Fault-driven stress testing of distributed real-time software based on UML models


**Published in:**
Software Testing Verification and Reliability

**Document Version:**
Peer reviewed version

**Queen's University Belfast - Research Portal:**
[Link to publication record in Queen's University Belfast Research Portal](https://www.qub.ac.uk/researchportal)

**Publisher rights**
Copyright 2020 Wiley. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

**General rights**
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date: 27. Nov. 2020
Fault-driven Stress Testing of Distributed Real-Time Systems based on UML Models

Vahid Garousi
Software Engineering Research Group (SERG)
www.serg.ucalgary.ca
Department of Electrical and Computer Engineering, University of Calgary
2500 University Drive NW, Calgary, AB Canada T2N 1N4
vgarousi@ucalgary.ca

Abstract. In a previous work, we reported and experimented with a stress testing methodology to detect network traffic-related Real-Time (RT) faults in Distributed Real-Time Systems based on the design UML model of a System Under Test (SUT). The stress methodology, referred to as Test Location-driven Stress Testing (TLOST), aimed at increasing the chances of violations in RT constraints associated with a given stress test location (a network or a node under test). As we demonstrate and experiment in this article, although useful to stress different test location-, but TLOST does not guarantee to target (test) all RT constraints in a SUT. This is because the durations of message sequences bounded by some RT constraints might never be exercised by TLOST. To address the above limitation of TLOST in not being able to target all possible RT faults in a SUT, we propose in this work an extended stress test methodology, referred to as Real-Time Fault-driven Stress Testing (RTFAST), which guarantees to target (test) all RT constraints in a SUT and detect their potential RT faults. Using a case study, we show that RTFAST is capable of targeting (and possibly revealing) the RT faults not detected by our previous methodology (TLOST).

Keywords. Stress testing, model-based testing, fault-driven testing, distributed systems, UML, network traffic.

1 INTRODUCTION

Distributed Real-Time Systems (DRTS)¹ are becoming more important to our everyday life. Examples include command and control systems, aircraft aviation systems, robotics, and nuclear power plant systems [1].

Sources of failures in the United States Public Switched Telephone Network (PSTN), as a very large DRTS, are investigated in [2]. It is reported that in the 1992-1994 time period, although only 6% of the outages were overloads, they led to 44% of the PSTN’s service downtime. In the system under study, overload was defined as the situation in which service demand exceeds the designed system capacity. So it is evident that although overloads do not happen frequently, the failure resulting from them can be quite expensive.

¹ For reading convenience, the glossary of acronyms is provided in the appendix.
Therefore the motivation for our work can be stated as follows: Because DRTSs are by nature concurrent and can be data-intensive [1], there is a need for methodologies and tools for testing and debugging DRTSs under stress conditions such as heavy user loads and intense network traffic. These systems should be tested under stressing conditions before being deployed in order to assess their robustness to distribution- and network-specific problems. In this work, our focus is on network traffic, one of the fundamental distribution-specific factors affecting the behavior of DRTSs.

Distributed nodes of a DRTS regularly need to communicate with each other to perform system functionality. Network communications are not always successful and on time as problems such as congestion, transmission errors, or delays might occur. On the other hand, many real-time and safety-critical systems have hard deadlines for many of their operations, where if the deadlines are not met, serious or even catastrophic consequences will happen. Furthermore, a DRTS might behave well with normal network traffic loads (e.g., in terms of amount of data, or number of requests), but the communication might turn out to be poor and unreliable if many network messages or high loads of data are concurrently transmitted over a particular network or towards a particular node.

Since 1997, UML [3] has become the de-facto standard for modeling object-oriented software for nearly 70 percent of IT industry [4]. As we expect UML to be increasingly used for DRTSs, it is therefore important to develop automatable UML model-driven, stress test methodologies.

Assuming that the UML design model of a DRTS is in the form of Sequence Diagrams (SD) annotated with timing information, and the system’s network topology is given in a specific modeling format, we proposed in [5] a methodology to derive test requirement to stress the system with respect to network traffic in a way that will likely reveal robustness problems. We introduced in [5] a systematic methodology to automatically generate an interleaving\(^1\) that will stress the network traffic on a network or a node so as to analyze the system under strenuous but valid conditions. If any network traffic-related failure is observed, designers will be able to apply any necessary fixes to increase robustness before system delivery.

Our methodology in [5], referred to as Test LOcation-driven Stress Testing (TLOST), aimed at increasing the chances of violations in RT constraints associated with a given stress test location (a network or a node under test). As we demonstrate and experiment in this article, while testers can exhaustively apply TLOST on all stress test locations in a System Under Test (SUT), TLOST does not guarantee to target (test)\(\text{\(^1\) A network interaction interleaving is a possible sequence of network interactions among a subset of objects on a subset of nodes.}\)
all RT constraints in a SUT. This is due to how TLOST is designed: it chooses Control Flow Paths (CFPs) which entail the maximum possible traffic on a given network or node, and even if all networks and nodes of a system are stress tested with such an approach, some particular CFPs might never be chosen as stress test cases. In such a case, a few particular RT constraints specified inside those CFPs will never be exercised by TLOST (the limitation is discussed in more detail in Section 4).

To address the above limitation of TLOST, we propose in this work a modified stress test methodology, referred to as Real-Time FAult-driven Stress Testing (RTFAST), which guarantees to target (test) all RT constraints of a SUT with maximum stress. To do so, RTFAST picks specific control flow paths from the UML sequence diagrams of a SUT and also specific test locations as test requirements, which if triggered, will increase the chances of RT faults in a given RT constraint. By using RTFAST, all RT constraints of a SUT can be checked one by one to ensure that they are met in most stressed conditions of a system.

Note that RTFAST is not intended to replace TLOST [5], but instead to complement it, i.e., both RTFAST and TLOST should be used to stress test a DRTS as they each have a different stress testing objective. The goal of TLOST [5] is to increase the chances of RT faults triggered when network traffic in a network or a node is maximized, while RTFAST’s objective is to increase the chances of RT faults in a given RT constraint.

The rest of this article is structured as follows. A survey of related works is presented in Section 2, Section 3 presents a background on real-time constraints. An overview of our TLOST stress test methodology in [5] is presented in Section 4, where we discuss in detail its limitation in targeting all RT constraints. The modified stress test methodology (RTFAST) is presented in Section 5, and is applied in Section 6 to a prototype DRTS. Conclusions and future works are discussed in Section 7.

2 RELATED WORKS

To the best of our knowledge, no existing work addresses the automated derivation of test requirements from UML models for performance stress testing of DRTSs from the perspective of increasing the chances of exhibiting RT faults in given RT constraints. There have not been many works on systematic generation of stress and load test suites for software systems, with the notable exception of [5-9].

Our previous work in [5] is a stress test methodology aimed at increasing chances of discovering RT faults originating from network traffic overloads in DRTSs. The methodology uses the UML 2.0 [3] model of a DRTS, augmented with timing information, and is based on an analysis [10] of the control flow in UML sequence diagrams. It yields stress test requirements that are made of specific CFPs along with time values indicating when to trigger them.
Avritzer and Weyuker [6] propose a class of load test case generation algorithms for telecommunication systems which can be modeled by Markov chains. The proposed black-box techniques are based on system operational profiles\(^1\) [11]. The Markov chain that represents a system’s behavior is first built. The operational profile of the software is then used to calculate the probabilities of the transitions in the Markov chain. The steady-state probability solution of the Markov chain is then used to guide the generation process of the test cases according to a number of criteria, in order to target specific types of faults. For instance, using probabilities in the Markov chain, it is possible to ensure that a transition in the chain is involved many times in a test case so as to target the degradation of the number of calls that can be accepted by the system. From a practical standpoint, targeting only systems whose behavior can be modeled by Markov chains can be considered a limitation of this work. Furthermore, using only operational profiles to test a system may not lead to stressing situations.

Yang proposes a technique [8] to identify potentially load sensitive code regions to generate load test cases. The technique targets memory-related faults (e.g., incorrect memory allocation/de-allocation, incorrect dynamic memory usage) through load testing. The approach is to first identify statements in the module under test that are load sensitive, i.e., they involve the use of `malloc()` and `free()` statements (in C) and pointers referencing allocated memory. Then, data flow analysis is used to find all data Definition-Use (DU)-pairs\(^2\) that trigger the load sensitive statements. Test cases are then built to execute paths for the DU-pairs.

Briand et al. [7] propose a methodology for the derivation of test cases that aims at maximizing the chances of deadline misses in RT systems. They show that task deadlines may be missed even though the associated tasks have been identified as schedulable through appropriate schedulability analysis. The authors note that although it is argued that schedulability analysis simulates the worst-case scenario of task executions, this is not always the case because of the assumptions made by schedulability theory. The authors develop a methodology that helps identify performance scenarios that can lead to performance failures in a system. This stress testing technique uses RT job schedules to find the worst-case scenario test cases and is not based on stress conditions due to network traffic usage.

---

\(^1\) The operational profile of a system is defined as the expected workload of the system once it is operational [11].

\(^2\) A data definition and a data use statement in a source code, where the data use uses the value defined in the data definition [12].
Zhang and Cheung [9] describe a procedure, similar to ours (both [5] and the current work), for automating stress test case generation in multimedia systems. The authors consider a multimedia system consisting of a group of servers and clients connected through a network as a SUT. Stringent timing constraints as well as synchronization constraints are present during the transmission of information from servers to clients and vice versa. The authors identify test cases that can lead to the saturation of one kind of resource, namely CPU usage of a node in the distributed multimedia system. The authors first model the flow and concurrency control of multimedia systems using Petri-nets coupled with timing constraints. A specific flavor of temporal logic is used to model temporal constraints. The following are some of the limitations of their technique: (1) The technique cannot be easily generalized to generate test cases to stress test other kinds of resources, such as network traffic, as this would require important changes in the test model; (2) The resource utilization (CPU) of media objects is assumed to be constant over time, but in the current work we consider variable resource utilization over time by executing each control flow path in a SUT; (3) If the technique in [9] is applied in a UML-based development, it requires additional knowledge (Petri Nets and a specific flavor of temporal logic) which can be an impediment to its use.

3 BACKGROUND ON REAL-TIME CONSTRAINTS

Real-Time (RT) constraints are timing constraints on operations in DRTSs. For example, the specification of a nuclear power plant system might require that an over-heated reactor should be cooled down within 5 seconds, or a catastrophic result will happen.

There are usually two types of RT constraints and RT systems: hard and soft [1]. A hard RT constraint on an operation enforces that the operation must complete within the specified time frame (e.g., 2 seconds) or the operation is, by definition, incorrect, unacceptable, and usually has no value. On the other hand, in the case of a soft RT constraint for an operation, the value of the operation declines steadily after the deadline expires. Tasks completed after their respective deadlines are less important than those whose deadlines have not yet expired [1].

Since our stress testing methodology is in the context of DRTSs and UML-driven development and our goal is to increase the chances of faults in RT constraints, we present next how RT constraints can be specified in UML models. To model RT constraints in UML models, the UML profile for Schedulability, Performance, and Time (UML-SPT) [13] proposes comprehensive modeling constructs to model timing information. Although UML-SPT briefly mentions soft and hard RT constraints (Section 2.2.3 of [13]), it does not propose any specific stereotypes to distinguish between hard and soft RT constraints in UML models.
It should be noted that explicit distinction of soft and hard RT constraints when modeling RT systems can be beneficial since it can help analysts, developers and testers to distinguish between the two types and perform different types of analysis for each of them. For example, stress testing with the intention to find a hard RT fault (violation of a hard RT constraint) is more cost-effective than targeting soft RT faults, since the failure costs due to the former type of faults are generally more than those of the latter.

In order to model hard and soft RT constraints, we proposed in [14] two extensions to the $RTaction$ stereotype of the UML-SPT referred to as $HRTaction$ (hard RT action) and $SRTaction$ (soft RT action). Furthermore, in order to model the statistical threshold probability up to which SRT constraints are allowed to be violated, we defined in [14] a tagged-value referred to as $RTmissProb$ for SRT constraints which ranges in $[0...1]$. In the context of our stress testing methodology, one of the potential benefits of specifying such statistical values for RT constraints is that we might be able to prioritize (order) test objectives (i.e., RT constraints) w.r.t. the allowed missing probability (i.e., criticality) of missing each RT constraint. The extreme values in the range of $[0…1]$ for $RTmissProb$ values, i.e., 0 and 1, denote “that no violation is allowed” and that ironically all violation instances are allowed, respectively. In the former case ($RTmissProb=0$), a SRT constraint turns to be a HRT constraint since any violation is interpreted as RT fault. In the latter case, ($RTmissProb=1$), a SRT constraint is in fact not a constraint since all of its violation instances are allowed and not considered a RT fault.

Similarly, we presented a tagged-value referred to as $RTcriticality$ for HRT constraints, which is a real number in the range of $[0...1]$ indicating the degree to which the consequences of missing a hard RT deadline are unacceptable. The extreme values in the range of $[0…1]$ for $RTcriticality$ values, i.e., 0 and 1, denote virtually no serious consequences and very catastrophic results for violating a HRT constraint, respectively. The closer to one the $RTcriticality$ value of a HRT constraint, the more severe the consequences of missing it.

Example usages of the $SRTaction$ and $HRTaction$ stereotypes in a UML diagrams are presented and discussed in our case study (Section 6).

4 TLOST STRESS TEST METHODOLOGY

An overview of our TLOST model-based stress test methodology is presented using an activity diagram in Fig. 1. Note that only the steps in gray background were addressed by [5].
A UML model of a SUT, following specific but realistic requirements, is used in input. A test model is then built to facilitate subsequent automation steps. The test model and a set of stress test parameters (objectives) set by the user are then used by an optimization algorithm to derive stress test requirements. Test requirements can finally be used to specify test cases to stress test a SUT. The detailed steps of Fig. 1 are described in [5]. To help the reader clearly understand the new RTFAST methodology (Section 5) and its differences with TLOST, we review next TLOST’s heuristics to derive stress test requirements [5].

4.1 HEURISTICS TO DERIVE STRESS TEST REQUIREMENTS

Given a specific network (or node) to stress test, TLOST identifies a message (or a set of messages) in different CFPs of SDs which imposes maximum traffic on the network (or node). We referred to such messages as maximum stress messages [5]. The network traffic generated by each message is estimated using a resource usage prediction technique [5] which is based on the data sizes of parameters or return values carried by each distributed message (sent between two distributed objects).

Using the start times of the maximum stress messages selected in each CFP, the selected set of CFPs can be scheduled in such a way that the maximum stressing messages are all sent concurrently. This concurrent schedule of CFPs will cause a maximum possible traffic on the selected network, which in turn will increase the probability of exhibiting distributed traffic-related faults in the SUT.

We designed TLOST [5] to stress a SUT under the most stressed but valid conditions. One type of such valid conditions is the sequential constraints among SDs of a SUT which define a set of valid SD sequences: e.g., the Login SD of an ATM system should be executed before the Withdraw SD. To formalize those sequential constraints in [5], we defined the notion of Independent SD Set (ISDS) as the maximal set of SDs that can be executed concurrently, i.e., finding as many SDs as possible to be triggered concurrently (to aim at maximum stress) provided that there are no constraints between any two of the
SDs in the set to prevent it. The ISDSs of a SUT are derived from its UML Interaction Overview Diagram (IOD)\(^1\) using an algorithm discussed in [5]. Only SDs (i.e., their respective CFPs) that are members of one ISDS are considered by TLOST to ensure we comply with constraints among SDs.

The above heuristic can be informally visualized by the example in Fig. 2. Given the Network Traffic Usage Patterns (NTUP) of the CFPs of SDs in an ISDS, the heuristic in [5] was to first find the CFP, among all CFPs of a SD, which has a message with the maximum traffic. In Fig. 2, \(NTUP(cfp, net, t)\) is a function [5] which returns the network traffic usage on network \(net\) in time instance \(t\) when the CFP \(cfp\) is executed. In Fig. 2, we assume we have three SDs and the corresponding CFPs are denoted \(CFP_{i,max}\) uniquely identifying SDs. The CFPs are then scheduled such that their maximum stressing messages are sent concurrently (right-hand side diagram in Fig. 2).

\(^1\) Interaction Overview Diagrams (IOD) were introduced as a new UML diagram in UML 2.0 (Section 14.4 of [15]). IODs “define interactions through a variant of activity diagrams in a way that promotes overview of the control flow” [15]. IODs are specializations of UML activity diagrams where object nodes are themselves sequence diagrams.

**4.2 DIFFERENT TEST STRATEGIES**

We investigated in [14] different TLOST test strategies for heavy network workloads focusing on a combination of the following four aspects of DRTSs:

- Stress location (networks or nodes)
- Traffic direction (in and out) which only applies when testing nodes
- Stress type (data traffic or number of requests)

---

\[\text{NTUP}(CFP_{i,max}, \text{net}, t)\]
\[\text{Heuristics}\]
\[\text{A stress test requirement}\]
Stress duration (instant versus interval stress)

The reason why traffic direction (in and out) do not apply in our stress testing approach when testing networks is that networks, unlike nodes, are not the end points of traffic, i.e. any traffic entering a network goes out of it, therefore distinguishing in and out traffic in case of a network is unnecessary.

In other words, the above four aspects indicate the different variations of stress test strategies we can apply on a system of nodes and networks. Example TLOST strategy names, based on a naming convention presented in [14], are: StressNetInsDT, StressNetInsMT, and StressNetPetDT, e.g., StressNetInsDT is one variant designed to identify and to stress test a SUT at the time instant (Ins) when data traffic (DT) on a network (Net) is maximal.

4.3 LIMITATION OF TLOST IN BEING UNABLE TO TARGET (TEST) ALL POSSIBLE RT FAULTS

Although TLOST [5] guarantees to entail maximum stress on a given network or a node, it does not guarantee to target (test) all RT constraints in a SUT. We discuss this limitation of TLOST using a hypothetical example next.

Consider a hypothetical SUT with the UML Interaction Overview Diagram (IOD) presented in Fig. 3, which has two Hard Real-Time (HRT) constraints: HRTC$_1$ and HRTC$_2$. The tagged values RTduration and RTcriticality denote the RT deadline and criticality of the two constraints (as defined in [14]). HRTC$_1$ and HRTC$_2$ enforce a constraint on the durations of SD5 and SD6 (from the beginning of SD5 to the end of SD6), and on the duration of SD10, respectively.

Assume that network traffic of messages in different SDs are such that, no matter which stress test location of this hypothetical SUT is stress tested nor which TLOST stress test strategy is applied, none of CFPs in SD10 is ever chosen as part of a stress test requirement by TLOST. According to the heuristics discussed in Section 4.1 (e.g., Fig. 2), this is possible when the network traffic value of any message in SD10 is less than that of any message in all other SDs.

For example, assume that, without loss of generality, SD10 has only one CFP called cfp1 which has only one message called m1. Further assume that the network traffic values of m1 are less than all the messages in SD9. Thus, between the choices of SD9 and SD10, TLOST always chooses a CFP from SD9 while cfp1 (in SD10) is always left out from the generated stress test requirements.

As a result, the duration of HRTC$_2$ (specified in cfp1) never gets a chance to be exercised (tested) by TLOST, no matter which stress test location or which stress test strategy is chosen. However, for instance, if we could execute a specific CFP of SD10 together with a specific set of CFPs from a specific set of other SDs in a specific schedule, a violation of HRTC$_2$ could have occurred.
In other words, exhaustive stress testing of a SUT with TLOST (with all stress test locations and all test strategies) does not ensure that all possible RT faults are targeted, since the durations of some RT constraints might never be exercised by TLOST. Note that the above discussion does not mean there is a shortcoming in TLOST, but we only discussed a limitation arisen from the way TLOST was designed in the first place [5]. While TLOST guarantees to devise stress test requirements yielding maximum possible network traffic, it doesn’t guarantees to exercise (test) all RT constraints.

As we can see in Fig. 3, \textit{HRTC}_2 has more criticality value than \textit{HRTC}_1 in this hypothetical SUT, which means the former has more critical consequences if it is violated after the system is deployed. Thus, we need to have a stress test methodology other than TLOST to stress test RT constraints not testable (reachable) using TLOST.

Recall from Section 1 that RTFAST is not intended to replace TLOST [5], but instead to complement it, i.e., both RTFAST and TLOST should be used to stress test a DRTS as they each have a different stress testing objective. The objective of the former is to increase the chances of RT faults in a given RT constraint, while the goal of the latter is to increase the chances of RT faults by triggering the maximum possible network traffic (stress) on a specific network or a node. We discussed such a difference above using the hypothetical SUT in Fig. 3.

### 5 Real-Time Fault-driven Stress Test Methodology

To address the above limitation of TLOST, we propose in this work a modified stress test methodology, referred to as \textit{Real-Time FAult-driven Stress Testing (RTFAST)}, which is driven by faults in RT constraints. In other words, given a RT constraint, RTFAST generates a stress test requirement to maximize the chances of RT faults in \textit{that} RT constraint.
To better understand RTFAST, the conceptual relationships and differences between the two stress test methodologies (TLOST and RTFAST) are illustrated using a black-box UML activity diagram in Fig. 4.

Given a stress test location, TLOST derives stress test requirements which target the stress test location in particular, and maximize the network traffic on the node or the network under stress test. It thus maximizes the chances of violating only RT constraints associated with that test location. As we discussed using an example in Section 4.3, even if we use all test strategies of TLOST to stress test all test locations in a SUT, the durations of some RT constraints might never be exercised and their RT faults might stay hidden (undetected).

The above idea is illustrated in Fig. 4 as the stress test requirements generated by TLOST target only a subset of possible RT faults, i.e., even if we stress test with all test strategies of TLOST based on a full coverage on all stress test locations, we can in general only achieve a partial coverage on possible RT faults.

On the other hand, given a RT constraint, RTFAST should derive a stress test requirement which targets the constraint in particular, and maximizes the chances of violating it. If we stress with RTFAST based on a full coverage on all RT constraints, we can achieve a full coverage on all possible RT faults under stressed conditions.

To address the above limitation of TLOST [5] and to achieve the goal of increasing the chances of RT faults in a given RT constraint, we carefully extended RTFAST from TLOST [5]. Similar to different variants of TLOST (Section 4), RTFAST also has different stress test strategies. For brevity and also due to space constraints, we only present one of RTFAST test variants in the following. For the discussion on other RTFAST variants, the reader is referred to [16].

Fig. 4- The conceptual relationships and differences between TLOST and RTFAST.
5.1 Heuristics

We use the Venn diagrams [17] in Fig. 5 to present the heuristics of RTFAST and to illustrate the key difference between TLOST and RTFAST in terms of their test objectives and how they increase the chances of RT faults based on the hypothetical SUT of Fig. 3.

Recall from Section 4 that we defined an Independent SD Set (ISDS) as a maximal set of SDs that can be executed concurrently. Using the algorithm defined in [5] to derive the ISDSs of this SUT, four of ISDSs in this system have been derived and illustrated in Fig. 5: ISDS1, ... ISDS4, e.g., ISDS1 indicates that both SD1 and SD7 can be executed concurrently, since there are no sequential constraints between them.

As discussed in Section 4 and illustrated in Fig. 5, TLOST chooses an ISDS in the SUT which entails maximum possible traffic on a given network or node under test. Two ISDSs have been gray highlighted in Fig. 5-(a), i.e., ISDS1 and ISDS4, denoting that applying two different hypothetical test strategies of TLOST has chosen two different ISDSs in this SUT as the output test requirements.

As we can see in Fig. 5-(a), SD10 has not been chosen in neither of those test requirements since we assume that its maximum stressing messages entail less traffic (on any given network or node) compared to the SDs in ISDS1 and ISDS4. On the other hand, to design a testing methodology (i.e., RTFAST) which addresses this limitation and tests all RT constraints, we have to only stress test those networks and nodes and search in those SDs and CFPs which are related to a given RT constraint. The notion of such a relationship is clarified next.

In general, network traffic on only a subset of all networks in a SUT has an impact on the duration of a RT constraint, and thus, RTFAST only considers such networks. For example, consider HRTC2 in Fig. 3 which is a constraint bound to the duration of SD10. Without loss of generality, assume that this SD has only one CFP and only one message which is transmitted through a particular network. Thus, network traffic on only this network has an impact on the duration of HRTC2, and stress testing other networks will not maximize the chances of violations in HRTC2. Again, assume that the only message of SD10 entails less
traffic on this network compared to other SDs and, for this reason, it has never a chance to be exercised by TLOST.

If a RT constraint (provided as an input to RTFAST) is specified in an IOD (e.g., HRTC2 in Fig. 3), the constraint is bound to the duration of one or several SDs, referred to as bounded SDs (e.g., SD10). In this case, RTFAST searches only among the SDs which are in the same ISDSs with any of the bounded SDs. For example, considering the SUT in Fig. 3 and according to what we discussed above, the maximum stressing messages of SD10 entail less traffic (on any given network or node) compared to the SDs in ISDS1 and ISDS4. However, to target (test) HRTC2, we design RTFAST is a way that it limits its ISDS, SD and CFP search spaces to, respectively, only ISDSs which include SD10, SDs which are in the same ISDSs as SD10, and CFPs inside SD10. As illustrated hypothetically in Fig. 5, by applying RTFAST on this hypothetical SUT to test HRTC2, ISDS2 is chosen as the test requirement.

5.2 A RTFAST STRESS TEST STRATEGY

Due to space constraints, we present in this article only one RTFAST test variant targeting RT constraints specified in IODs, i.e., RTFAST.IO.D.Net.Ins.DT(RTC), a test strategy designed to identify test requirements to stress test the networks (Net) of a SUT with respect to data traffic (DT) at a time instant (Ins) so that the chances of revealing RT faults in a given RT constraint specified in a IOD is maximal. Note that, as we discussed in [16], using RTFAST to test a RT constraint is performed slightly differently for RT constraints specified in SDs and IODs which stems from how each of the two variants search and filter messages, CFPs and SDs to choose stress test requirements.

We present a pseudo code for the above test strategy in Fig. 6, where RTC is a given IOD-level RT constraint. The algorithm is derived by modifying the StressNetInsDT strategy from TLOST [5]. SDs(RTC), Nets(RTC) and GetNetworkPaths(source, target) in Fig. 6 are utility functions and are discussed next.

SDs(RTC) returns the set of SDs bounded by an IOD-level RT constraint. For example, considering the IOD in Fig. 3, SDs(HRTC1) returns [SD5, SD6]. Nets(RTC) returns the set of networks which messages of SDs in the set SDs(RTC) go through. For example, consider the details of SD10 and the network topology shown in Fig. 7 for the SUT discussed in Section 4. In this case, Nets(HRTC2) returns {network1, network3} since messages of SD10 go through those networks, i.e., the communication of nodes n1, n2 and n3 is possible through those networks.

GetNetworkPaths(source, target) in Fig. 6 is a function (defined in [14]) which returns the network paths between a source and a target node in a distributed system topology. Note that, in general, there can be several network paths between those nodes.
The outermost loop of Fig. 6 (Step 1) limits the search to only networks in \( \text{Nets RTC} \). The loop in Step 1.1 searches for maximum stressing CFPs of SDs which are in the same ISDSs with any of the bounded SDs, the set of \( \text{SDs RTC} \).

**Function RTFAST.IOD.Net.Ins.DT(RTC): StressTestRequirement**

1. For each network \( \text{net} \) in \( \text{Nets RTC} \)

   // Find maximum stressing CFP of each SD on \( \text{net} \)

   1.1. For each SD where SDs and SDs RTC are in the same ISDSs

      1.1.1. For each CFP \( \rho_{ij} \) of SD

         \[
         \text{MaxNetInstDTValue}(\rho_{ij}, \text{net}) = \max\{\text{NetInstDT}(\rho_{ij}, \text{net}, t)\}
         \]

      \[
      \text{MaxNetInstDTTime}(\rho_{ij}, \text{net}) = t_{\text{max}} \mid t_{\text{max}} \text{ makes } \text{NetInstDT}(\rho_{ij}, \text{net}, t) \text{ maximum}
      \]

      1.1.2. Among all CFPs of SD/s, find the CFP with maximum stress value:

         \[
         \text{MaxNetInstDTCFP}(\text{SD}, \text{net}) = \left\{ \begin{array}{ll}
         \rho_{\text{max}} & \text{if } \rho_{\text{max}} \in \text{CFP(} \text{SD}) ; \\
         \rho_{\text{max}} & = \text{MaxNetInstDTValue}(\rho_{\text{max}})
         \end{array} \right.
         \]

2. Find the ISDS (Independent-SD Set) and network with maximum impact on RTC

2.1. For each network \( \text{net} \) in \( \text{Nets RTC} \)

   2.1.1. For each ISDS, such that any of SDs in SDs RTC are in ISDS

      \[
      \text{MaxNetInstDTValue(ISDS, net)} = \sum_{\text{SD} \in \text{ISDS}} \text{MaxNetInstDTValue(MaxNetInstDTCFP(} \text{SD}, \text{net}) , \text{net})
      \]

2.2 Find the maximum \( \text{MaxNetInstDTValue(ISDS, net)} \) and refer to the selected ISDS and network as \( \text{ISDS max} \) and \( \text{net max} \)

3. Schedule the SDs of \( \text{ISDS max} \) in the same way as TLOST

4. Return all \( \text{StressTest Schedule} \ = (\rho_{\text{max}} , \text{net}_{\text{max}}) \) and \( \text{net}_{\text{max}} \) as outputs.

**Fig. 6- Pseudo-code of the RTFAST.IOD.Net.Ins.DT test strategy.**

In searching for maximum stressing CFPs, two intermediate matrices \( \text{MaxNetInstDTValue} \) and \( \text{MaxNetInstDTTime} \) are calculated (Step 1.1.1). The former is the maximum instant data traffic value on the network search counter \( \text{net} \) by executing the CFP search counter \( \rho_{ij} \). The latter is the time instant in which the above maximum instant data traffic value is entailed. Both those values are calculated using
NetInsDT(\(\rho, \text{net}, t\)), one of our UML-based resource usage prediction functions [5], which returns the estimated amount of network traffic on network \(\text{net}\) on time \(t\) by triggering CFP \(\rho\). Step 2 of the pseudo-code finds the ISDS and the network with maximum impact on the given RT constraint, i.e., executing which ISDS will entail the maximum stress on which network while impacting the duration of the given constraint the most.

Once we find the \(\text{ISDS}_{\text{max}}\) which has the maximum impact on the duration of the given constraint (in Step 2.2), we schedule the SDs of \(\text{ISDS}_{\text{max}}\) in the same way as TLOST (right-hand side of Fig. 2).

Since the test requirement search phase of RTFAST is carefully designed to target a given RT constraint, stress testing using the RTFAST.IO.D.Net.Ins.DT strategy will make sure that the duration of a given IOD-level RT constraint is exercised and the chances of its violation is increased.

6 CASE STUDY

Our TLOST stress test methodology was used in [5] to stress test SCAPS, a prototype SCADA-based Power System (e.g., [18, 19]). This system controls the power distribution grid across a nation consisting of several provinces, composed of cities and regions. Each city and region has several local power distribution grids, each with a Tele-Control unit (TC), which gathers the grid data and can also be controlled remotely.

There is a nation-wide central server in SCAPS, and each province has one central server that gathers the SCADA data from TCs from all over the province and sends them to the central server. The central server performs the following real-time data-intensive safety-critical functions as part of the power application software [20]: (1) Overload monitoring and control, (2) Detection of separated power systems and (3) Power restoration after network failure. We designed SCAPS to be used in Canada. To simplify the design and implementation of a prototype version of SCAPS, we considered only two Canadian provinces in the system: Ontario (ON) and Quebec (QC). The complete UML design model of SCAPS is presented in [14]. We discuss next the main parts of the design model used in this article.

The UML model was defined and the system was implemented using Borland Delphi [21], which is a well-known Integrated Development Environment for rapid application development.

Based on the system’s business logic derived from the SCADA-based power systems literature (e.g., [18, 19]), we defined for SCAPS ten SD-level and one IOD-level RT constraints. Of those 11 RT constraints, six are soft RT and the remaining five are hard RT constraints. The constraints are defined in the UML models of the system [16] using our extended stereotypes «SRTaction» and «HRTaction» based on the UML profile for Schedulability, Performance, and Time (UML-SPT) [13]. For brevity, we only present two
of those 11 RT constraints in this article: IOD_HRTC in the system IOD (Fig. 8), and HRTC₃ in SD OC (Fig. 9).

As we have discussed in [5, 22], the RT constraints in this system are realistic estimates of message duration times used in SCADA power systems (e.g., [18, 19]). IOD_HRTC in the system IOD (Fig. 8) enforces that the start-up procedure for the overload monitoring (OM_STARTUP SD) should be executed in less than 1000 ms. This SD is a procedure to set up and establish a communication link between the nodes and initialize several buffers to store overload data [14].

Fig. 8- The Interaction Overview Diagram (IOD) of SCAPS.

HRTC₃ (Fig. 9) enforces that all the load data stored in the primary national server should be backed up in the backup national server in less than 1500 ms. According to the SCADA-based power systems literature (e.g., [18, 19]), backing up critical data in such systems is very important. This is done so that, in case of a failure in the primary national server (SEV_CA1), the backup server (SEV_CA2) can continue to control the power system without interruption in service. Blackouts such as the northeast North American blackout on August 2003 have occurred partly due to failures in both primary and backup servers [23].
The objective of our case study was to apply both TLOST and RTFAST methodologies to the case study SUT and compare their ability and effectiveness in detecting RT faults in this SUT. To objectively analyze the effectiveness of TLOST and RTFAST, we also measure the durations for RT constraints of SCAPS when testing (executing) the system based on an operational profile. We considered test cases based on an operational profile to be useful baselines of comparison as they represent a “typical” situation in which the system can be exercised, and are used as a common testing practice to assess a system based on its expected usage in the field [11]. To derive operational profile test cases, we took into account SCAPS’ business logic in the context of SCADA-based power systems and developed an operational profile for this system [14]. For example, overload and power failure situations are expected to be fairly rare in a power grid [20].

To automate derivation of TLOST and RTFAST test requirements, we developed a prototype C tool based on the pseudo-codes of TLOST [5] and RTFAST (Fig. 6). Our tool is referred to as STERT (Stress TEst Requirement Tool for distributed real-time systems) and was implemented similarly to a tool we developed previously: GARUS (GA-based test Requirement tool for real-time distribUted Systems) [24]. Note that the GARUS tool was developed for a variation [25] of the TLOST methodology in which we took into account time arrival patterns (e.g., periodic) for events in DRTSs which have a major impact on the way stress test requirement could be derived [25]. However, the current work (RTFAST) does not take into account event arrival patterns and is thus applicable to only DRTS without arrival patterns.
To investigate TLOST’s ability in targeting RT faults in the SUT, we exhaustively applied all TLOST test strategies (12 for testing nodes and 4 for networks [5]) on all networks and nodes under test in this system.

According to SCAPS network deployment topology (Fig. 10), the system has 20 stress test locations: 7 networks and 13 nodes (shown as rectangles and ellipses in Fig. 10, respectively.)

![SCAPS network deployment topology](image)

Therefore, we used all 12 node stress test strategies of TLOST (e.g., StressNodInInsDT) and all of the 4 network test strategies (e.g., StressNetInsDT) to stress test each of the above 13 nodes and 7 networks, respectively. Note that stress testing here means executing the TLOST or RTFAST stress test cases derived by our STERT tool.

We also followed some specific provisions to be as precise as possible in our stress testing, e.g., (1) to eliminate the impacts of unwanted network activities, we ran the SUT (SCAPS) and also the stress test cases in a dedicated network whose connections to the outside world (Internet) were disconnected, and (2) all the networking applications (except SCAPS) on the involved machines were shut down during our stress tests.

For each pair of test strategies and test location, TLOST stress testing was repeated for 10 times to reduce the effects of randomness as the behavior of a DRTS can change across multiple executions. Application of some of the stress testing strategies resulted in revealing one or more RT faults, while some did not reveal any RT faults.

We consider TLOST as being able to detect a fault in a RT constraint if any of its test strategies revealed a fault during our tests. Based on this definition, Table 1 presents the RT fault detection ability and stress test results of TLOST and RTFAST in which the first column of the table is the list of all 11 RT constraints in this SUT. The first and the second sub-columns under TLOST indicate the name of a TLOST test strategy (if any) which detected a fault in a RT constraint (presented in rows), and a node/network under stress test whose stress testing led to finding a RT fault. Note that, although we observed RT faults of in
some of RT constraint by using several TLOST test strategies, but we only present in Table 1 the name of one such strategy in each row, e.g., one of the TLOST test strategies which helped us detect a RT fault in SRTC$_1$ was StressNodInInsDT when we used it to stress the node SEV$_{ON}$. RTFAST test results are also shown in Table 1 and are discussed after discussing TLOST test results. We discuss next three main observations based on TLOST stress test results in Table 1:

1. Even after exhaustive stress testing using TLOST (applying all its test strategies on all test locations in SCAPS), five of the 11 RT constraints in this SUT are “not tested”, i.e., their duration never had a chance to be exercised. Those five RT constraints are: SRTC$_2$, SRTC$_3$, SRTC$_5$, HRTC$_4$ and IOD_HRTC. This is due to the limitation of TLOST which was described hypothetically in Section 4.C. The reason why, e.g., SRTC$_2$ (a RT constraint in Fig. 9) was never tested (exercised) by applying TLOST was that the keepOldLoadPolicy() message, the message bound by SRTC$_2$, has a very small data size since it has no parameter. This reason led to the a situation where keepOldLoadPolicy() was never part of a TLOST test requirement, and thus SRTC$_2$ was never exercised. Therefore, we consider TLOST unable to test the SUT w.r.t. those five RT constraints, a limitation which is expected to be addressed by testing using RTFAST (discussed in the next page).

2. We observed violations in three constraints: SRTC$_1$, SRTC$_4$, and HRTC$_2$. Since several TLOST strategies caused a RT fault in each of those three constraints, the name of only one test strategy and one node or a network under stress test that revealed a fault in each of those constraints appears in Table 1, e.g., applying StressNodInInsDT on node SEV$_{ON}$ caused a fault in SRTC$_1$. Therefore, we consider TLOST as being able to test the SUT w.r.t. those three RT constraints and reveal at least a RT fault in those constraints.

3. Exhaustive stress testing of the SUT using TLOST did not reveal any fault in three constraints: SRTC$_6$, HRTC$_1$, and HRTC$_3$, i.e., the duration of messages (or SDs) bounded by these constraints were always less than their corresponding deadlines in all stress tests performed by all test strategies. Although TLOST was able to test the SUT w.r.t. those three RT constraints and did not reveal any RT faults in those constraints, we still attempted to test those RT constraints by RTFAST to assess if they are still met when applying RTFAST.

Furthermore, since we observed violations in three constraints (SRTC$_1$, SRTC$_4$, and HRTC$_2$) during testing with TLOST, there was no need to re-test them again by RTFAST (as shown in the RTFAST column of Table 1). Hence, we only tested the other 8 constraints whose RTFAST results are shown in Table 1 and are discussed next.

Table 1- RT fault detection ability and stress test results of TLOST and RTFAST on SCAPS.

---

19
TLOST was unable to test SCAPS w.r.t. five RT constraints: SRTC_2, SRTC_3, SRTC_5, HRTC_4 and IOD_HRTC. Therefore, we used the RTFAST methodology to aim those constraints, to maximize their chances of being violated, and to assess whether they are met under stressed conditions. Similar to testing with TLOST, we observed values of durations bound to each of those constraints by running test cases derived by using each RTFAST strategy. Also, similar to TLOST test results, since several RTFAST strategies caused RT faults in some of the constraints, the name of only one test strategy that revealed a fault in each of such constraints appears in Table 1, e.g., test strategy RTFAST.SD.Net.Ins.DT revealed a RT fault in SRTC_6.

Further note that there is no column for RTFAST testing in Table 1 showing the node or network under stress test since RTFAST strategies do not take such a test location as its input parameter (Section 5), but rather only a RT constraint. RTFAST test results for each of the 8 constraints (not violated using TLOST) are discussed next.

- **SRTC_2**: The RTFAST.SD.Net.Ins.DT strategy was able to reveal a RT fault in this SRT constraint. This test verdict stems from the fact that the RTmissProb value of this constraint was 0.5 [5], but we observed durations over 500 ms (SRTC_2’s deadline) in more than 50% of test executions.

- **SRTC_3**: No RT fault was detected in this constraint after applying TLOST. In other words, the 500 ms deadline on the duration of keepOldLoadPolicy (Fig. 9) was not violated in more than 50% (its RTmissProb value) of test executions.

- **SRTC_5**: No RT fault was detected in this constraint after applying all RTFAST strategies.

- **SRTC_6**: The RTFAST.SD.Net.Ins.MT strategy was able to reveal a RT fault in this SRT constraint.

- **HRTC_1**: No RT fault was detected in this HRT constraint after applying all RTFAST strategies. In other words, the 500 ms deadline of querying all Ontario TCs and receiving their replies [5] was
never violated in any of the 500 test executions. This means that the 500 ms deadline is a reliable
threshold which can never be violated under stress conditions.

- **HRTC3**: The RTFAST.SD.Nod.Out.Int.DT strategy was able to reveal a RT fault in this HRT constraint.
  Considering that the deadline of this constraint was 1500 ms (Fig. 9), we observed several durations
  over this deadline in the 500 test executions (discussed next). The RT faults denote that the 1500 ms
deadline for this HRT constraint is not a reliable threshold and can be violated under stress
conditions.

- **HRTC4**: No RT fault was detected in this constraint after applying all RTFAST strategies.

- **IOD_HRTC**: The RTFAST.IOD.Nod.Ins.DT strategy was able to reveal a RT fault in this HRT
  constraint.

To demonstrate an example case of exhibiting a RT fault in this system, we discuss the test results of RT
constraint **HRTC3** when running Operational Profile Tests (OPT), RTFAST and TLOST stress test cases.
**HRTC3** (Fig. 9) is one of the hard RT constraints and specifies that all the load data stored in the primary
national server should be backed up in the backup national server in less than 1500 ms.

To minimize the indeterministic behavior of SCAPS (as a RT system), we ran the above three test cases for
a large number of times (i.e., 500). Fig. 11 shows the observed values of the duration bounded by **HRTC3**
when running 500 OPT, 500 TLOST and 500 RTFAST stress tests. The X-axis shows the test type and the
Y-axis the duration in milliseconds. The quantile regions and the histograms of the three distributions are
also depicted, and are reported in Table 2.

Some of the main reasons why testing in distributed environments is prone to indeterminism in message
transmission times are: (1) different delay times in network links and routers, and (2) different load
situations in nodes or networks. Due to such an indeterminism, the duration of distributed messages can
be different across different executions, hence the variance in the distributions of Fig. 11. The 500 ms hard
deadline of **HRTC3** is illustrated by a horizontal bold line in Fig. 11 and all OPT and TLOST test
executions satisfy it. In contrast, it is violated in 98% (490/500) of RTFAST stress test cases. Furthermore,
the differences in average and median value between OPT and RTFAST and TLOST distributions are
large too, illustrating the ability of RTFAST and TLOST test cases to stress the system.

Although testing with TLOST did not reveal any RT faults in three constraints (**SRTC6**, **HRTC1** and
**HRTC3**), testing those RT constraints by RTFAST showed that two of those constraints (**SRTC6** and
**HRTC3**) can be violated when our fault-driven stress testing methodology is used. This is due to the fact
that RTFAST tries to pick specific messages and CFPs and schedule them in such a way that maximum
stress is generated with the goal of increasing faults in specific RT constraints (e.g., **HRTC3** in Fig. 11).
Fig. 11- Duration of the message sequence bounded by $HRTC_3$ when running Operational Profile Test cases (OPT), RTFAST and TLOST stress test cases.

Table 2-Quantiles of the distributions in Fig. 11. Values are in milliseconds.

<table>
<thead>
<tr>
<th>Level</th>
<th>Minimum</th>
<th>10%</th>
<th>25%</th>
<th>Median</th>
<th>75%</th>
<th>90%</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>927.0681</td>
<td>1052.436</td>
<td>1102.419</td>
<td>1161.622</td>
<td>1209.026</td>
<td>1252.693</td>
<td>1415.622</td>
</tr>
<tr>
<td>RTFAST</td>
<td>1489.481</td>
<td>1508.59</td>
<td>1514.468</td>
<td>1520.302</td>
<td>1527.35</td>
<td>1532.609</td>
<td>1551.601</td>
</tr>
<tr>
<td>TLOST</td>
<td>1382.377</td>
<td>1409.363</td>
<td>1417.828</td>
<td>1428.102</td>
<td>1439.061</td>
<td>1448.205</td>
<td>1477.338</td>
</tr>
</tbody>
</table>

7 CONCLUSIONS AND FUTURE WORKS

In a previous work [5], we reported and experimented with a stress testing methodology to detect network traffic-related Real-Time (RT) faults in Distributed Real-Time Systems (DRTSs) based on the design UML model of a System Under Test (SUT). Our methodology, referred to as Test LOcation-driven Stress Testing (TLOST), aimed at maximizing the chances of violations in RT constraints associated with a given stress test location (a network or a node under test).

As we demonstrated and experimented in this article, while testers can exhaustively apply TLOST on all stress test locations in a SUT, TLOST does not guarantee to target (test) all RT constraints in a SUT. In other words, exhaustive stress testing of a SUT with TLOST (given all stress test locations) does not ensure that all possible RT faults are targeted (and detected), since the durations of message sequences bounded by of some RT constraints might never be exercised by TLOST.

To address the above limitation of TLOST in being unable to target (test) all possible RT faults in a SUT, we proposed in this work an extended stress test methodology, referred to as Real-Time FAult-driven Stress Testing (RTFAST), which guarantees to target (test) all RT constraints in a SUT. By using RTFAST, all RT constraints of a SUT can be checked one by one to make sure they are met in most stressed conditions of a system. The new methodology can help testers build high quality and more dependable DRTSs by finding all RT faults before a system is deployed.
We applied the test methodology to a prototype distributed system designed and implemented using a real-world distributed system specification, and described, for that particular system, how the stress test cases are derived and executed using our methodology. The stress test results indicated that RTFAST is significantly more effective at detecting RT faults when compared to test cases based on an operational profile. We also showed that RTFAST is capable of revealing the RT faults which are not detectable by our previous methodology (TLOST).

Some of our future works include: (1) Experimenting with Software Performance Engineering (SPE) [26] methodologies such as our recent Iterative Stress-Test Performance Engineering (ISTPE) technique [27] to eliminate chances of RT faults found by RTFAST; (2) Performing a risk assessment and fault analysis of distributed-type faults; (3) generalizing the RTFAST methodology to target RT faults by stressing other types of resources, e.g., CPU and memory; and (4) Utilizing performance modeling and analysis techniques such as asymptotic bounds analysis (e.g., [28]) to prevent RT faults once we found them. We also plan to stress test more complex DRTSs and investigate the effectiveness of our new test methodology.

**ACKNOWLEDGEMENTS**

This work was supported by a start-up grant from Department of Electrical and Computer Engineering and the Schulich School of Engineering of the University of Calgary. The author was further supported by a discovery grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). The author would like to thank Diwakar Krishnamurthy for his helpful comments and suggestions on the early drafts of this article.

**REFERENCES**


APPENDIX- GLOSSARY OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA</td>
<td>Automatic System Agent</td>
</tr>
<tr>
<td>CFP</td>
<td>Control Flow Path</td>
</tr>
<tr>
<td>DRTS</td>
<td>Distributed Real-Time System</td>
</tr>
<tr>
<td>DT</td>
<td>Data Traffic</td>
</tr>
<tr>
<td>HRT</td>
<td>Hard Real-Time</td>
</tr>
<tr>
<td>IOD</td>
<td>Interaction Overview Diagram</td>
</tr>
<tr>
<td>ISDS</td>
<td>Independent SD Set</td>
</tr>
<tr>
<td>MT</td>
<td>Message Traffic</td>
</tr>
<tr>
<td>NTUP</td>
<td>Network Traffic Usage Pattern</td>
</tr>
<tr>
<td>RT</td>
<td>Real-Time</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control and Data Acquisition System</td>
</tr>
<tr>
<td>SCAPS</td>
<td>a SCAda-based Power System</td>
</tr>
<tr>
<td>SD</td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td>OPT</td>
<td>Operational Profile-based Test</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Constraint</td>
</tr>
<tr>
<td>RTFAST</td>
<td>Real-Time FAult-driven Stress Testing</td>
</tr>
<tr>
<td>SRT</td>
<td>Soft Real-Time</td>
</tr>
<tr>
<td>STERT</td>
<td>Stress TEst Requirement Tool for distributed real-time systems</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>TC</td>
<td>Tele-Control unit</td>
</tr>
<tr>
<td>TLOST</td>
<td>Test LOcation-driven Stress Testing</td>
</tr>
</tbody>
</table>