Automated Interpretation and Reduction of In-Vehicle Network Traces at a Large Scale

Artur Mrowca, Thomas Pramsohler
Bayerische Motoren Werke AG, Germany
artur.mrowca@bmw.de

Sebastian Steinhorst, Uwe Baumgarten
Technical University of Munich, Germany
sebastian.steinhorst@tum.de

ABSTRACT

In modern vehicles, high communication complexity requires cost-effective integration tests such as data-driven system verification with in-vehicle network traces. With the growing amount of traces, distributable Big Data solutions for analyses become essential to inspect massive amounts of traces. Such traces need to be processed systematically using automated procedures, as manual steps become infeasible due to loading and processing times in existing tools. Further, trace analyses require multiple domains to verify the system in terms of different aspects (e.g., specific functions) and thus, require solutions that can be parameterized towards respective domains. Existing solutions are not able to process such trace amounts in a flexible and automated manner. To overcome this, we introduce a fully automated and parallelizable end-to-end preprocessing framework that allows to analyze massive in-vehicle network traces. Being parameterized per domain, trace data is systematically reduced and extended with domain knowledge, yielding a representation targeted towards domain-specific system analyses. We show that our approach outperforms existing solutions in terms of execution time and extensibility by evaluating our approach on three real-world data sets from the automotive industry.

KEYWORDS

data-driven verification, data mining, big data, automotive, in-vehicle network traces, trace analysis

1 INTRODUCTION AND RELATED WORK

Many functions in modern vehicles communicate on common channels across multiple Electronic Control Units (ECUs). Also, cars are developed by multiple domains, have over 100 million lines of source code on-board, and up to 15 ECUs communicating per function with 2 million messages per minute. This results in high system complexity. To verify such systems during development, integration tests such as off-board analyses of massive network traces (e.g. at BMW Group 500 cars produce 1.5 TB per day), are used (see Fig. 1). However, due to the increasing amount of recorded data, such traces become increasingly difficult to analyze with existing tools such as Wireshark [3]. Instead, Big Data frameworks, such as Apache Spark, become essential for analysis of such traces. Further, high test coverage requires various domains to investigate traces in terms of different aspects. Thus, a parameterizable analysis approach for traces is required that yields a format that can be used for inspection. E.g. function specialists might focus on functional, while communication analysts might be interested in channel-level information. Also, for low costs, extraction needs to be performed automatically on all traces. No existing solution covers all of those aspects combined. Thus, we propose the first distributable preprocessing framework for in-vehicle traces that requires one-time parameterization, automatically producing a domain-specific representation for further analyses on massive amounts of traces. Designing such a framework has the following challenges.

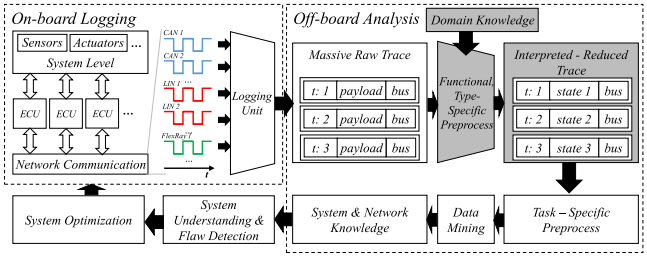

Figure 1: Data is recorded on-board of vehicles and stored in common traces. Those are analysed off-board by various domains on different aspects which requires domain-specific and automated processing. The grey shaded parts represent the scope of our contribution.

Handling massive data: Massive traces require a scalable design including effective bytes to signal mappings. *Handling heterogeneous data:* Over 10 000 signal types are verified, data is recorded as bytes using different protocols and data types. This requires per-signal analyses to achieve a common (=homogeneous) data format over all signals for further analysis. *Parameterization towards specific domain:* Per domain, different granularities and contents are analyzed. Thus, parameterization in terms of reduction and association of meta-data with domain knowledge, such as knowledge from documentation, needs to be included automatically. *Preserving determinism:* As we focus on systematic analyses such as fault diagnosis, unlike approximating approaches, our approach aims to produce replicable results for analyses. *Exploiting reduction potential:* Trace data is highly redundant and exploitable for lossless reduction. E.g., information is sent cyclically without changes. However, important state changes such as violations of cycle times need to be preserved.

Related Work: As little comparable frameworks exist, related work includes work on in-vehicle networks and user-aided trace analysis frameworks.

In [13], a framework is presented where vehicle data is collected from test vehicles at a large scale and analyzed by experts for faults. There, the focus is on rule-based visual error-diagnosis from data subsets while we emphasize the task-specific preprocessing of data from traces, where tasks refer to one aspect that is to be analyzed (e.g. analyzing the behavior of the wiper position). Tools such as Wireshark [3] or other OEM's in-house tools [5] are monitoring software that allow for offline analyses of recorded traces. Similar to our solution, they allow to inspect traces by mapping protocol information onto traces. However, they are not designed for massive traces and, unlike our method, unable to be distributed. Also, our framework allows for reductions, extensions and data-dependent processing, while those tools are for monitoring only. In [11], an approach to identify specifications from simulation traces and, in [8], a scalable specification mining tool for general digital circuits is presented. Similar to our approach, automated preprocessing of trace data is performed. However, those works focus on extracting specifications while we aim on the generalizability of

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196000

preprocessing as a basis for various tasks. Detecting faults with CAN signals was proposed in [6] and [4] where signals are partitioned into segments and features are extracted. In terms of preprocessing, the authors of [1] introduced a clustering approach for vehicular-sensor data to optimize it for Data Mining. That work, unlike our work, only covers the reduction aspect of our framework and uses clustering for reduction. In [12], the authors aggregated vehicle signal data to predict compressor faults. Different to us, their preprocessing uses aggregated or subsets of numerical signals. In the software context, user-involved analysis frameworks for fault detection are proposed such as approaches for application failures [10] and for problem detection in internet services [2]. Those works focus on analyses aimed towards certain Data Mining tasks and with or without user interactions for simplification, while we focus on automated extraction and reduction of in-vehicle data at a large scale. Our Contributions are as follows.

- Existing frameworks for in-vehicle trace analysis are targeted towards specific tasks, such as specification mining, or are not designed for massive amounts of data, as data is processed on one machine. We, thus, introduce the first distributable general end-to-end preprocessing framework that handles massive amounts of in-vehicle traces.
- Inspection of relevant signals in existing solutions requires several manual steps such as loading, filtering or merging data from different channels. We automate those steps in our approach such that a common (homogeneous) data format results, allowing for direct further analyses in terms of various aspects such as fault inspection.
- Processing of massive traces is inflexible in existing solutions as they are preprocessed for specific aspects only. However, our framework is designed highly parameterizable to allow for general definitions of extensions and reductions while being executed directly within a distributed Big Data framework. With this, our approach paves the way for general data analysis on massive traces, ultimately lowering costs and optimizing test coverage.

We first introduce a formalization of traces in Sec. 2 before presenting a distributed approach to message-to-signal mapping and domain specific selection in Sec. 3. Next, the parameterization in terms of extensions and reductions are discussed together with homogenization in Sec. 4. The approach is evaluated in Sec. 5.

2 NETWORK TRACES

In this section we introduce a formalization that is used as a basis to present our framework. In order to execute a functionality (e.g., the wiper) ECUs exchange messages on common channels in in-vehicle networks. Per channel, various protocols are used to transmit information from sensors, actuators or ECUs which are logged via monitoring devices and stored in a common trace K_b for off-line analyses (see Fig. 1).

Formal Definitions: For any protocol, communication is performed via defined messages of type m assigned to a unique identifier m_id. Each occurrence of a message type at a point in time is a message instance $\hat{m}$. Similarly, each signal type s has an identifier s_id and occurrences $\hat{s}$. All existing vehicle signal types are denoted as alphabet $\Sigma = \{s_1, s_2, ..., s_z\}$. Each message type has a set of defined signal types $S \subseteq \Sigma$ that each instance of it carries and the channel on which it occurs b_id. I.e. $m = (S, m_\text{id}, b_\text{id})$ where $|S|$ can vary per message. Each signal instance $\hat{s} = (v, s_\text{id})$ has a value v at its time of occurrence and a type assigned via s_id. Per s_id information on either a function (e.g. steering angle), a control unit (e.g. reset) or the network (e.g. frame qualifier) is exchanged. In the example on Fig. 2 there is a type m' with $m_\text{id} = 3$ holding all signal types of the wiper function in S' and is sent on the FA-CAN bus with $b_\text{id} = \text{FC}$. There $S' = (s_\text{wpos}, s_\text{wvel})$ has s_wpos as the signal type wiper position and s_wvel as type wiper velocity. A message instance $\hat{m}'$ of type m' occurs at a certain point in time such as $\hat{m}' = ((\hat{s}_\text{wpos}, \hat{s}_\text{wvel}), 3, \text{FC})$ with $\hat{s}_\text{wpos} = (45°, wpos), \hat{s}_\text{wvel} = (1\ \text{rad/min}, wvel)$.

When recorded from the channel, data is given as an ordered byte sequence $K_\text{b} =< k_{\text{b}1}, k_{\text{b}2}, ..., k_{\text{b}w} >$ with $|K_\text{b}| = w$, byte tuple $k_{\text{b}j} = (t_j, l_j, b_{\text{id}j}, m_{\text{id}j}, m_{\text{info}j})$ with l as message payload in byte format, b_id as channel identifier, m_id as message identifier and m_info as protocol specific message fields used for protocol specific translation (e.g. DLC). In CAN, m_id is the CAN identifier. Each payload l can be translated to all signal instances $\hat{S}$ it contains using its unique identifier m_id. Thus, after interpretation K_b can be written as:

$$K_\text{n} =< (t_1, \hat{m}_1), (t_2, \hat{m}_2), ...(t_w, \hat{m}_w) >= \\ < (t_1, (\hat{S}_1, \hat{m}_{\text{id}1})), (t_2, (\hat{S}_2, \hat{m}_{\text{id}2}), ...(t_w, (\hat{S}_w, \hat{m}_{\text{id}w}) >$$

If all signal instances $\hat{s}$ are extracted separately from $\hat{S}$, per time of occurrence, this can be written as:

$$K_\text{s} =< (t_1, \hat{s}_{11}, b_{\text{id}1}), (t_1, \hat{s}_{12}, b_{\text{id}1}), ...(t_w, \hat{s}_{w1}, b_{\text{id}w}), ... >$$

Per signal type, using s_id all $\hat{s}_{ij}$ of same type can be grouped as each type represents one sent information such as $K_\text{s}^{s_\text{id}=wvel} = \sigma_{s_\text{id}=wvel}(K_\text{s})$. An example of this concept together with all formalizations is shown on Fig. 2. We use this formalism to present our framework. Our approach transforms raw byte sequences to extended, interpreted and domain-specific homogeneous representations. Its main steps are structuring, interpretation, user-defined extension and reduction. The detailed algorithm for this approach is defined using relational algebra and presented in Algorithm 1. An overview of our framework is given in Fig. 3. The main steps are described in the following sections.

Algorithm 1:
Reduction, Interpretation and Homogenization of raw Byte sequences
Input: ***trace*** K_b***, preselected interpretation rules*** U_{rel}***, reduction rules*** C***, extension rules*** E
Output: ***homogeneous, reduced, interpreted Sequence*** R_{out}

$R_\text{out} = \emptyset$ ▷ Output
$U_\text{comb} \subseteq U_\text{rel}$ ▷ Preselection
$K_\text{pre} = \sigma_{(m_\text{id}, b_\text{id}) \in U_\text{comb}}(K_\text{b})$
$K_\text{join} = K_\text{pre} \bowtie_{K_\text{pre}.b_\text{id}=U_\text{rel}.b_\text{id} \cap K_\text{pre}.m_\text{id}=U_\text{rel}.m_\text{id}} U_\text{rel}$ ▷ Interpretation
$K_\text{join2} = \mathcal{F}_{u_1}(K_\text{join})$
$K_\text{s} = \mathcal{F}_{u_2}(K_\text{join2})$
for each $s_i^* \in \Sigma^*$ **do** ▷ signal Splitting
  $K_\text{s}^{s_i^*\text{id}} = \sigma_{K_\text{s}.s_\text{id}=s_i^*\text{id}}(K_\text{s})$
  $(K_{\text{sep}_i}, K_{s_\text{cor}}^{s_i^*\text{id}}) = e(K_\text{s}^{s_i^*\text{id}})$
  $K_{\text{cond}_i} = K_{\text{sep}_i} \bowtie_{K_{\text{sep}_i}.s_\text{id}=C.s_\text{id}} C$ ▷ Reduction
  $K_{\text{red}_i} = \sigma_{K_{\text{cond}_i}.e=true}(K_{\text{cond}_i})$
  $W_i = \mathcal{F}_E(K_{\text{red}_i})$ ▷ Extension
  $\tau = type(K_{\text{red}_i}, Z)$ ▷ Type-dependent processing
  if τ is α **then**
    $(K_{\text{num}_i}, K_{\text{nom}_i}) = typeSplit(K_{\text{red}_i})$
    $(K_{\text{num}_{i\text{out}}}, K_{\text{num}_{i\text{rep}}}) = outlier(K_{\text{num}_i})$
    $K_{\text{num}_{i\text{clean}}} = m(K_{\text{num}_{i\text{rep}}})$
    $K_{\text{res}_i} = K_{\text{num}_{i\text{out}}} \cup K_{\text{num}_{i\text{clean}}} \cup K_{\text{nom}_i}$
  end if
  if τ is β **then**
    $(K_{\text{F}_i}, K_{\text{V}_i}) = functionSplit(K_{\text{red}_i})$
    $(K_{\text{F}_{i\text{out}}}, K_{\text{F}_{i\text{clean}}}) = outlier(K_{\text{F}_i})$
    $K_{\text{F}_{i\text{clean}}} = addGradient(K_{\text{F}_{i\text{clean}}})$
    $K_{\text{res}_i} = K_{\text{F}_{i\text{out}}} \cup K_{\text{F}_{i\text{clean}}} \cup K_{\text{V}_i}$
  end if
  if τ is γ **then**
    $K_{\text{res}_i} = K_{\text{red}_i}$
  end if
  $R_\text{out} = R_\text{out} \cup K_{\text{res}_i} \cup W_i$ ▷ Merge
end for

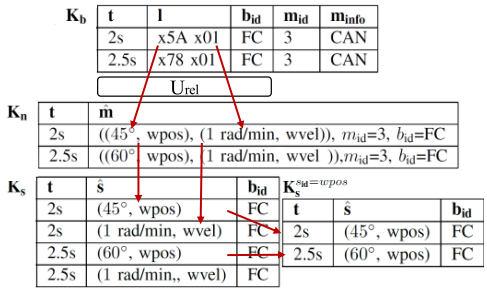

Figure 2: Wiper function: Example of the formalization used. Assuming l' to be the first two bytes (=wpos) per message and l'' the last two bytes (=wvel) the rules for mapping between K_b onto K_n are $v = 0.5 \cdot l'$ and $v = l''$.

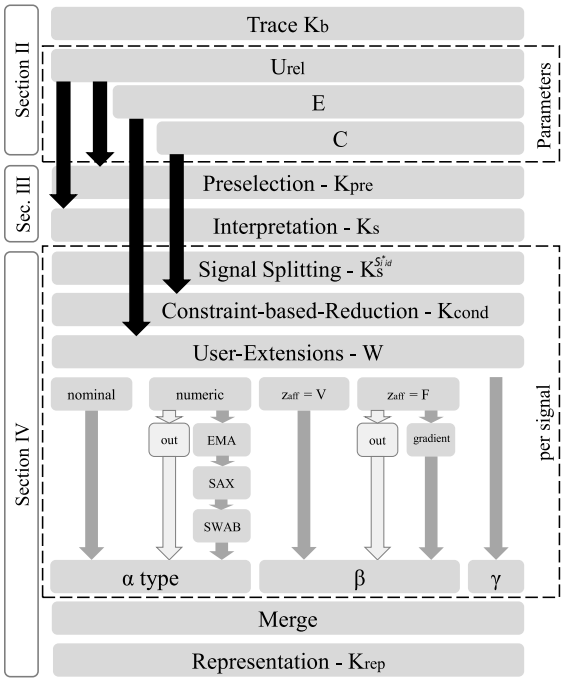

Figure 3: Overview of the flow of our framwork with the respective sections detailing the process stated.

3 INFORMATION EXTRACTION

Different domains require the analysis of different signal subsets. Thus, one main goal of our approach is an automated and parameterizable way for each domain to extract representations that contain signals relevant to it. For this, first, per-domain, signals K_s relevant to the analyzing domain need to be extracted from the raw trace K_b (see Fig. 2). When dealing with small traces, this mapping can be performed sequentially on one machine which is infeasible for massive traces due to computational limitations. Thus, processing in Big Data frameworks is required. To be able to operate in a distributed manner, they require tabular operations for signal extraction. They are run as batch and are ideally suited to be parameterized individually per-domain to automatically extract relevant representations per domain. In current tools there is no automated way to retrieve such relevant signals. Instead, such tools parse sequentially through all messages of a trace and lookup each signal per message. Our framework solves this using a distributed approach, enabling automated domain-specific large-scale analyses of large fleets of vehicles. To efficiently map K_b to a relevant K_s here, first, a parameterization input is proposed that specifies relevant signals to extract. Next, an efficient tabular approach to interpretation of relevant signals is presented. Interpretation cost is kept low as relevant messages are filtered prior to interpretation.

3.1 Structuring and Preselection

Multiple signal types are transmitted in a trace. Translating all signal instances in all message instances is inefficient, as usually per domain a subset of bytes per message is of interest. Those signals are documented and known per domain.

Structuring: For efficient retrieval of signal values $\hat{s}$ from K_b a set $U_{comb} \subseteq U_{rel}$ of translation tuples u_{rel} needs to be input to the algorithm (with U_{rel} as all signals possible). An expert inspecting the Wiper might select the position wpos and velocity wvel as U_{comb}. Each signal to extract has format $u_{rel} = (s_{id}^{rel}, b_{id}, m_{id}, u_{info})$, where u_{info} contains information needed for extraction s_{id}^{rel} ids relevant to the analysis (wpos and wvel) and b_{id} and m_{id} as all according channel and message ids on which the signals occur.

Preselection: To perform less interpretations, reductions need to be performed directly on K_b, which is done by filtering K_b for all m_{id}s and b_{id}s of signals in U_{comb}. E.g., to extract wpos and wvel in Fig. 2, those signals would be chosen as U_{comb}. With this, all rows in K_b, with $m_{id} \in (3)$ and $b_{id} \in (FC)$ would be filtered, resulting in a byte sequence with only relevant message types K_{pre}. A simple example for U_{rel} (same schema as U_{comb}) is given in Table 1. Thus, per-domain, U_{comb} needs to be given as parameter.

s_{id}^{rel}	b_{id}	m_{id}	u_{info}
wpos	FC	3	Int.rule: $v = 0.5 \cdot l$; rel.B = l' = (1,2)
wvel	FC	3	Int.rule: $v = l$; rel.B = l' = (3,4)
wtype	K-LIN	11	Int.rule: $v = l + 2$; rel.B = l' = (1)
wstat	SOME/IP	212	Int.rule: $v = l$; rel.B = l' = (10,22)

Table 1: Example for U_{rel} with relevant bytes to extract: Bytes 1 and 2 for wpos in messages with id 3, Bytes 3 and 4 for wvel. From SOME/IP the wiper status wstat and from K-Lin the wiper type wtype could be extracted from messages with respective ids 11 and 212. I.e. our approach allows to combine multiple protocols into this extraction.

3.2 Information Interpretation

For efficiency, per message, interpretation is performed on relevant bytes only. Also, it requires scalable database operations and a format that allows for row-wise extraction of signal instances with interpretation rules u_{rel}. Consequently, we propose the following interpretation format and procedure.

Approach: As discussed above, tabular operations are required for distributed processing of massive traces in Big Data frameworks. In particular, interpretation needs to be made processible per row for allocability of this computation. This is achieved as follows. Starting with the traces K_{pre}, the signals we want to extract and extraction information per signal in U_{comb}, we aim to map K_{pre} to K_s efficiently (mapping format in Fig. 2 from K_b to K_s; algorithm lines 4 - 6). As it allows for parallelism, an effective approach is to join all extraction instructions in U_{comb} on (m_{id}, b_{id}) (same table schema as U_{rel}) with all its according raw messages in K_{pre}, resulting in a table K_{join} with columns $(t, l, m_{info}, s_{id}^{rel}, b_{id}, m_{id}, u_{info})$. Thus, all u_{info} of all signals to extract are in a row with its corresponding messages and signals in K_{pre}. With rules given in u_{info} per row of K_{join}, a mapping u can be applied to extract the signal value per trace element. As shown in Table 1, u_{info} needs to contain the byte positions at which the signal value v can be found in the payload l and how it can be evaluated to the signal value v. This value v can be trivial linear mappings or complex rules and is required in protocols such as SOME/IP, e.g., rules where values of preceding

bytes define the presence of a signal type in succeeding bytes. Also, evaluation information need to be given, such as data types, coding, protocol based fields and translation rules (e.g. mapping of a Hex to a categorical value). Thus, given K_{join} (row-wise) first, the relevant payload bytes l_{rel} are extracted using $u_1 : (l, u_{\text{info}}) \mapsto l_{\text{rel}}$ giving K_{join2} before then, the interpretation information is used to get signal values using $u_2 : (l_{\text{rel}}, m_{\text{info}}, u_{\text{info}}) \mapsto k_s = (t, s) = (t, (v, s_{\text{id}}))$ resulting in K_s . Existing tools do not support extraction of signals in massive traces as this operation is performed sequentially. Our approach allows to process large traces by exploiting tabular operations (e.g. join, filter) in order to be distributable. Further, to keep memory efficiency high (which is required in large scale systems) we store traces in raw format K_b which is more efficient than translating all K_b to K_s as, e.g., per CAN message 8 bytes could contain 8 signals which would result in a K_s of 8 times the size of K_b. At the same time, we allow for automated signal extraction for relevant aspects when needed. In contrast to that, existing tools need to ingest traces and cache traces of all journeys under inspection for each analysis performed, which can take up to 2 hours per journey. However, once parameterized, our approach allows to extract signals from multiple journeys of massive size in comparably short time.

4 DATA TRANSFORMATION

To be able to analyze relevant data, a reduced format targeted towards the analyzing domain is required. Extension (i.e. extraction of new information from given signals) and association of meta-data is required to add system knowledge during preprocessing. Those steps need to be parameterizable and integrated in a tabular approach to allow for distributed computation on massive traces in order to be executable in an automated manner. At the same time, we aim to provide a common data format as a result of our framework. This (for each domain), first, enables to directly inspect data and, second, to directly apply consequent analyses with Big Data frameworks onto this representation. Thus, we present a targeted reduction and extension approach that is parameterized with domain knowledge resulting in a homogeneous representation.

4.1 Reduction Technique

Traces can be reduced by removing repeated data points, by mapping multiple trace segments on a representative symbol, by clustering or by using sampling techniques. We offer a general description formalization that allows to use such techniques as part of our framework by defining them formally. For this, a condition set is specified by the analyzing entity (e.g. expert) for reduction. In particular, automotive traces contain characteristics that can be well exploited for reduction (e.g., communication patterns, sending conditions).

Signal Splitting: As processing needs to be done per signal type, in line 8, we split K_s by its remaining signal types $\Sigma^* = \{s_1^*, s_2^*, ..., s_r^*\}$, with $\Sigma^* \subseteq \Sigma$ giving $K_s^{s_i^* \text{id}}$ as one sequence per signal type. An example for such a sequence after splitting is the position of the wiper at each time $K_s^{s_{\text{id}}=wpos}$ (Fig. 2). When signals are forwarded through gateways they are recorded multiple times in the trace. Thus, by exploiting that identical signal instances are routed on multiple channels computational cost is reduced by processing signal instances for one channel only and using the result for corresponding signal instances. For this to be valid, in line 9, equality of signal instances is checked with $e : K_s^{s_i^* \text{id}} \mapsto (K_{s_{\text{rep}}}^{s_i^* \text{id}}, K_{s_{\text{cor}}}^{s_i^* \text{id}})$ resulting in a representative sequence $K_{s_{\text{rep}}}^{s_i^* \text{id}}$ and a set $K_{s_{\text{cor}}}^{s_i^* \text{id}}$ of corresponding sequences $K_{s_{\text{cor}}}^{s_i^* \text{id}}$. For better readability, we abstract each sequence as $K_{\text{sep}} := K_{s_{\text{rep}}}^{s_i^* \text{id}}$.

numeric

t	ŝ	$\mathbf{b_{id}}$
2s	(45°, wpos)	FC
2.5s	(60°, wpos)	FC
...	...	...

ordinal

t	ŝ	$\mathbf{b_{id}}$
2.1s	(high, heat)	K-LIN
2.7s	(medium, heat)	K-LIN
...	...	...

nominal

t	ŝ	$\mathbf{b_{id}}$
1.0s	(driving, state)	DC
50.1s	(parking, state)	DC
...	...	...

binary

t	ŝ	$\mathbf{b_{id}}$
1.4s	(ON, belt)	FC
22.2s	(OFF, belt)	FC
...	...	...

Figure 4: Example of four K_{red} each consisting of signal instances of one signal type with four different data types. Those signals need to be processed based on their data type.

Constraint Reduction: For efficient reduction, we exploit above characteristics by marking and filtering relevant elements of K_{sep}. To mark a set of constraints, we use $C = \{c_i | 0 < i < m, i, m \in \mathbb{N}\}$ and $c = (s_{\text{id}}, d, F)$, where s_{id} defines the sequence for which c is applied. If d is true, all functions $f \in F$ are applied. f can be a row-wise or an aggregation operation which are inherently distributable in Big Data systems. For instance, f could compute the temporal gap between subsequent rows and evaluate to *true*, if this gap exceeds a defined cycle time. Those constraints are applied in line 10 by joining C with K_{sep} on their signal type to get K_{cond}. Then, we apply all $f \in F$, if d holds yielding value e as follows:

$$f : k_{\text{cond}}.e = \begin{cases} true & \text{if } \exists f_i \in F, \text{ with } f_i(k_{\text{sep}}) = true \\ false & \text{otherwise} \end{cases} \quad (1)$$

If any f is true, so is e. Thus, in line 11, K_{red} is found by filtering for f being false leaving task-relevant elements only.

Extension Rules: For further knowledge we associate meta-data (e.g., expected cycle times) to traces by adding relevant information $W = \{w_1, w_2, ...\}$ to the sequence. Applying a function on each K_{red} in line 12 adds meta-data such as the gap to previous elements or results from computations based on other signals to K_{red} as new sequence elements $\hat{w}$. For this, we define w similar to signal types as $w = (v, w_{\text{id}})$ with v as value, w_{id} as identifier associating w to the corresponding signal type and $\hat{w}$ as instance (example shown in Table 2).

t	$\hat{\mathbf{w}} = wposGap$	$\mathbf{b_{id}}$
2s	(0.5, wposGap)	FC
2.5s	(0.4, wposGap)	FC
2.9s	(0.45, wposGap)	FC
...	...	...

Table 2: Extension: Gap between wpos signals from sequence $K_s^{s_{\text{id}}=wpos}$.

4.2 Type-Dependent Processing

The K_{red} are of various data types (examples in Fig. 4). Further, we aim to homogenize all traces to a common data format in order to enable low inspection costs and further automated analyses. This is done by symbolizing sub sequences K_{red} dependent on their data type which is described in this section. For this classification of each sub trace and subsequent type-dependent processing is required. The resulting symbolized sub sequences are merged to a common representation which is described in Sec. 4.3. Criteria for classification were determined through inspection of more than 1000 signal types and three processing classes of signals were found. Each signal instance sequence K_{red} is classified using criteria $Z = (z_{\text{type}}, z_{\text{rate}}, z_{\text{num}}, z_{\text{val}})$. The data type $z_{\text{type}} \in \{S, N\}$ is String S or Numeric N. The affiliation $z_{\text{aff}} \in \{F, V\}$ distinguishes between values that express a functional property F and the ones

z_{type}	z_{rate}	z_{num}	z_{val}	Data Type	Processing Branch
N	H	> 2	$true$	numeric	α
N	L	> 2	$true$	ordinal	β
S	$H \cup L$	> 2	$true$	ordinal	β
S	$H \cup L$	$= 2$	$true$	binary	γ
S	$H \cup L$	> 2	$false$	nominal	γ
N	$H \cup L$	$= 2$	$true$	binary	γ

Table 3: Map signal instance sequences to data type and processing branch.

that define validity V of either the sent message (e.g. message invalid), the signal instance (e.g. signal invalid) or a functional component (e.g. object invalid). For numeric values we use

$$z_{\text{rate}} = \begin{cases} H & \text{if } \frac{n}{\Delta t} > T \\ L & \text{otherwise} \end{cases} \tag{2}$$

to differentiate between values changing at a high rate H and slow rate L, using a threshold T, that is determined by domain knowledge. z_{rate} is the number of values n in active segments of duration Δt. z_{num} is the number of different values of a functional property information. If signal values of a type contain a comparable valence is defined as $z_{\text{val}} \in \{true, false\}$; $z_{\text{num}} \in \mathbb{N}$. Using Z each K_{red} is assigned a processing branch according to Table 3.

In Branch α, numerical and nominal elements are processed by removing outliers and applying smoothing, segmentation, trend estimation with SWAB [7] and symbolization with SAX [9] (= mapping m) giving a tuple of trend and symbol per segment. Then, the outliers as potential errors are merged back to form K_α. A similar approach is used for ordinal values in Branch β, where K_{red} is split on each element using z_{aff} to get a nominal part K_{V} where V holds and a functional part K_{F} where F holds. K_{F} is translated into a numerical equivalent, analyzed for outliers and the trend is determined using the gradient. Outliers are stored and merged similar to α. In Branch γ, if less values are given, no transformation is needed and all values are treated as nominal values with a similar splitting as in β. With this approach, all signal types are automatically transformed into a common data format that can be merged to a common state representation that allows for a common processing of all sequences in further analyses which is described in Sec. 4.3.

4.3 Representation

All resulting signal instances K_α, K_β and K_γ are merged to form a common sequence K_{rep} of unified shape.

State Representation: Different representations can be formed from K_{rep}, while for Data Mining, (especially fault diagnosis) an adequate format is exemplified in Table 4. It shows each signal type as column and a value at all time stamps where its value occurred. Signal types without values at that time are filled with the value of its last occurrence. Each row resembles the state of all signal instance at a time. This representation is formed efficiently as concatenation of signal instances during merging and lag operations are scalable database operations. A main benefit of this representation is that subsequent automatic Data Mining approaches can be directly applied.

4.4 Applications

The generated state representation can be directly used for domain-specific error inspection or various subsequent analyses. Possible applications to verification are the following.

- Outliers as potential errors are automatically discovered with our framework which allows to check the state of the car when the outlier occurred and the chain of states prior to it. Thus, the cause of an error can be isolated. By enriching the trace with extensions, further error types can be analyzed. For instance, by extending traces with expected cycle times, locations of violations of such times can be detected.

t	$s_{\text{headlight}}$	$s_{\text{levercontrol}}$	s_{speed}	$s_{\text{indicatorlight}}$	$s_{\text{lightswitch}}$
2	off	default	(high,increasing)	off	default
4	off	**pushed up**	(high,increasing)	off	default
4.25	off	pushed up	(high,increasing)	**left on**	default
7	off	**default**	(high,increasing)	left on	default
7.22	off	default	(high,increasing)	**off**	default
14	off	default	**(high,steady)**	off	default
20	off	default	(high,steady)	off	**turned halfway**
20.1	**parklight on**	default	(high,steady)	off	turned halfway
22	parklight on	default	**outlier** $v = 800$	off	turned halfway
23	parklight on	default	(high,steady)	off	**turned full**
23.5	**headlight on**	default	(high,steady)	off	turned full

Table 4: Exemplary state representation of signal instances of the function lights combined with driving speed.

	SYN	LIG	STA
# signal types	13	180	78
# signal types - α	6	27	6
# signal types - β	4	71	1
# signal types - γ	3	82.	71
# examples	13,197,983	12,306,327	4,807,891
$\varnothing$ signal types per message	1,47	5.11	3.66

Table 5: Statistics of our three data sets.

- Association Rule Mining can be used to detect IF-THEN rules, when each row is considered an item-set and columns are used as antecedents. This allows to inspect causes of errors by inspecting rules such as "IF $T < -10$ and WiperActivated THEN WiperErrorBlocked".
- Transition graphs can be generated that allow for visual inspection of error causes and event chains prior to errors. This can be defined by linking all rows of the state representation to its consequent row and aggregating the number of times a transition occurred. With this, rare transitions indicate potential errors and error causes are isolated through path analysis.
- Using Anomaly Detection, hot-spots can be detected in large databases. Detected anomalies can be ranked in terms of severity and presented to the developer or can automatically be transformed into extensions w to detect similar anomalies in further runs.

5 EXPERIMENTAL RESULTS

Next, we evaluate our approach in comparison to an existing solution. We show that it outperforms state-of-the-art tools in terms of computation time required for signal extraction and that Big Data solutions are essential for massive traces.

5.1 Performance Analysis

Setup: We evaluate lines 3 to 11 of Algorithm 1 using Apache Spark on a cluster with 70 servers, Infiniband QSFP, 20 virtual CPUs, per node two Intel ® Xeon ® processors E5-2680v2 with 256 GB DDR3 RAM using 5 virtual CPUs and 10 servers with 3 GB RAM per executor and 4 GB RAM per driver node. The remaining part of the algorithm is not considered, as methods used there are evaluated in respective literature, e.g., SWAB Algorithm [7]. Signal instances are often sent repeatedly without change of values. Thus, identical subsequent signal instances are removed as reduction (line 10). Also, one channel per signal type is analyzed. Three representative data sets are inspected, all recorded from one modern premium vehicle during 20 hours of driving. Its statistics are shown in Table 5. From this trace, per data set, message instances are extracted such that all message instances in K_{b} contain at least one signal instance of the corresponding data set. The *LIG* data set $K_{\text{b}}^{\text{lig}}$ has signal types $U_{\text{comb}}^{\text{lig}}$ exchanged within all light functions (e.g. light brightness), the *STA* data set (with $K_{\text{b}}^{\text{sta}}$ and $U_{\text{comb}}^{\text{sta}}$) has signal types about the car's state and the synthetic data set *SYN* ($K_{\text{b}}^{\text{syn}}$ and $U_{\text{comb}}^{\text{syn}}$) is generated from 13 representative signal types from different functions. For *SYN*, the first 3.9 million entries were

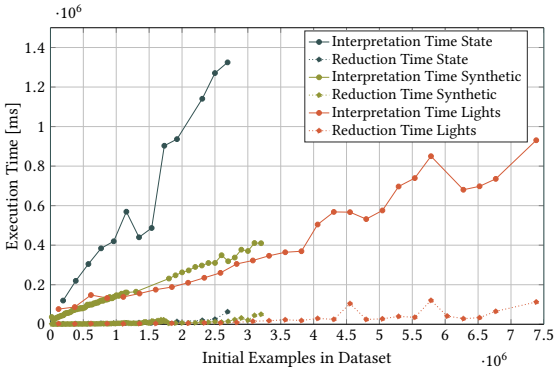

Figure 5: Execution time after interpretation and reduction.

Journeys	Trace rows $\cdot 10^9$	Extracted rows $\cdot 10^6$	# Extracted signals	Extraction time Proposed [min]	Extraction time in-house [min]
1	0.481	12.751	9	9.58	41.66
1	0.481	79.466	89	168.05	41.66
7	4.286	94.013	9	62.00	372.88
7	4.286	586.124	89	183.25	372.88
12	5.901	133.619	9	87.62	504.27
12	5.901	833.066	89	269.65	504.27

Table 6: Signal extraction times for massive traces.

used while for *STA* and *LIG* all examples are evaluated.

Execution performance: We run our approach with a constant number of signal types and step-wise increase a subset of K_b. Per data set, all signal types are extracted giving one K_{red} per signal type. I.e., in each data set, all entries are interpreted. As Fig. 5 shows, more examples increase execution times. As interpretation is performed row-wise, more examples are processed per node. Processing has complexity $O(n)$, yielding a linear curve. The curve shows fluctuations as cluster computation is communicated during distribution. Further fluctuation occurs as added message instances may contain different numbers and types of signal instances that are processed and interpreted type-dependent. It can be seen that interpretation is expensive in terms of execution time and, thus, early reduction is required in our approach. Our approach yields reasonable results for *STA* and *LIG*. E.g., interpretation of 2.6 million examples is performed in 1324 seconds and 7.4 million examples in 930 seconds. We restricted our computational power to 10 nodes yielding a lower bound of execution performance.

Comparison: We compared our framework to an OEM's in-house tool [5] (comparable to Wireshark) on an HP™ Z-840 equipped with two Intel® Xeon® E5-2640 v3 2.60GHz CPUs and 96GB of RAM. Here, our approach is run with 10 nodes, 5 cores and 10 GB of RAM per executor and 20 GB of RAM per driver. Signal extraction is performed on massive traces with billions of rows. Results are shown in Table 6. To extract signals, the in-house tool requires to ingest signals to process them while performing interpretation on ingest. Thus, we measure the time for ingest as extraction time while, for our approach, we measure the time it takes to perform interpretation followed by writing the results to the database. As input to both methods, we provide the same amount of traces to extract a fixed number of signals, which is the second column in Table 6. We evaluate the execution times in both methods for traces of various sizes. Per domain usually between 9 and 100 signals are extracted. Thus, we compare the approaches within this range. We can see that for 89 extracted signals from 12 journeys a reasonable extraction time of 269.65 minutes is required while the existing solution requires 504.27 min giving a speedup of factor 1.8. For less signals extraction time is within 87.62 min for 9 signals while the existing tool requires 504.27 min for the same task which shows that extraction of signals with our framework is about 5.7 times faster here. The existing approach requires to loop through all data points in order to determine relevant signals. Thus, extraction time scales linearly with rows to interpret. This extraction time does not change with the number of extracted signals as extraction is done within one loop. For processing of single journeys, this is sufficient. However, when extracting signals from massive traces (i.e. multiple journeys) this becomes inefficient and distributed approaches become essential. Thus, our distributed approach results in lower processing times and reasonable results for massive traces, thus, outperforming existing solutions.

6 CONCLUSION

We presented a distributable general end-to-end preprocessing framework that allows to preprocess massive in-vehicle traces paving the way for automated large-scale analyses of traces from vehicle fleets. It allows for parameterization of constraints and extensions to extract domain-specific knowledge. Being designed extensible and for parallel Big Data frameworks, it allows to be used as basis for subsequent Data Mining, which is required in growing massive traces as specification and visual verification become infeasible. It includes a homogenization approach for signal content that enables further Data Mining on the resulting format. Those characteristics allow to lower testing costs using our framework. We evaluated our framework in terms of execution time and showed that even in large traces relevant signals are extracted in reasonable time. We showed that our approach outperforms existing solutions in terms of time for signal extraction for massive traces.

ACKNOWLEDGMENT

This work was supported by the BMW Group. With the support of the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

REFERENCES

[1] Puneet Agarwal, Gautam Shroff, Sarmimala Saikia, and Zaigham Khan. 2015. Efficiently discovering frequent motifs in large-scale sensor data. In *Proc. of IKDD CODS 2015.*

[2] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. 2002. Pinpoint: Problem determination in large, dynamic internet services. In *Proc. of DSN 2002.*

[3] Gerald Combs et al. last modified: 2007. Wireshark. *Web page: http://www.wireshark.org/* (2007).

[4] Jacob A Crossman, Hong Guo, Yi Lu Murphey, and John Cardillo. 2003. Automotive signal fault diagnostics-part I: signal fault analysis, signal segmentation, feature extraction and quasi-optimal feature selection. *IEEE Transactions on Vehicular Technology* 52, 4 (2003), 1063–1075.

[5] Codemanufaktur GmbH. last modified: 2017. CARMEN Analyzer. *Web page: https://codemanufaktur.com/projekte/bmw- car-measurement-environment/* (2017).

[6] Hong Guo, Jacob A Crossman, Yi Lu Murphey, and Mark Coleman. 2000. Automotive signal diagnostics using wavelets and machine learning. *IEEE transactions on vehicular technology* 49, 5 (2000), 1650–1662.

[7] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2001. An online algorithm for segmenting time series. In *Proc. of ICDM 2001.*

[8] Wenchao Li, Alessandro Forin, and Sanjit A Seshia. 2010. Scalable specification mining for verification and diagnosis. In *Proc. of the DAC 2010.*

[9] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2004. A symbolic representation of time series, with implications for streaming algorithms. In *Proc. of SIGMOD DMKD 2004.*

[10] Alexander V Mirgorodskiy, Naoya Maruyama, and Barton P Miller. 2006. Problem diagnosis in large-scale computing environments. In *Proc. of SC 2006.*

[11] Ahmed Nassar, Fadi J Kurdahi, and Salam R Zantout. 2016. Topaz: Mining high-level safety properties from logic simulation traces. In *Proc. of DATE 2016.*

[12] Rune Prytz, Sławomir Nowaczyk, Thorsteinn Rögnvaldsson, and Stefan Byttner. 2015. Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data. *Engineering applications of artificial intelligence* 41 (2015), 139–150.

[13] Yilu Zhang, Gary Gantt, Mark Rychlinski, Ryan Edwards, John Correia, and Calvin Wolf. 2009. Connected vehicle diagnostics and prognostics, concept, and initial practice. *IEEE Trans. on Reliability* 58, 2 (2009).