode and Client in Request Mode

In this last state, the service is silent, which means that it does not send any offer messages unless it receives a find message. The client is sending find messages in its repetition phase with the base interval of $CltRepDel_{i,j} \ne 0$. Because find messages are ignored in the initial phase of the service, $SvcInitDel_{n,j}$ should be configured to 0. Nevertheless, the following formula is applicable to every value. In detail, the worst-case parameter set is determined by first calculating $A \in \mathbb{N}$ such that A is maximum and

$$\begin{aligned} & \left(2^{A+1} - 1\right) \cdot CltRepDel_{i,j} < \max\left(t_{S_{n,j}Init}\right) \\ \Leftrightarrow \quad & A = \left\lfloor \log_2 \left(\frac{\max\left(t_{S_{n,j}Init}\right)}{CltRepDel_{i,j}} + 1 \right) \right\rfloor - 1 \end{aligned} \tag{9}$$

Note that, if $A > (CltRepMax - 1)$, in the worst case the client will never find its service. After knowing A, $t_{C_{i,j}Init}$ needs to be determined such that $CltInitDel_{i,j}$ is maximum with respect to:

$$\begin{gathered} t_{C_{i,j}Init} + \left(2^{A+1} - 1\right) \cdot CltRepDel_{i,j} < \max\left(t_{S_{n,j}Init}\right) \\ SdClientTimerInitialFindDelayMin \le CltInitDel_{i,j} \\ CltInitDel_{i,j} \le SdClientTimerInitialFindDelayMax \end{gathered}$$

Now, there are two subcases that need to be distinguished:

1) $t_{C_{i,j}Init} + t_c(n,i) \ge t_{S_{n,j}Init}$:

$$t^W_{i,j} = CltInitDel_{i,j} + SvcAnsDel_{n,j} + 2\,t_c(n,i) \tag{10}$$

2) $t_{C_{i,j}Init} + t_c(n,i) < t_{S_{n,j}Init}$:

$$\begin{aligned} t^W_{i,j} = \; & t_{C_{i,j}Init} - CltBootDel_{i,j} \\ & + \left(2^{X_S+1} - 1\right) \cdot CltRepDel_{i,j} \\ & + SvcAnsDel_{n,j} + 2\,t_c(n,i) \end{aligned} \tag{11}$$

Based on these client waiting times it is now possible to derive the delay between power-on (t_0) and the system being able to communicate for each client that uses SOME/IP-SD to configure its communication relations. In the first step the relative waiting time of the single clients $t^W_{i,j}$ needs to be normalized to a startup delay $t^D_{i,j}$, which starts at t_0 by adding the boot delay of each client:

$$t^D_{i,j} = t^W_{i,j} + CltBootDel_{i,j} \tag{12}$$

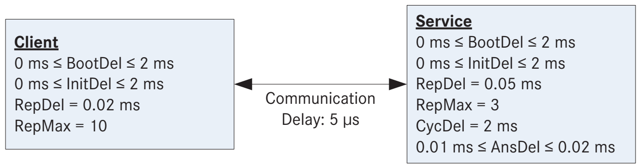


Figure 4 – Configuration used for the proof of concept.

In the second step, the startup delay of the whole system is calculated. This is done by iterating over all clients on every node in the system. Every node adds its corresponding startup delay to a set of all waiting times and the worst-case delay is given by looking at the maximum value within this set.

V. Proof of Concept

To validate the correctness of formula (6)–(11) a system consisting of two nodes was built up as an experiment and modeled in a simulation environment. Node 1 implements a client that wants to subscribe to service 1 implemented on node 2. The configuration parameters of the system can be seen in Figure 4. The method described above is implemented in GNU Octave and is available free to use on request. The case described in Section IV-A is a combination of Section IV-B and Section IV-C and the correctness of the former relies on the correctness of the latter two. Thus, Section IV-B and Section IV-C were implemented in a simulation and an experimental setup and are now discussed in more detail for the system depicted in Figure 4. At this point, it is not expedient to build up a more complex system because the communication delay is a fixed parameter in the calculation and it is more important to ensure the conservativeness of the model introduced in this paper.

A. Scenario 1

The experiment uses the same boards for node 1 and 2 and both are connected to the same power source. As a result, both nodes start nearly synchronously and their boot times are very close to each other: $\left|CltBootDel_{1,1} - SvcBootDel_{2,1}\right| \leq 2\,ms$. In this scenario, the client is configured in listen-mode and service 1 in offer-mode.

For the parameters depicted in Figure 4, the introduced model calculates:

$$t^{W}_{1,1} = 4.005\,ms$$

In this easy example, the outcome is comprehensible: It results from the following worst-case parameter set and formula (6):

$$\begin{aligned} SvcBootDel_{2,1} &= 2\,ms \\ SvcInitDel_{2,1} &= 2\,ms \\ CltBootDel_{1,1} &= 0\,ms \\ CltInitDel_{1,1} &= 0\,ms \end{aligned}$$

A development version of the RTaW-Pegase library [12] was used for the simulative results. The system was configured as shown in Figure 4. The simulated time is configured to start with the Initial Phase of the client and stop as soon as the client receives an offer from service 1. Figure 5 shows the histogram of $t^{W_{sim}}_{(1,1)}$ with 10 000 simulative runs taken.

$$\max\left(t^{W_{sim}}_{(1,1)}\right) = 3.984\,ms < t^{W}_{1,1}$$

This suggests to us, that the proposed model is a valid worst-case for this simulation and the gap to the worst case calculated by the presented method is less than 1 %. In the experiment, the client was configured on node 1 with IP-address 192.168.1.3 and the service on the node 2 with IP-address 192.168.1.2. Table I

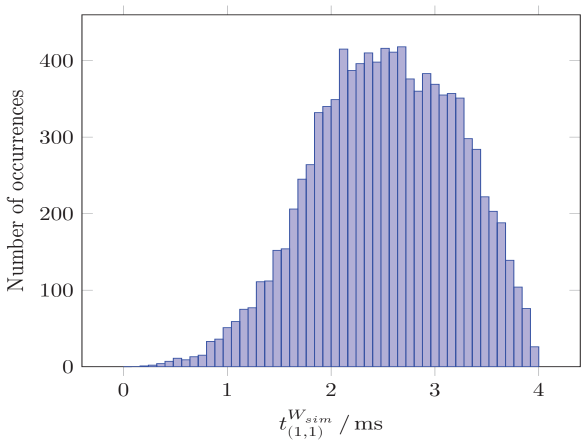


Figure 5 – Startup delays obtained by simulation for scenario 1 with the parameters as described in Figure 4.

time [ms]	source	destination	type
9.028	system	controller	start client and service
11.091	192.168.1.2	231.101.101.101	offer
11.114	192.168.1.3	192.168.1.2	subscribe
...	...	...	...

Table I – Experimental results for scenario 1.

shows the important excerpt of the captured trace regarding the startup delay. This trace shows, that the client will receive the first offer message of service 1 between 11.096 ms and 11.101 ms, which results in a maximum waiting time of

$$t^{W_{exp}}_{(1,1)} = 2.073\,ms$$

As $t^{W_{exp}}_{(1,1)} < t^{W}_{1,1}$, the worst-case assumption holds true for the experiment also.

B. Scenario 2

In the second scenario, the service is now configured to be in silent mode and the client is in find mode. Contrary to the SOME/IP specification, the offer message of the service which will be sent automatically after the service enters its Main Phase, was disabled to enforce a behavior which differs from Scenario 1. The introduced analytical models returns

$$t^{W}_{1,1} = 6.58\,ms$$

and the following parameter set:

$$\begin{aligned} SvcBootDel_{2,1} &= 2\,ms \\ SvcInitDel_{2,1} &= 2\,ms \\ CltBootDel_{1,1} &= 0\,ms \\ CltInitDel_{1,1} &= 1.45\,ms \end{aligned}$$

Again, the system was modeled using RTaW-Pegase and the results are displayed in Figure 6. The maximum which was seen in the 10 000 runs was:

$$\max\left(t^{W_{sim}}_{(1,1)}\right) = 6.495\,ms < t^{W}_{1,1}$$

This again provides evidence, that the introduced model holds its worst-case claim. However, note that the calculated case is a possible case and thus there is no over approximation in the presented method. The results from the experimental setup are depicted in Table II. In this case, the offer is received latest at 14.116 ms, which results in a subscription time of

$$t^{W_{exp}}_{(1,1)} = 2.075\,ms < t^{W}_{1,1}$$

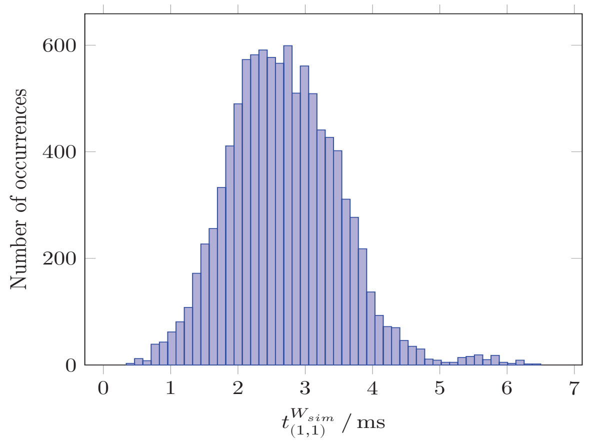


Figure 6 – Startup delays obtained by simulation for scenario 2 with the parameters as described in Figure 4.

time [ms]	source	destination	type
12.036	system	controller	start client and service
12.321	192.168.1.3	231.101.101.101	find
12.341	192.168.1.3	231.101.101.101	find
12.381	192.168.1.3	231.101.101.101	find
12.461	192.168.1.3	231.101.101.101	find
14.098	192.168.1.2	231.101.101.101	offer
14.121	192.168.1.3	192.168.1.2	subscribe
...	...	...	...

Table II – Experimental results for scenario 2.

Again, the worst-case assumption holds true, even though it is only one experimental result.

VI. Conclusion and Outlook

This paper describes a new mathematical method to calculate the subscription time of any client within an automotive network using AUTOSAR SOME/IP-SD. Additionally, a worst-case parameter set and system startup time is returned. The method was evaluated using a timing-accurate simulation and an experimental setup. It is shown that the worst-case assumption of the method holds true. In both scenarios, the difference between the seen maximum in the simulation and the calculated worst-case is less than 1.3 %.

The next steps should include a comprehensive analysis of the design space of SOME/IP-SD to find design rules that will help in the development process of automotive networks. For the experimental results, AUTOSAR was configured in interrupt mode. In normal mode, AUTOSAR will collect all instances in one main cycle and send everything bundled at the end. This will add one cycle time as additional waiting time and needs to be investigated in more detail. Finally, it is needed to include the proposed analysis in a complete wcrt-analysis which does consider real network latencies.

References

[1] AUTOSAR. *Specification of Service Discovery*. 2013. URL: http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/system-services/standard/AUTOSAR_SWS_ServiceDiscovery.pdf.

[2] AUTOSAR. *System Template*. 2013. URL: https://www.autosar.org/fileadmin/files/releases/4-1/methodology-templates/templates/standard/AUTOSAR_TPS_SystemTemplate.pdf.

[3] A Bottaro et al. “Dynamic Web Services on a Home Service Platform”. In: *Proc. of AINA*. Mar. 2008, pp. 378–385.

[4] Robert I Davis et al. “Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised”. In: *Real-Time Systems* 35.3 (2007), pp. 239–272.

[5] Jonas Diemer, Daniel Thiele, and Rolf Ernst. “Formal worst-case timing analysis of ethernet topologies with strict-priority and AVB switching”. In: *Proc. of SIES*. IEEE. 2012, pp. 1–10.

[6] J-P Georges, Thierry Divoux, and Eric Rondeau. “Strict Priority versus Weighted Fair Queueing in Switched Ethernet networks for time critical applications”. In: *Proc. of IPDPS*. IEEE. 2005, pp. 141–141.

[7] Erik Guttman. “Service location protocol: Automatic discovery of IP network services”. In: *Internet Computing, IEEE* 3.4 (1999), pp. 71–80.

[8] Sumi Helal. “Standards for service discovery and delivery”. In: *Pervasive Computing, IEEE* 1.3 (2002), pp. 95–100.

[9] Rafik Henia et al. “System level performance analysis–the SymTA/S approach”. In: *IEE Proceedings-Computers and Digital Techniques* 152.2 (2005), pp. 148–166.

[10] Andreas Kern. “Ethernet and IP for Automotive E/E - Architectures”. PhD Thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2012.

[11] Andreas Kern et al. “Gateway strategies for embedding of automotive CAN-frames into ethernet-packets and vice versa”. In: *ARCS* (2011), pp. 259–270.

[12] Joern Migge and Nicolas Navet. *RTaW-Pegase : analyzing AFDX & switched Ethernet networks*. Oct. 2014. URL: http://www.realtimeatwork.com/software/rtaw-pegase/.

[13] Robert B Miller. “Response time in man-computer conversational transactions”. In: *Proc. of the joint computer conference*. 1968, pp. 267–277.

[14] Rene Queck. “Analysis of ethernet avb for automotive networks using network calculus”. In: *Proc. of ICVES*. IEEE. 2012, pp. 61–67.

[15] Kasper Revsbech et al. “Worst-case traversal time modelling of Ethernet based in-car networks using real time calculus”. In: *Smart Spaces and Next Generation Wired/Wireless Networking*. Springer, 2011, pp. 219–230.

[16] Kai Richter, Marek Jersak, and Rolf Ernst. “A formal approach to MpSoC performance verification”. In: *Computer* 36.4 (2003), pp. 60–67.

[17] Simon Schliecker et al. “System level performance analysis for real-time automotive multicore and network architectures”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.7 (2009), pp. 979–992.

[18] Daniel Thiele et al. “Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet”. In: *Proc. of CODES+ ISSS*. IEEE. 2013, pp. 1–10.

[19] Daniel Thiele et al. “Improving Formal Timing Analysis of Switched Ethernet by Exploiting Traffic Stream Correlations”. In: *Proc. of CODES+ISSS*. 2014.

[20] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. “Real-time calculus for scheduling hard real-time systems”. In: *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*. Vol. 4. IEEE. 2000, pp. 101–104.