Local TBASCEM - Tight Bounds with Arrival and Service Curve Estimation by Measurements

Oliver Kardos¹, Christoph Funda², Reinhard German¹, and Kai-Steffen Hielscher¹

¹Friedrich-Alexander Universität Erlangen-Nürnberg
²ZF Mobility Solutions GmbH

March 30, 2024
Local TBASCEM - Tight Bounds with Arrival and Service Curve Estimation by Measurements

Oliver Kardos¹, Christoph Funda², Reinhard German¹, and Kai-Steffen Hielscher¹

¹ Friedrich-Alexander Universität Erlangen-Nürnberg, Erlangen, Germany
² ZF Mobility Solutions GmbH, Marie-Curie-Straße 5a, 85055 Ingolstadt, Germany

Abstract. Our goal is to solve the challenge of quantifying the performance of Hardware-in-the-Loop (HIL) computer systems that are used for data re-injection. In such a system, there are multiple queues and a server system operating on a First-In, First-Out (FIFO) basis. Here, the challenge lies in establishing tight bounds on end-to-end delay and system backlog. With that, buffer and pre-buffer time configurations can be optimized. To achieve this, Network Calculus (NC) is chosen as the basic analytical framework. For NC calculations, different techniques of estimating arrival and service curves from measurement data can be used from the literature. We have chosen four of these methods, which can be applied to data sets of industrial Timestamp Logging (TL). However, here the problem is that these conventional methods could produce too large bounds (by factor of 1000 or more) than the measured maximum values. This can lead to an ineffective design of HIL system parameters and inefficient resource usage. In our proposed approach, called local TBASCEM, we introduce a local reverse engineering approach. It derives from the global TBASCEM and relies on linear NC equations for estimating the parameters of arrival and service curves. For test purposes, we imposed constraints on the equation variables and employed non-linear optimization. So that, we achieve tighter bounds on service curves in comparison to four other state-of-the-art methods. Furthermore, TBASCEM in general eases the run-time measurement process. This is done by supporting real-time data acquisition to evaluate and optimize HIL system performance. It also enhances observability for the adaption of the HIL configuration to new sensor data. Efficient performance logging of arrival and service curve parameters and the derivation of tighter bounds in HIL systems make TBASCEM a strong tool for optimizing and monitoring applications in non-hard-real-time environments.

Keywords: Performance Evaluation and Monitoring · Hardware-in-the-loop Test System · Streaming System · Network Calculus · Arrival and Service Curve Estimation

1 Introduction

For verification of the perception technology on the Device Under Test (DUT) in autonomous vehicle technology, millions of kilometers of recorded sensor data...
must be replayed with HIL systems. A lot of time and energy are required for this process, because the recordings need to be transmitted from the DUT to the HIL in real-time. Furthermore, ensuring high quality HIL performance, the playback must exactly represent a real-world scenario. Timing to the DUT and no data loss in the HIL system are of high significance. In the second section, this will be discussed.

Many HIL systems fall under the area of soft and non-real-time systems. Although their hardware interfaces to the DUT are real-time systems.

Now, it has to be questioned how such non-hard real-time systems can satisfy the requirements of the DUT. An approach could be the usage of a playback buffer on the hardware interface to the DUT. To guarantee the continuous streaming to the DUT this buffer can be filled with the necessary amount of data.

But designing buffers and pre-buffers accordingly is challenging. The worst-case end-to-end delay and backlog bounds for the buffer size dimensions and the pre-buffer time parameter must be determined.

In order to minimize the idle time before streaming to the DUT the pre-buffering has to be minimized as much as possible. This will also reduce the energy consumption of computing resources.

Avoiding an empty playback buffer during re-injection is decisive. This disturbs the DUT’s precise timing, leading to failed tests and again a loss of time and energy at the HIL computing resources. So, it is essential to adjust the pre-buffer time to be at least equal to or greater than the end-to-end delay and to fill the playback buffer with a minimum of the burst parameter of the arrival curve of the sensor stream. This is to achieve an efficient HIL performance.

Our method includes measuring the HIL system before operation, designing it accordingly, and monitoring its performance during operation, optimizing its parameters in case of necessity. The usage of measured maximum end-to-end delay would directly overestimate the required time, as waiting time in a queue could be included. That is why, considering the queuing system theory, it is needed to be accomplished by employing NC - a queuing system theory. The usage of NC allows computing the bounds for the backlog and end-to-end delay of a queuing system from its arrival and service curves. For our use case and our system, the bounds must be as tight as possible. But the NC bounds must not always be tight. Here, we have the major challenge.

Basic concepts are mathematically defined in the NC framework and in all basic literature about NC. To summarize, an arrival curve is the upper constraint of an input flow, and a service curve is the lower constraint of a flow provided by a service. We use burst-rate curves and rate-latency curves, which are the basic linear arrival curves and service curves. In hard real-time systems, their parameters can be found easily. They are often directly defined. But, in our non-real-time HIL system under study, this is not the case. So, the challenge is to measure them.

In this paper, we used one of the approaches mentioned in the survey study by Fidler et al. It is about generating a strict service curve based on TL.
of the input and output of queuing server systems by Alcuri et al. [1]. But, our results show, that the bounds are not tight at all with these methods. They often exceed a factor of 1000.

All the measurement methods, which are additionally discussed in the survey, refer to the TL of the ingress and egress of the system. Our main interest is to reduce the TL being able to do the run-time measurements at the HIL system during operation, by not generating massive data and performance overhead for the network or processor. On one side, TL occupies the memory. On the other side, the TL has to be sent via network, which would occupy the network during operation and the processor.

In [3] a new approach is introduced called local TBASCEM - Tight Bounds with Arrival and Service Curve Estimation by Measurements to close the gap in estimating a service curve by measurements. It is derived from the global TBASCEM approach, which is explained in [7] (paper content serves as basis for this one). The target is to provide tight delay and backlog bounds of a streaming process within a computer system, without the production of much TL. TBASCEM in general aims to provide a performant and efficient estimation method from TL. It also includes a technique for reducing the necessary measurement data for arrival and service curve estimation, being able to calculate tight bounds with NC.

So, the novel TBASCEM approach fills the gap for an effective and efficient method to measure and estimate service curves. This paper uses the global TBASCEM approach and presents a variant of it called local TBASCEM. The target of it is to determine arrival and service curve estimations for each packet in a data stream. Since the already described global TBASCEM approach looks at the incoming data stream as a whole, we call it global TBASCEM in this paper. This is for distinction.

This paper is structured as follows: In section 2 we describe the basic technical background of a HIL systems. The next sect. 3 introduces our local TBASCEM approach. Here, we describe the conceptional idea. After that we explain the local engineering approach for arrival and service curve parameter estimation. In the following section 4 our research question will be defined, including a short explanation of the methodology and our hypothesis. This is followed by a description of the experimental setting. Finally, we discuss our results. In the next section 5 we describe former work in literature and its relation to our work. In the last section 6 we finalize with a short summary and the contribution of our work.

2 Fundamentals

2.1 Real-Time Constraints of HIL Systems

In HIL systems, the computing machine on which the processes run greatly influences the processing time of software or network processes. In hard real-time systems, the worst-case processing time is a critical performance indicator,
leading to strict requirements on the computing machine. Hard real-time systems are expensive. But they offer a formal evaluation and guarantees of their [Worst-Case Execution Time (WCET)] through measurements of processing cycles and CPU frequency.

However, many [HIL] systems, beyond their hardware interfaces to the [DUT] are not part of the category of hard real-time systems. They rather fall into the category of soft and non-real-time systems. For such systems, formal evaluation cannot be done, and because of various factors, no guarantees for a [WCET] can be given. Caching mechanisms have an impact on processing times as well as memory system hierarchies, CPU frequency fluctuations during run-time, hardware travel times, interrupt requests, context switches, and other features of modern computers and operating systems. Furthermore, delays and processing times may arise when software systems use middleware. This makes accurate measurement difficult with high variation. For instance, interprocess communication when using a localhost network connection in the robot operating system (ROS) [13] shows this complexity, like applied in our [HIL] test system [8].

In these non-real-time systems, relying totally on an observed [WCET] can also be very pessimistic. This can show up bottleneck assumptions and infinite bounds in theory, whereas practical results are much more optimistic, as demonstrated by us in [9]. Despite their limitations, non-real-time systems are more cost-effective, and their software development is simpler and less expensive. In many cases, a service based on mean-rate or even stochastic bounds is sufficient. If overshoots are monitored and documented in the test results, it is unlikely that catastrophic events will occur.

In practice, monitoring mechanisms to detect simulation service performance during run-time are often employed by closed-loop simulations running on [HIL] systems. Any task overruns are logged as warnings or errors, prompting re-evaluation of tests in cases of excessive, unwanted overruns [4].

If an open-loop re-injection occurs, the [HIL] system functions as a streaming device. It streams measured input data as previously captured sensor data to the [DUT]. The [DUT] processes the data, and the output is streamed back to the [HIL] system for evaluation as test results. Provided with insights into their behaviour and performance in real-world scenarios, this setup allows for comprehensive testing and evaluation of complex cyber-physical systems.

### 2.2 Design of [HIL] Test Systems with [NC]

[NC] is a possible analytical framework to use to provide bounds for end-to-end delay and buffer size design of queuing systems. It was introduced by Cruz et al. in 1991 [3]. The analytical solutions for streaming devices have been derived by Le Boudec et al. in [12]. It supports in setting tight but safe bounds for the buffer and delay streaming systems like this.

We adapted and applied the [NC] solutions by linear equations on measurement data from a [HIL] test system in [9]. However, to make them usable in practice, there are improvements to the derived bounds needed for the estima-
tion methods of the system service curves. The results in [9] showed this. The derived bounds often grow to infinity with the WCET method.

So, we started a review study on practical measurement methods for arrival and service curves and applied them to our HIL system published in [6] to fill this gap. We applied in [6] the proposed methods of Helm et al. [10] and Wandeler et al. [15] for estimating service curves by measurement data on TL from a HIL test system. We made use of realistic industrial workload for streaming data to CAN and Automotive Ethernet interfaces for re-injection to the DUT. The study demonstrated that the estimated service curves’ delay and backlog bounds are not sufficiently tight in all instances to be effective in practice. The calculated bounds were of a factor of over, 10000 higher compared to the maximum measured backlog or end-to-end delay.

The next chapter introduces our new local TBASCEM approach. It enables monitoring of the streaming performance, like latency, rate, and backlog, of our HIL systems during operation. It estimates arrival and service curves by using measurements, which provide tight bounds for delay and backlog.

3 TBASCEM Approach in general

In our proposed concept, we will measure and estimate arrival and service curves for non-hard real-time software services within a streaming chain of a HIL system. But, it could also be applied to any other type of streaming system. Its worst-case latency and backlog can be determined, as well as the burst parameter and estimated arrival and service curves. This allows the estimation of deterministic NC bounds for the underlying streaming system.

The approach is based on a local online algorithm that analyzes the incoming packets in real-time. It substitutes the conventional TL used for evaluating the processing performance of software modules. This helps to reduce the amount of logging data significantly. Our method only requires saving three variables in total for one software process, based on timestamps and the number of messages in the queue. So, we can waive of saving two timestamps per message. The iterative calculation process is computationally efficient, making it suitable for real-time operation of a HIL test bench in streaming mode.

Offering insights into any system influences and changes during operation, the estimated service curves give an overview of the HIL system’s performance. Those curves can be stored in a database, together with logging data from the system. This eases the analysis of system influences and allows the calculation of the probability of these service curves. This is then usable for the stochastic NC framework. It also allows for the detection of any kind of bottlenecks.

In case new input data with a different arrival curve behaviour needs to be processed by the HIL system, the buffer size and pre-buffer time parameter require adjustments. Our concept can propose an appropriate pre-buffer time to prevent buffer underflow at the playback buffer. If the recommended pre-buffer time is significantly lower compared to the initially designed value, it can also
save time. Additionally, it allows for predicting the required buffer size between each software service.

3.1 Concept Idea of local TBASCEM

Available measurement points and the streaming SW in the HIL system abstract the queue and server systems. In figure 1 a part of the HIL system is shown. There is $T_1$ representing the input to a queue of a server and $T_3$ representing the output of the server. All measured and computed values by the local TBASCEM refer to the queues between input and output.

![Fig. 1. Queue and software service (see [7])](image)

The local TBASCEM approach goes iteratively through the packets of the input stream. The objective is to calculate the service latency and rate locally. This means that only a few packets are analyzed at the same time. This makes it possible to detect local events inside the incoming data stream, e.g., impending bottlenecks or empty queues. Table 4 (see appendix) shows an overview of all the following variables.

In the beginning, the overall arrival rate $a_r$ is computed over all incoming packets. (see 1)

\[
a_r = \text{packet.size} / (T_{in}(n) - T_{in}(0))
\] (1)

The arrival rate is assumed to be the mean arrival rate of the input data stream and is always used in the next computations. Next, the algorithm uses windowing as a method to measure the queue length and end-to-end delay. It fills the window with packets as long as the incoming timestamp of a packet is smaller than the outgoing timestamp of the first packet in the window. Once that is reached, the window is complete. This represents a window (the index of the window is $j$) in the whole computation of an input data stream. For each incoming packet into the window, a temporary backlog counter $B_j$ is increased, considering the incoming packet sizes. It reflects the size of the current window by adding up the sizes of all packets in the window. The value of the backlog $B_j$ is used for the computation of service latency and rate. When the window was taken, the upper arrival rate for that window was computed. It is a linear function and estimates upwards the current arrival flow. For that, the burst of the window is computed iteratively. The packet size of the first packet in the window represents the initial burst $b_j$ of the window. Then 2 (upper arrival curve) is applied on each incoming
packet inside the window. The values of $T$ only refer to the packets inside the window, on the contrary to $1$

$$\text{bytes}(T) = a_r * (T_{j-in}(i) - T_{j-in}(i)) + b_j$$

(2)

The values of $\text{bytes}(T)$ represent the expected number of bytes at the given point in time. That means how many bytes are expected to have already come in until $T_{j-in}(i)$. This is then compared to the actual number of arrived bytes at a given point of time $T_{j-in}(i)$. The difference between the actual and the expected number of bytes is calculated for $T_{j-in}(i)$ in the window. Three options can occur:

1. The expected number of arrived bytes is smaller than the actual number of arrived bytes at $T_{j-in}(i)$. The difference between both values is negative.
2. The expected number of arrived bytes is higher than the actual number of arrived bytes at $T_{j-in}(i)$. The difference between the two values is positive.
3. The expected number of arrived bytes is equal to the actual number of arrived bytes at $T_{j-in}(i)$.

![Fig. 2. Evaluation of appropriate upper arrival curve burst](image)

After all values were computed, the following list shows the computation of the appropriate burst of the upper arrival curve. For that, the maximum value of option 1 (see enumeration above) $\text{max}(\Delta b_{neg})$ and the minimum value of option 2 $\text{min}(\Delta b_{pos})$ are taken. $\text{max}(\Delta b_{neg})$ describes the minimum necessary increase of the burst, and $\text{min}(\Delta b_{pos})$ the maximal necessary decrease of the burst. This is also seen in 2 The red graph shows the arriving packets. The green shows
the initial upper arrival curve, which has to be adjusted. The initial burst is the first incoming packet from the window. Then the values of \( \max(\Delta b_{neg}) \) and \( \min(\Delta b_{pos}) \) are added to the initial burst based on the following rules:

- If \( |\max(\Delta b_{neg})| \) is greater than 0, it means that at least one value lies above the initial upper arrival curve. In consequence, the burst has to be increased by \( |\max(\Delta b_{neg})| \). So that, all values are under the upper arrival curve. In this case, \( \min(\Delta b_{pos}) \) can be neglected.
- If \( \max(\Delta b_{neg}) \) is 0, it means that there is no need to move the upper arrival curve up. All values are under it. In this case, \( \min(\Delta b_{pos}) \) comes into account:
  - If \( \min(\Delta b_{pos}) \) is equal to 0, the initial upper arrival curve fits well and can be left as it is.
  - If \( \min(\Delta b_{pos}) \) is greater than 0, the initial upper arrival curve is overestimated and can be moved down. For that, \( \min(\Delta b_{pos}) \) is added to \( b_j \). So, \( b_j \) becomes smaller.

After that, the upper arrival curve fits the incoming packets of windows \( j \) and \( b_j \). So, the maximum delay \( D_{j_{\max}} \) of the window is computed. This is done by taking the highest difference between \( T_{j_{in}} \) and \( T_{j_{out}} \) of each packet in the window. Now, the service latency \( L_j \) and the service rate \( s_{r_j} \) are computed with the equations 3 and 4.

\[
L_j = (B_j - b_j)/a_r \quad \text{(3)}
\]

\[
s_{r_j} = b_j/(D_{j_{\max}} - L_j) \quad \text{(4)}
\]

Both equations can be comprehended with Fig. 3. It illustrates the already-determined upper arrival curve (represented by the arrival rate and burst). Based
on the relationship between burst, backlog and arrival rate, the latency is computed (left upper triangle). The service rate is computed with the ratio of burst and the difference of delay and latency (right lower triangle).

Looking at both equations, the following exceptions must be handled:

- $B_j < b_j$: $B_j$ is set to $b_j$
- $b_j = 0$: $s_{r_j}$ is set to infinite because no valid service rate can be computed, meaning the input stream cannot be used to get a service rate.
- $D_{j_{max}} = L_j$: $s_{r_j}$ is set to infinite. Division by 0 is not allowed.
- $D_{j_{max}} < L_j$: This would lead to a negative $s_{r_j}$. $D_{j_{max}}$ is already at the maximum of the window. So, $L_j$ has to be adjusted. It is computed from the already fixed $B_j$ and $a_r$. So, only the $b_j$ has to be changed. Looking at $L_j$ can only be decreased by increasing $b_j$. That is why $b_j$ is increased as long as $D_{j_{max}} \leq L_j$ and $b_j < B_j$. This ensures a positive $s_{r_j}$.

The results of each window can be stored and compared to the results of the other windows. After one window is processed, the backlog gets updated by removing the oldest packet (the first packet of the last window). Now, the algorithm generates the next window by removing the oldest packet from the last window and letting in new packets as long as their timestamp is older than the outgoing timestamp of the new oldest packet in the new window. In the event that the next incoming timestamps are newer than the new oldest packet in the new window, no new packet is added to the new window. This leads to a decrease in the window size. The advantage of this dynamical adaption of the window size is that it is not necessary to determine static window sizes. This would have the disadvantage that one would need to define such a static window size. So, there would be work and time involved in determining this. And it is also not guaranteed that the determined static window size is optimal for every kind of input stream. The code for the local TBASCEM can be comprehended in https://github.com/olik0815/Local_tbascem.

4 Evaluation and Results

In this chapter, we treat two research questions. First, we describe the importance of the questions and the methodology to answer them. Then we state a hypothesis and explain our experimental setup. Finally, we discuss the results.

RQ1: How tight are the end-to-end delay and backlog bounds by the estimated arrival and service curves derived by different estimation methods based on TL from real industrial workload?

Explanation: In the literature, there are various methods for estimating service curves from TL. We aim to decrease the TL and enable a Run-Time Measurement (RTM) technique during operation. For this, we developed the local TBASCEM method. However, before implementing the RTM in software, it is worth evaluating the tightness of the bounds. This we can generate with the TL which is already collected from the HIL system. Methodology: We implement the method in MATLAB to evaluate the tightness. Then we apply it offline on
Furthermore, we compare it to other methods from the literature, which we apply to the same dataset. We also compare it to the global TBASCEM method.

**Hypotheses:** Our Hypothesis is that the local TBASCEM derives the tightest bounds compared to all other applied methods.

**Experimental Setup:** Our test data we collected in our prototype HIL streaming system. Here, we made use of software timestamps as a baseline for our performance evaluations of the HIL streaming system. They are shown in figure 4. Eight different queuing systems were considered, e.g., the ones between measurement points $T_{ROS1}$ and $T_{ROS2}$ or measurement points $T_{LV1}$ and $T_{LV2}$. At the incoming measurement point, the incoming timestamp and the packet size were taken. At the outgoing measurement point, the outgoing timestamp of the same packet was taken. In total, we recorded 81 data streams with around 62000 packets each. This leads to more than 5 mio captured data packets. Since eight queuing systems are considered, we have more than 40 mio data points. We use them to assess the performance of various arrival and service curve algorithms. Emulating radar data being sent as Ethernet packets over a 100 Mbps channel to the DUT we made use of realistic industrial workload. Local TBASCEM is implemented in MATLAB and is subsequently applied to the recorded timestamps being generated by the software processes. We extract latency and backlog bounds. We also implemented four other state-of-the-art algorithms, namely Best-Case Execution Time (BCET), Mean-Case Execution Time (MCET), WCET and the approach by Alcuri et al. The global TBASCEM was added to the comparison, too. We also used the arrival and service curves to determine both bounds. Furthermore, we perform a comparison of the bounds’ tightness. This is done by dividing the NC bound by the maximum measured values of queue length and end-to-end delay.

**Results:**
We use TL to determine the tightness of the TBASCEM and the other algorithms offline. After that, we compare the computed bounds of all methods as tightness factors by dividing them by the maximum measured delay or back-
log. These comparisons can be comprehended in figure 5. A tightness factor of smaller than 1 is an undershot. Therefore, it is rated as a violation of the bound by the maximum measured value. Tightness factors between 1 and 10 are rated as having high tightness. A medium tightness is considered with a factor between 10 and $10^3$. A tightness factor over $10^3$ is rated as low tightness. However, in the end, the absolute value needs to be taken into account if the tightness factor is usable as a safety factor for the playback buffer size or pre-buffer time. For instance, if the measured service delay is in the µs range and the maximum measured backlog is in the byte range, a safety factor of $10^4$ or even $10^6$ would still be technically possible. So, in practice, it is feasible, but not an efficient use of resources.

Remarkably, the bounds computed by the local TBASCEM approach prove to be the tightest among all the algorithms used. The tightness factor lays for these systems at 1 for the delay tightness and below 2 for the backlog tightness. This is a safety and usability factor to consider when designing a system in practice. Even, if the backlog tightness is at 2, it is only a bit worse than global TBASCEM or Alcuri et al. But compared to all other algorithms, the standard deviation is 0. It applies to both the backlog and the delay tightness. This makes the algorithm very reliable compared to the other algorithms. Their tightness values scatter more and their whiskers are partly very long.

The tables 1 and 2 show the values of the measurement in figure 5.

**RQ2: Who long is the runtime of the local TBASCEM method compared to the other estimation methods?**

**Explanation:** For this research question, the same algorithms as in RQ1 are used, namely BCET, WCET, MCET, Alcuri et al. and global TBASCEM. This is done to figure out the runtime performance of the local TBASCEM in relation to other algorithms. Since, the delay and backlog tightness calculation is very strong, we need to figure out whether the calculation is performed within a reasonable time.

**Methodology:** We used the implementation of the mentioned algorithms again. For the time measurement, we used the "tic toc" method of Matlab. They
<table>
<thead>
<tr>
<th>Alcuri</th>
<th>BCET</th>
<th>MCET</th>
<th>WCET</th>
<th>gT</th>
<th>lT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Outlier</td>
<td>4120.3</td>
<td>19082.0</td>
<td>6271.6</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Upper Whisker</td>
<td>124.6</td>
<td>609.3</td>
<td>139.8</td>
<td>58111.9</td>
<td>122.1</td>
</tr>
<tr>
<td>0.75 Quantile</td>
<td>80.2</td>
<td>378.1</td>
<td>110.2</td>
<td>31879.8</td>
<td>79.7</td>
</tr>
<tr>
<td>Median</td>
<td>2.3</td>
<td>260.7</td>
<td>16.2</td>
<td>24651.5</td>
<td>1.7</td>
</tr>
<tr>
<td>IQR</td>
<td>79.0</td>
<td>345.4</td>
<td>106.4</td>
<td>18174.2</td>
<td>78.5</td>
</tr>
<tr>
<td>0.25 Quantile</td>
<td>1.23</td>
<td>32.7</td>
<td>3.8</td>
<td>13705.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Lower Whisker</td>
<td>1.0</td>
<td>1.9</td>
<td>1.0</td>
<td>318.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Lower Outlier</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Table 1. Values of the backlog tightness measurement (see [5]). In this table gT stands for globalTBASCEM and lT for localTBASCEM.

<table>
<thead>
<tr>
<th>Alcuri</th>
<th>BCET</th>
<th>MCET</th>
<th>WCET</th>
<th>gT</th>
<th>lT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Outlier</td>
<td>5.2</td>
<td>5962.3</td>
<td>494.2</td>
<td>NaN</td>
<td>1.1</td>
</tr>
<tr>
<td>Upper Whisker</td>
<td>1.0</td>
<td>1637.1</td>
<td>97.3</td>
<td>61265.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.75 Quantile</td>
<td>1.0</td>
<td>813.2</td>
<td>47.7</td>
<td>60801.9</td>
<td>1.0</td>
</tr>
<tr>
<td>Median</td>
<td>1.0</td>
<td>517.9</td>
<td>26.3</td>
<td>60558.1</td>
<td>1.0</td>
</tr>
<tr>
<td>IQR</td>
<td>0.0</td>
<td>569.9</td>
<td>33.3</td>
<td>639.7</td>
<td>0.0</td>
</tr>
<tr>
<td>0.25 Quantile</td>
<td>1.0</td>
<td>243.2</td>
<td>14.4</td>
<td>60162.2</td>
<td>1.0</td>
</tr>
<tr>
<td>Lower Whisker</td>
<td>1.0</td>
<td>1.9</td>
<td>1.1</td>
<td>59582.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Lower Outlier</td>
<td>1.0</td>
<td>NaN</td>
<td>NaN</td>
<td>37874.1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 2. Values of the delay tightness measurement (see [5]). In this table gT stands for globalTBASCEM and lT for localTBASCEM.
surround the function call of each algorithm. ”Tic” starts the time measurement, and ”toc” returns the measured time in ”seconds”. The measurements were carried out on an Intel i7 processors of the 11th generation. Since, the relative time between the algorithms is of interest, the absolute times can be neglected. The absolute time is only useful if the algorithm runs on its target hardware.

Hypotheses: Since our algorithms iterates through the incoming stream, we expect a similar runtime as the global TBASCEM. This can be approached as O(n). However, the fine adjustment of the burst parameter (described in 3) rather leads to an O(n^2). So, it is probably more expected to have a longer runtime.

Experimental Setup: The setup is the same as in RQ1. Only the ”tic toc” time measurement was added. The user servers and input data streams are the same.

Results: Figure 6 shows the runtime of each algorithm at each server. The local TBASCEM algorithm shows relatively stable run times. It is mostly between 0.55 and 0.9s. Compared to the other algorithms, this is nearly by a factor 10 slower as the global TBASCEM and factor 100-1000 slower as WCET, BCET and MCET. Only the Alcuri et al. algorithm takes up to 5s. This is up to 10 times slower compared to our local TBASCEM. As stated, the absolute values have no significance. But the relative values give a good indication of the runtime. Regarding the relative reliability of the runtime, it shows robust and stable results. The absolute deviations are higher. Based on that, the local TBASCEM is not the first choice regarding speed. However, with a good robustness, it keeps up with the other algorithms.

Table 3 shows the values of the time measurement of 6.
5 Related Work

For the past years, algorithms for estimating service curves using NC have been an often-treated topic in the literature [1,12,15,10,7]. In this section, we present some examples of them that relate to our use-case of a FIFO queuing server system.

The work of Alcuri et al. [1] defines a method for the estimation of service curves for various types of systems. Among others, it includes non-First-In-First-Out ones. The algorithm divides the input and output traffic measurements into backlogged periods. These are periods when the buffer is not empty. The start time, end time, and quantity of output traffic for each backlogged period are iteratively determined. The throughput $r$ of each period is determined by the bits leaving the system divided by the duration of the period. Among all backlogged periods, a maximum estimation technique figures out the maximum throughput $r_{\text{max}}$. The delay $T$ is computed by tracing a line with a slope of $r_{\text{max}}$ at all points of the departure process. Then it is projected onto the horizontal axis. After that, the maximum delay $T_{\text{max}}$ of each backlogged period is obtained. The rate $r_{\text{max}}$ and latency $T_{\text{max}}$ describe the service curve as a rate-latency curve [1]. This approach harmonizes with the definition of strict service curves (see Le Boudec et al. in [12]).

The first method is the global TBASCEM which was already mentioned a lot and relates to the local TBASCEM. The global TBASCEM is a reverse engineering approach. It is based on linear NC equations for estimating the parameters of arrival and service curves. It achieves tighter bounds on service curves compared to the next four state-of-the-art methods. This is done by imposing constraints on the equation variables and employing non-linear optimization. While taking the whole input data stream as a whole, it assumes that maximum backlog and delay occur at the same time. [7]
In the next method, WCET is used to estimate the service curve. This was proposed by Helm et al. [10]. The service curve is calculated by utilizing the maximum measured processing latency and the minimum measured rate as the latency and rate. It needs the condition that the minimum system service rate is higher than the mean input rate. Thus, this approach fits well for hard real-time requirements. For systems with limited-length streams and playback buffers, uninterrupted streaming is possible. This is also the case if the mean system rate is not higher than the mean input rate.

To address this issue, the second approach is based on the MCET and estimates the service utilizing the measured mean rate. It calculates the system’s latency. This is done by taking into account the maximum and minimum deviations between the input flow and the mean flow, and effectively accounting for latency outliers. Helm et al. [10] inspires for this approach.

The fourth method is based on BCET. It orders the measured data with cumulative latencies in descending order for the derivation of a service curve. A tangent at the rate point of interest is traced. The intersection point with the time axis determines the system latency. But, this method could result in highly overestimated bounds. This is particularly true if patterns in the service flow exist. It can also be employed to estimate the arrival curve by sorting measured data in ascending order by the inter-arrival time. The burst parameter is determined by the intersection between the tangent and the y-axis (bytes). The sliding window approach inspires this. It is mentioned in the real-time calculus (RTC) framework by Wandeler et al. [15]. This technique proves to be highly advantageous in evaluating the feasibility of streaming a stream with a time limit.

6 Conclusions

This paper introduces the local TBASCEM methodology derived from the global TBASCEM. The corresponding algorithm offers the opportunity to compute the backlog and delay for each outgoing data packet of a data stream. The associated arrival and service curves are determined as well. It is also applicable to state-of-the-art TL from the input flow and output flow of any FIFO server queuing system. We applied the local TBASCEM method on more than 40 million data points of our HIL system. With that, we received very tight bounds. So, we found a gap for service curve estimation from measurement data producing tight bounds.

Based on our empirical investigation, the local TBASCEM method provides tighter bounds compared to other methods that can be found in the literature. It is applicable to either TL of input and output flows of a server queuing system.

The local TBASCEM RTM is usable in any streaming system to generate local arrival and service curves. This helps to identify local properties of an arrival data stream, which could be very useful to detect local events such as bottlenecks. Because of the local analysis, we do not need to assume all maximum parameters are occurring at the same time. This is an advantage compared to
the global TBASCEM. However, runtime measurements showed that the computation of the parameters and the iteration through the whole data stream takes a while. This leads to longer runtimes than most of the other estimation methods. In a subsequent work, it could be figured out the runtime behaviour in a real target HIL system to evaluate the performance also regarding other resource consumption. This would bring further insights into the practical applicability of this method. Furthermore, the results of the local TBASCEM could be extended to stochastic NC to apply a formal analysis of the overall system. But for that the method has to be extended.

As mentioned above, local events can be detected easily. Another piece of follow-up work could involve defining rules for local bottleneck detection. In a live system, this could help to alarm in case of a bottleneck and could lead to consequent action, such as throttling the incoming data stream or allocating buffer space.

References


### Appendix

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_r$</td>
<td>overall arrival rate</td>
</tr>
<tr>
<td>$T_{in}(i)$</td>
<td>incoming timestamp of i-th packet in whole data stream</td>
</tr>
<tr>
<td>$\text{packet_size}$</td>
<td>Size of a packet in byte</td>
</tr>
<tr>
<td>$j$</td>
<td>index of a window</td>
</tr>
<tr>
<td>$B_j$</td>
<td>backlog in a window $j$</td>
</tr>
<tr>
<td>$b_j$</td>
<td>burst in window $j$</td>
</tr>
<tr>
<td>$T_{j-in}(i)$</td>
<td>incoming timestamp of i-th packet inside window $j$</td>
</tr>
<tr>
<td>bytes(T)</td>
<td>expected number of bytes at the given point of time</td>
</tr>
<tr>
<td>$\text{min}(\Delta b_{pos})$</td>
<td>minimum positive difference if the expected number of arrived bytes exceeds the actual number of arrived bytes inside a window</td>
</tr>
<tr>
<td>$\text{max}(\Delta b_{neg})$</td>
<td>maximum negative difference if the actual number of arrived bytes exceeds the expected number of arrived bytes inside a window</td>
</tr>
<tr>
<td>$D_{j\text{max}}$</td>
<td>maximum delay in window $j$</td>
</tr>
<tr>
<td>$L_j$</td>
<td>latency in window $j$</td>
</tr>
<tr>
<td>$s_{r_j}$</td>
<td>service rate in window $j$</td>
</tr>
</tbody>
</table>

**Table 4.** Mapping table of server numbering to timestamps
Definition of Terms and Acronyms

**BCET** Best-Case Execution Time (BCET) is the lowest observed execution time of a software process running on a dedicated computing machine. 10

**DUT** Device Under Test (DUT) is the technical device, what is integrated into the hardware-in-the-loop simulator and stimulated by measurement data from the real-world device or by simulation. 1, 2, 4, 5, 10

**FIFO** First-In, First-Out (FIFO) refers to a principle where the first item to enter a system or queue is the first to be processed or served. 1, 14, 15

**HIL** Hardware-in-the-Loop (HIL) test system or simulator or test bench is a methodology and a technical system for testing and validation of a technical product. See 2 and 13 for details. 1–6, 9, 10, 15, 16

**MCET** The Mean-Case Execution Time (MCET) is the mean or average observed execution time of a software process running on a dedicated computing machine. 10, 15

**NC** Network Calculus (NC) is a system theoretical approach for calculating delay and backlog bounds by min-plus algebra. 1–5, 10, 14, 16

**RTM** Run-Time Measurement (RTM) approach are measurement methods for performance evaluation of software during system operation. In this paper mainly used to measure the maximum end-to-end delay and backlog between the input and output flow of a message stream in a software service process with queues in between. Additionally, the maximum burst parameter of the input and output flow is estimated. 9, 15

**TL** Timestamp Logging (TL) is a state-of-the-art method for performance evaluation of software timing. Timestamp Logs (TL) are the respective data-files generated by the logging. In this paper mainly used to measure the input and output flow of a message stream in a software service process with queuing. 1–3, 5, 9, 10, 15

**WCET** Worst-Case Execution Time (WCET) is the highest observed execution time of a software process running on a dedicated computing machine. 4, 6, 10, 15